Івано-Франківський національний технічний університет нафти і газу

Інститут інженерної механіки
Кафедра комп'ютеризованого машинобудування

Бурак Олег Володимирович

УДК 62-52

БАКАЛАВРСЬКА РОБОТА

Система автоматизованого проектування лабораторних стендів для вивчення мехатронних систем

Інженерія мехатронних систем
(назва освітньої програми)

131- Прикладна механіка
(шифр і назва спеціальності)

_____О. В. Бурак

(підпис, ініціали та прізвище здобувача освітнього ступеня)

Науковий керівник Копей Володимир Богданович, д-р. техн. наук, доцент

(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)

Допущено до захисту

Завідувач кафедри

професор_____Панчук В. Г.

(посада) (підпис) (дата) (ініціали та прізвище)

Рецензент

_____

(посада) (підпис) (дата) (ініціали та прізвище)

Робота містить результати власних розробок. Використання ідей, результатів і текстів інших авторів мають посилання на відповідне джерело

м.Івано-Франківськ-2021 рік

Івано-Франківський національний технічний університет нафти і газу

(повне найменування закладу вищої освіти)

Інститут інженерної механіки

Кафедра комп'ютеризованого машинобудування

Освітній рівень - бакалавр

Спеціальність 131-Прикладна механіка

(шифр і назва)

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____
«____» _____20__ року

.З А В Д А Н Н Я
НА БАКАЛАВРСЬКУ РОБОТУ СТУДЕНТОВІ

Бурак Олег Володимирович
(прізвище, ім'я, по батькові)

1. Тема роботи Система керування автоматизованого проектування лабораторних стендів для вивчення мехатронних систем

Керівник роботи Копей Володимир Богданович, д-р. техн. наук, доцент
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом закладу вищої освіти від "10" березня 2021 року № 92/7
2. Строк подання студентом роботи 21.06.2021 р.
3. Вихідні дані до роботи Побудова лабораторних стендів за допомогою моделювання мовою Modelica та Arduino
4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
1.Аналіз компонентів для побудови лабораторних стендів.
2. Принципи проектування лабораторних стендів за допомогою Modelica та Arduino
3. Побудова геометричних моделей універсальних деталей для лабораторного стенда
4. Аналіз працездатності маятника в SolidWorks

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
1. Моделювання сдендів в OpenModelica
2. Лабораторний стенд з машинним зором
3. Моделі деталей у SolidWorks
4. Симуляція маятника у Solidworks
5. Маятник

6. Консультанти розділів роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
1-4	Копей В.Б., проф. каф. КМВ	**10.03.21**	**10.03.21**

7. Дата видачі завдання **10.03.21**

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів бакалаврської роботи	Термін виконання етапів роботи	Примітка

Студент _____ _____
(підпис) (прізвище та ініціали)

Керівник роботи _____ _____
(підпис) (прізвище та ініціали)

Зміст

ВСТУП 8
РОЗДІЛ 1. АНАЛІЗ КОМПОНЕНТІВ ДЛЯ ПОБУДОВИ ЛАБОРАТОРНИХ СТЕНДІВ 10
1.1. Огляд датчиків в мехаронних системах 10
1.2. Огляд актуаторів 13
1.3. Опис платформи Arduino 15
1.4. Опис універсальних деталей #Структор 16
1.5. Алюмінієвий профіль як універсальний ресурс 16
РОЗДІЛ 2 ПРИНЦИПИ ПРОЕКТУВАННЯ ЛАБОРАТОРНИХ СТЕНДІВ ЗА ДОПОМОГОЮ MODELICA ТА ARDUINO 18
2.1 Принципи вивчення мехатронних систем за допомогою Modelica та Arduino 18
2.2 Лабораторний стенд на основі Arduino, Python, OpenCV та scikit-image для вивчення мехатронних систем 32
2.3 Програма для автоматизованого створення та симуляції Modelica-моделей лабораторних стендів 37
РОЗДІЛ 3. ПОБУДОВА ГЕОМЕТРИЧНИХ МОДЕЛЕЙ УНІВЕРСАЛЬНИХ ДЕТАЛЕЙ ДЛЯ ЛАБОРАТОРНИХ СТЕНДІВ 42
3.1 Побудова геометричної моделі планки широкої П-подібної 42
3.2 Побудова моделі кутника 43
3.3 Побудова моделі планка П-подібна 44
3.4 Побудова моделі планки 46
3.5 Побудова моделі короткого кутника 47
3.6 Побудова моделі колеса 48
РОЗДІЛ 4. АНАЛІЗ ПРАЦЕЗДАТНОСТІ МАЯТНИКА В SOLIDWORKS 51
4.1. Аналіз працездатності маятника в SOLIDWORKS Motion 51
4.2. Статичний аналіз моделі кутника в SOLIDWORKS Simulation 56
ВИСНОВКИ 61
ЛІТЕРАТУРА 62
Додаток А - Код Arduino-скетча для мехатронної системи "маятник" 64
Скетч 64

Python-програма 65
Додаток Б - Код програми для лабораторного стенда на основі Arduino, Python, OpenCV та scikit-image 67
Додаток В - Код програми для автоматизованого створення та симуляції Modelica-моделей лабораторних стендів 68
Модуль ModelicaCAD.py 68
Модуль ModelicaCADgui.py 77

АНОТАЦІЯ

кваліфікаційної бакалаврської роботи "Система автоматизованого проектування лабораторних стендів для вивчення мехатронних систем"

Розрахунково-пояснювальна записка: 77 с., 64 рис., 2 табл., 3 додатки, 25 джерела.

Графічна частина: 5 аркушів формату А1.

Розроблено принципи проектування мехатронних лабораторних стендів на основі Arduino та інших недорогих універсальних компонентів, які полягають у використанні мови Modelica для їхнього моделювання. Показано приклад лабораторного стенда на основі Arduino, Python, OpenCV та scikit-image для вивчення мехатронних систем з машинним зором. Розроблено програму для автоматизованого створення та симуляції Modelica-моделей лабораторних стендів, яка основана на побудові морфологічних матриць з різними сполученнями різнотипних компонентів. За допомогою САПР SolidWorks розроблено геометричні параметричні моделі універсальних деталей для побудови механічної частини. Набір деталей містить мінімальну кількість типів деталей, які можуть бути надруковані на 3D принтері або виготовлені з алюмінієвого листа. За допомогою Modelica та SolidWorks розроблено моделі та виготовлено простий лабораторний стенд "маятник", який складається з універсальних компонентів. Розроблено програмне забезпечення для його системи керування. Проведено симуляцію роботи стенда за допомогою Modelica, SolidWorks/Motion та SolidWorks/Simulation, отримано кінематичні та динамічні характеристики, які можуть бути використані для оптимізації параметрів конструкції, а також оптимізовано конструкцію деталей за критеріями маси і напружень.

Студент Бурак О.В.

ABSTRACT

of the bachelor's work "System of automated design of laboratory stands for the study of mechatronic systems"

Paper: 77 pages, 64 figures, 2 tables, 3 attachments, 25 references.

The graphical part: 5 sheets of A1 format.

The principles of designing mechatronic laboratory stands based on Arduino and other inexpensive universal components, which consist in using the Modelica language for their modeling, have been developed. An example of a laboratory stand based on Arduino, Python, OpenCV and scikit-image for studying mechatronic systems with machine vision is shown. A program for automated creation and simulation of Modelica-models of laboratory stands, which is based on the construction of morphological matrices with different combinations of different components, has been developed. With the help of CAD SolidWorks geometric parametric models of universal parts for the construction of the mechanical part are developed. The part set contains the minimum number of part types that can be printed on a 3D printer or made of aluminum sheet. With the help of Modelica and SolidWorks, models of a simple laboratory stand "pendulum", which consists of universal components, have been developed, and the stand has been assembled. Software for its control system has been developed. The stand was simulated using Modelica, SolidWorks/Motion and SolidWorks/Simulation, kinematic and dynamic characteristics, which can be used to optimize the design parameters, were obtained, and the design of parts is optimized according to the criteria of mass and stress.

Student Burak O.V

ВСТУП

Актуальність роботи. Ефективне вивчення мехатроніки потребує наявності різноманітних лабораторних стендів, які доступні для кожного студента. Для створення таких стендів можна використовувати недорогі та універсальні компоненти, зокрема, платформу Arduino, сенсори і актуатори для неї, універсальні деталі для збирання механічних частин мехатронної системи.

Мета і задачі досліджень. Метою роботи є створення принципів проектування таких лабораторних стендів, а також створення компонентів системи їхнього автоматизованого проектування.

Для досягнення мети потрібно:

1.Провести аналіз існуючих компонентів для побудови в лабораторних стендів: платформи Arduino, сенсорів та актуаторів для неї, універсальних деталей для побудови механічної частини.

2.Розробити принципи проектування лабораторних стендів за допомогою Modelica та Arduino.

3.Розробити програму для автоматизованого створення та симуляції Modelica-моделей таких лабораторних стендів.

4.Розробити геометричні моделі універсальних деталей для побудови механічної частини за допомогою САПР SOLIDWORKS.

5.Розробити моделі за допомогою Modelica та SOLIDWORKS простого лабораторного стенда, який складається з вказаних компонентів, і виготовити його.

6.Провести симуляцію роботи стенда за допомогою Modelica, SOLIDWORKS/Motion та SOLIDWORKS/Simulation, отримати кінематичні та динамічні характеристики, а також оптимізувати конструкцію деталей за критеріями маси і напружень.

Об’єкт дослідження - лабораторні стенди для вивчення мехатроніки.

Предмет дослідження - методи і засоби моделювання та проектування лабораторних стендів для вивчення мехатроніки.

Методи дослідження. Для моделювання мехатронних систем використовували OpenModelica та САПР SOLIDWORKS. Для створення автоматизованої системи проектування лабораторних стендів використовували мову програмування Python та мову моделювання Modelica. Для створення геометричних моделей універсальних деталей використовували САПР SOLIDWORKS. Для симуляції напружено-деформованого стану деталей використовували метод скінченних елементів, який реалізований в модулі SOLIDWORKS Simulation.

Опубліковані праці. За результатами досліджень зроблені доповіді на двох міжнародних, одній міжвузівській та одній кафедральній (каф. КМВ ІФНТУНГ) конференціях та опубліковані три тези.

РОЗДІЛ 1. АНАЛІЗ КОМПОНЕНТІВ ДЛЯ ПОБУДОВИ ЛАБОРАТОРНИХ СТЕНДІВ

1.1. Огляд датчиків в мехаронних системах

Датчик (сенсор) – пристрій, який призначений для перетворення інформації, що поступає на його вхід у вигляді деякої фізичної величини, в іншу функціональну величину, для подальшого використання в елементах мехатронних систем [1].

Енкодери можна використовувати в Arduino проектах, коли потрібно вирахувати кут нахилу або точно покрутити сервопривід. Також енкодер можна використовувати для регулювання швидкістю мотора чи крокового двигуна. В деяких приладах енкодер використовують як джойстик, замість того щоб багато раз клацати по кнопкам можна прокрутити енкодер [2].

Види енкодерів:

Абослютні енкодери. Основною характеристикою абсолютних енкодерів - як оптичних, так і магнітних - є число кроків, тобто унікальних кодів на оберт і кількість таких обертів. При цьому не потрібне початкове встановлення «нуля» та ініціалізації. Найпоширеніші типи вихідного сигналу — це паралельний код [2].

Оптичні енкодери. Оптичні енкодери мають жорстко закріплений на валу скляний диск із прецизійною оптичною шкалою. При обертанні об’єкта з закріпленим на ньому диском оптопара зчитує інформацію, а електронна схема перетворює її в послідовність дискретних електричних імпульсів. Абсолютні оптичні енкодери – це датчики кута повороту, де кожному положенню вала відповідає унікальний цифровий вихідний код, який разом з числом обертів є основним робочим параметром пристрою [2].

Магнітні енкодери. Магнітні енкодери з високою точністю реєструються проходженням магнітних полюсів магнітного елемента, що

обертається на малій відстані від чутливого елемента, і перетворюють цю інформацію у відповідних цифровий код [2].

Механічні та оптичні енкодери з послідовним виходом. Мають у своєму складі діелектричний чи скляний диск з нанесеним на нього випуклими, провідними чи непрозорими ділянками, відповідно до конкретної конструкції приладу. Визначення абсолютного кута повороту диска виконується лінійкою вимикачів чи контактів у випадку механічної схеми, або лінійкою оптопар, у випадку оптичної схеми. Вихідні сигнали являють собою код Грея, який дозволяє позбутися неоднозначності інтерпретації сигналу [2].

Рис. 1.1 – Енкодер класичний

Рис. 1.2 – Енкодер удосконалений

Рис 1.3 – Енкодер промисловий

Модуль датчика швидкості. FC-03 – оптичний щілинний датчик (рис 1.4), в який поміщається об'єкт для зчитування кількості і частоти обертань. Оптична пара, є основним елементом датчика. Використовується компактор LM393. У корпус FC-03 вбудовані інфрачервоний світлодіод і фототранзистор, які розташовані протилежно один одному. Коли в щілини між елементами з’являється об’єкт, який здатний стримувати інфрачервоні випромінювання, сигнали від світлодіода перекриваються і транзистор закривається. Якщо транзистор закритий, тоді на виході перетворюються параметри в цифрові

сигнали. Так само можна визначити частоту обертання. Для цього поміщається об’єкт з отворами в щілину датчик, який має виріз в 5мм. При оборотах диска, в прорізі проявляються отвори. Потім датчик перетворює чергування в імпульси, відповідно імпульси в цифрові сигнали [3].

Рис 1.4 – Датчик швидкості

Акселерометр ADXL345. Це крихітний мікропотужний трьохосьовий акселерометр з високою розподільною здатністю (13 біт). Діапазон вимірювання положення коливається до 16g. Результат вимірювання дається у вигляді 16-розрядних чисел в додатковому коді і через цифрові інтерфейси SPI/12 C. Даний акселерометр ADXL345 (рис 1.5) ідеально підходить для визначення прискорення (викликаного гравітацією) в задачах визначення відхилення, або динамічного прискорення, викликаного рухом або ударами. Заявлена висока розподільна здатність акселерометра (4 * 10-3 g / LSB) дозволяє точно відслідковувати зміну відхилення менш ніж на 1.0 градусів. Режим зниженого енергоспоживання датчика дозволяє реалізувати інтелектуальне управління живленням системи [4].

Рис 1.5 – Акселерометр

1.2. Огляд актуаторів

Актуатор (actuator) – виконавчий пристрій регулює або керує системою. У нашому випадку такою системою є робот, і рух його складових відбувається саме завдяки наявності актуаторів. До найбільш поширених можна віднести електричні, гідравлічні і пневматичні актуатори [5].

Електричний привід. До цього типу відносяться електричні двигуни різних видів. Існуюча класифікація двигунів, виділяє дві основні групи: двигуни постійного струму і двигуни змінного струму (рис 1.6). Суть цього поділу впливає з самої назви [5].

Рис 1.6 – Двигуни постійного струму (AC motors)

Серводвигун. Серводвигунами також називаються керовані двигуни (рис 1.7). Вони використовуються в системах управління зворотним зв’язком як вихідні приводи і не використовуються для безперервного перетворення енергії. Принцип роботи сервомотора аналогічний принципу роботи іншого електромагнітного двигуна, але конструкція і операція різні. Їх потужність змінюється від частки ват до декількох сотень ват. Інерція ротора двигунів низька і має високу швидкість реагування. Ротор двигуна має довжину і менший діаметр. Вони працюють на дуже низькій швидкості, а іноді навіть на нульовій швидкості [5-6].

Рис 1.7 - Серводвигун

Застосування серводвигуна. Застосування сервомотора такі:

• Вони використовуються в радіолокаційній системі та контролері процесів.

• Сервомотори використовуються в комп'ютерах і робототехніці.

• Вони також використовуються в верстатах [5, 6].

• Системи відстеження та наведення.

Лінійний ауктуатор. Лінійний актуатор являє собою систему позиціонування, в основі якої лежить перетворення обертального моменту електодвигуна в поступальний рух штока (рис 1.8). Як правило, такий пристрій включає в себе сам двигун, редуктор, датчик повороту ротора двигуна і кінцевий вимикач. Довжина висунутою чистини штока у типових ЛА варіюється від 50мм до 500мм. Швидкість руху штока до 50мм / с в залежності від навантаження. Прикладена приводом до об'єкта сила приймає значення 200Н до 10000Н в залежності від моделі [5] .

Рис 1.8 – Лінійний актуатор

Двигуни змінного струму. Двигуни цього типу, самі по собі, є найдешевшими у виготовленні і в обслуговуванні. Такі електродвигуни в основному застосовуються в промисловості і в побутових приладах, де немає необхідності плавно змінювати швидкість обертання ротора. Двигуни змінного струму поділяються на синхронні і асинхронні. У перших, обертання ротора відбувається згідно з обертанням магнітного поля статора. В асинхронному двигуні такої залежності немає, і часто поле обертається швидше ротора [5].

Крокові двигуни. Для цілей робототехніки нам найбільше цікавий один з видів синхронного двигуна, званий кроковим двигуном (КД, stepper motor). Крокові двигуни дозволяють здійснювати обертання ротора на строго заданий кут без використання датчика кута повороту (рис 1.9). КД можуть бути використані замість серво, оскільки мають досить великий крутний момент і точне позиціонування вала. Подібні приводи використовуються в системах де необхідне позиціювання, таких як наприклад принтери, сканери, плоттери і верстати з ЧПУ (CNC) [5].

Рис 1.9 – Крокový двигун

1.3. Опис платформи Arduino

Arduino – це невелика плата з власним процесором і пам’яттю. На платі також є пара десятків контактів, до яких можна підключати всілякі компоненти: датчики, мотори, роутери, магнітні дверні замки і взагалі все, що працює від електрики.

У мікроконтролер Adruino (рис 1.10) можна завантажувати програму, яка буде керувати всіма цими пристроями за заданим алгоритмом. Таким чином можна створити нескінченну кількість унікальних гаджетів, зроблених своїми руками і за власним задумом [7].

Рис. 1.10 – Плата Arduino

1.4. Опис універсальних деталей #Структор

#Структор – це ґратчастий конструктор (рис 1.11) для швидкої збірки корпусів і механічних вузлів вашого розумного пристрою. Це зручні кріплення для поширеного хобі-електроніки та електромехіки. Це детальки, за допомогою яких можна швидко перетворити грудочку заплутаних проводів в закінчений, охайний пристрій. Саморобка може стати приємним елементом декору [8].

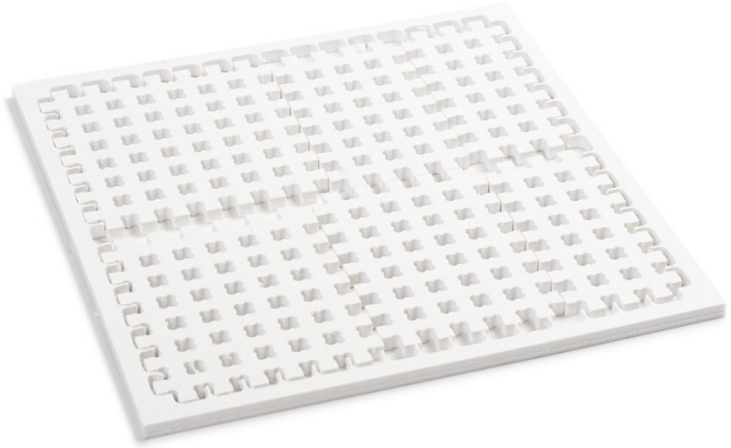


Рис. 1.11 - #Структор

1.5. Алюмінієвий профіль як універсальний ресурс

Ще недавно профіль типу Т-слот (T-slot) був не найпопулярнішим, але після того, як його стали застосовувати в конструкціях багатьох моделей 3D-принтерів, він з’явився по всіх усюдах. Тепер він використовується для складання тих же 3D-принтерів, лазерних різаків, верстатів з ЧПУ. Крім того профіль підходить для виготовлення верстатів, освітлювальних приладів, навіть рамок для фотографій, якщо, звичайно, така задумка виникне.

Говорячи про профілі ми маємо на увазі одночасно кілька типів виробів з алюмінію з різною геометрією. Найчастіше зустрічається профіль у формі квадрата або прямокутника. Найбільш розповсюджений різновид – профіль з квадратним перетином, поздовжнім отвором в центрі і Т-образними пазами

для підключеня самих різних об’єктів. Також є профілі, виготовлені по метричній системі. Називають профіль (в даному випадку квадратний) по його розмірності. Наприклад квадратний профіль (рис 1.12) довжиною 20мм буде називатися профіль 20х20. Офіційно такий профіль назиється <<алюмінієвий верстатний профіль 20х20>> [9]. Для простих деталей можна використовувати профіль, який виготовлений шляхом гнуття алюмінієвого листа.

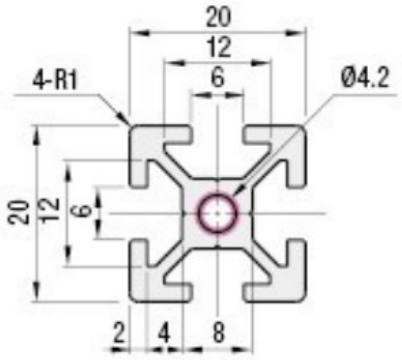


Рис. 1.12 – Профіль

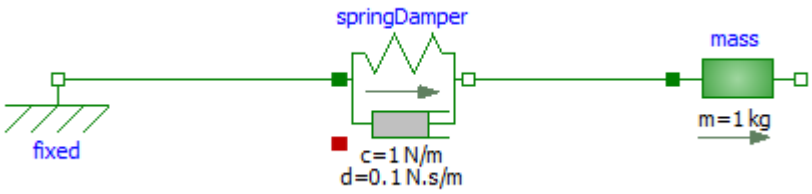
РОЗДІЛ 2 ПРИНЦИПИ ПРОЕКТУВАННЯ ЛАБОРАТОРНИХ СТЕНДІВ ЗА ДОПОМОГОЮ MODELICA ТА ARDUINO

2.1 Принципи вивчення мехатронних систем за допомогою Modelica та Arduino

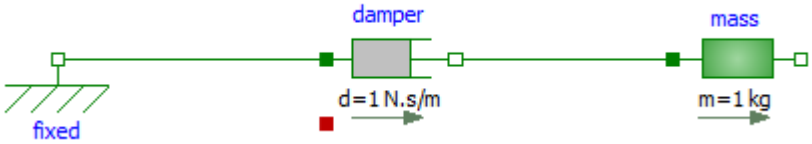
Вивчення мехатроніки потребує ефективних засобів імітаційного моделювання мехатронних систем та недорогих засобів для реалізації різноманітних лабораторних стендів і експериментування з ними. Одними з таких засобів є мова моделювання Modelica та програмно-апаратна платформа Arduino [10].

В розділі описано принципи вивчення мехатроніки за допомогою середовища OpenModelica, бібліотек PlanarMechanics [11] та Arduino [12], деталей для плати Arduino, а також мови програмування Python.

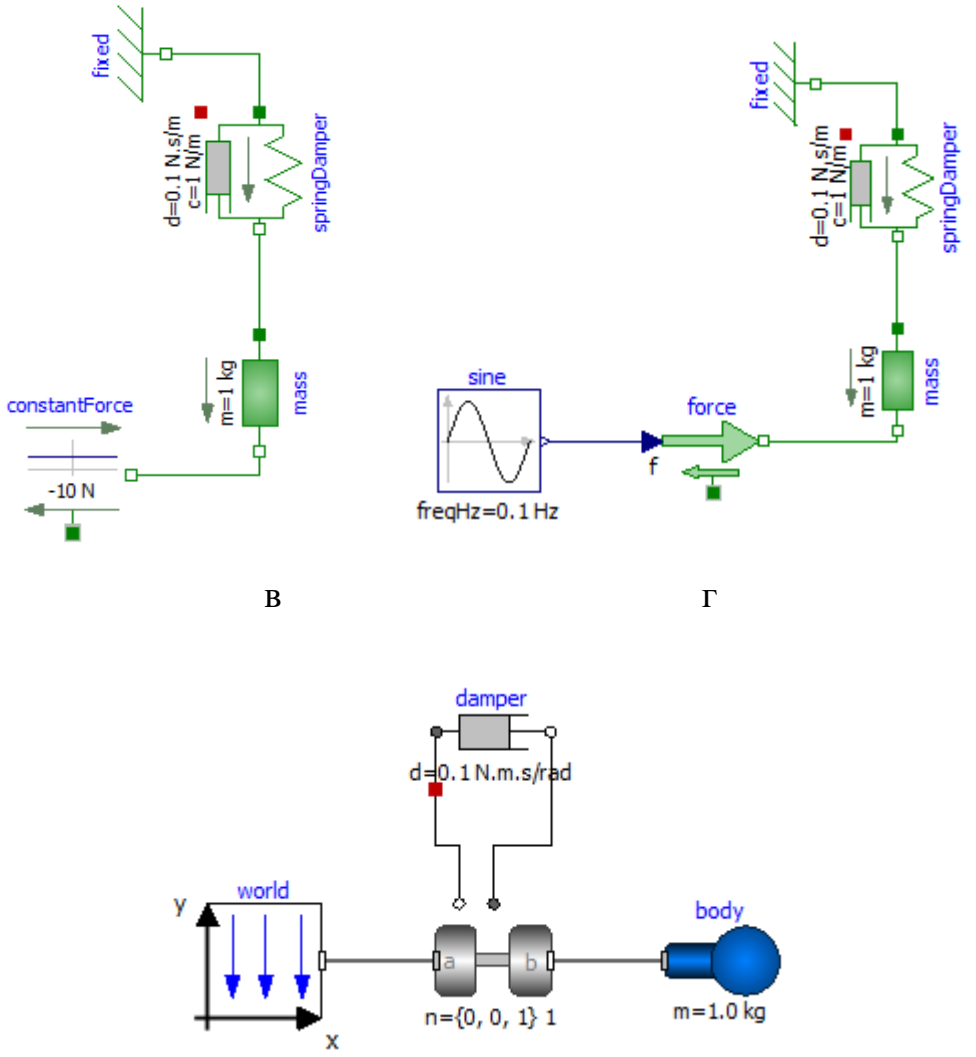
OpenModelica дозволяє легко створювати моделі мультидоменних систем. У першу чергу механічних систем (рис 2.1-а,б,в,г,д). На рис 2.1 показані моделі систем керування і мехатронних систем.



а



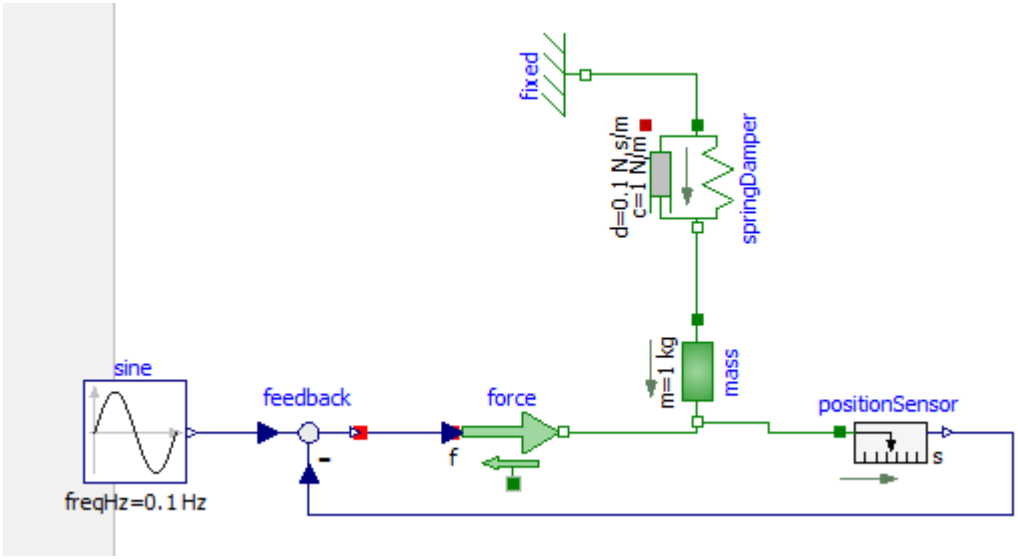
б



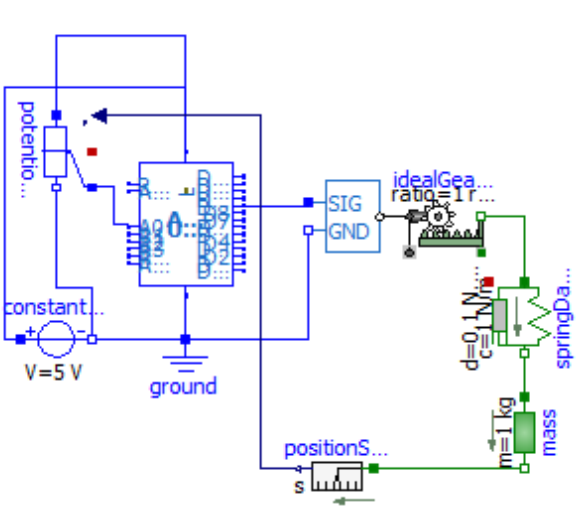
в г

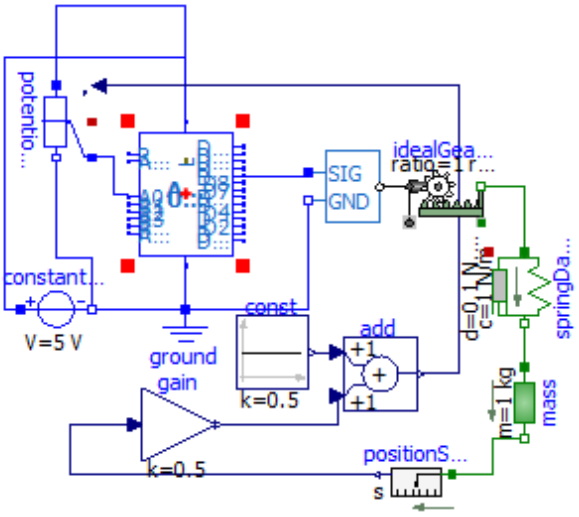
д

Рис 2.1. - Моделі механічних систем: поступальні пружина-демпфер-маса (а), демпфер-маса (б), пружина-демпфер-маса з постійною силою (в), пружина-демпфер-маса зі змінною силою (г), маятник з демпфером (д)



а





б в

Рис 2.2. Моделі мехатронних систем: система керування зі зворотним зв'язком (а), система керування з мікроконтролером: спрощена (б) і удосконалена (в)

На рис 2.3 показано модель простого лабораторного стенда для вивчення мехатронних систем [13]. До цифрового піна D9 Arduino Uno під'єднано сервопривід. До сервопривода під'єднано пружину кручення, а до пружини - компонент диск з моментом інерції. Кут повороту диска визначається за допомогою сенсора кута повороту. Сигнал від сенсора кута повороту передається на компонент, який додає до цього сигналу константу. Після цього

сигнал передається на змінний резистор, який змінює напругу на аналоговому піні A0. Скетч, який виконуються на Arduino Uno, отримує значення на аналоговому піні A0 і в залежності від нього повертає вал сервопривода на заданий кут.

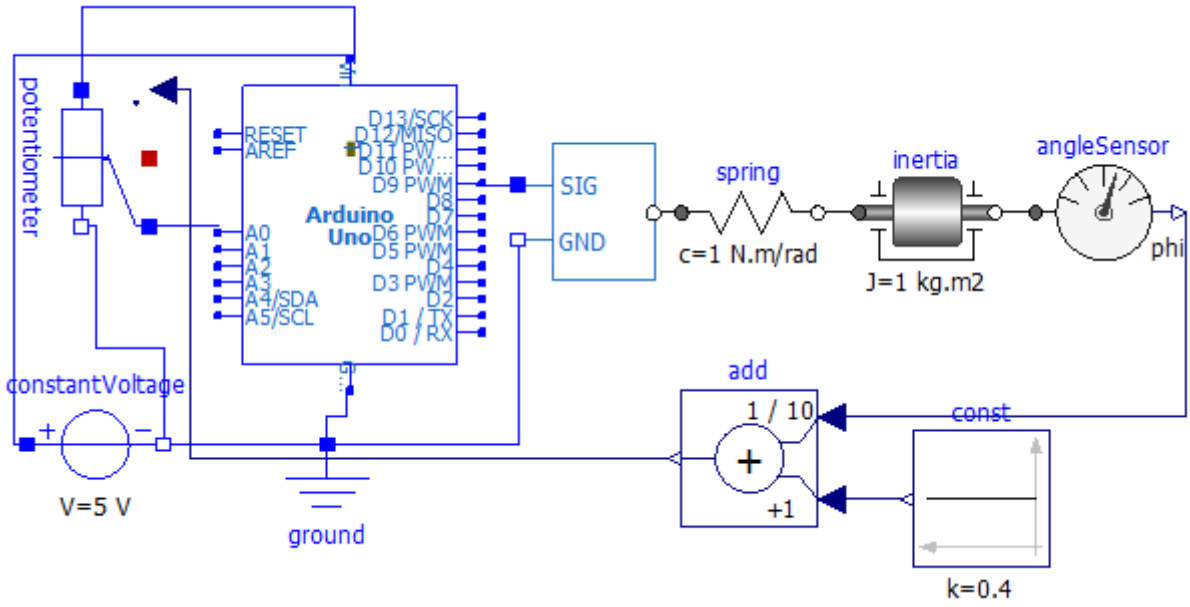


Рис 2.3 - Модель мехатронної системи в OpenModelica

Нижче описано приклад створення лабораторного стенда "маятник" для вивчення мехатронних систем [14]. На рис 2.4 показана модель простого маятника, зроблена за допомогою компонентів PlanarMechanics. На тіло діє сила гравітації. Початкове положення маятника є паралельним до горизонталі. Це спричиняє вільні коливання (рис 2.5).

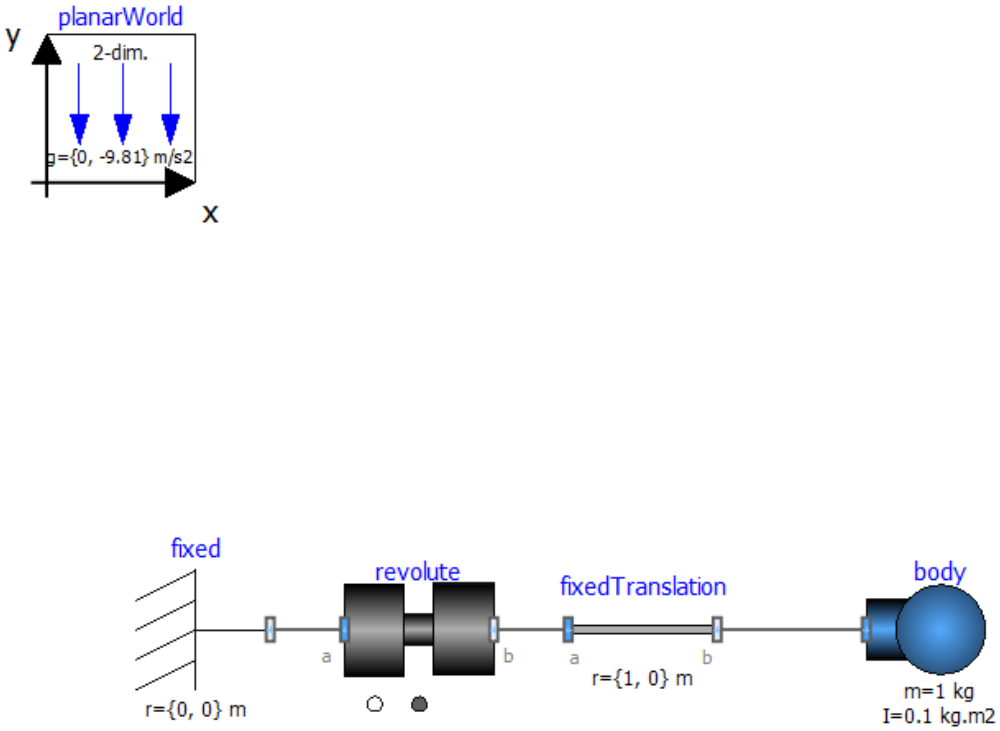


Рис 2.4 - Модель маятника, зроблена за допомогою компонентів PlanarMechanics

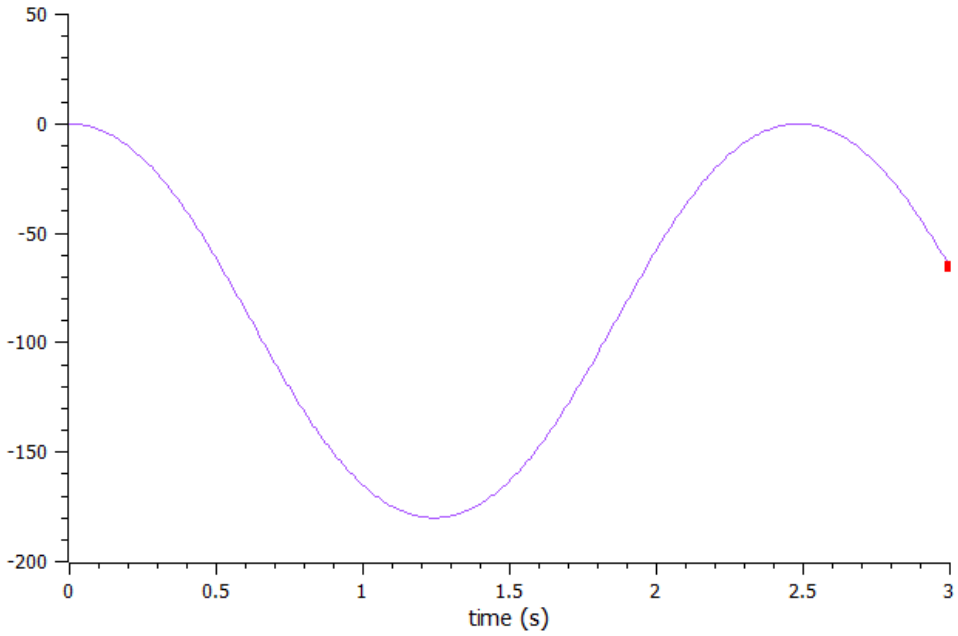


Рис 2.5 - Результати симуляції - кут повороту (град.)

В цю модель було додано зовнішню силу, яка підштовхує маятник за синусоїдальним законом (рис 2.6).

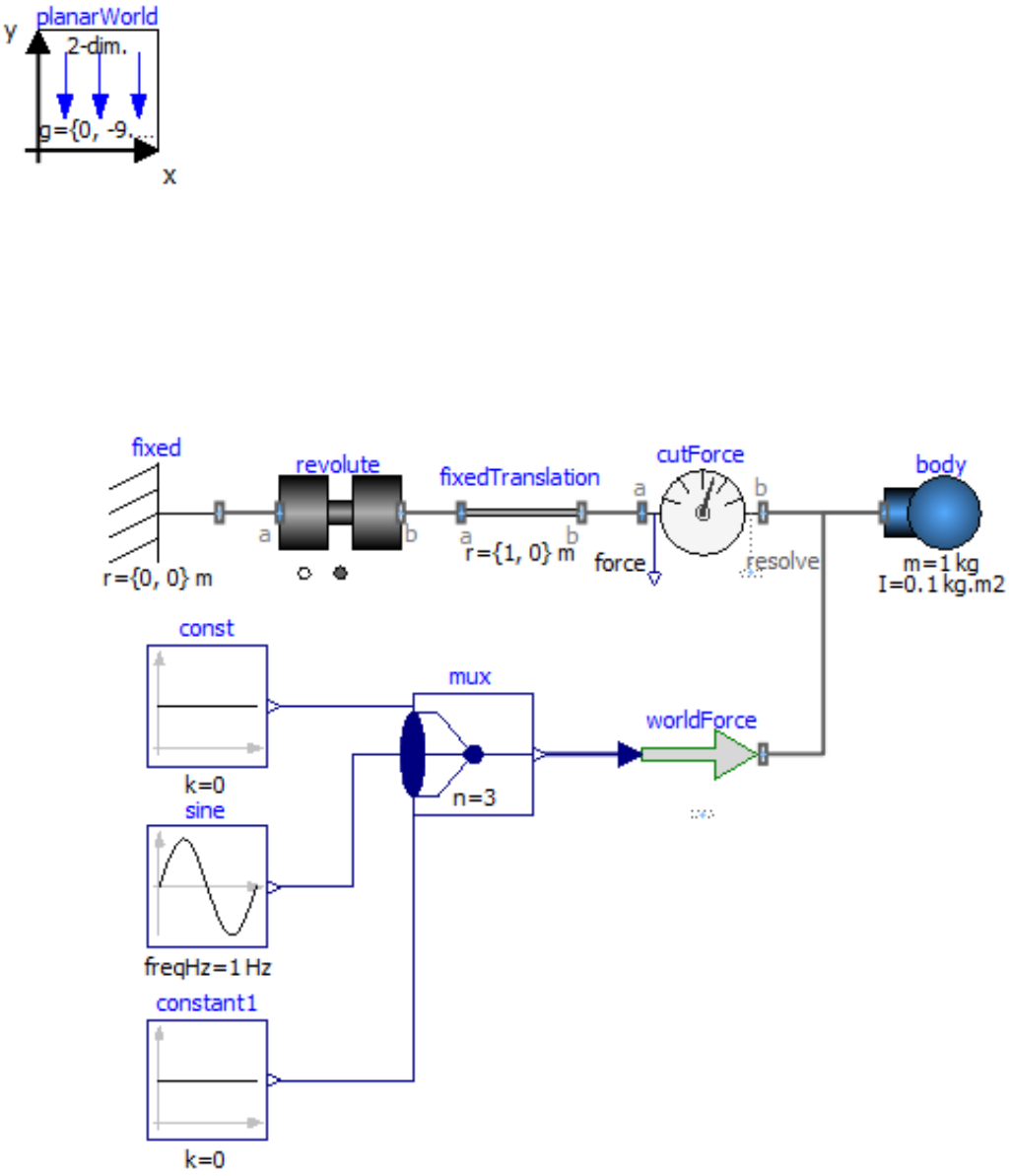


Рис 2.6 - Модель маятника, який підштовхується зовнішньою силою

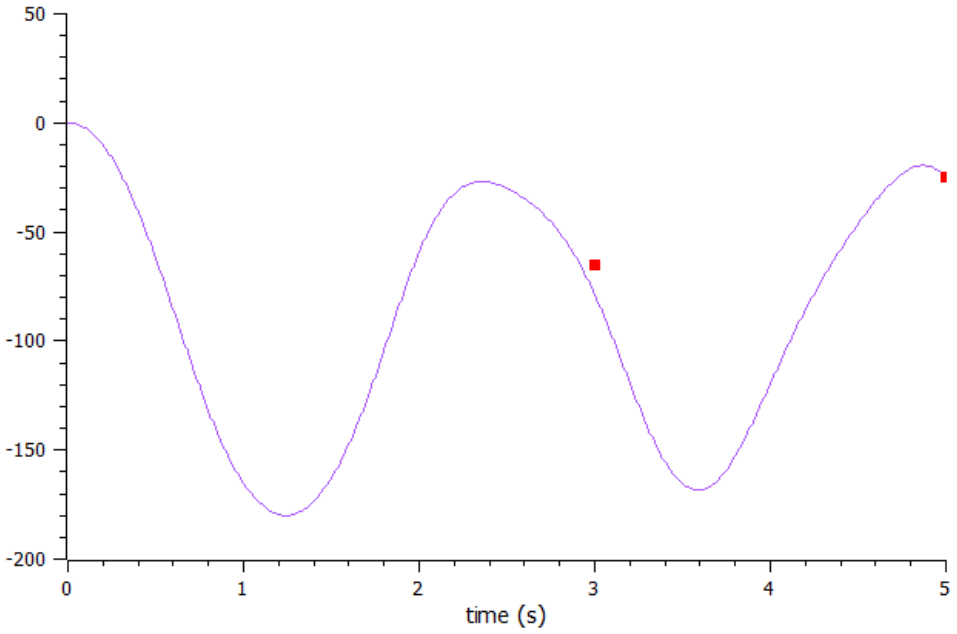


Рис 2.7 - Результати симуляції - кут повороту (град.)

Інший спосіб підштовхування (шляхом обертання фланця шарніра) показаний на рис.2.8.

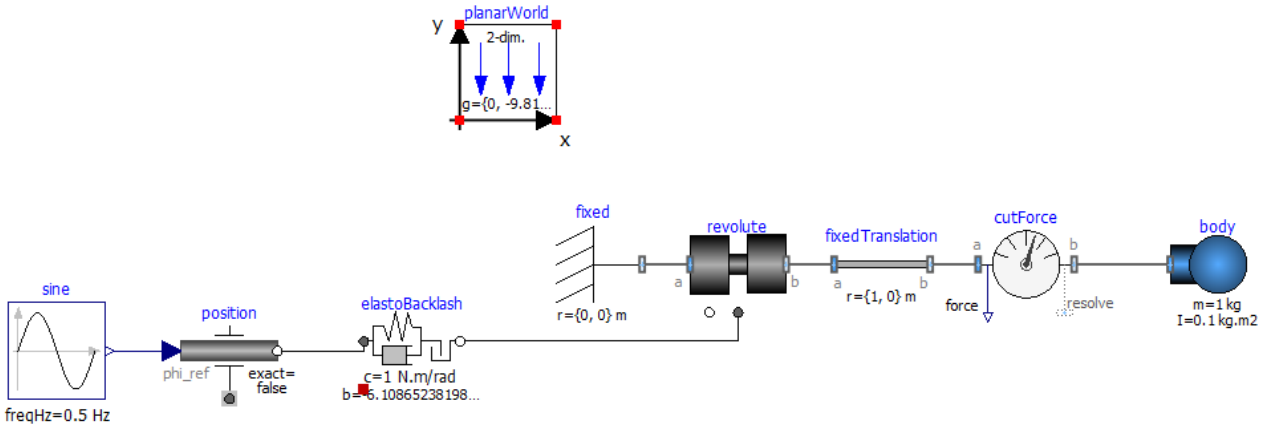


Рис 2.8 - Інший спосіб побудови моделі

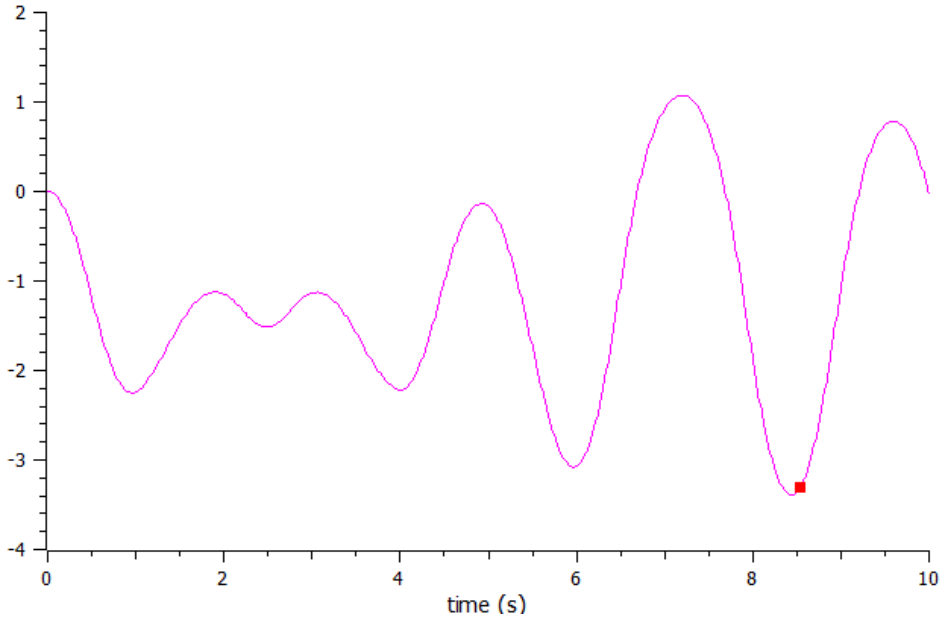


Рис 2.9 Результати симуляції - кут повороту (град.)

На рис 2.10 показано повністю модель простого лабораторного стенда "маятник" для вивчення мехатронних систем [14].

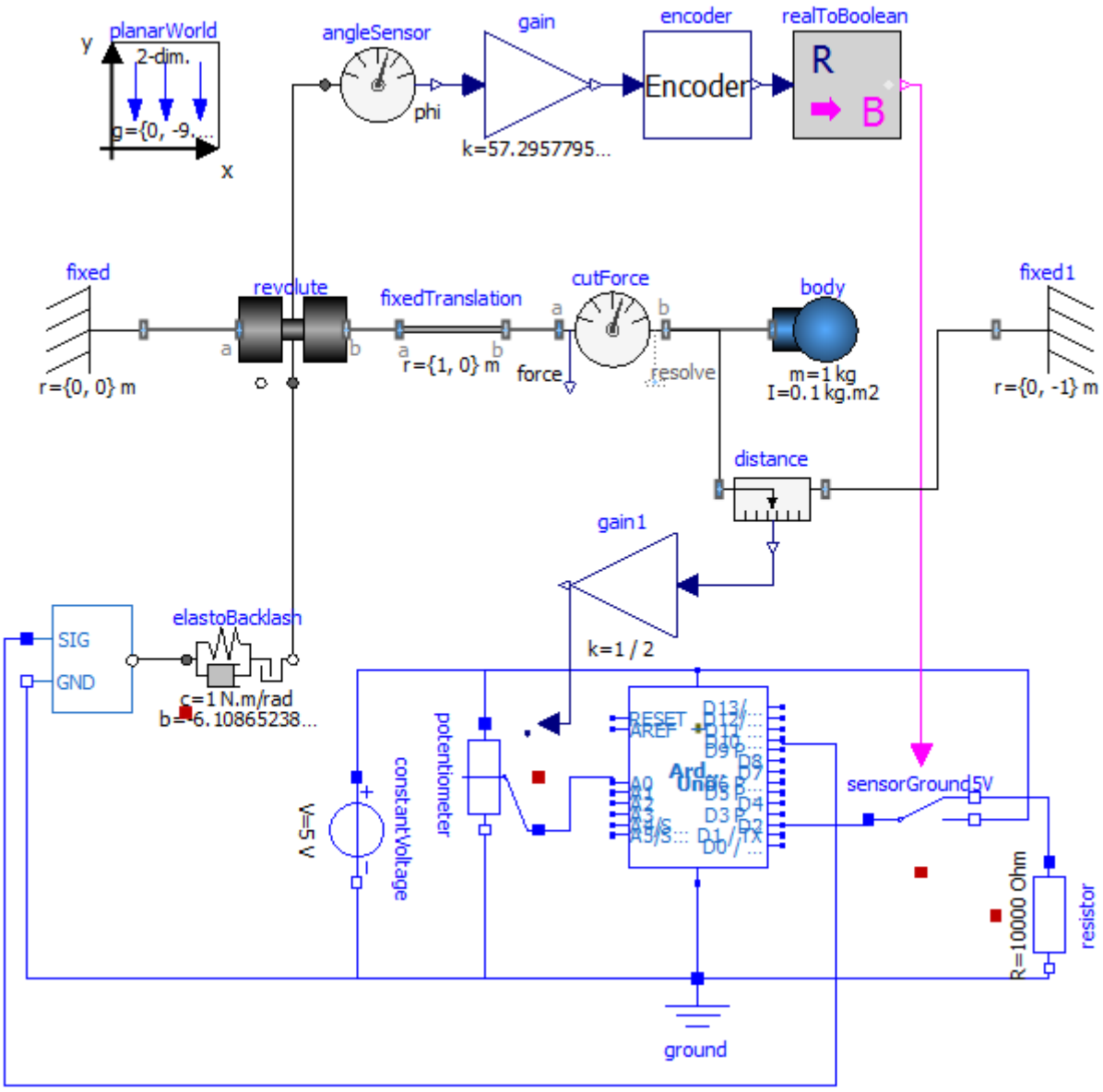


Рис 2.10 - Модель мехатронної системи "маятник"

Механічна частина стенда являє собою маятник. Для побудови моделі маятника використано компоненти бібліотеки PlanarMechanics. Альтернативою можуть бути 3D компоненти MultiBody стандартної бібліотеки мови Modelica, але використання 2D компонентів набагато простіше початківцям. Зауважимо, що OpenModelica 1.16.2 має помилку, яка полягає в несумісності установленої версії Modelica 3.2.3 та PlanarMechanics 1.5. Потрібно завантажити стару версію PlanarMechanics 1.4.1 з офіційного репозиторію і розпакувати в папку OpenModelica1.16.2-64bit\lib\omlibrary\PlanarMechanics 1.4.1. Відкрити цю бібліотеку в OpenModelica Connection Editor можна шляхом перетягування файлу

package.mo на вікно браузера бібліотек. Перед використанням бібліотеки рекомендується ознайомитись з вбудованими прикладами – у першу чергу з прикладом Pendulum (вільні коливання плоского маятника) та PendulumExcited (вимушені коливання плоского маятника). Маятник (рис. 1) будується з компонентів fixed (зафіксований фланець), revolute (шарнір), fixedTranslation (жорстка ланка заданої довжини без інерції), body (тіло з масою та моментом інерції), planarWorld (система координат та поле гравітації). Компонент cutForce не є обов'язковим і може бути використаний для вимірювання сили між фланцями.

Перед студентами можуть бути поставлені різні завдання програмування мікроконтролера – гальмування вимушених коливань, підтримка заданої амплітуди тощо. В даному випадку ціллю системи автоматичного керування є підтримування коливань маятника. Система досягає цієї цілі шляхом визначення положення маятника за допомогою сенсорів та підштовхування маятника за допомогою актуатора, якщо рух відсутній. Використовуються два сенсори – інкрементний обертовий енкодер та сенсор відстані на основі світлодіода та фоторезистора. Модель енкодера складається з компонентів angleSensor (сенсор кута повороту), gain (множення на 57.3 градуса), encoder (перетворює дійсний сигнал кута повороту в булевий), sensorGround5V (перемикач, в залежності від стану якого цифровий порт Arduino D2 отримує 0 або 1). Модель сенсора відстані складається з компонентів distance, який вимірює відстань між body і нерухомим фланцем fixed1 і передає сигнал на potentiometer (керований цим сигналом змінний резистор, середній контакт якого з'єднаний з аналоговим входом A0). Для моделювання актуатора використовується модель сервопривода з бібліотеки Arduino та компонент elastoBacklash (1D обертова пружина-демпфер з зазором), який під'єднаний до 1D обертового фланця шарніра. Застосовується бібліотека Arduino 0.1.0 [12], яка створена для використання в Dymola, тому є певні особливості її використання в OpenModelica 0.16. На комп'ютері повинна бути установлена Microsoft Visual Studio 14. Командний файл \lib\omlibrary\Arduino

0.1.0\Resources\Source\Arduino\build_sketch.bat дозволяє відкомпілювати довільний скетч Arduino. Але у будь-якому випадку шлях до нього повинен бути незмінним: \lib\omlibrary\Arduino 0.1.0\Resources\Sketches\Blink.ino. В результаті компіляції отримується бібліотека ModelicaArduino.dll, яку потрібно скопіювати в папку з моделлю перед стимуляцією. Компонент encoder відсутній в стандартній бібліотеці Modelica, але його можна легко створити шляхом успадкування класу блокового компонента SISO. Для прикладу код моделі encoder може бути таким [15]:

```
block Encoder
  extends Modelica.Blocks.Interfaces.SISO;
  parameter Real angle(start=90);
equation
  if noEvent(mod(u, angle) >= 0 and mod(u, angle) <= angle / 2) then
    y = 1;
  else
    y = 0;
  end if;
end Encoder;
```

Тут для ідентифікації отвору на диску оптичного енкодера використовуються операція остачі від ділення mod. Змінна u – це кут повороту диску, y – цифровий вихід енкодера, параметр angle – кутовий крок отворів. Модель для тестування енкодера з параметром angle=90 та результати її симуляції показані на рис. 2.11-12.

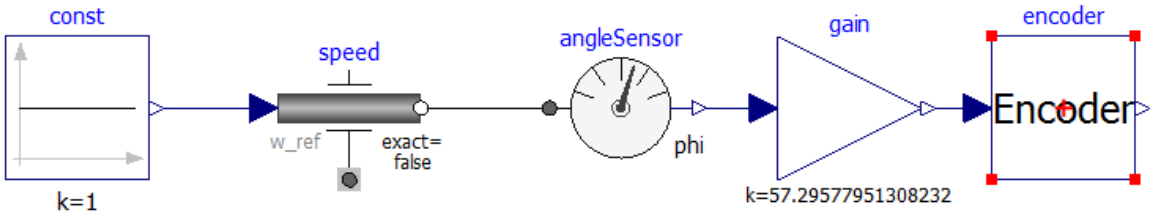


Рис 2.11 - Модель для тестування енкодера з параметром angle=90

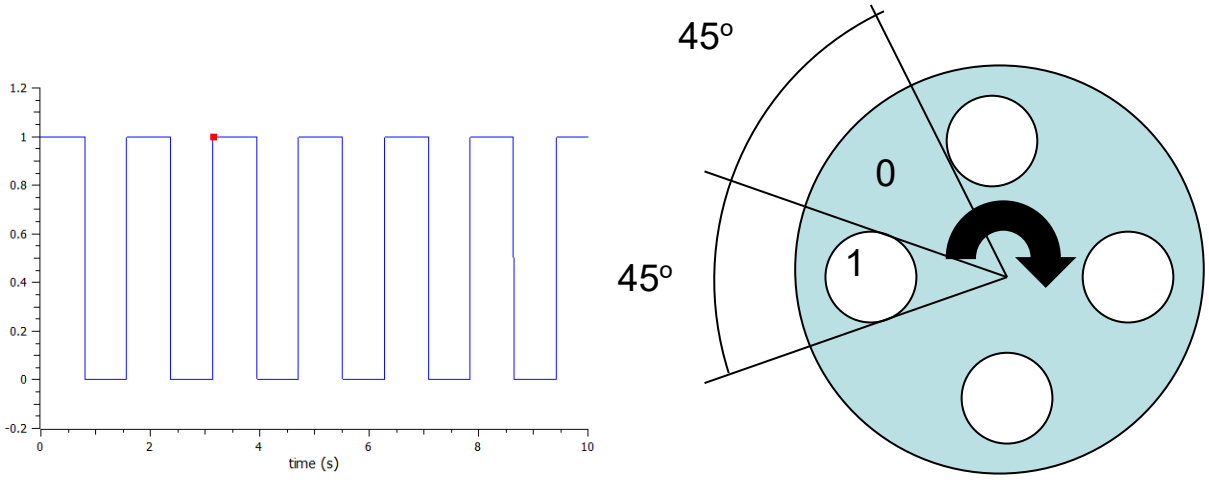


Рис 2.12 - Результати симуляції - сигнал на виході енкодера

Ще одна модель для тестування енкодера з використанням компонента Arduino показана на рис. 2.13

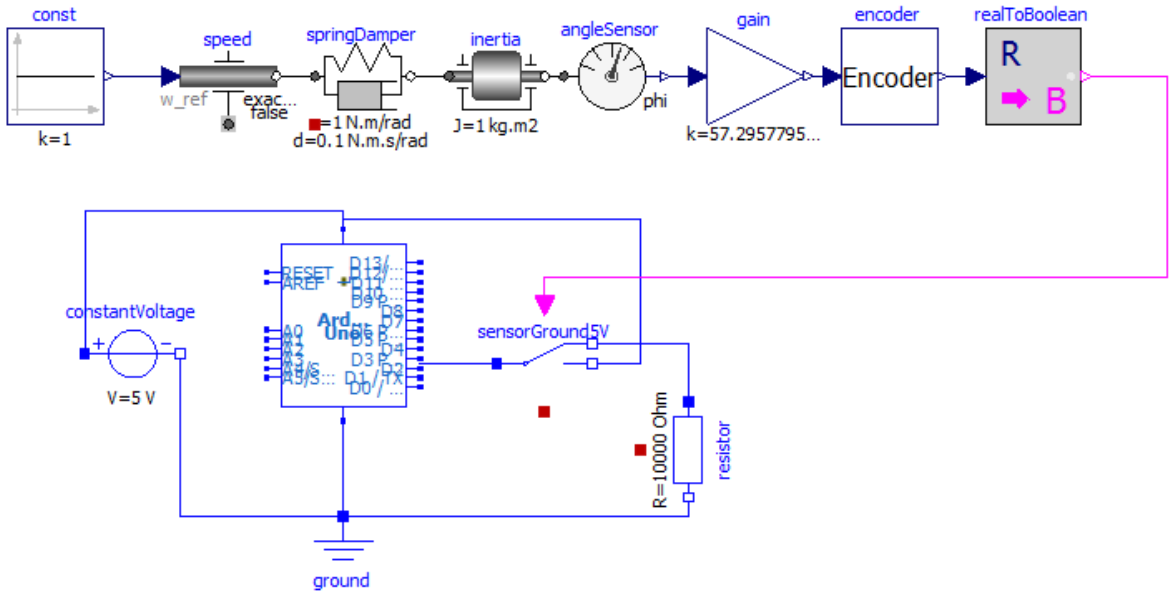


Рис 2.13 - Модель для тестування енкодера з використанням компонента Arduino

Скетч для Arduino для тестування енкодера:

```
volatile int encCounter;
volatile boolean state, lastState;

void int0() {
  state = digitalRead(2);
  if (state != lastState)
     encCounter += 1;
  lastState = state;
}

void setup() {
  Serial.begin(9600);
  attachInterrupt(0, int0, CHANGE);
}

void loop() {
  Serial.println(encCounter);
  delay(100);
}
```

Модель "маятник" реалізована в простому лабораторному стенді на основі недорогих компонентів: Arduino Uno, сервопривід SG90, енкодер на LM393, світлодіод, фоторезистор, деталі металевого конструктора (рис.2.14) [13]. В Arduino завантажено скетч, що дозволяє обмінюватися даними з персональним комп'ютером за допомогою протоколу firmata [16]. На ПК виконується Python-програма, яка керує системою (додаток А). Керувати можна також виключно за допомогою Arduino-скетча (додаток А), але Python

і ПК мають більше можливостей, зокрема, дозволяють візуалізувати сигнал датчиків в реальному часі (рис.2.15) та застосовувати більш складні алгоритми керування. Код скетча та програми мовою Python, яка працює на ПК та обмінюється даними з Arduino по протоколу Firmata, наведено в додатку.

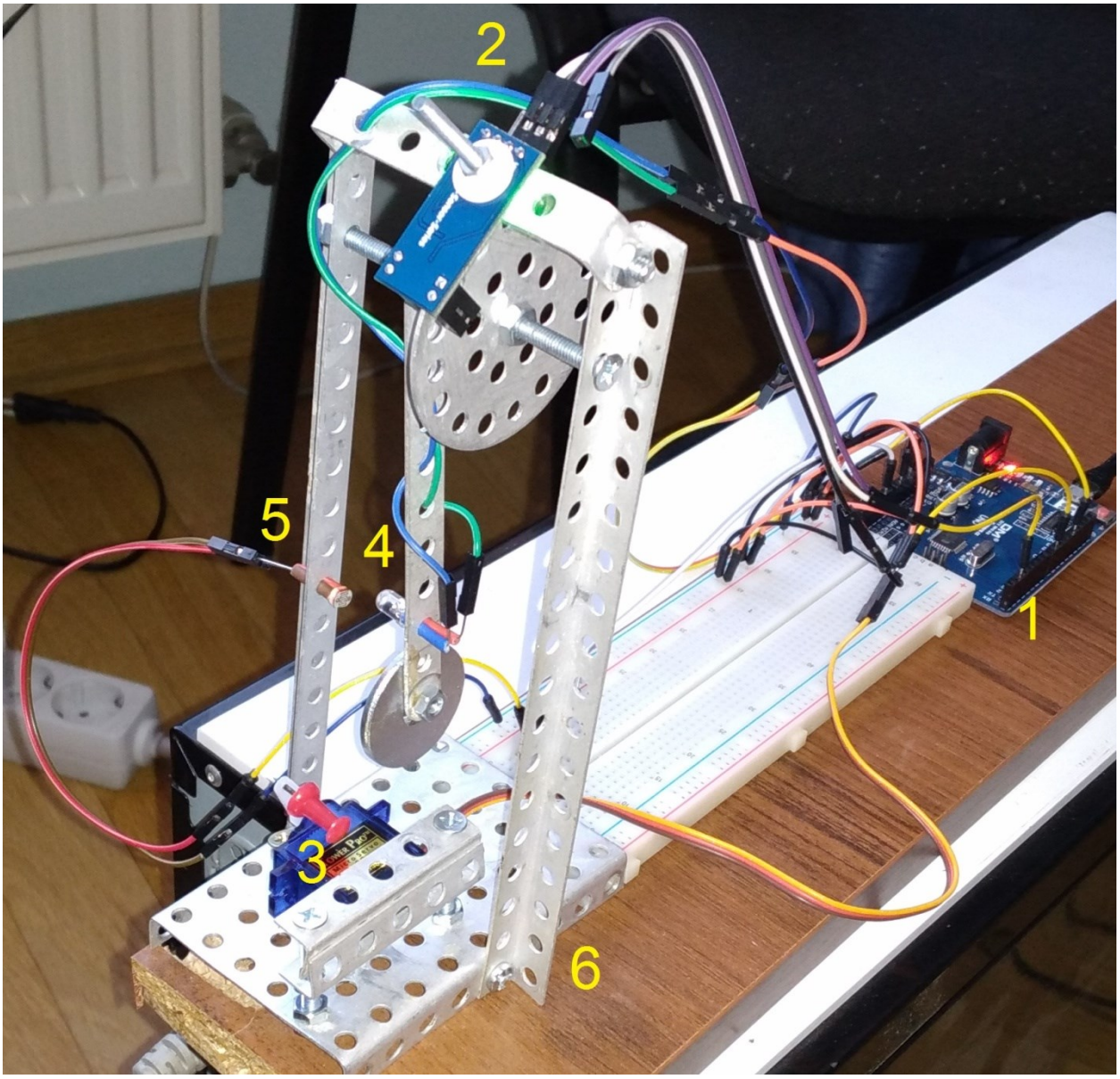


Рис 2.14 - Лабораторний стенд на основі Arduino: Arduino Uno (1), енкодер на LM393 (2), сервопривід SG90 (3), світлодіод (4), фоторезистор (5), деталі металевого конструктора (6)

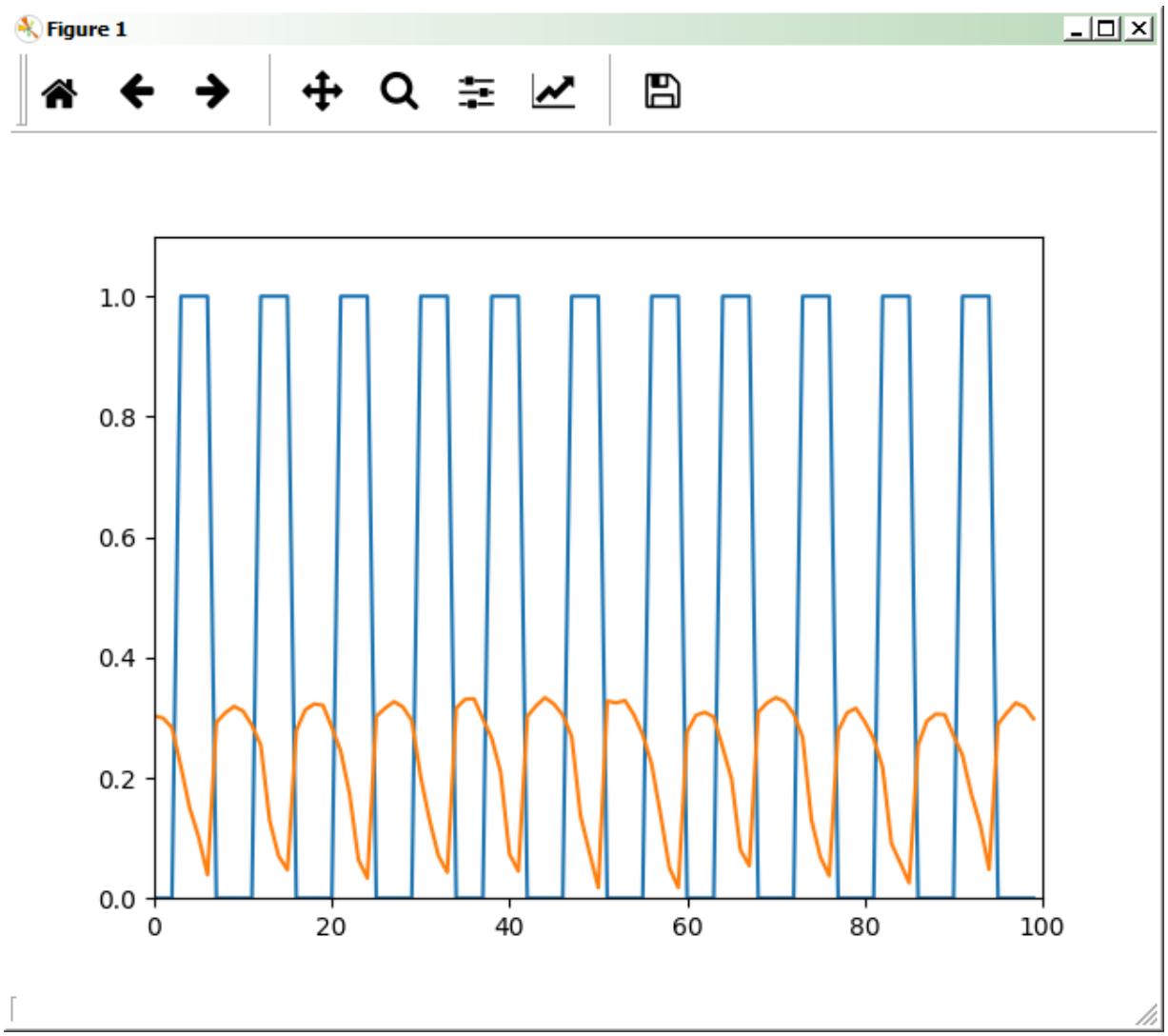


Рис 2.15 - Сигнали енкодера і фоторезистора

Такий підхід до вивчення мехатронних систем має свої переваги. Можна створювати будь-які мультидоменні моделі (у тому числі з використанням Arduino) у вільному середовищі OpenModelica, а також можливо відразу реалізувати систему з недорогих компонентів та верифікувати результати симуляції шляхом експерименту. З кодом моделей і програм, які розроблені автором для розглянутої системи, можна ознайомитись на GitHub [15].

2.2 Лабораторний стенд на основі Arduino, Python, OpenCV та scikit-image для вивчення мехатронних систем

Сьогодні мехатронні системи широко застосовуються в різних галузях техніки, зокрема в машинобудуванні. Ефективне вивчення мехатроніки потребує наявності у кожного студента простих засобів для створення власних

мехатронних систем і експериментування з ними.

Нижче описано принципи розроблення простого, гнучкого і недорогого лабораторного стенда для вивчення мехатронних технологічних систем з машинним зором. Стенд повинен використовувати доступні усім компоненти. Тому було обрано такі компоненти: плата Arduino UNO, сервопривід SG90, персональний комп'ютер, веб-камера та вільне програмне забезпечення (Python, pyFirmata, StandardFirmata, OpenCV, scikit-image).

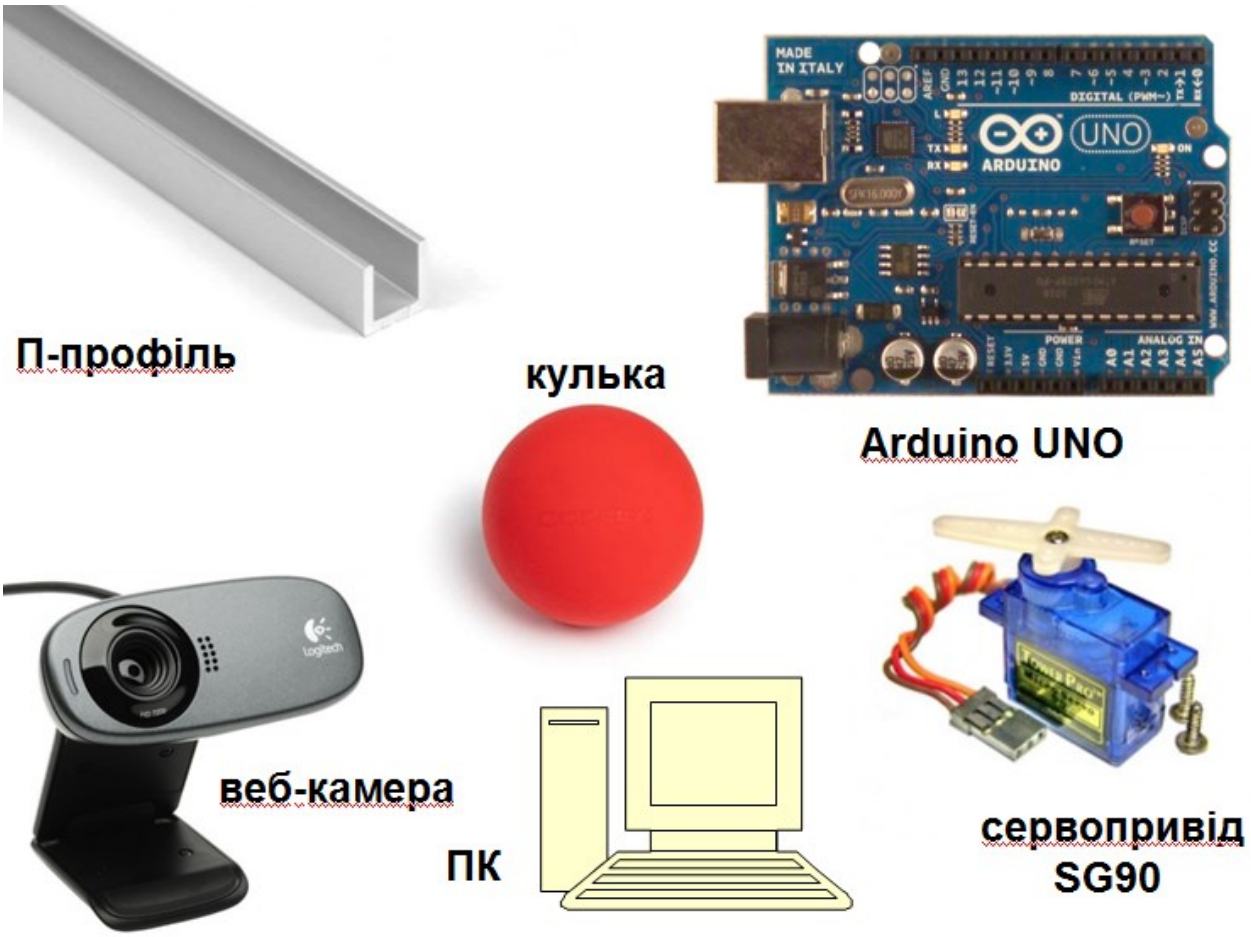


Рис 2.16 Компоненти для лабораторного стенда

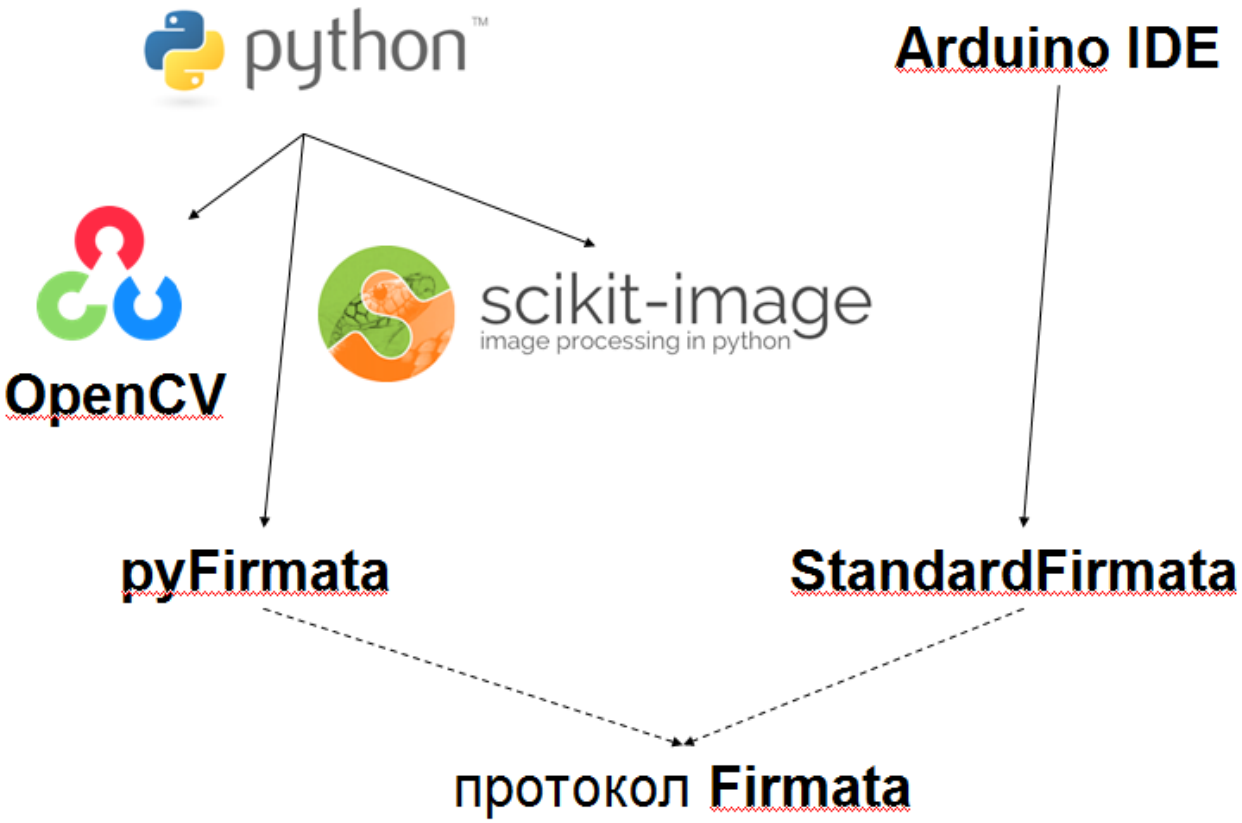


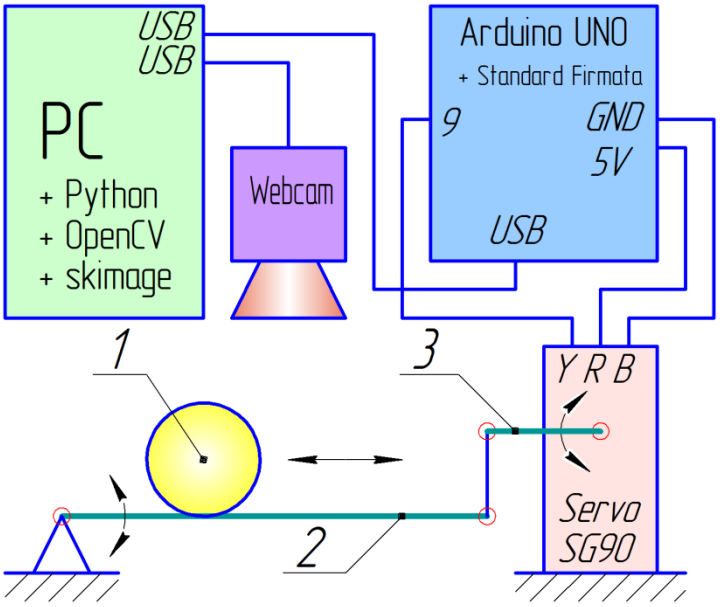
Рис 2.17 - Безкоштовне програмне забезпечення

Arduino Uno – це найбільш популярна і документована мікроконтролерна плата, основана на ATmega328P [10]. Вона має 14 контактів цифрового введення/виведення (з яких 6 можуть використовуватись для ШІМ-виведення), 6 аналогових входів, керамічний резонатор на 16 МГц, USB з'єднання, роз'єм живлення, ICSP і кнопку скидання. Популярна високорівнева мова програмування загального призначення Python ідеально підходить для вивчення програмування мехатронних систем завдяки її здатності до системної інтеграції різнотипних програмних компонентів та до простого вивчення і застосування. Firmata – це загальний протокол для комунікації мікроконтролерів з головним комп'ютером [16]. OpenCV – програмна бібліотека з відкритим кодом для комп'ютерного зору і машинного навчання, яка має оптимізовані швидкодіючі алгоритми з фокусом на оброблення зображень в реальному часі [17]. Вона також має Python-інтерфейс. scikit-image – це колекція гнучких процедур мовою Python для обробки зображень, побудована на основі scipy.ndimage [18]. Цей пакет спрямований на вивчення обробки зображень та має детальну документацію і

велику кількість прикладів.

На рис 2.18 показано схему лабораторного стенда. Кулька 1 може перекочуватись вздовж кривошипа з жолобом 2. Цей кривошип з'єднаний з кривошипом (коромислом) сервоприводу 3 за допомогою гнучкої ланки. Сервопривід (SG90 9g Micro Servo) може повертати кривошип вправо або вліво на заданий кут. Ціллю мехатронної системи є виставлення кульки в задане положення. Сервопривід приєднаний до Arduino UNO трьома провідниками: жовтим (Y-9), червоним (R-5V), коричневим (B-GND). Arduino UNO з'єднана з комп'ютером (PC) за допомогою USB. Фактично для передачі даних між Python програмою, що виконується на комп'ютері, і Arduino використовується віртуальний COM-порт і протокол Firmata. Для цього в мікроконтролер Arduino завантажена програма StandardFirmata, а Python використовує пакет pyFirmata [19]. Для визначення положення кульки використовується USB-веб-камера та програмне забезпечення для комп'ютерного зору OpenCV і scikit-image. В даному випадку OpenCV використовується тільки для отримання зображення з камери, а scikit-image реалізує алгоритми машинного зору для визначення положення кульки. Проте, з метою підвищення швидкодії системи, користувач може використовувати тільки процедури OpenCV.

В розробленій програмі (додаток Б) [20] створюється об'єкт для комунікації з Arduino, об'єкт servo для управління сервоприводом та об'єкт cap для отримання відеокадрів з камери. Основний алгоритм являє собою цикл, в якому отримується відеокадр, зображення конвертується у відтінки сірого, передається у функцію користувача detObj, яка призначена для визначення координат кульки за допомогою процедур scikit-image. Ці координати використовуються об'єктом servo для управління сервоприводом.



1 - кулька, 2 - кривошип з жолобом, 3 - кривошип сервоприводу

Рис 2.18 - Схема лабораторного стенда

```
while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cx, cy, r = detObj(gray)
    servo.write(cx/20)
```

У detObj для визначення границь об'єктів використовується алгоритм Кенні. Ці границі і список значень радіусів кілець R, які потрібно виявити, передаються функціям для визначення кільцевих форм на основі перетворення Хафа. Функції знаходять найбільш імовірні координати центра кільця (cx, cy):

```
edges = canny(image, sigma=2, low_threshold=10,
high_threshold=20)
R = [16]; hough_res = hough_circle(edges, R)
accums, cx, cy, r = hough_circle_peaks(hough_res, R,
                                       total_num_peaks=1)
```

Розроблений стенд та програма [21] можуть бути основою для проектування більш складних систем для автоматизації машинобудівного виробництва з простим приєднанням інших компонентів, таких як датчики, актуатори чи Python-пакети.

2.3 Програма для автоматизованого створення та симуляції Modelica-моделей лабораторних стендів

Розроблена програма (додаток В) призначена для автоматизації побудови Modelica-моделей та їхньої симуляції шляхом створення морфологічних матриць з різними Modelica-компонентами.

Морфологічний аналіз — методика творчості у сфері винахідництва, яка допомагає розглядати різні можливі рішення проблеми розбиваючи її на окремі атрибути та комбінуючи можливі реалізації цих атрибутів [22]. В нашому випадку програма будує морфологічну матрицю з усіма можливими сполученнями Modelica-компонентів. Користувач може обрати будь-які сполучення, щоб за ними була автоматизовано побудована модель.

Програма складається з двох модулів: ModelicaCAD (класи компонентів) та ModelicaCADgui (графічний інтерфейс). Перший модуль містить клас controlPart, який містить такі атрибути як components і equation. Це компоненти і рівняння мовою Modelica [23], які описують систему керування в лабораторному стенді, тобто Arduino, сервопривід і датчик. Наступний клас Flange описує поняття Modelica конектора компонента з такими атрибутами, як сам компонент, ім’я конектора і його координати. Наступний клас описує поняття Modelica-компонента, в конструкторі __init__ вказуються параметри компонента, координати і ім’я. Усі ці атрибути є необов’язковими. Також цей клас містить функцію className, яка повертає Modelica-назву класу, code, яка повертає код Modelica (код декларації компонента в моделі) та функція anno - анотація, як частина Modelica-коду. На основі цього класу розроблені наступні класи, які описують конкретні механічні компоненти стандартної бібліотеки

Modelica. Ці класи успадковують клас Component. Клас Modelica_Mechanics_Translational_Components_Fixed описує поняття зафіксованого фланця, наступний клас Modelica_Mechanics_Translational_Components_Spring описує поняття поступальної пружини, наступний клас Modelica_Mechanics_Translational_Components_Mass описує поняття поступального компонента маса. Також можна додати і інші класи. Наступний клас Model описує поняття Modelica-моделі, де використовуються такі атрибути як name (ім'я моделі), connected (список пар з'єднаних компонентів) і eqs (список рівнянь). Функція code_eqs повертає Modelica-код опису рівнянь, функція connect дозволяє з'єднати два компонента, також ще одна функція connect_ повертає Modelica–код з'єднання компонентів з анотацією. Функція code_gen генерує Modelica-код моделі. Функція autoOrigin автоматично розташовує компоненти, задає їм певні координати на робочому просторі. Функція saveCode зберігає код Modelica, а функція run виконує симуляцію за допомогою інтерфейсу Python-Modelica. Також тут є список D, який містить компоненти, які будуть використовуватись для побудови морфологічних матриць, але в самих морфологічних матрицях будуть застосовуватися фланці цих компонентів. Ці матриці будуть будуватися за допомогою списку C.

Вкінці коду описано приклад побудови моделі. Спочатку створюється модель, в модель додається зафіксований фланець, пружина, маса, також можемо задати координати цих компонентів і з'єднуємо їх за допомогою функції connect. Після цього функція Code_gen генерує код Modelica, autoOrigin розташовує компоненти в заданих координатах, зберігаємо наш код і виконуємо симуляцію.

Другий модуль використовує бібліотеку Tkinter для створення графічного інтерфейсу користувача, щоб можна було в ній побудувати морфологічну матрицю і її заповнити. Дальше створюється об'єкт root - це головне вікно програми. Створюється морфологічна матриця, яка містить множину прапорців Checkbutton і створюється одна кнопка Button1, яка

призначена для побудови моделі після заповнення морфологічної матриці. Коли ця кнопка натискається створюється модель - визначається, які Checkbutton були вибрані і, в залежності від них, з'єднуються певні компоненти, генерується код і зберігається.

Якщо цю програму запустити, з'являється морфологічна матриця, і вибираємо, що ми хочемо з'єднати (рис 2.19).

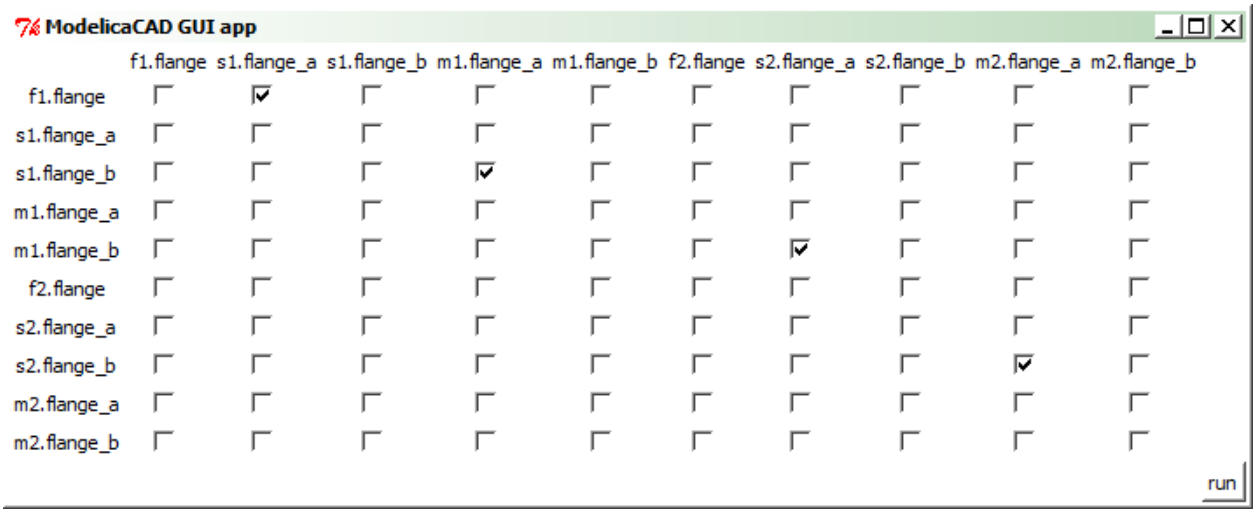


Рис 2.19 - Вікно програми для заповнення морфологічної матриці

Відповідна морфологічна матриця:

0 1 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 1 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 1 0

0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0

Якщо згенерований код відкрити через OpenModelica то ми зможемо побачати згенеровану модель лабораторного стенда (рис 2.20). Користувач

може доповнити цю модель, сюди можна додати компонент IdealGearR2T, який перетворює обертовий рух в поступальний, і з'єднати його з сервоприводом. Можна додати PositionSensor (сенсор переміщення) і передати сигнал датчику. Таким чином розроблена програма дозволяє полегшити створення моделей таких лабораторних стендів.

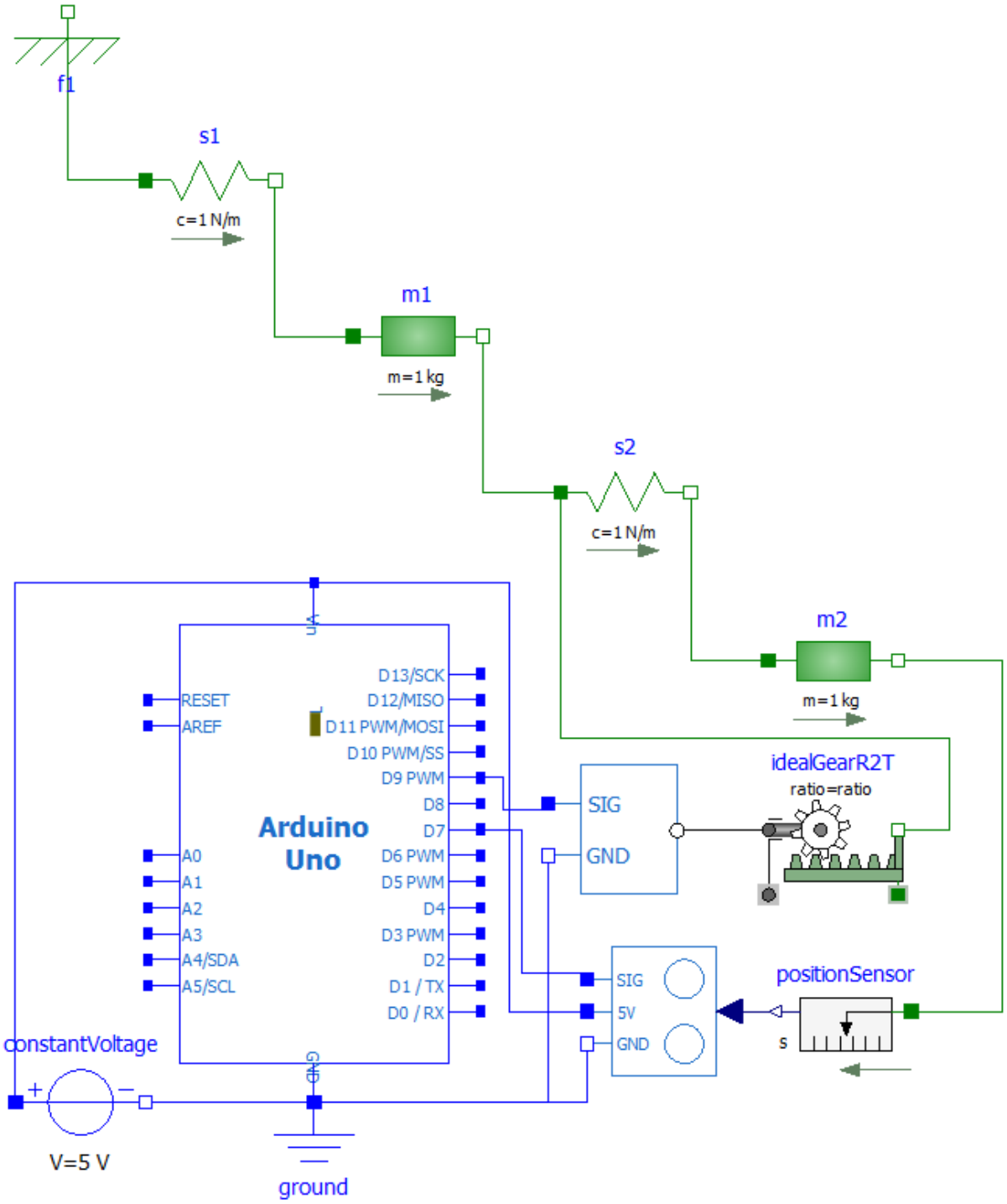
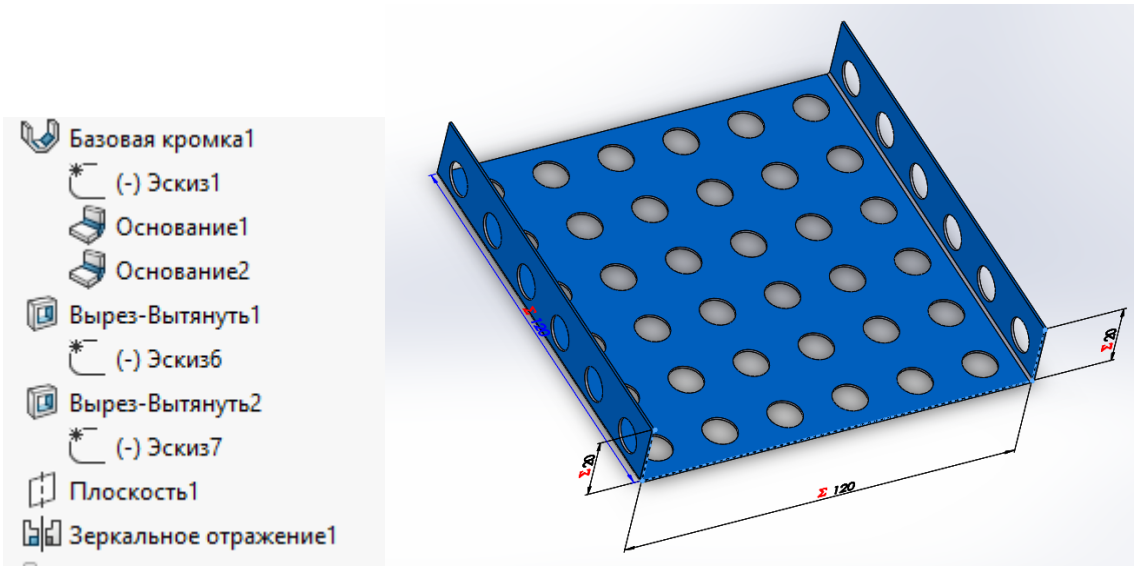


Рис 2.20- Модель, яка автоматично побудована за морфологічною матрицею

РОЗДІЛ 3. ПОБУДОВА ГЕОМЕТРИЧНИХ МОДЕЛЕЙ УНІВЕРСАЛЬНИХ ДЕТАЛЕЙ ДЛЯ ЛАБОРАТОРНИХ СТЕНДІВ

3.1 Побудова геометричної моделі планки широкої П-подібної

Параметричну геометричну модель планки широкої П-подібної будували в САПР SOLIDWORKS 2016 [24]. На рис. 2.1 показано параметричну геометричну тривимірну модель планки. Модель побудована за допомогою елементів Базовая кромка1 в Эскиз1 основа моделі, за допомогою Вырез-вытянуть1 ми вирізали отвори в середині планки, Вырез-вытянуть2 ми вирізали отвори на боковій стінці планки, за допомогою плоскость1 ми дзеркально відобразили отвори на бокових стінках.



а б

Рис 3.1 – Дерево побудови (а) і модель (б) планки

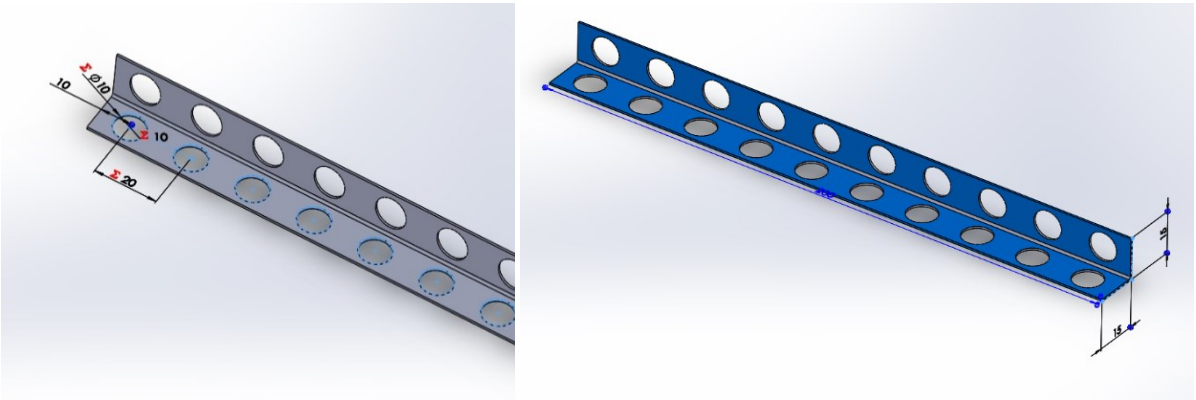
Для параметричної моделі були розроблені рівняння (рис. 2.2), які створені у моделі вузла. Використання рівнянь дозволить спростити перебудову моделі.

Имя	Значение / Уравнение	Равняется
Глобальные переменные		
"d"	= 10	10
"L"	= 120	120
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D5@Эскиз6"	= "L" / ("d" * 2)	6
"D1@Эскиз6"	= "d"	10мм
"D4@Эскиз6"	= "L" / ("d" * 2)	6
"D3@Эскиз7"	= "L" / ("d" * 2)	6
"D2@Эскиз1"	= "L"	120мм
"D6@Эскиз6"	= "d" * 2	20мм
"D2@Эскиз6"	= "d"	10мм
"D2@Базовая кромка1"	= "L"	120мм
"D4@Эскиз7"	= "d" * 2	20мм
"D7@Эскиз6"	= "d" * 2	20мм
"D3@Эскиз1"	= "d" * 2	20мм
"D1@Эскиз1"	= "d" * 2	20мм

Рис. 3.2 – Рівняння моделі

3.2 Побудова моделі кутника

На рис. 2.3 показано параметричну геометричну тривимірну модель кутника. Модель була побудована за допомогою елементів Базовая кромка1 в Эскиз основа моделі. В Эскиз6 було створене коло діаметром 10мм і за допомогою елемента Линейный масив эскиза ми побудували отвори на всю довжину планки, на боковій стінці планки ми повторили цю процедуру і за допомогою елемента Вырез-вытянуть1 та Вырез-вытянуть2 ми вирізали отвори.



а) б)

Рис 3.3 – Побудова отворів (а) і модель (б) кутника

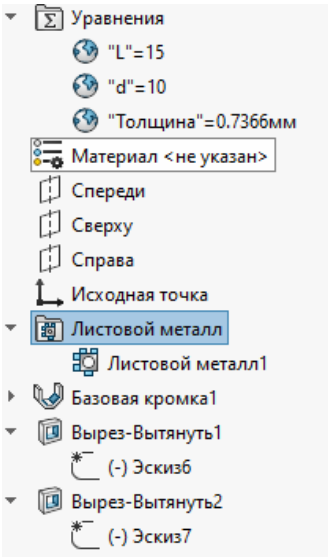


Рис 3.4 – Дерево побудови моделі кутника

Для параметричної моделі були розробленні рівняння рис. 2.5 які створені у моделі вузла. Використання рівнянь дозволить спростити перебудову моделі.

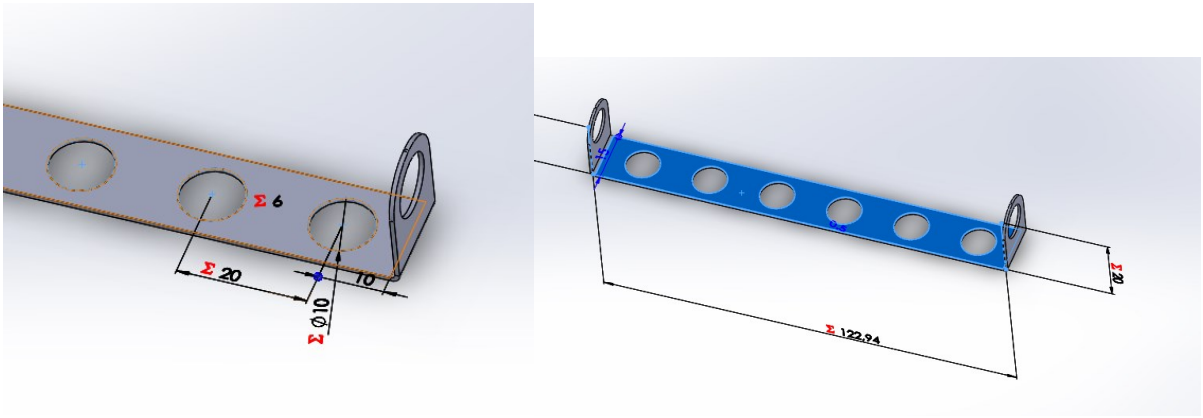
Имя	Значение / Уравнение	Равняется
Глобальные переменные		
"L"	= 15	15
"d"	= 10	10
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D1@Эскиз6"	= "d"	10мм
"D3@Эскиз6"	= "L" / ("d" * 2)	1
"D3@Эскиз7"	= "L" / ("d" * 2)	1
"D4@Эскиз6"	= "d" * 2	20мм
"D4@Эскиз7"	= "d" * 2	20мм

Рис 3.5 – Рівняння моделі

3.3 Побудова моделі планка П-подібна

На рис. 2.6 показано параметричну геометричну тривимірну модель планки П-подібної. Модель була побудована за допомогою елементів базовая кромка1 в Эскиз основа моделі. В Эскиз6 було створене коло діаметром 10мм і за допомогою елемента Линейный масив эскиза ми побудували отвори на всю довжину планки, після чого ми використали елемент Вырез-вытянуть1 ми

вирізали отвори,на боковій стінці був створений отвір діаметром 10мм, після чого ми використали такий елемент як Поскость1 і розмістили її посередині планки і використали елемент Зеркальное отражение1 для того щоб відобразити наш отвір на протилежній стороні стінки і знову використали елемент Вырез-вытянуть2, ще був використаний такий елемент як Скругление1 і Скругление2 за допомогою цього ми заокруглили стінки планки.



а) б)

Рис 3.6 (а) Побудова отворів і модель (б) планки

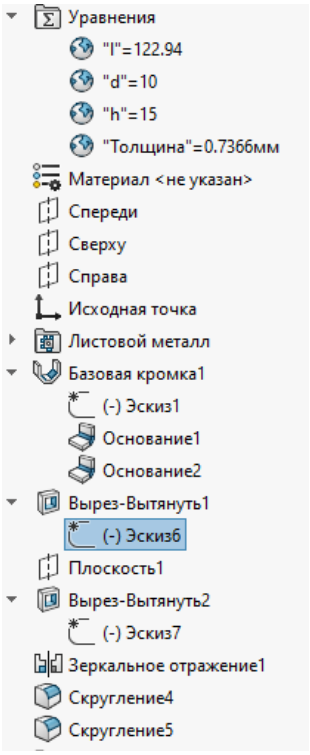


Рис 3.7 – Дерево побудови моделі

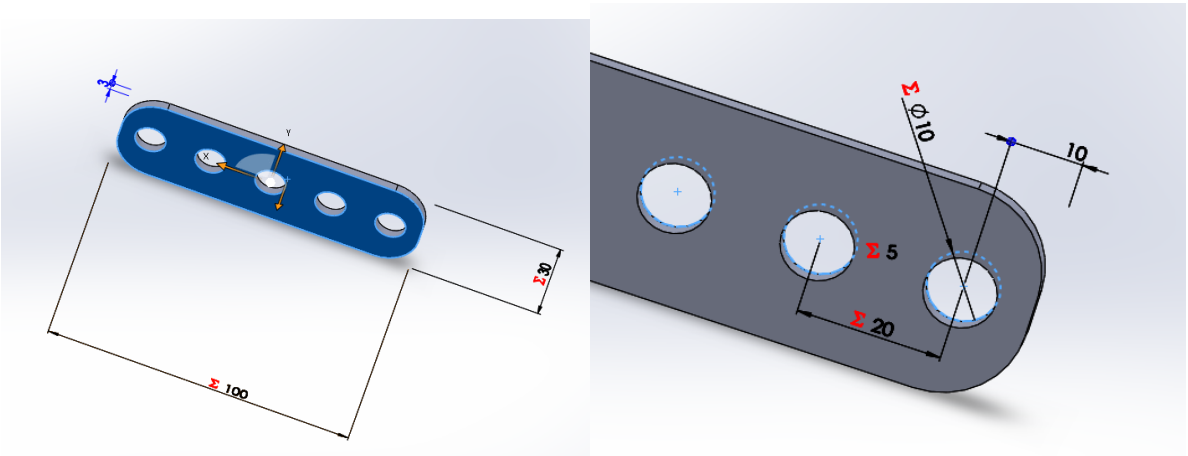
Для параметриної моделі були розроблені рівняння рис. 2.8 які створені у моделі вузла. Використання рівнянь дозволить спростити перебудову моделі.

Имя	Значение / Уравнение	Равняется
Глобальные переменные		
"l"	= 122.94	122.94
"d"	= 10	10
"h"	= 15	15
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D1@Эскиз1"	= "l"	122.94мм
"D1@Эскиз6"	= "d"	10мм
"D3@Эскиз6"	= "l" / ("d" * 2)	6
"D2@Эскиз1"	= "d" * 2	20мм
"D4@Эскиз6"	= "d" * 2	20мм

Рис 3.8 – Рівняння моделі

3.4 Побудова моделі планки

На рис 2.9 показано параметричну геометричну тривимірну модель плоскої планки. Модель була побудована наступним чином: Эскиз1 основа моделі, після чого ми використали елемент Бобышка-вытянуть1, дальше ми на нашій моделі побудували коло діаметром 10мм і використали елемент Линейный масив эскиза для того щоб побудувати отвори на всю довжину планки і використали елемент Вырез-вытянуть1 щоб вирізати отвори, також ми задіяли елемент Скругление щоб заокруглити стінки планки.



а) б)

Рис 3.9 Побудова отворів (а) і модель (б) планки

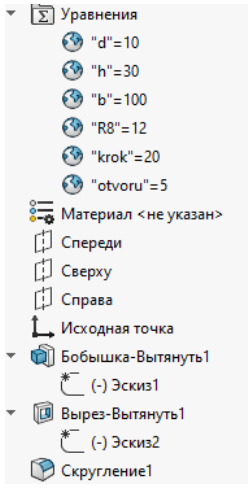


Рис 3.10 – Дерево побудови моделі

Для параметричної моделі були розроблені рівняння рис. 2.11 які створені у моделі вузла. Використання рівнянь дозволить спростити перебудову моделі.

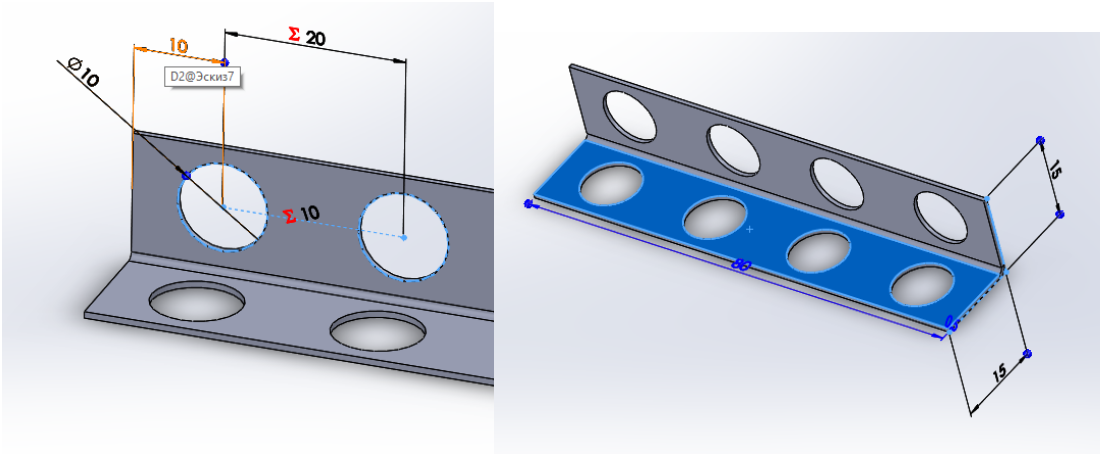
"d"	= 10	10
"h"	= 30	30
"b"	= 100	100
"R8"	= 12	12
"krok"	= 20	20
"otvoru"	= 5	5
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D2@Эскиз2"	= "d"	10мм
"D1@Эскиз1"	= "h"	30мм
"D2@Эскиз1"	= "b"	100мм
"D1@Скругление1"	= "R8"	12мм
"D4@Эскиз2"	= 2 * "d"	20мм
"D3@Эскиз2"	= "b" / "krok"	5

Рис 3.11 – Рівняння моделі

3.5 Побудова моделі короткого кутника

На рис. 2.12 показано параметричну геометричну модель короткого кутника. Модель побудована за допомогою елементів Базовая кромка1 в Эскиз основа моделі. В Эскиз6 було створене коло діаметром 10мм і за допомогою

елемента Линейный масив эскиза ми побудували отвори на всю довжину планки, на боковій стінці планки ми повторили цю процедуру і за допомогою елемента Вырез-вытянуть1 та Вырез-вытянуть2 ми вирізали отвори. Така модель уже застосована, тільки тут інші розміри.



а) б)

Рис 3.12 – (а) Побудова отворів і модель (б) кутника

Для параметричної моделі були розробленні рівняння на рис. 2.13 які створені у моделі вузла. Використання рівнянь дозволить спростити перебудову моделі.

Глобальные переменные		
"L"	= 15	15
"d"	= 10	10
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D1@Эскиз6"	= "d"	10мм
"D3@Эскиз6"	= "L" / ("d" * 2)	1
"D3@Эскиз7"	= "L" / ("d" * 2)	1
"D4@Эскиз6"	= "d" * 2	20мм
"D4@Эскиз7"	= "d" * 2	20мм

Рис 3.13 – Рівняння моделі

3.6 Побудова моделі колеса

На рис. 2.14 показано модель колеса яка може служити колесом, кривошипом, маховиком і диском енкодера в залежності від ситуації. Модель побудована за допомогою елементів Бобышка-вытянуть1 в Эскиз1 основа моделі, за допомогою Вырез-вытянуть1 було вирізано коло посередині колеса,

Вырез-вытянуть2 ми вирізали центральні кола діаметром 12, Вырез-вытянуть3 ми вирізали кола діаметром 10, для створення цих кіл ми використали елемент Круговой масив, щоб створити кола по цілому колесі, також ми був використаний елемент Скругление1, щоб скруглити краї колеса, потім у вигляді справа було створене ще одне коло для того щоб зробити виріз і потім використали такий елемент як Вырез-повернуть1 і вирізали нашу деталь по кругу.

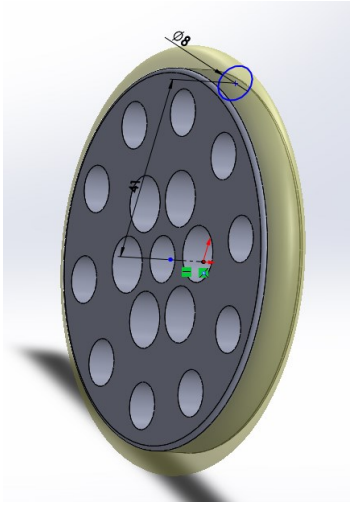
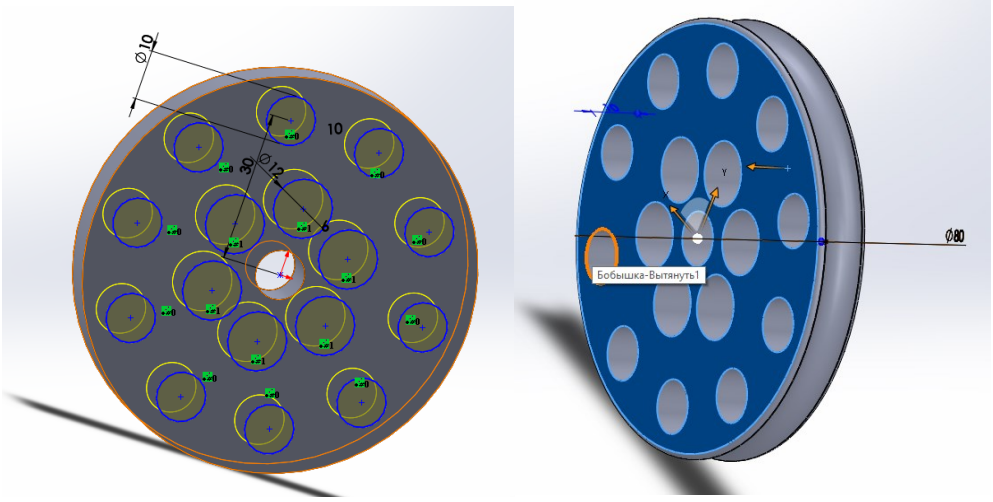


Рис 3.14 – Принцип побудови вирізу



а) б)

Рис 3.14 –Побудова отворів (а) і модель (б) колеса

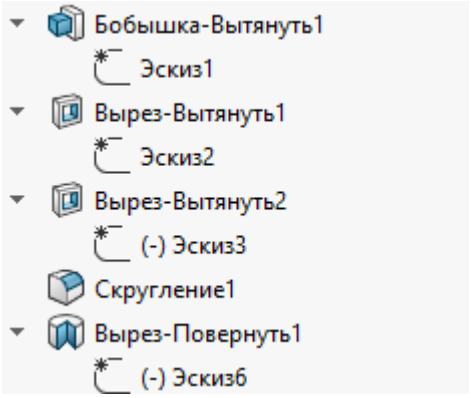


Рис 3.15 – Дерево побудови моделі

РОЗДІЛ 4. АНАЛІЗ ПРАЦЕЗДАТНОСТІ МАЯТНИКА В SOLIDWORKS

4.1. Аналіз працездатності маятника в SOLIDWORKS Motion

Для того щоб симулювати модель була використана така програма як SOLIDWORKS Motion, На рис 4.1 зображено модель маятника, вона була спрощена для того щоб не помилитись під час створення зв'язків,залишили тільки серводвигун з кривошипом, маятник і вісь. Спочатку потрібно перевірити щоб була включена програма SOLIDWORKS Motion. Якщо вона буде виключеною, то ми зможемо зробити тільки простий аналіз, а нам потрібно Анализ Движения. Цей аналіз є найбільш реалістичним і враховує усі типи рухів об'єктів і забезпечує точність всіх результатів.

На рис 4.1 показана група сполучень. Concetric1 - вісь і отвір концентричні, Distance1 - відстань від осі до планки для того щоб планка не переміщалася вздовж осі, Concentric2 - концентричність отвору серводвигуна і отвору кривошипа, Coincident2 - збіг поверхонь двигуна і кривошипа, Coincident3 - збіг бічної поверхні планки і серводвигуна, Distance2 - відстань серводвигуна до центру координат, Distance3 - відстань від серводвигуна до центру координат , але уже по вертикалі.

На рис 4.2 показано дерево побудови симуляції, дальше включаємо Motion Analisis і додаємо Rotary Motor. Це обертовий двигун. На рис 4.3 зображено його параметри. Потрібно вибрати циліндричну поверхню кривошипа і вказати напрямок обертання кривошипа і закон руху (в даному випадку він задається за допомогою Data Points). Після цього потрібно задати контакт Solid Body Contact - потрібно задати контакт двох поверхонь, кривошипа і маятника. Цей контакт зображений на рис. 4.4. Дальше потрібно задати Gravity. Гравітація зображена на рис 4.5. Вона задана таким чином, що сила тяжіння напрямлена вниз по осі y. Також ми застосували обертовий Damper, зображений на рис 4.6. Демпфер потрібен для того, щоб коливання маятника були затухаючі. Дальше виставили тривалість симуляції на 10

секунд. Спочатку у нас буде один результат - це кутове переміщення маятника (рис 4.7). Після чого нажимаємо кнопку Calculate. Кривошип рухається по заданому закону і вдаряється в маятник, а маятник коливається. На рис 4.8 показано результати симуляції, і можна побачити як міняється нахил кривошипа з часом. На графіку можна побачити, що коливання затухають, після цього кривошип підштовхує маятник, потім коливання знову затухають і т.д. На рис. 4.9 показано графік кутової швидкості, на рис 4.10 - графік кутового прискорення.

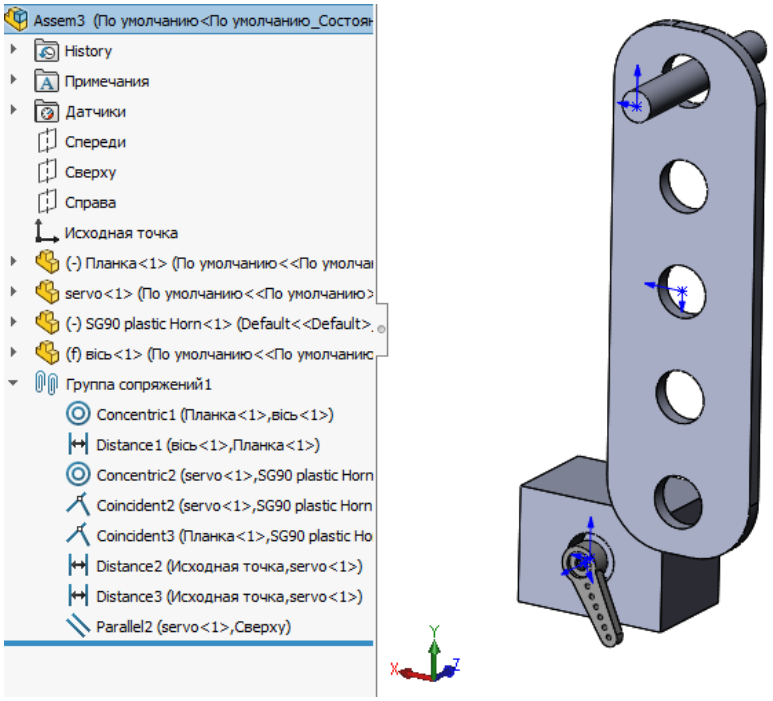


Рис 4.1 – Модель маятника та група сполучень

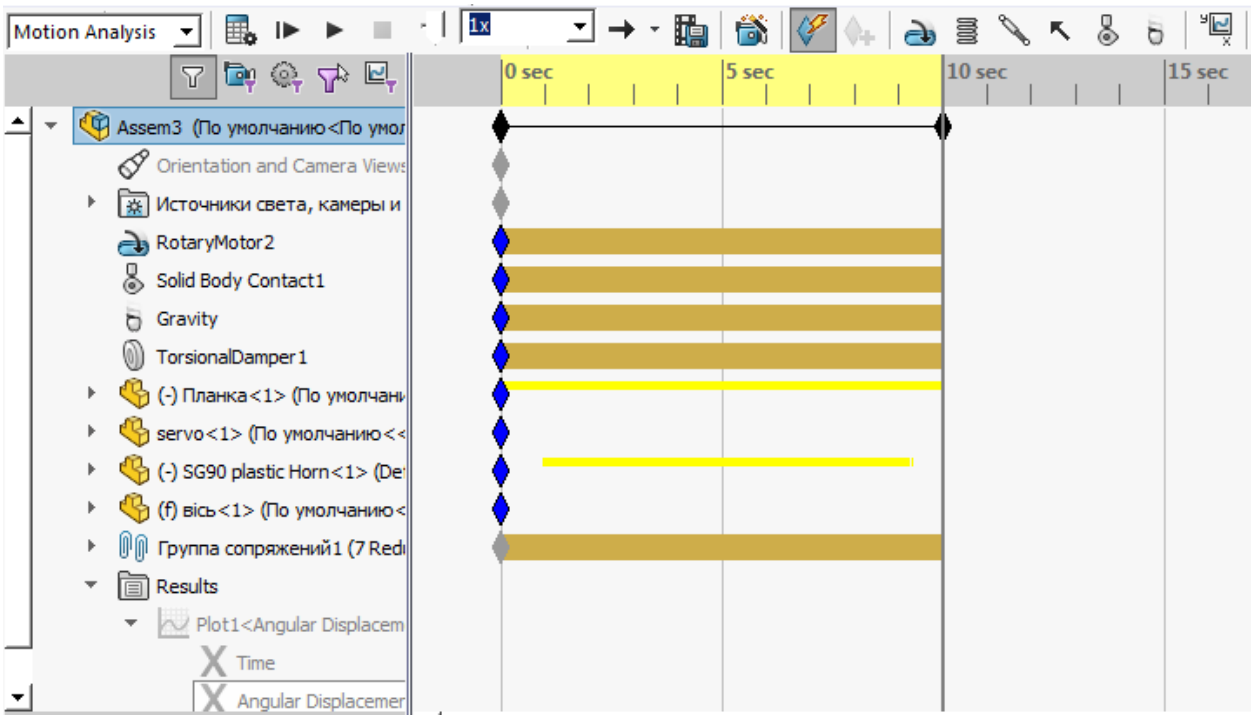


Рис 4.2 – Дерево побудови симуляції

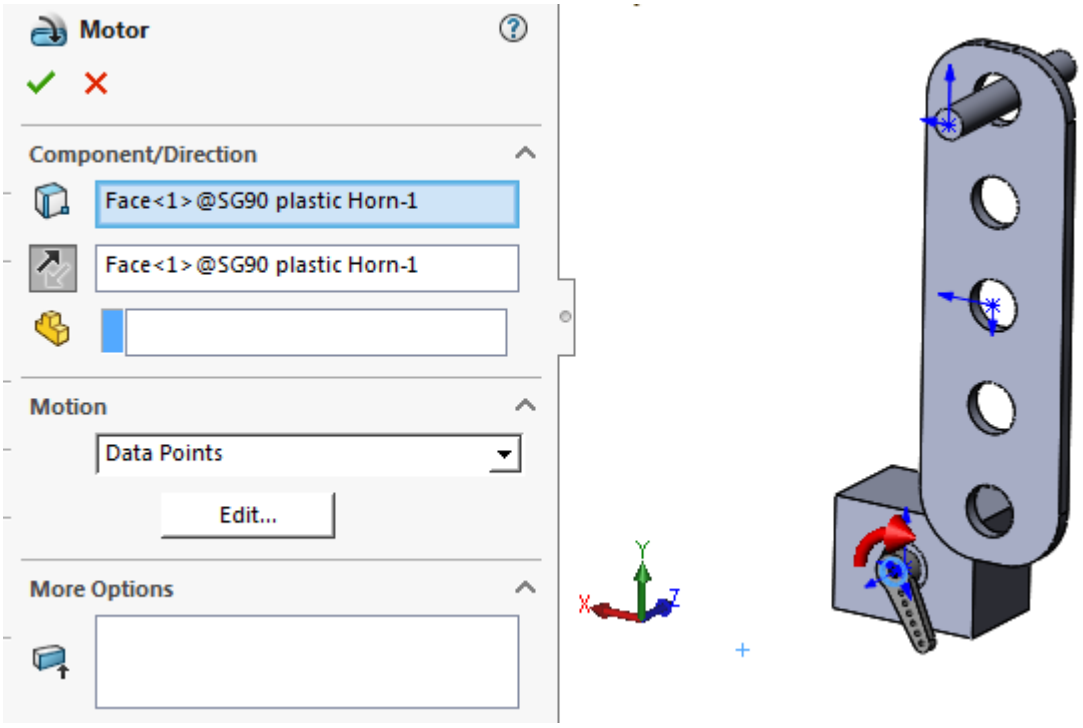


Рис 4.3 – Параметри Rotary Motor

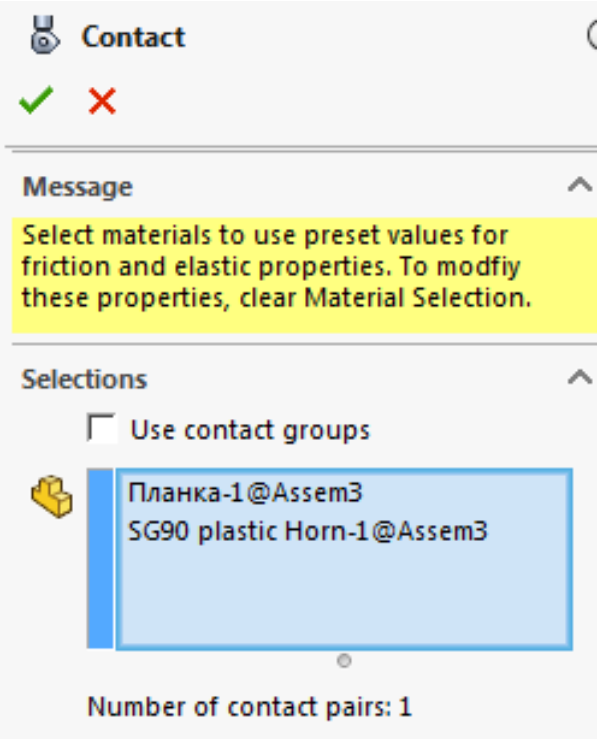


Рис 4.4 – Параметри контактів

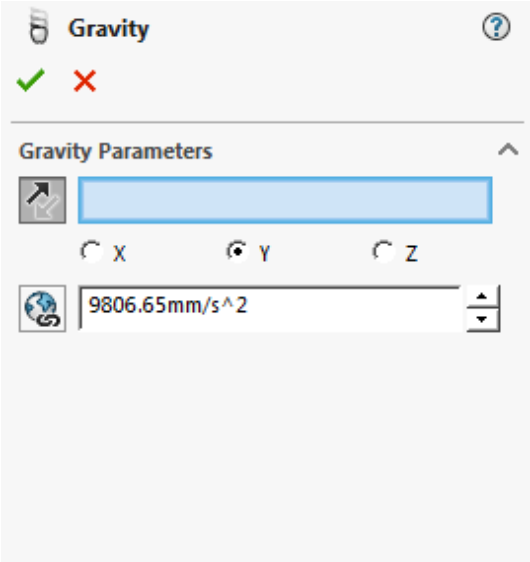


Рис 4.5 – Гравітація моделі

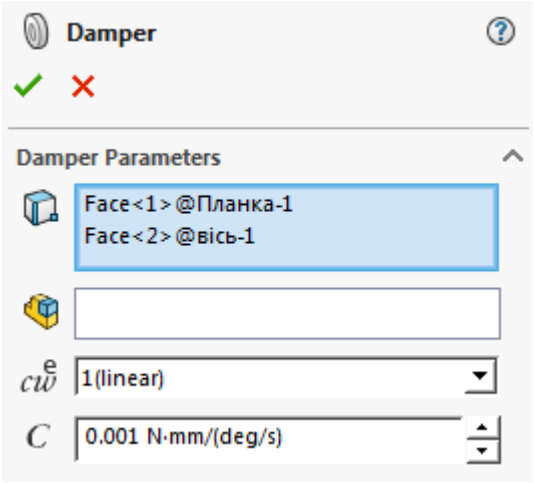


Рис 4.6 – Параметри демпфера

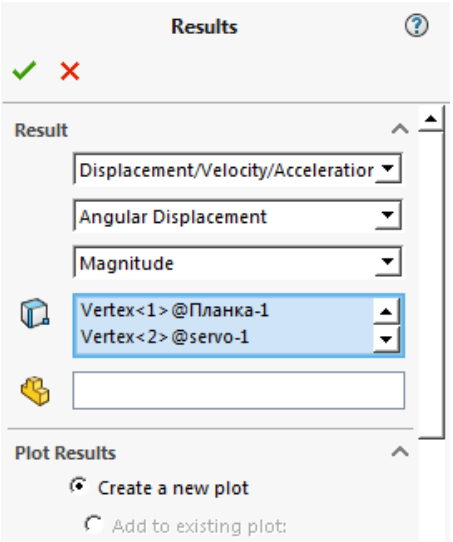


Рис 4.7 – Параметри результату - кутового переміщення маятника

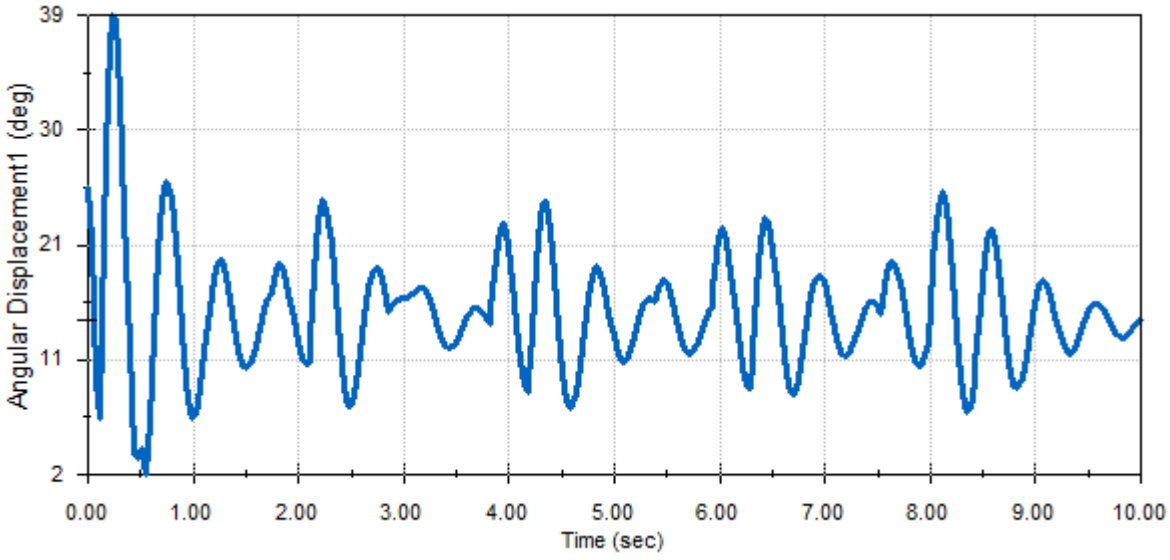


Рис 4.8 – Графік кутового переміщення

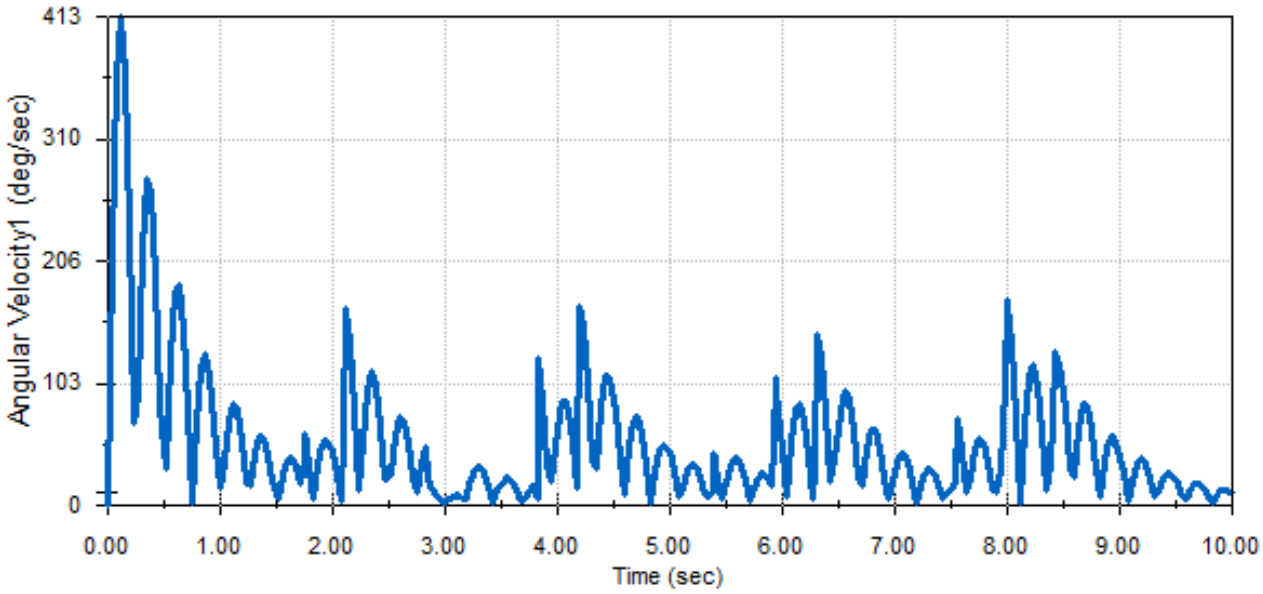


Рис 4.9 – Графік кутової швидкості

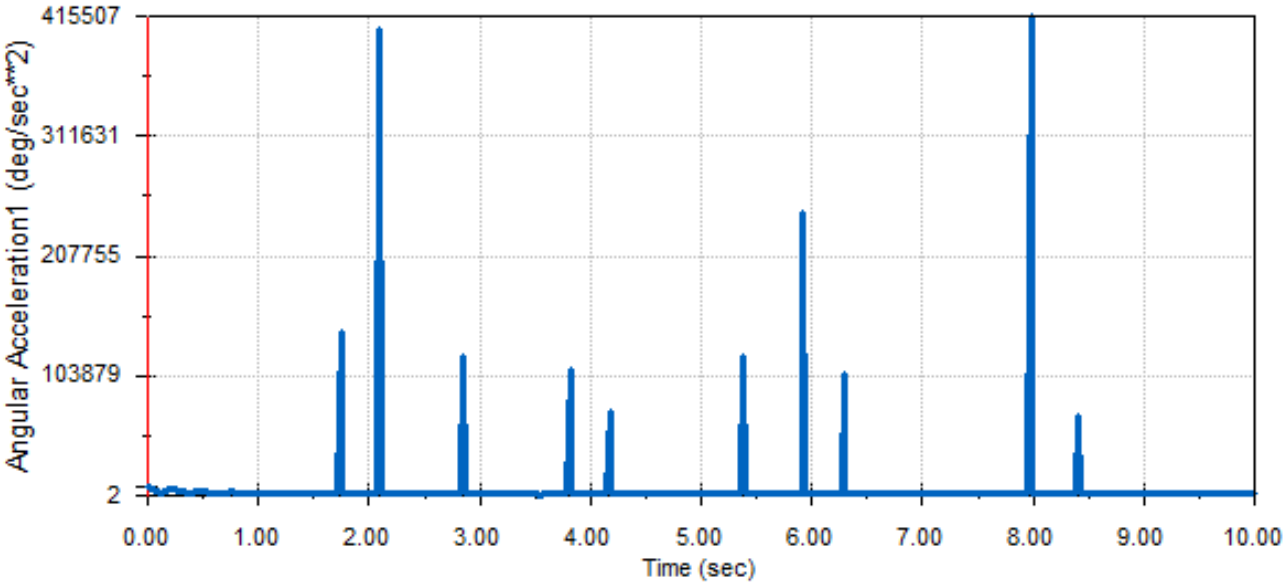


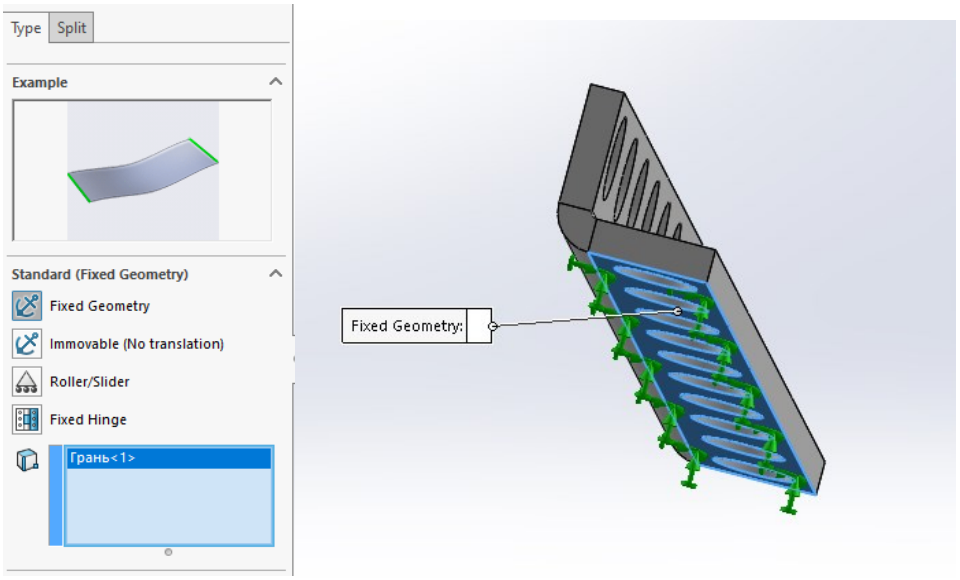
Рис 4.10 – Графік кутового прискорення

4.2. Статичний аналіз моделі кутника в SOLIDWORKS Simulation

Для того щоб провести статичний аналіз напружено-деформованого стану моделі кутника, була використана програма SOLIDWORKS Simulation [25]. Статичний аналіз ми проводили на двох матеріалах - це пластик ABS і алюміній Н-12. Також аналіз проводився для різної товщини кутника.

Для проведення статичного аналізу спочатку потрібно запустити програму SOLIDWORKS Simulation, в верхній панелі потрібно вибрати Новое

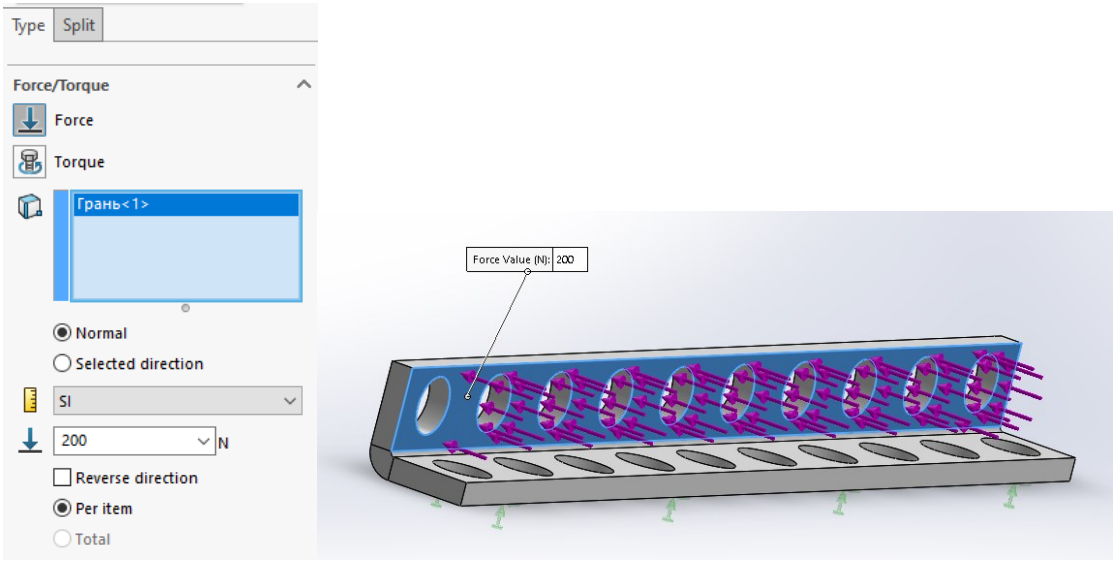
исследовние, після чого потрібно вибрати тип аналізу, який ми хочемо провести. Ми використовуємо статичний аналіз Static. Дальше в головному меню потрібно вибрати Консультант по креплениям і вибираємо Фисированая геометрия і фіксуємо нижню поверхню кутника рис 4.11.



а) б)

Рис 4.11 – Тип граничних умов (а) і місце фіксації (б) кутника

Також нам потрібно задати силу. Для того, щоб це зробити вибираємо Консультант по внешним нагрузкам і вибираємо Сила. Дальше ставимо силу 200Н і задаємо куди буде діяти наша сила (рис 4.12).



а) б)

Рис 4.12 – Параметри навантаження (а) і місце дії сили (б) на кутник

Для симуляції потрібно задати матеріал як і в дереві побудови самої моделі так і в дереві статичного аналізу. Дальше потрібно створити сітку. Переходимо в Mesh, вибираємо Create Mesh і вибираємо розмір сітки. Після того як сітка створилась вибираємо Запустить исследование і получаємо результат. Аналіз проводився для різних товщин починаючи від 1мм до 4мм з кроком 0,5мм як для пластику ABS так і для алюмінію H-12. За цими результатами були побудовані графіки. Щоб побачити оптимальну товщину, всі дані аналізу записувались в таблицю 4.13. Перша колонка - це є товщина кутника, друга колонка - це максимальне напруження по von Mises і третя колонка - це маса кутника.

Таблиця 4.1 – Результати аналізу пластика

Товщина	Напруженння	Маса
1	6,009	4,61
1,5	2,744	7,04
2	1,872	9,55
2,5	1,289	12,13
3	1,3	14,8
3,5	6,31	17,55
4	7,028	20,38

За цими даними були створені графіки рис 4.14 На графіку по шкалі y, показано дані по напруженням Мізеса-Губера, по шкалі x - товщина кутника. За цими даними ми підібрали оптимальну товщину 2,5 мм. Також на рис 4.15 показано графік маси кутника. Шкала y - це маса кутника і шкала x - це товщина кутника.

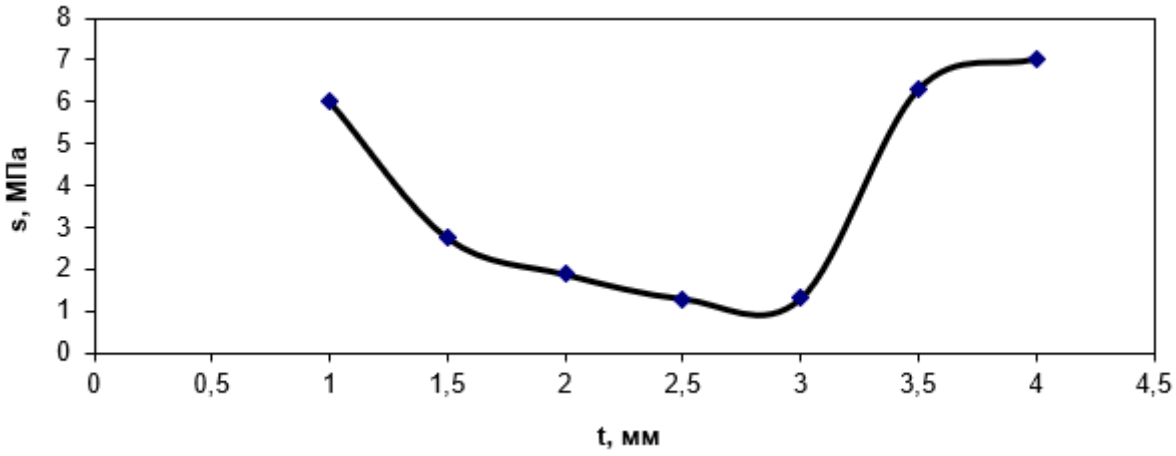


Рис 4.13 – Графік напружень для пластика

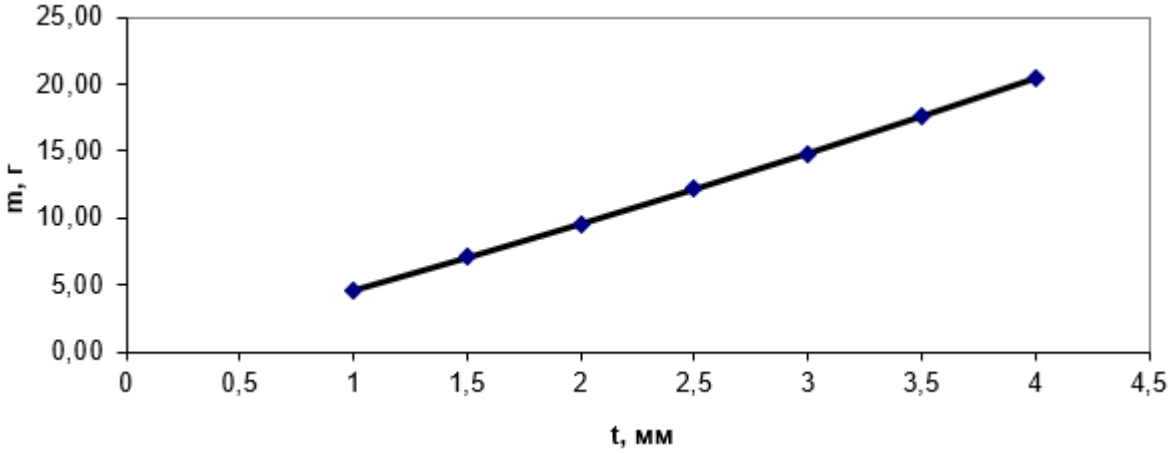


Рис 4.14 – Графік зміни маси

Також був проведений аналіз для алюмінію Н-12 і всі дані ми занесли в таблицю 4.2. Перша колонка - це є товщина кутника, друга колонка - це максимальне напруження по von Mises і третя колонка - це маса кутника.

Таблиця 4.2 – Результати аналізу алюмінію

Товщина	Напруження	Маса
1	5,914	12,23
1,5	2,912	18,67
2	1,994	25,32
2,5	1,325	32,18

3	1,48	39,25
3,5	6,603	46,54
4	7,325	54,04

За цими даними були створені графіки (рис 4.17). На графіку по шкалі у показано дані по напруженням Мізеса, по шкалі х - товщина кутника. За цими даними ми підібрали оптимальну товщину 2,5 мм. Також на рис 4.18 показано графік маси кутника. Шкала у - це маса кутника і шкала х - це товщина кутника.

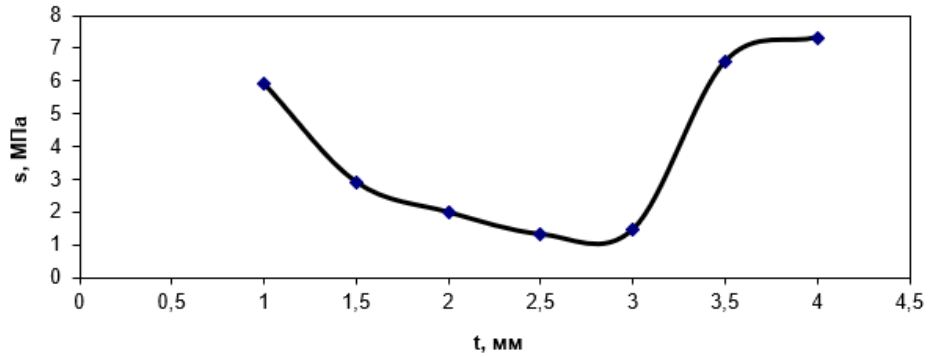


Рис 4.15 – Графік напруження для алюмінію

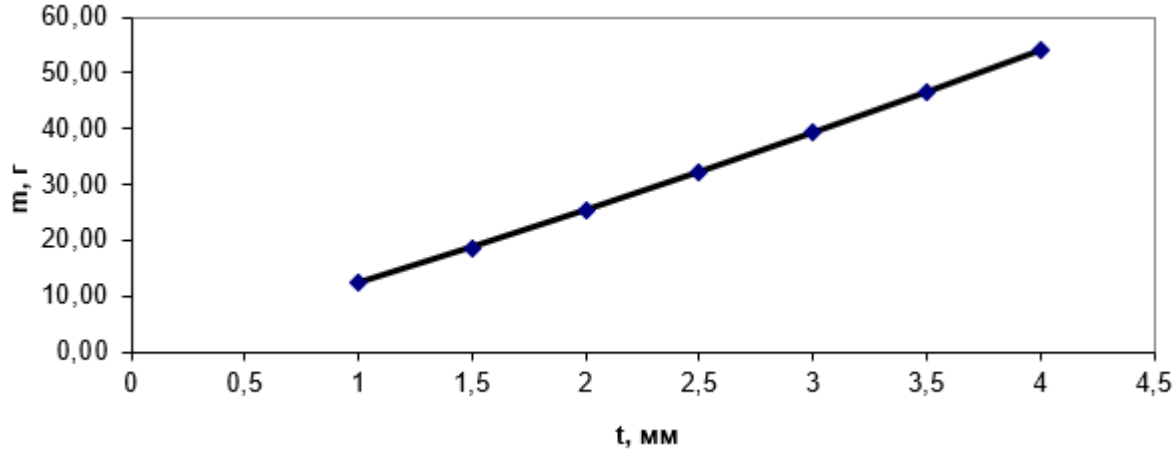


Рис 4.16 – Графік зміни маси

ВИСНОВКИ

1.Проведено аналіз існуючих компонентів для побудови в лабораторних стендів. Виявлено що побудову різноманітних лабораторних стендів можна легко реалізувати за допомогою недорогих компонентів: платформи Arduino, сенсорів та актуаторів для неї, вільної мови програмування Python та її пакетів, універсальних деталей для побудови механічної частини, таких як деталі-структори, алюмінієві профілі або листи та деталі надруковані на 3D принтері.

2.Розроблено принципи проектування лабораторних стендів, які полягають у використанні мови Modelica та її бібліотеки Arduino для їхнього моделювання та платформи Arduino і інших недорогих універсальних компонентів для реалізації. Показано приклад лабораторного стенда на основі Arduino, Python, OpenCV та scikit-image для вивчення мехатронних систем з машинним зором.

3.Розроблено програму для автоматизованого створення та симуляції Modelica-моделей лабораторних стендів, яка основана на побудові морфологічних матриць з різними сполученнями різнотипних компонентів.

4.За допомогою САПР SolidWorks розроблено геометричні параметричні моделі універсальних деталей для побудови механічної частини. Набір деталей включає мінімальну кількість типів деталей, які можуть бути надруковані на 3D принтері або виготовлені з алюмінієвого листа.

5. За допомогою Modelica та SolidWorks розроблено моделі простого лабораторного стенда "маятник", який складається з універсальних компонентів. Цей стенд виготовлено та розроблено програмне забезпечення для його системи керування.

6.Проведено симуляцію роботи стенда за допомогою Modelica, SolidWorks/Motion та SolidWorks/Simulation, отримано кінематичні та динамічні характеристики, які можуть бути використані для оптимізації параметрів конструкції, а також оптимізовано конструкцію деталей за критеріями маси і напружень.

ЛІТЕРАТУРА

1. Ловейкін В.С., Ромасевич Ю.О., Човнюк Ю.В. Мехатроніка навчальний посібник. Київ 2012.
2. https://arduino.ua/art93-pidkluchennya-enkodera-do-ardyino
3. https://www.robostore.com.ua/ua/modul-datchika-skorosti-dlya-arduino/
4. https://arduino.ua/prod2516-gy-291-adxl345
5. http://www.poprobot.ru/ideologia/aktuator
6. https://illustrationprize.com/uk/561-servo-motor.html
7. https://amperka.ru/page/what-is-arduino
8. https://amperka.ru/product/structor-middle
9. https://habr.com/ru/company/selectel/blog/561264/
10. Arduino Uno Rev3. URL: https://store.arduino.cc/arduino-uno-rev3.
11. A free Modelica library for planar mechanical multi-body systems. URL: https://github.com/dzimmer/PlanarMechanics.
12. Simulate circuits and sketches on a virtual Arduino Uno in Modelica. URL: https://github.com/CATIA-Systems/Modelica-Arduino.
13. Бурак, О. Моделювання мехатронних систем на основі Arduino в OpenModelica та їхнє виготовлення / О. Бурак, В. Копей // Міжвузівська науково-практична конференція "Сучасна наука: стан, проблеми та перспективи розвитку": тези доповідей, 24 травня 2021р, Івано-Франківськ. - Івано-Франківськ : ІФФ Університету «Україна», 2021. - С.4-6.
14. Копей, В. Досвід вивчення мехатронних систем за допомогою Modelica та Arduino / В. Копей, О. Бурак // Інформаційні моделюючі технології, системи та комплекси (ІМТСК-2021): збірка тез 3-ї міжнародної науково-практичної конференції. 27-28 травня 2021 р., Черкаси. - Черкаси : Черкаський національний університет імені Богдана Хмельницького, 2021. - С.154-157.

15. Arduino Pendulum - Modelica-model and implementation. URL: https://github.com/vkopey/mechatronics2.
16. firmata/protocol. URL: https://github.com/firmata/protocol.
17. Real-time computer vision with OpenCV / K. Pulli, A. Baksheev, K. Kornyakov, V. Eruhimov // Communications of the ACM. – June 2012. – Vol. 55, No. 6. – P. 61-69. – DOI: 10.1145/2184319.2184337.
18. scikit-image: Image processing in Python / Stéfan van der Walt et al // PeerJ. – 2014. – 2:e453. – Available at: https://doi.org/10.7717/peerj.453.
19. tino/pyFirmata. – Available at: https://github.com/tino/pyFirmata.
20. vkopey/mechatronics1: Simple mechatronic system based on Arduino, Python, OpenCV and scikit-image. – Available at: https://github.com/vkopey/mechatronics1.
21. Копей, В.Б. Лабораторний стенд на основі Arduino, Python, OpenCV та scikit-image для вивчення мехатронних систем / В.Б. Копей, О.В. Бурак // Машинобудування очима молодих: прогресивні ідеї – наука – виробництво: матеріали Дев'ятнадцятої міжнародної молодіжної науково-технічної конференції (м. Суми, 25–26 листопаду 2020 року) / редкол.: В. О. Залога, О. В. Івченко.. - Суми : Сумський державний університет, 2020. - С.155-158.
22. Fritz Zwicky: Morphologische Forschung. Winterthur, 1959, Neuaufl. Glarus: Baeschlin, 1989.
23. Fritzson, Peter A. Principles of object oriented modeling and simulation with Modelica 3.3: a cyber-physical approach / Peter Fritzson. - 2nd edition. - Wiley-IEEE Press, 2014. - 1256 p.
24.Тику Ш. Эффективная работа: SolidWorks 2004. — СПб.: Питер, 2005. — 768 с.: ил.
25. Алямовский А. А. SolidWorks 2007/2008. Компютерное моделирование в инженерной практике. — СПб.: БХВ-Петербург, 2008. — 1040 с.: ил.

Додаток А - Код Arduino-скетча для мехатронної системи "маятник"

Скетч

```
#include <Servo.h>
Servo myservo;
volatile int encCounter;
volatile boolean state, lastState;
boolean x=1,xp=1;
int y;



void int0() {
  state = digitalRead(2);
  if (state != lastState)
     encCounter += 1;
  lastState = state;
}

void servopush() {
   myservo.write(90);
   delay(100);
   myservo.write(0);
   delay(100);
}

void setup() {
  Serial.begin(9600);
  attachInterrupt(0, int0, CHANGE);
```

```
  myservo.attach(9);
}

void loop() {
  Serial.println(encCounter);
  y=analogRead(A0);
  Serial.println(y);
  x=state;
  if (x && xp || y==0)
       servopush();
  xp=x;

  delay(100);
}
```

Python-програма

```
# -*- coding: utf-8 -*-

import time
from pyfirmata import Arduino, util
import matplotlib.pyplot as plt
board = Arduino('COM22')
it = util.Iterator(board)
it.start()
board.analog[0].enable_reporting()
servo=board.get_pin('d:9:s')
pin11 = board.get_pin('d:2:i')
```

```
board.digital[2].enable_reporting()
pin13 = board.get_pin('d:13:o')
pin13.write(1)



def push():
    servo.write(30)
    time.sleep(0.1)
    servo.write(1)
    time.sleep(0.1)

xp, xpp, xppp=1,1,1
y=0
X,Y=[],[]
ln1,=plt.plot(range(len(X)),X)
ln2,=plt.plot(range(len(Y)),Y)
plt.axis([0, 100, 0, 1.1])
while 1:
    x=pin11.read()
    y=board.analog[0].read()
    #print x,
    if all([x,xp,xpp,xppp]):
        push()
    xppp=xpp
    xpp=xp
    xp=x
    X+=[x]
    if len(X)>100: X.pop(0)
```

```
    ln1.set_data(range(len(X)), X)
    Y+=[y]
    if len(Y)>100: Y.pop(0)
    ln2.set_data(range(len(Y)), Y)
    plt.pause(0.0001)
    #if len(X)%20==0:
    #    plt.pause(0.0001)
board.exit()
```

Додаток Б - Код програми для лабораторного стенда на основі Arduino, Python, OpenCV та scikit-image

```
#-*- coding: utf-8 -*-
import numpy as np
import cv2
from skimage.feature import canny
from skimage.transform import hough_circle, hough_circle_peaks
from skimage import io
import pyfirmata

board=pyfirmata.Arduino('COM13')
#print board.get_firmata_version()
servo=board.get_pin('d:9:s')

def detObj(image):
    edges = canny(image, sigma=2, low_threshold=10, high_threshold=20)
    #io.imsave("edges.bmp", edges)
    hough_radii = [16]
    hough_res = hough_circle(edges, hough_radii)
```

```
    accums, cx, cy, r = hough_circle_peaks(hough_res, hough_radii,
total_num_peaks=1)
    return cx, cy, r

cap = cv2.VideoCapture(0)
while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    cx, cy, r=detObj(gray)
    print cx, cy, r
    #cv2.circle(gray, (cx, cy), r, 255, -1)

    servo.write(cx/20)

    #cv2.imshow('frame',gray)
    #if cv2.waitKey(1) & 0xFF == ord('q'): break

cap.release()
cv2.destroyAllWindows()
board.exit()
```

Додаток В - Код програми для автоматизованого створення та симуляції Modelica-моделей лабораторних стендів

Модуль ModelicaCAD.py

```
#encoding: utf-8
import os, sys
```

```
##
class controlPart:
   components="""  Arduino.Components.ArduinoUno arduinoUno annotation(
   Placement(visible = true, transformation(origin = {-46, -38}, extent = {{-16, -20},
{16, 20}}, rotation = 0)));
  Arduino.Components.Servo servo annotation(
   Placement(visible = true, transformation(origin = {-6, -36}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Arduino.Components.SEN136B5B sen136b5b annotation(
   Placement(visible = true, transformation(origin = {0, -64}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Electrical.Analog.Basic.Ground ground annotation(
   Placement(visible = true, transformation(origin = {-46, -80}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));
  Modelica.Electrical.Analog.Sources.ConstantVoltage constantVoltage(V = 5)
annotation(
   Placement(visible = true, transformation(origin = {-74, -70}, extent = {{-10, -10},
{10, 10}}, rotation = 0)));"""

   equations="""  connect(ground.p, arduinoUno.GND) annotation(
   Line(points = {{-46, -70}, {-46, -58}}, color = {0, 0, 255}));
  connect(constantVoltage.n, ground.p) annotation(
   Line(points = {{-64, -70}, {-46, -70}}, color = {0, 0, 255}));
  connect(constantVoltage.p, arduinoUno.Vin) annotation(
   Line(points = {{-84, -70}, {-84, -18}, {-46, -18}}, color = {0, 0, 255}));
  connect(arduinoUno.D9, servo.SIG) annotation(
   Line(points = {{-30, -33}, {-16, -33}}, color = {0, 0, 255}));
  connect(servo.GND, ground.p) annotation(
```

```
    Line(points = {{-16, -40}, {-16, -70}, {-46, -70}}, color = {0, 0, 255}));
  connect(arduinoUno.D7, sen136b5b.pinSig) annotation(
    Line(points = {{-30, -37}, {-24, -37}, {-24, -60}, {-10, -60}}, color = {0, 0, 255}));
  connect(arduinoUno.Vin, sen136b5b.input5V) annotation(
    Line(points = {{-46, -18}, {-22, -18}, {-22, -64}, {-10, -64}}, color = {0, 0, 255}));
  connect(ground.p, sen136b5b.ground) annotation(
    Line(points = {{-46, -70}, {-10, -70}, {-10, -69}}, color = {0, 0, 255}));
"""

##
class Flange(object):
    "Modelica-коннектор компонента"
    def __init__(self, comp, name, origin):
        self.comp=comp
        self.name=name
        self.origin=origin
    #def setName(self):

##
class Component(object):
    """Modelica-компонент"""

    def __init__(self, params="", origin=[0, 0], name=''): # model ?
        self.params=params
        self.origin=origin
        self.name=name

    def className(self):
```

```
        "Повертає Modelica-назву класу"
        return self.__class__.__name__.replace("_", ".")

    def code(self, anno=True):
        "Modelica-код декларації компонента в моделі"
        if anno:
            anno=self.anno()
        else:
            anno=""
        if self.params:
            return "%s %s(%s) %s;"%(self.className(), self.name, self.params, anno)
        else:
            return "%s %s %s;"%(self.className(), self.name, anno)

    def anno(self):
        "Анотація як частина Modelica-коду"
        return """annotation(
    Placement(visible = true, transformation(origin = {%d, %d}, extent = {{-10, -10},
{10, 10}}, rotation = 0)))"""%tuple(self.origin)

##
class Modelica_Mechanics_Translational_Components_Fixed(Component):
    def __init__(self, params="", origin=[0, 0], name=''):
        Component.__init__(self, params, origin, name)
        self.flange=Flange(self, "flange", [0,0])

##
class Modelica_Mechanics_Translational_Components_Spring(Component):
```

```
    def __init__(self, params="", origin=[0, 0], name=''):
        Component.__init__(self, params, origin, name)
        self.flange_a=Flange(self, "flange_a", [-10, 0])
        self.flange_b=Flange(self, "flange_b", [10, 0])

##
class Modelica_Mechanics_Translational_Components_Mass(Component):
    def __init__(self, params="", origin=[0, 0], name=''):
        Component.__init__(self, params, origin, name)
        self.flange_a=Flange(self, "flange_a", [-10, 0])
        self.flange_b=Flange(self, "flange_b", [10, 0])

##
class Model(object):
    """Modelica-модель"""
    name="myModel"
    connected=[] # список пар зєднаних компонентів
    eqs=[] # список рівнянь (connect(...))

    def code_com(self):
        "Modelica-код опису компонентів"
        s=""
        for n,c in self.__dict__.items():
            if Component in c.__class__.__bases__:
                c.name=n
                s+=c.code()+"\n"
        return s
```

```
    def code_eqs(self):
        "Modelica-код опису рівнянь"
        self.eqs=[]
        for a,b,anno in self.connected:
            self.connect_(a,b,anno)
        return "\n".join(self.eqs)

    def connect(self, a, b, anno=True):
        "Додає пару зєднаних фланців у connected"
        self.connected.append([a,b,anno])

    def connect_(self, a, b, anno):
        "Modelica-код зєднання компонентів з анотацією"
        s="connect(%s, %s)"%(a.comp.name+'.'+a.name, b.comp.name+'.'+b.name)
        #a=a.comp.origin
        #b=b.comp.origin
        a=a.comp.origin[0]+a.origin[0], a.comp.origin[1]+a.origin[1]
        b=b.comp.origin[0]+b.origin[0], b.comp.origin[1]+b.origin[1]
        if anno: s+=""" annotation(
    Line(points = {{%d, %d}, {%d, %d}}, color = {0, 127, 0}))"""%(a[0],a[1],b[0],b[1])
        self.eqs.append("%s;"%s)

    def code_gen(self):
        "Генерує Modelica-код моделі"
        s="""model %s
%s

%s
```

```
equation

%s

%s

annotation(
   uses(Modelica(version = "3.2.3")));
end %s;"""
      return s%(self.name, self.code_com(), controlPart.components,
self.code_eqs(), controlPart.equations, self.name)

   def autoOrigin(self):
      "Автоматично розміщує компоненти"
      x,y=-90,90
      s=set()
      for a,b,anno in self.connected:
         if a.comp not in s:
            a.comp.origin=[x,y]
         if b.comp not in s:
            b.comp.origin=[a.comp.origin[0]+50, a.comp.origin[1]-50]
         x+=50
         y-=50
         s.add(a.comp)
         s.add(b.comp)
      """
      d={}
```

```
        for a,b,anno in self.connected:
            for i in (a,b):
                if d.has_key(i.comp):
                    d[i.comp]+=1
                else:
                    d[i.comp]=1
        print sorted(d.items(),key=lambda x: x[1], reverse=True)
        """



    def saveCode(self):
        "Зберігає код"
        f=open("%s.mo"%self.name,"w")
        f.write(self.code_gen())
        f.close()

    def run(self):
        "Виконує симуляцію"
        sys.path.insert(0, r"e:\OpenModelica1.16.2-
64bit\share\omc\scripts\PythonInterface") # шлях до модулів
        from OMPython import OMCSession, ModelicaSystem

        # перший спосіб - використання OMCSession:
        omc = OMCSession()
        omc.sendExpression('loadFile("myModel.mo")')
        #omc.sendExpression('setParameterValue(myModel, mass.m, 2)')
        omc.sendExpression('simulate(myModel)')
        omc.sendExpression('plot(mass.s)')
```

```
    print omc.sendExpression('val(mass.s , 1.0)') # результат time=1.0


D=[Modelica_Mechanics_Translational_Components_Fixed(name='f1'),
Modelica_Mechanics_Translational_Components_Spring(params="c = 1",
name='s1'),
Modelica_Mechanics_Translational_Components_Mass(params="m = 1",
name='m1'),
Modelica_Mechanics_Translational_Components_Fixed(name='f2'),
Modelica_Mechanics_Translational_Components_Spring(params="c = 1",
name='s2'),
Modelica_Mechanics_Translational_Components_Mass(params="m = 1",
name='m2')]
C=[]
for x in D:
  try:
    C.append(x.flange_a)
  except:
    pass
  try:
    C.append(x.flange_b)
  except:
    pass
  try:
    C.append(x.flange)
  except:
    pass

##
```

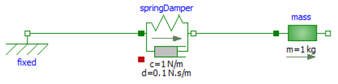
```
if __name__=='__main__':
    m=Model()
    m.fixed=Modelica_Mechanics_Translational_Components_Fixed()
    m.fixed.origin=[-50, 0]
    m.spring=Modelica_Mechanics_Translational_Components_Spring("c = 1")
    m.spring.origin=[0, 0]
    m.mass=Modelica_Mechanics_Translational_Components_Mass("m = 1, s(fixed
= true, start = 1)")
    m.mass.origin=[50, 0]
    m.connect(m.fixed.flange, m.spring.flange_a)
    m.connect(m.spring.flange_b, m.mass.flange_a)


    m.code_gen()
    m.autoOrigin()
    m.saveCode()
    #m.run()
```

Модуль ModelicaCADgui.py

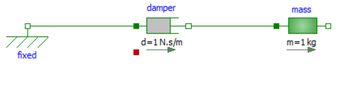
```
#encoding: utf-8
from ModelicaCAD import *
from Tkinter import * # імпортувати все з модуля Tkinter

def Button1Click(): # функція, яка викликається під час натиску на Button1
    S=set()
    m=Model()
    for i,y in enumerate(BV):
        s=''
        for j,x in enumerate(y):
            s=s+' %d'%int(x.get())
```

```
            if x.get():
                S.add(C[i].comp)
                S.add(C[j].comp)
                m.connect(C[i],C[j])
        print s

    for s in S:
        m.__setattr__(s.name, s)
    m.code_gen()
    m.autoOrigin()
    m.saveCode()

root = Tk() # головне вікно програми
root.title('ModelicaCAD GUI app') # надпис на вікні
#root.geometry("200x150+0+0") # розмір вікна
#C=[0,1,2,3]
BV=[]
for y,c in enumerate(C):
    Label(root,text=c.comp.name+'.'+c.name).grid(row=y+1, column=0)
    BV.append([])
    for x,c in enumerate(C):
        Label(root,text=c.comp.name+'.'+c.name).grid(row=0, column=x+1)
        bv=BooleanVar() # створити булеву змінну
        BV[-1].append(bv)
        check=Checkbutton(root,variable=bv) # прапорець, пов'язати зі змінною bv
        check.grid(row=y+1, column=x+1) # розмістити
Button1=Button(root, text="run", command=Button1Click) # створити кнопку,
пов'язати з функцією command1
```

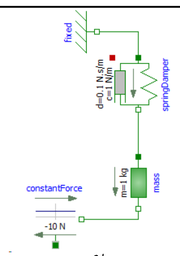
```
Button1.grid(row=y+2, column=x+2) # розмістити
root.mainloop() # головний цикл програми (для обробки подій)
```



а)

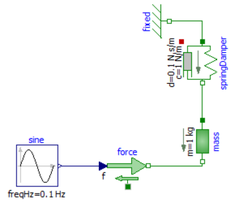


б)

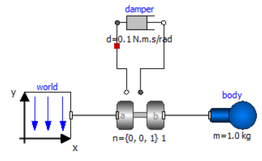


в)

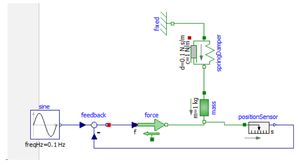
Моделі механічних систем



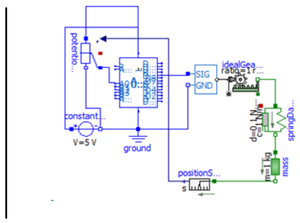
г)



д)

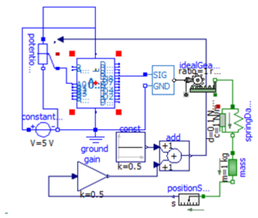


а)



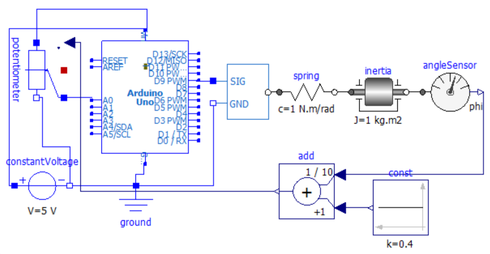
б)

Моделі мехатронних систем

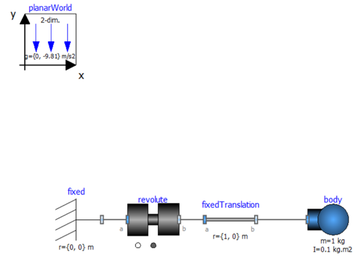


в)

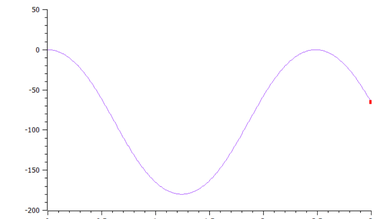
Модель мехатронної системи



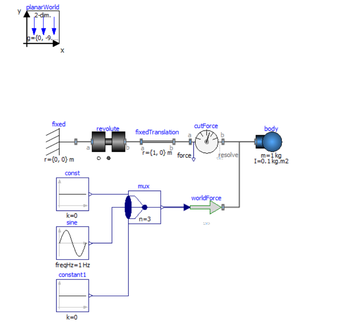
Модель мехатронної системи в OpenModelica



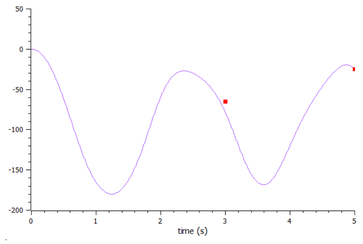
Модель маятника, зроблена за допомогою компонентів PlanarMechanics



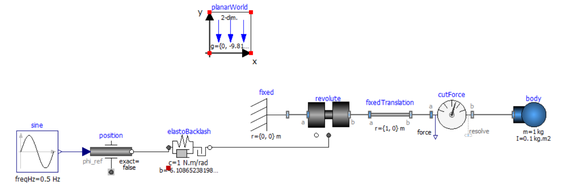
Результати симуляції – кут повороту (grad)



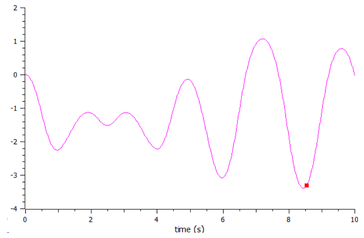
Модель маятника, який
підштовхується зовнішньою силою



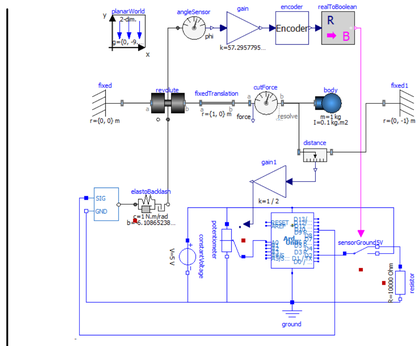
Результати симуляції – кут повороту (grad)



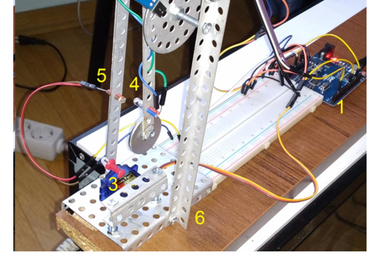
Інший спосіб побудови моделі



Результати симуляції – кут повороту (grad)

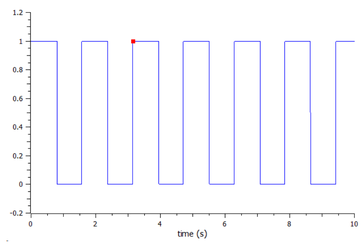


Модель мехатронної системи "маятник"

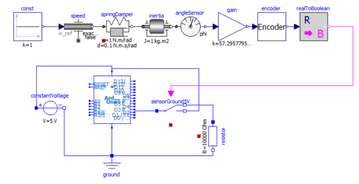


Лабораторний стенд на основі Arduino:
Arduino Uno (1), енкодер на LM393 (2)
сервопривід SG90 (3), тактовий (4),
фоторезистор (5)
деталі металевого конструктора (6)

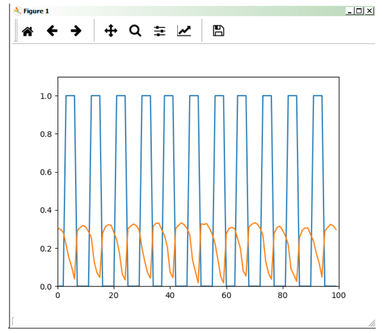
Модель для тестування енкодера з параметром angle=90



Модель для тестування енкодера з параметром angle=90



Модель для тестування енкодера з використанням компонента Arduino



Сигнали енкодера і фоторезистора

					БР.063.00.01	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Бирюк О.В.			Моделювання сценаріїв в OpenModelica	Лит. / Масса / Масштаб: 1:1
Пров.		Колей В.Б.				Лист / Листов: 1
Т.контр.						ПМІ-19-1К
Н.контр.						
Утв.						

Копировал

Формат А1

Перв. примен.

Справ. №

Подп. и дата

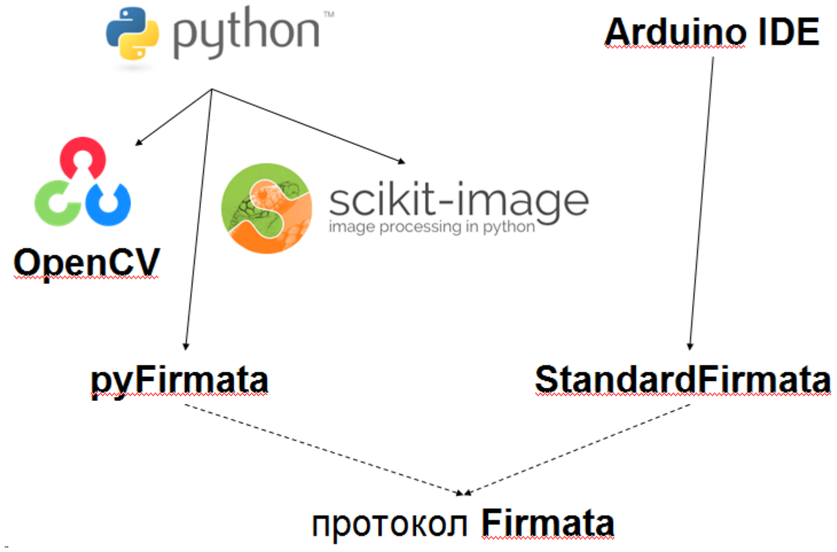
Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

Компоненти для лабораторного стенда



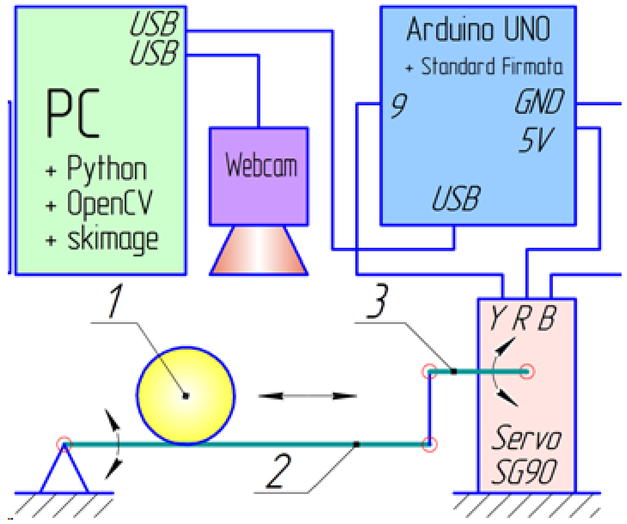
Безкоштовне програмне забезпечення

```
while(True):
    ret, frame = cap.read()
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    cx, cy, r = detObj(gray)
    servo.write(cx/20)
```

У detObj для визначення границь об'єктів використовується алгоритм Кенні.
Ці границі і список значень радіусів кілець R, які потрібно виявити
передаються функціям для визначення кільцевих форм на основі перетворення Хафа.

Функції знаходять найбільш імовірні координати центра кільця (cx, cy):

```
edges = canny(image, sigma=2, low_threshold=10, high_threshold=20)
R = [16]; hough_res = hough_circle(edges, R)
accums, cx, cy, r = hough_circle_peaks(hough_res, R,
                                       total_num_peaks=1)
```



1 – кулька, 2 – кривошип з жолобом, 3 – кривошип сервоприводу

Схема лабораторного стенда

					БР.063.00.02	
Изм.	Лист	№ докум.	Подп.	Дата		
Разраб.		Бурак О.В			Лабораторний стенд з машинним зором	Лит. / Масса / Масштаб: 1:1
Пров.		Копей В.Б				Лист / Листов: 1
Т.контр.						
Н.контр.						ПМІ-19-1К
Утв.						

Копировал

Формат А2

Перв. примен.

Справ. №

Подп. и дата

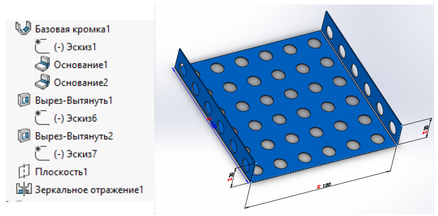
Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

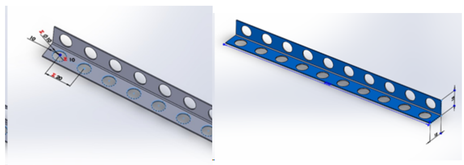
БР.063.00.03



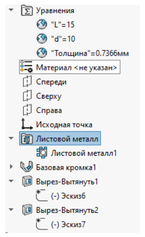
Дерево побудови і модель (5) планки

Имя	Значение / Уравнение	Равняется
Глобальные переменные		
"d"	= 10	10
"L"	= 120	120
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D5@Эскиз6"	= "L" / ("d" * 2)	6
"D1@Эскиз6"	= "d"	10мм
"D4@Эскиз6"	= "L" / ("d" * 2)	6
"D3@Эскиз7"	= "L" / ("d" * 2)	6
"D2@Эскиз1"	= "L"	120мм
"D6@Эскиз6"	= "d" * 2	20мм
"D2@Эскиз6"	= "d"	10мм
"D2@Базовая кромка1"	= "L"	120мм
"D4@Эскиз7"	= "d" * 2	20мм
"D7@Эскиз6"	= "d" * 2	20мм
"D3@Эскиз1"	= "d" * 2	20мм
"D1@Эскиз1"	= "d" * 2	20мм

Таблиця рівняння моделі



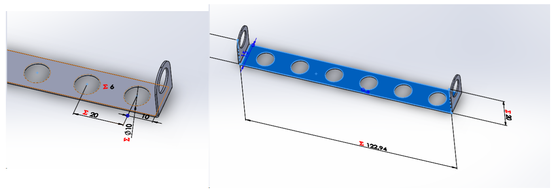
Побудова отворів і модель кутника



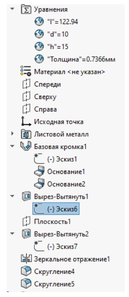
Дерево побудови моделі кутника

Имя	Значение / Уравнение	Равняется
Глобальные переменные		
"L"	= 15	15
"d"	= 10	10
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D1@Эскиз6"	= "d"	10мм
"D3@Эскиз6"	= "L" / ("d" * 2)	1
"D3@Эскиз7"	= "L" / ("d" * 2)	1
"D4@Эскиз6"	= "d" * 2	20мм
"D4@Эскиз7"	= "d" * 2	20мм

Таблиця рівняння моделі



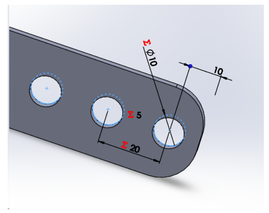
Побудова отворів і модель (6) планки



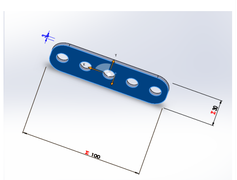
Дерево побудови моделі

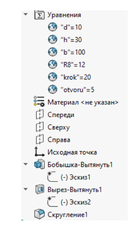
Имя	Значение / Уравнение	Равняется
Глобальные переменные		
"l"	= 122.94	122.94
"d"	= 10	10
"h"	= 15	15
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D1@Эскиз1"	= "l"	122.94мм
"D1@Эскиз6"	= "d"	10мм
"D3@Эскиз6"	= "l" / ("d" * 2)	6
"D2@Эскиз1"	= "d" * 2	20мм
"D4@Эскиз6"	= "d" * 2	20мм

Таблиця рівняння моделі



Побудова отворів (а) і модель (8) планки

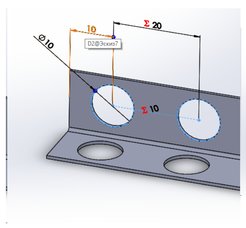




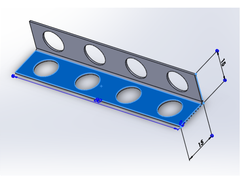
Дерево побудови моделі

"d"	= 10	10
"h"	= 30	30
"b"	= 100	100
"R8"	= 12	12
"krok"	= 20	20
"otvoru"	= 5	5
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D2@Эскиз2"	= "d"	10мм
"D1@Эскиз1"	= "h"	30мм
"D2@Эскиз1"	= "b"	100мм
"D1@Скругление1"	= "R8"	12мм
"D4@Эскиз2"	= 2 * "d"	20мм
"D3@Эскиз2"	= "b" / "krok"	5

Таблиця рівняння моделі

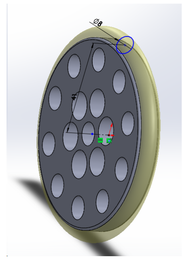


Побудова отворів і модель (8) кутника

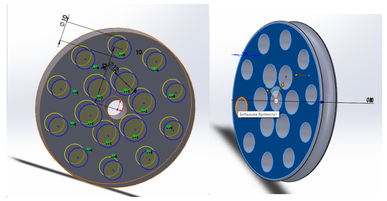


Глобальные переменные		
"L"	= 15	15
"d"	= 10	10
Толщина	= 0.74	0.74мм
Добавить глобальную переменную		
Элементы		
Добавить погашение элемента		
Уравнения		
"D1@Эскиз6"	= "d"	10мм
"D3@Эскиз6"	= "L" / ("d" * 2)	1
"D3@Эскиз7"	= "L" / ("d" * 2)	1
"D4@Эскиз6"	= "d" * 2	20мм
"D4@Эскиз7"	= "d" * 2	20мм

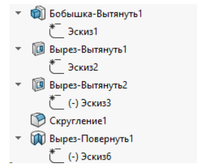
Таблиця рівняння моделі



Принцип побудови вирізу



Побудова отворів і модель колеса



Дерево побудови моделі

Изм.	Лист	№ докум.	Подп.	Дата
Разраб.		Бурак О.В		
Пров.		Копей В.Б		
Т.контр.				
Н.контр.				
Утв.				

БР.063.00.03

Моделі деталей у Solidworks

Лит.	Масса	Масштаб
		1:1
Лист	Листов 1	

ПМІ-19-1К

Копировал

Формат А1

Перв. примен.

Справ. №

Подп. и дата

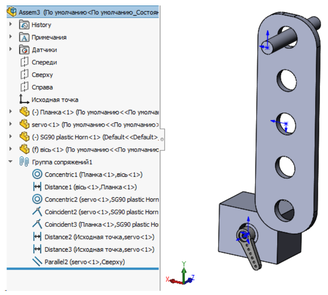
Инв. № дубл.

Взам. инв. №

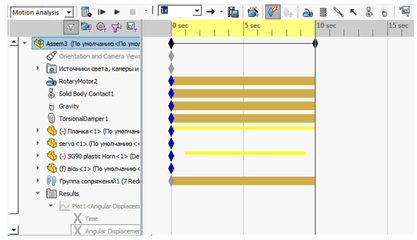
Подп. и дата

Инв. № подл.

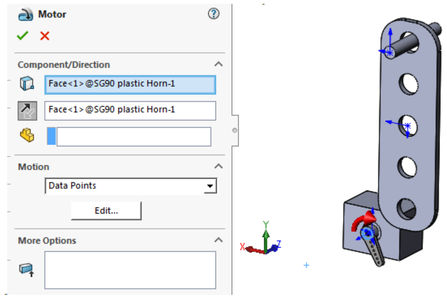
БР.063.00.04



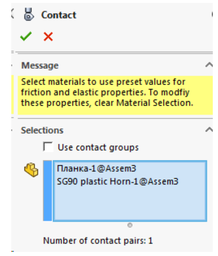
Модель маятника та група сполучень



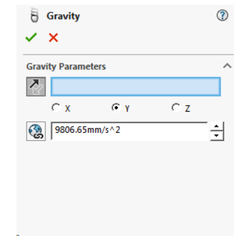
Дерево побудови симуляції



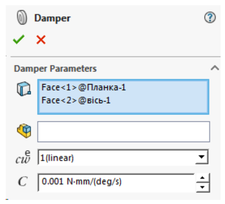
Параметри Rotary Motor



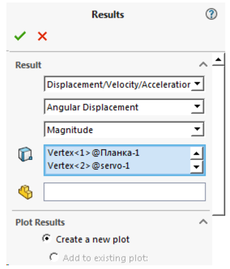
Параметри контактів



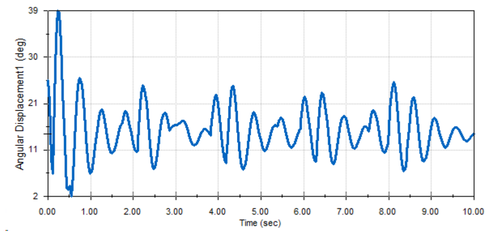
Гравітація моделі



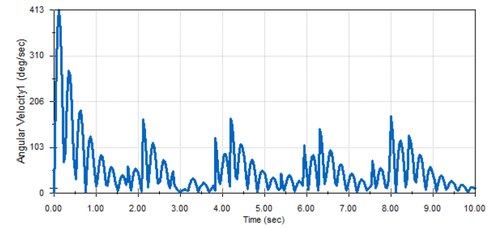
Контакти демпфера



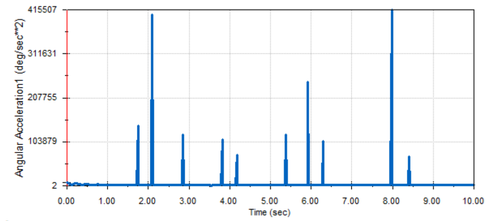
Кутове переміщення



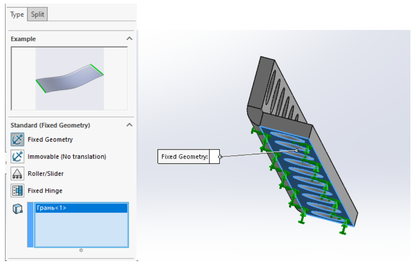
Графік кутового переміщення



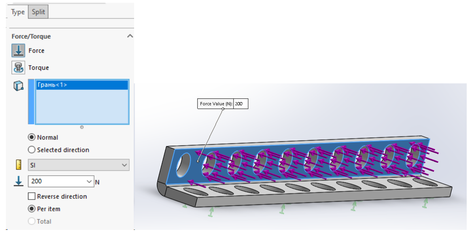
Графік кутової швидкості



Графік кутового прискорення



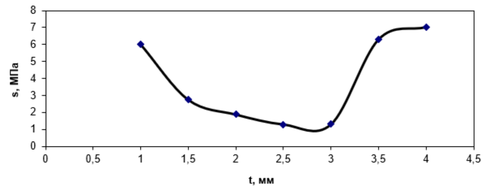
Тип фіксації і місце фіксації кутника



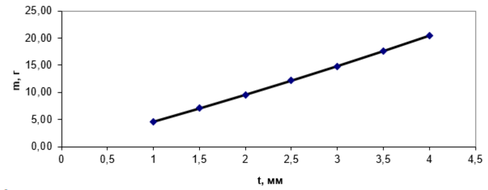
Параметри сили і місце дії сили на кутник

1	6,009	4,61
1,5	2,744	7,04
2	1,872	9,55
2,5	1,289	12,13
3	1,3	14,8
3,5	6,31	17,55
4	7,028	20,38

Результати аналізу пластика



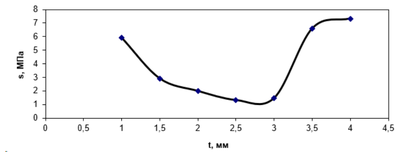
Графік напружень для пластика



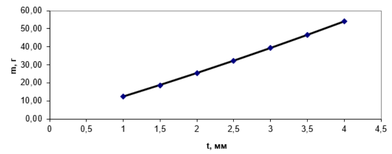
Графік зміни маси

1	5,914	12,23
1,5	2,912	18,67
2	1,994	25,32
2,5	1,325	32,18
3	1,48	39,25
3,5	6,603	46,54
4	7,325	54,04

Результати аналізу алюмінія



Результати напруження для алюмінія



Графік зміни маси

Изм.	Лист	№ докум.	Подп.	Дата	БР.063.00.04
Разраб.		Бурак О.В			Симуляція маятника у Solidworks
Пров.		Копей В.Б			Лит. / Масса / Масштаб: 1:1
Т.контр.					Лист / Листов: 1
Н.контр.					ПМІ-19-1К
Утв.					

Копировал

Формат А1

Перв. примен.

Справ. №

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.