МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний аерокосмічний університет ім. М. Є. Жуковського
«Харківський авіаційний інститут»
Факультет літакобудування
Кафедра інформаційних технологій проектування

Курсовий проект

Програмне забезпечення роботизованих систем
(назва дисципліни)

на тему: «Лінійний робот, лінійні переміщення за трьома координатами»

Виконав: студент 3 курсу групи № 137
спеціальності “Комп’ютерні науки”
(шифр і назва напряму підготовки (спеціальності))
Рєпін О. В.
(прізвище й ініціали студента)
Керівник:
старший викладач Єремєєв М. Б.
(посада, науковий ступінь, прізвище й ініціали)
Національна шкала: _____
Кількість балів: _____
Оцінка: ECTS _____

Члени комісії:

_____ _____
(підпис) (прізвище й ініціали)

_____ _____
(підпис) (прізвище й ініціали)

_____ _____
(підпис) (прізвище й ініціали)

Харків – 2023

ЗМІСТ

ЗАВДАННЯ

1. Проаналізувати завдання та обрати модель та алгоритм для виконання завдання.

2. Створити у середовищі Solid Works модель вузла робота.

3. Описати математичну модель для роботи робота

4. Створити блок-схему, яка описує поведінку алгоритму робота.

5. Розробити програму на будь-якій мові програмування, так щоб програма виводила кожен крок робота та реалізовувала кожен крок у окремій функції.

6. Перевірити програму тестуванням її при декількох різних значеннях.

1. ВСТУП

Розробити модель лінійного робота з лінійним переміщенням по трьом координатам. Для розробки такого робота було 3д-принтер так як він максимально схожий за концептом, приклад 3д-принтера зображено на рисунку 1.1.

Модель робота буде створено з програмним забезпеченням Solid Works 2022 та буде складатися з 6 деталей та 1 готової збірки.

Програма буде мати графічний інтерфейс та написана на мові програмування Python.

Розрахувати що кожен крок робота на 1 одиницю, цей крок не має певної одинці виміру а є мікро-кроком крокового двигуна робота

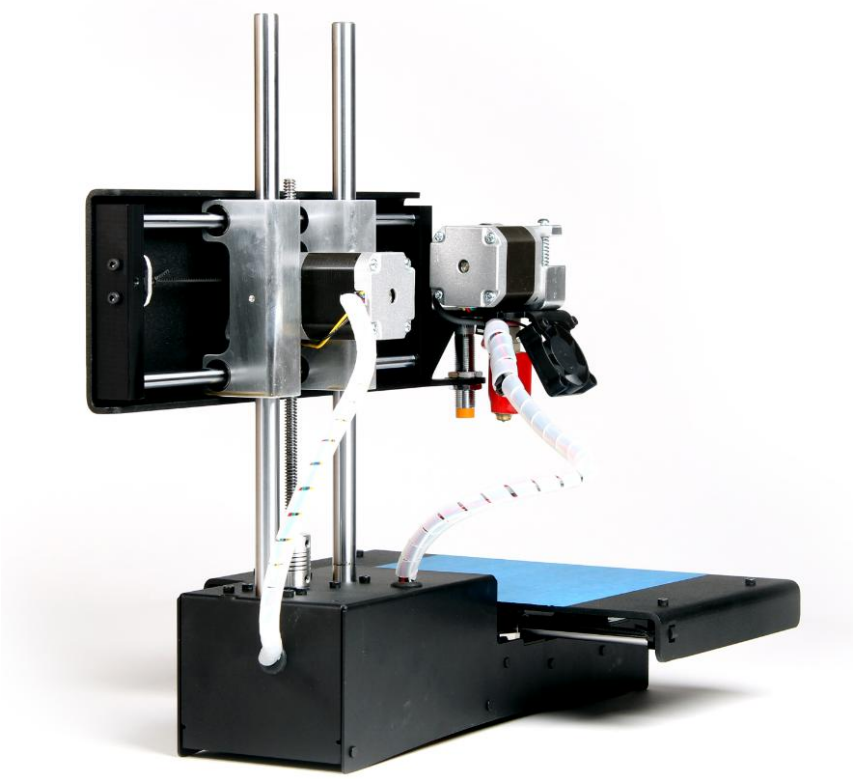

Рисунок 1.1 - 3д-принтер

2. МОДЕЛЮВАННЯ РОБОТА

Лінійний робот з лінійним переміщення найближче репрезентується 3д-принтером тому буде створюватися проста модель модель, яка буде репрезентувати рух робота в 3 координат з обмеженням у (0, 0, 0) в першій координаті та (500, 500, 500) у другій координаті. [1]

2.1 Створення деталей

Основа принтера - це деталь, яка буде реалізовувати рух принтера по осі Y, приклад деталі зображено на рисунку 2.1.1.

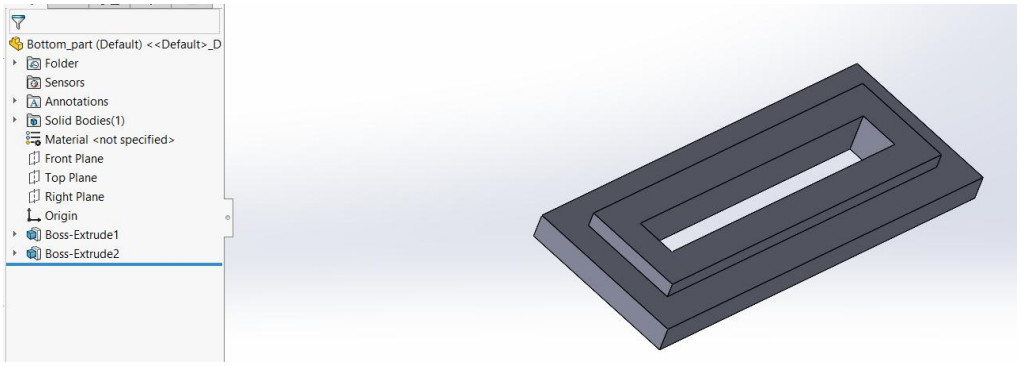

Рисунок 2.1.1 - Основа принтера

Другою основною деталю буде деталя, яка вставляється в основу для її руху по осі Y, та має виріз для руху деталі по осі Z, ця деталь зоброжена на рисунку 2.1.2.

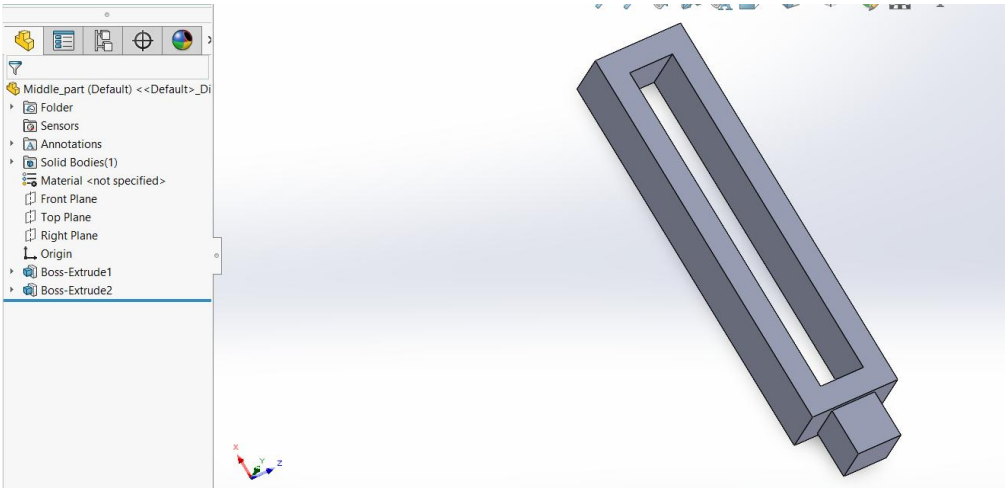

Рисунок 2.1.2 - Основна велика рухома деталь принтера

Для руху по осі Z створено 3 основну деталь, цю деталь зображено на рисунку 2.1.3, ця буде встановлюватися у другу деталь для руху по осі Y та для руху самої деталі по осі Z, також деталь має кріплення для пензля.

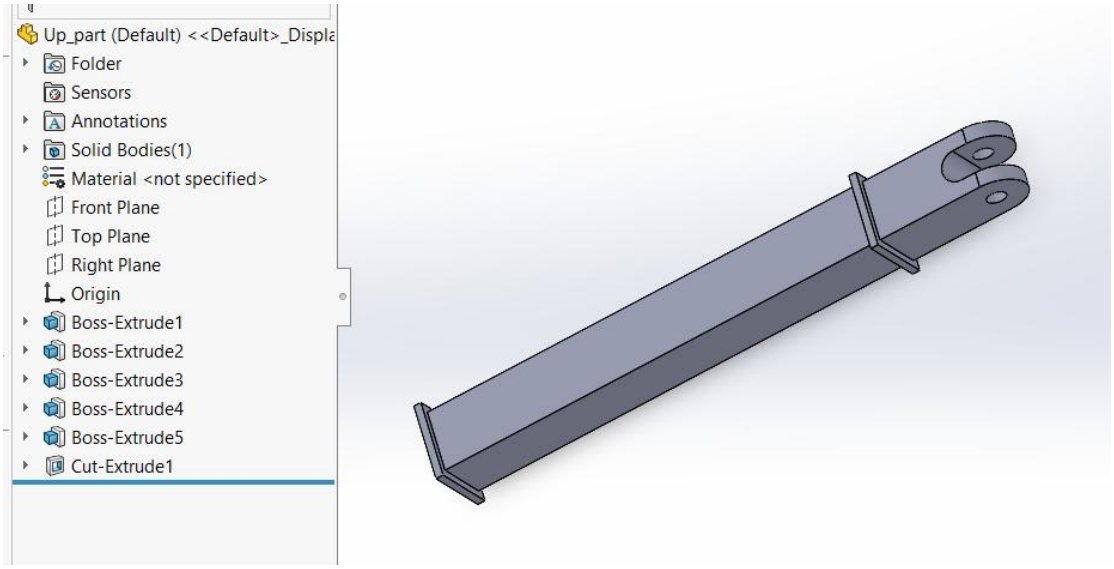

Рисунок 2.1.3 - Основна мала рухома деталь принтера

Пензель - додаткова деталь кінчик якої репрезентує точку яка буде рухатися в просторі, зображено деталь на рисунку 2.1.4.

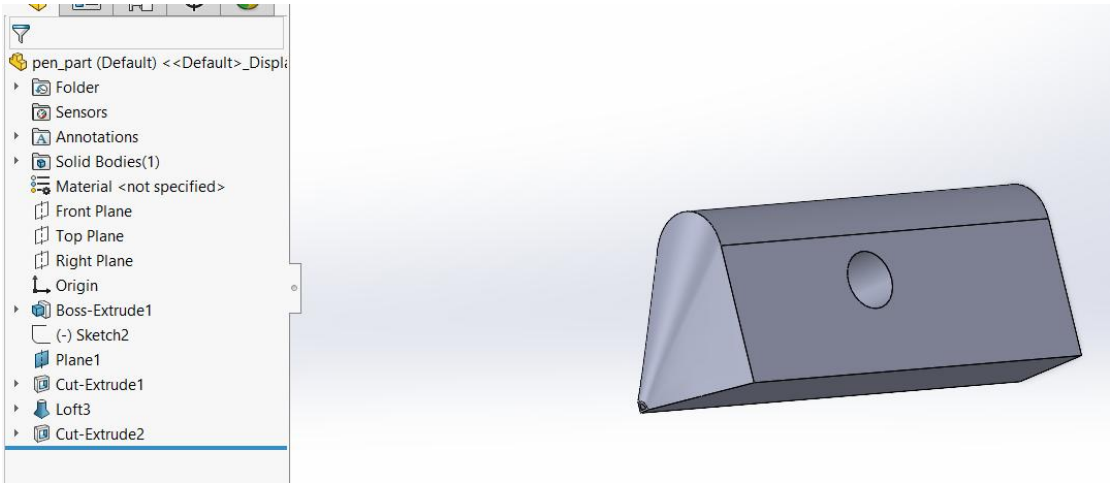

Рисунок 2.1.4 - Пензель принтера

Болт для кріпленя пензля до основної деталі, зображено на рисунку 2.1.5.

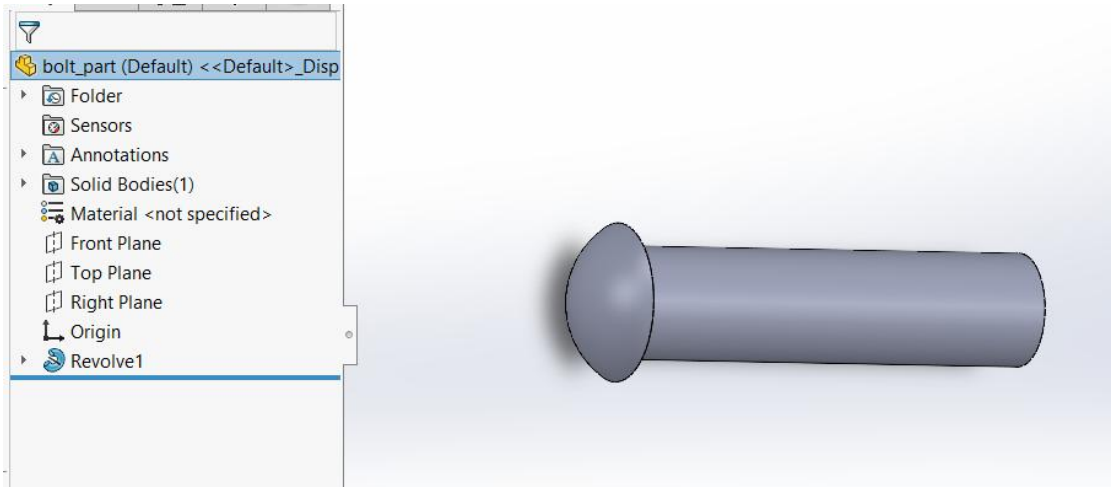

Рисунок 2.1.5 - Болт для кріплення пензля

Щоб показати розмір простору в якому можна рухатися та відкалібрувати координати буде використовуватися підставка, яка зображено на рисунку 2.1.6.

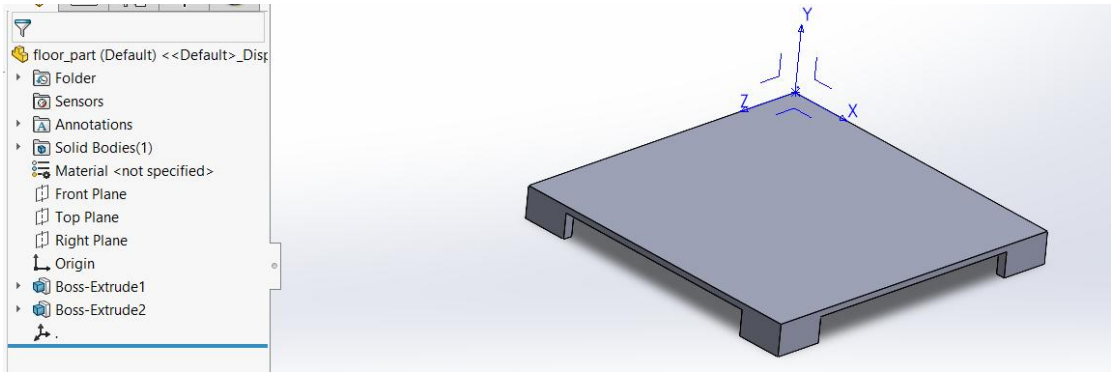

Рисунок 2.1.6 - Підставка для деталей

2.2 Збірка моделі

Створена з 6 деталей збірка має додаткові позначення положення точки та можливість переміщення по трьом координатам у межах від (0, 0, 0) до (500, 500, 500). Готову збірку можна побачити на рисунку 2.2.1.

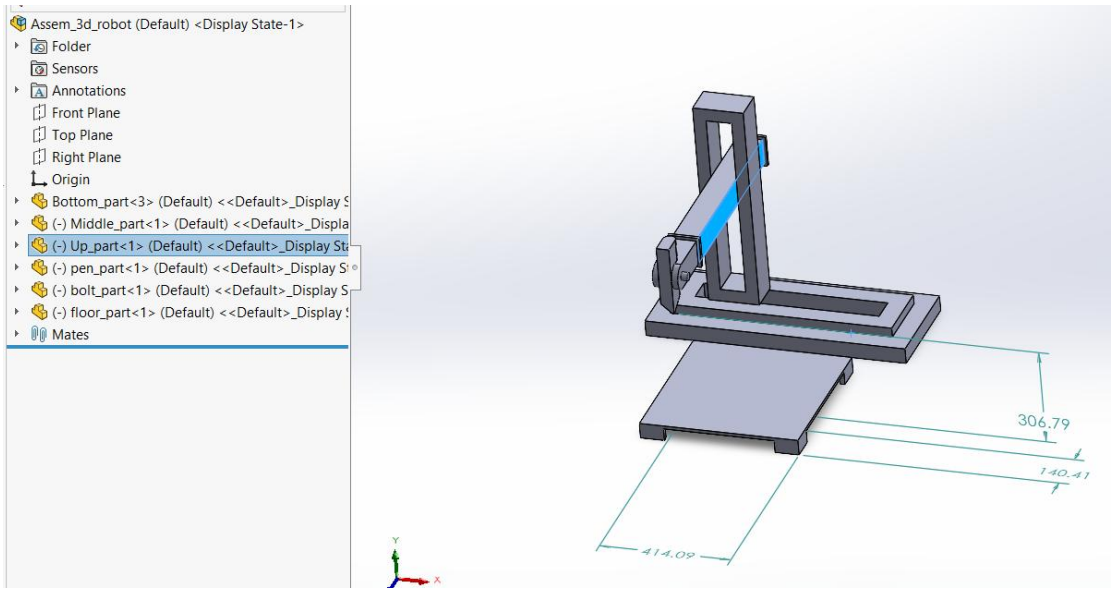

Рисунок 2.2.1 - Збірка моделі

2.3 Тест моделі

Для візуалізації розмірів рухомого простору принтера, представлено три приклади, які зображено на рисунку 2.3.1-2.3.3, де зображено положення у мінімальних, максимальних координатах та в середині координат.

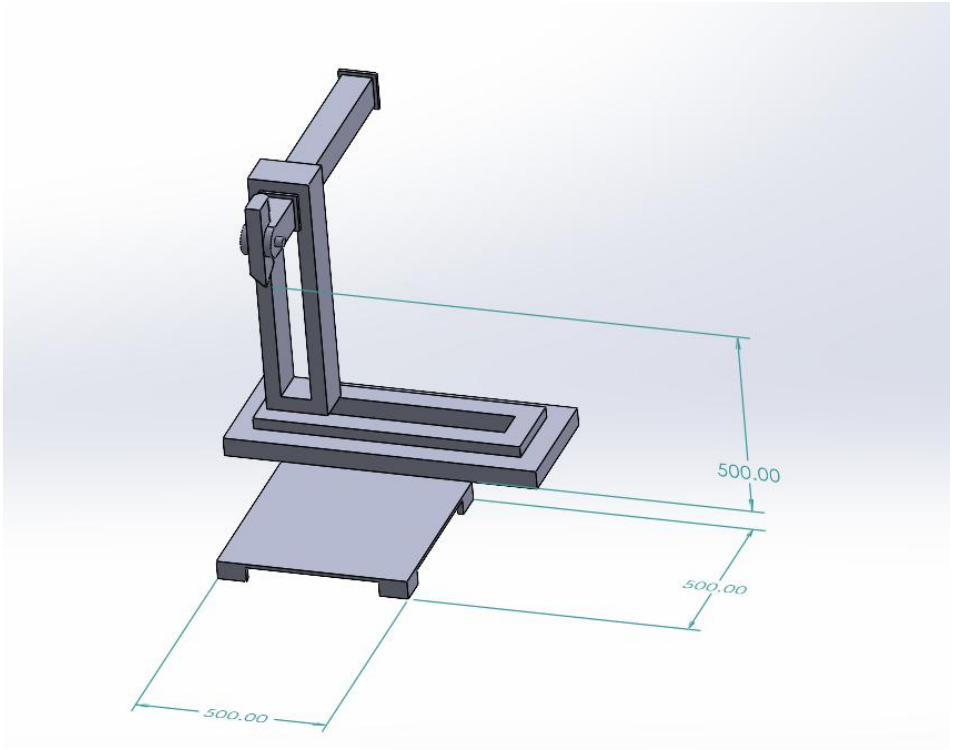

Рисунок 2.3.1 - Приклад 1

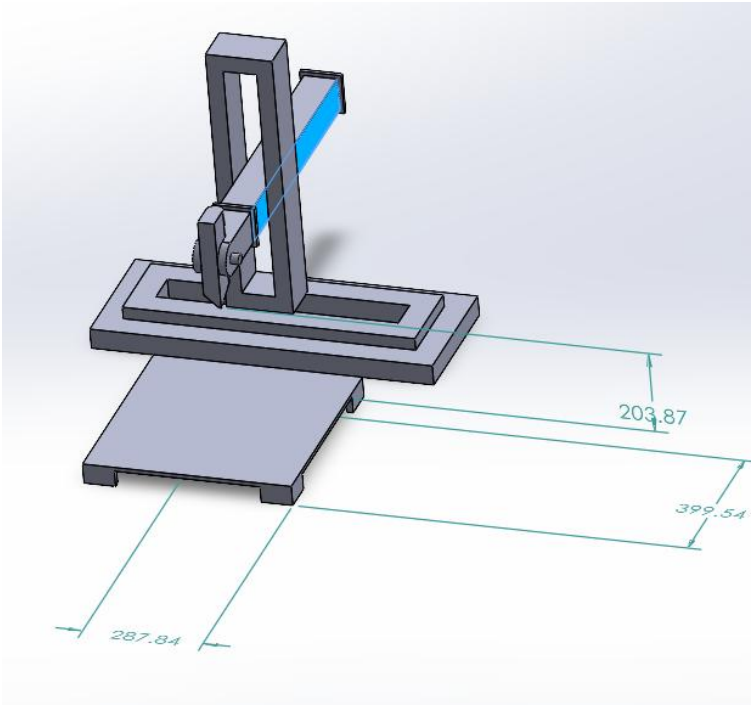

Рисунок 2.2.1 - Приклад 2

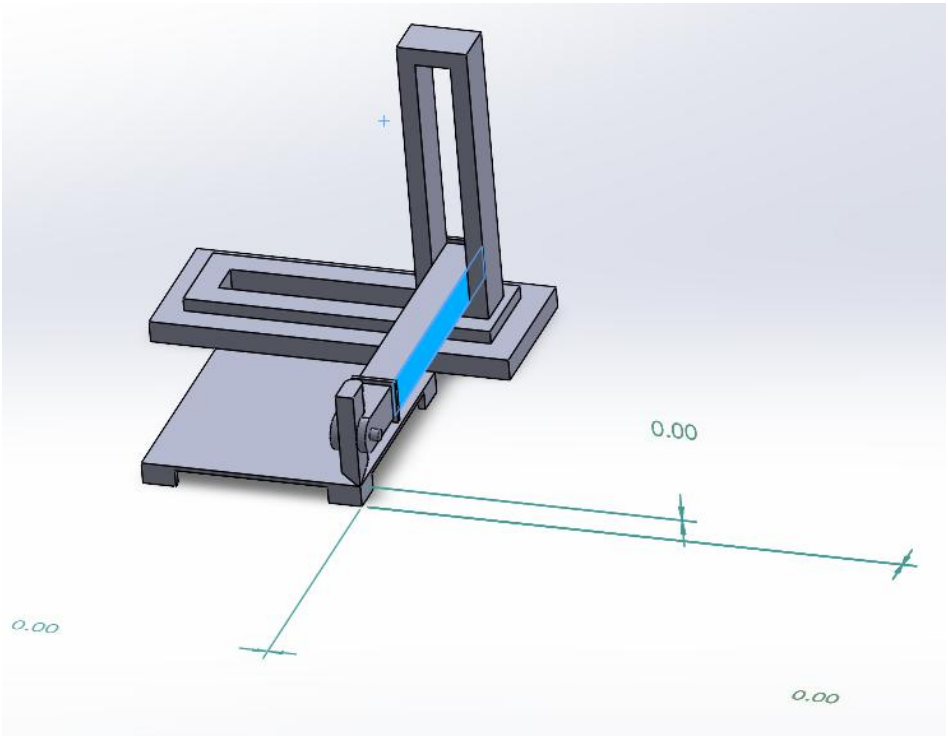

Рисунок 2.2.1 - Приклад 3

3. РОЗРАХУНОК МАТЕМАТИЧНОЇ МОДЕЛІ

Для руху робота було обрано розширений алгоритм Брезехема. [4]

Алгоритм Брезенхема для 3D-простору розширюється відносно алгоритму для 2D-площини. Він може використовуватися для побудови тривимірних ліній і відрізків у тривимірному просторі. Для цього ми будемо відстежувати, які пікселі мають бути засвічені, щоб намалювати тривимірну лінію між двома точками в 3D-просторі.

Нехай у нас є початкова точка $P_1(x_1, y_1, z_1)$ і кінцева точка $P_2(x_2, y_2, z_2)$, і ми хочемо намалювати лінію між ними.

Основна ідея алгоритму Брезенхема полягає в тому, що ми рухатимемося координатними осями і на кожному кроці обиратимемо наступний піксель для засвічення на основі поточного положення і помилки.

Визначення різниці між координатами двох точок:

$$\Delta x = x_2 - x_1$$

$$\Delta y = y_2 - y_1$$

$$\Delta z = z_2 - z_1$$

Визначення кроків по кожній з трьох осей:

$$step_x = \text{sign}(\Delta x)$$

$$step_y = \text{sign}(\Delta y)$$

$$step_z = \text{sign}(\Delta z)$$

де $\text{sign}(a)$ - функція знаку, яка повертає -1, 0 або 1, в залежності від знаку числа a.

Обчислення абсолютних значень різниць:

$$\Delta x = |\Delta x|$$

$$\Delta y = |\Delta y|$$

$$\Delta z = |\Delta z|$$

Визначення головної осі:

$$\text{main\_axis} = \begin{cases} x, \text{ if } \Delta x \geq \Delta y \text{ and } \Delta x \geq \Delta z \\ y, \text{ if } \Delta y \geq \Delta x \text{ and } \Delta y \geq \Delta z \\ z, \text{ if } \Delta z \geq \Delta x \text{ and } \Delta z \geq \Delta y \end{cases}$$

Ініціалізація поточних координат та початкової помилки:

$$x = x_1,\ y = y_1,\ z = z_1$$

$$\text{error} = 0$$

Початок циклу за головною осі (main_axis), ітеруючись від 0 до максимального значення зміни на головній осі (більшого з Δx, Δy, Δz):

a. Побудова поточного пікселя на основі поточних координат (x, y, z).

b. Корекція помилки на головній осі:

$$\text{error} = \text{error} + 2 * \Delta\text{main\_axis}$$

c. Якщо помилка перевищує Δx, Δy або Δz, то коригуємо координати відповідно до головної осі:

$$x = x - \text{step}_x$$

$$y = y - \text{step}_y$$

$$z = z - \text{step}_z$$

Завершення циклу.

Після завершення цього алгоритму ми отримаємо координати всіх точок, через які проходить лінія у тривимірному просторі від точки $P_1(x_1, y_1, z_1)$ до точки $P_2(x_2, y_2, z_2)$.

4. АЛГОРИТМ РОБОТИ РОБОТА

Для демонстрації роботи алгоритму Брезенхема у розширеному вигляді було створено блок-схему, нижня частина якої репрезентує роботу функції яка повертає крок, який потрібно зробити, блок-схема зображена на рисунку 4.1. [3]

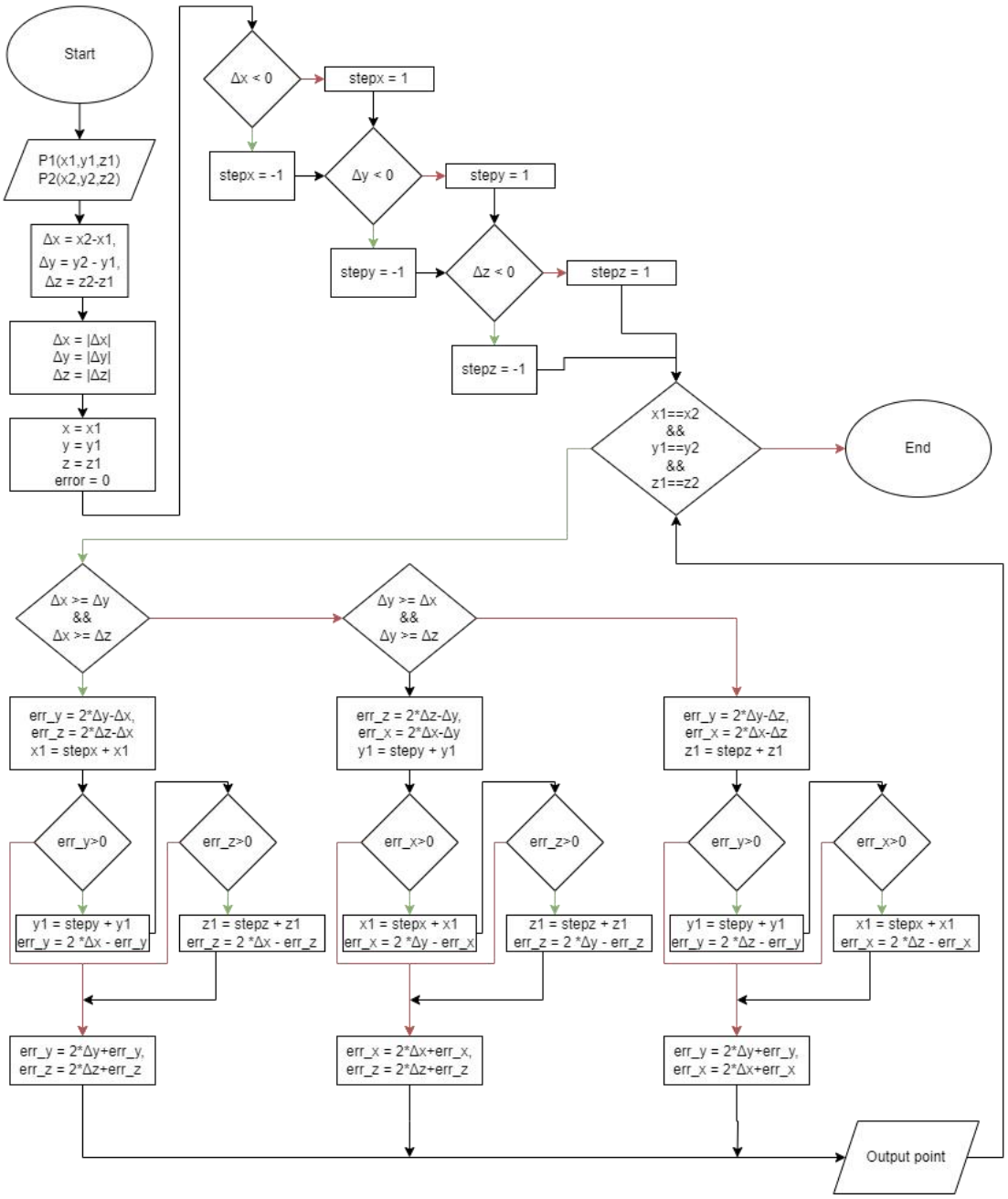

Рисунок 4.1 - Блок-схема

5. СТВОРЕННЯ ПРОГРАМИ

Програму створено за допомогою мови програмування Python, будуть використовуватися такі бібліотеки як tkinker (для створення графічного інтерфейсу), mathplotlib (для візуалізації графіку), time (для покрокової візуалізації) та threading (для можливості використання програми під час побудови шляху). [2]

5.1 Керівництво користувача

При відкриті програми з'явиться вікно Visualization 3D motion, приклад зображено на рисунку 5.1.1, програма має декілька частин це кнопки управління (рис. 5.1.5), місце відображення графіка, написи з поточними позиціями (рис. 5.1.4) та кнопку налаштувань.

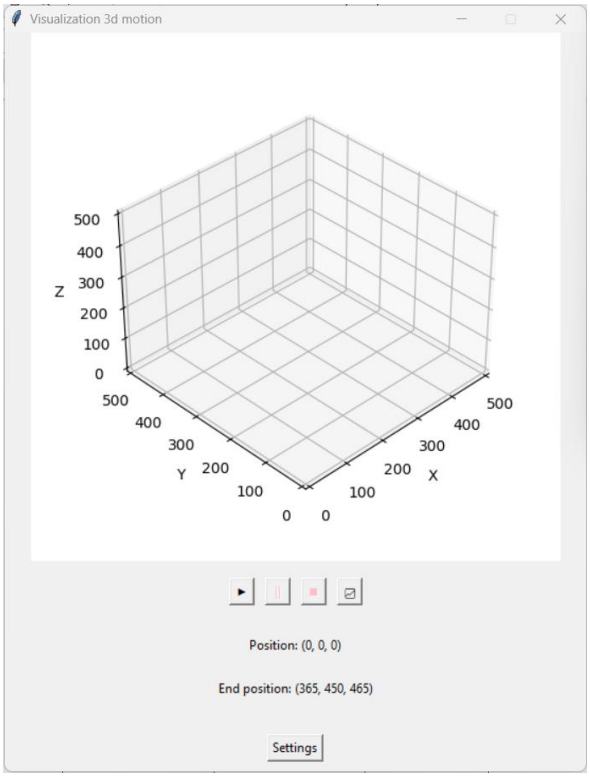

Рисунок 5.1.1 - Вигляд програми

Для початку роботи програми потрібно задати координати направлення, початкові координати задані в програмі та при запуску завжди дорівнюють (0,0,0), щоб задати координати направлення

потрібно натиснути 4 кнопку керування із іконкою графіку (рис.5.1.5), після чого відкриється меню задання координат Set Coords:, як на рисунку 5.1.3, де повзунками можна задати всі 3 координати, ці координати відображуються місце призначення, тобто кінцеві координати, щоб натиснути кнопку підтвердження потрібно щоб програма була закінчена. Далі Щоб побачити роботу програми потрібно натиснути кнопку “ ▶ ” (рис. 5.1.5), щоб почати виконання програми щоб подивитися поточне положення потрібно подивитися на поле Position(рис. 5.1.4), щоб поставити на паузу потрібно натиснути кнопку паузи “ ||” (рис. 5.1.5) та щоб продовжити знову кнопку “ ▶ ” (рис. 5.1.5) щоб перервати виконання потрібно натиснути на кнопку “▌ ” (рис. 5.1.5). Якщо швидкість малювання графіка або його колір вам не подобається потрібно натиснути на кнопку Settings (рис. 5.1.2) після чого з’явиться вікно Settings, у якому повзунок Speed відображає швидкість малювання графіка, Point Color колір кожної крапки, Line Color колір лінії та Size розмір кожної крапки.

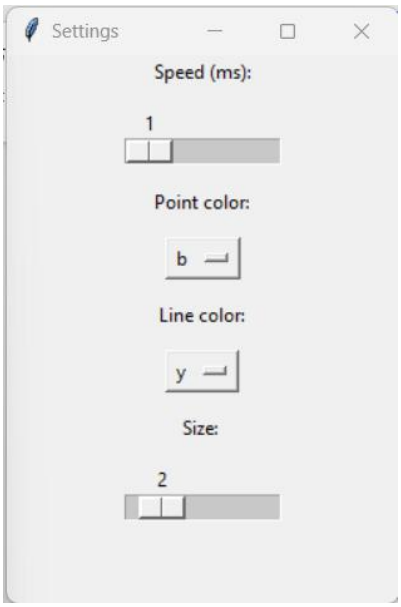

Рисунок 5.1.2 - Вікно налаштувань

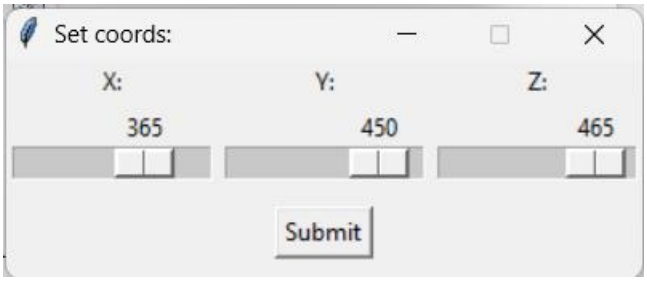

Рисунок 5.1.3 - Вікно для задання координат

Рисунок 5.1.4 - Позиції

Рисунок 5.1.5 - Кнопки для управління

Так як програма передбачає виведення графіку також цим графіком можна керувати при натисканні середньої кнопки миші графік скидається, якщо крутити колесо миші графік буде наближатися чи віддалятися, при затиснутій правій кнопці миші графік можна обертати. Також програма виводить кожен крок у консоль, як на рисунку 5.1.6.

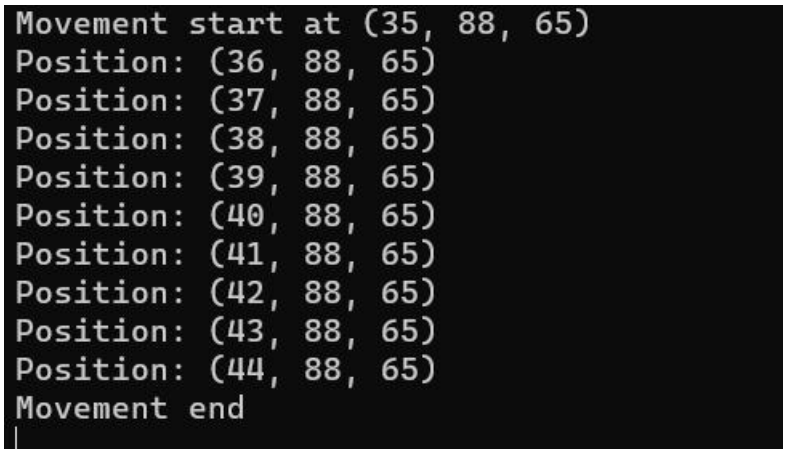

Рисунок 5.1.6 - Консольний вивід

5.2 Керівництво програміста

Для програми описане призначення функцій та глобальних змінних, основну частину коду програми займає створення графічного інтерфейсу. Виконавчий код програми знаходиться у Додаток 1.

Глобальні змінні:

Змінні speed_var, color_var, color_line_var, size_var змінні, які зберігають значення кольорів лінії та точок а також розмір точок, змінюються у налаштуваннях.

button_press_time змінна для скидання налаштувань графіка.

cur_pos - зберігає поточну позицію, а також єдиний спосіб задати поточну позицію

end_pos - зберігає кінцеву позицію змінюється в спеціальному вікні.

is_paused - зберігає значення чи на паузі зараз програма для кнопки паузи

is_running - зберігає значення чи запущена зараз програма

is_force_stopped - змінна для зупинки програми до кінця її виконання

colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w'] - змінна зберігає кольори, які можна задати для всіх графічних елементів, можливо додати більше за потреби.

Функція step_3d(current_pos, target_pos):

Основна функція, яка виконує крок по алгоритму Брезенхема, основною віссю в цій функції є вісь Z

Функція visualize_movement():

Виконує першу частину блок-схеми тобто задає виконується, тобто викликається функція step_3d, поки не координата кінця і

початку не будуть збігатися, також перевіряє значення для управління графіком (пауза, вимкнення).

Функція start_visualization():

Функція запускає окремий потік для малювання графіка та розрахунку кроків, для функції visualize_movement().

Функція toggle_pause():

Перемикає значення паузи для програми.

Функція force_stop_visualization():

Перемикає значення зупинки для програми, що при запущеному потоці зупиняє потік.

Функція open_settings_window():

Створює вікно Settings для налаштування швидкості, розміру та кольорів.

Функція open_coordinates_window():

Створює вікно Set coords: для задання координат X, Y, Z.

Функція set_coordinates(x, y, z, window):

Робить перевірку на введення даних та роботу алгоритму, якщо алгоритм вимкнено і дані правильно задає координати кінця, які були передані.

Функція zoom_zero:

Відслідковує чи було натиснуто середню клавішу миші щоб обнулити налаштування графіка

Функція zoom:

Відслідковіє рух колеса миші для наближення та віддалення графіка.

6. ТЕСТУВАННЯ ПРОГРАМИ

Для тестування програми було показано графік та консольне вікно з кроками.

Для першого тесту було обрано рух з координат 0, 0, 0 до 22, 22, 22 щоб показати рух по діагоналі, приклад на рисунку 6.1.

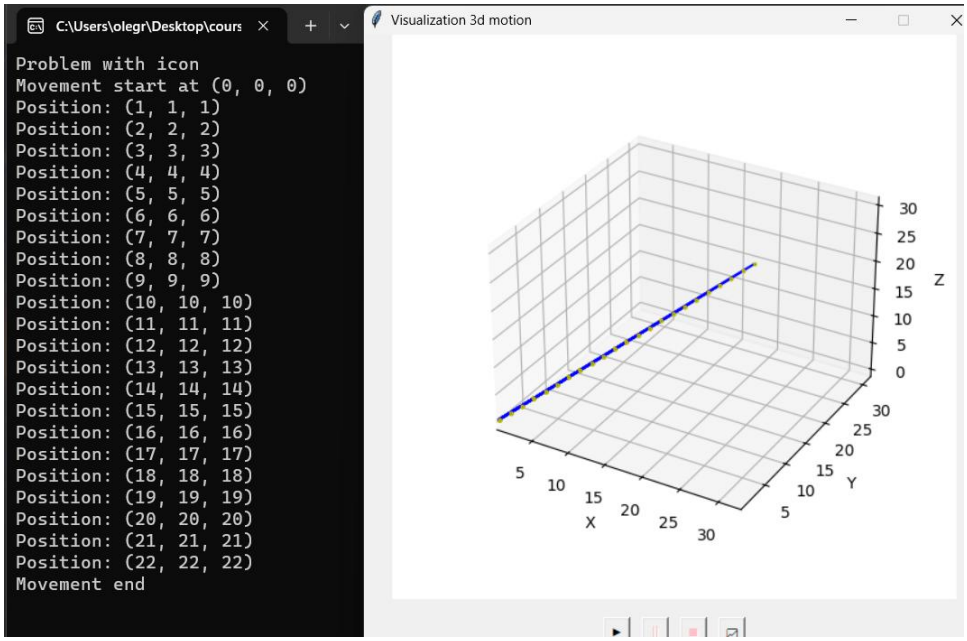

Рисунок 6.1 - Приклад роботи № 1

Для прикладу 2 координати кінця 37, 29, 29 тобто дві координати однакові, приклад можна побачити на рисунку 6.2.

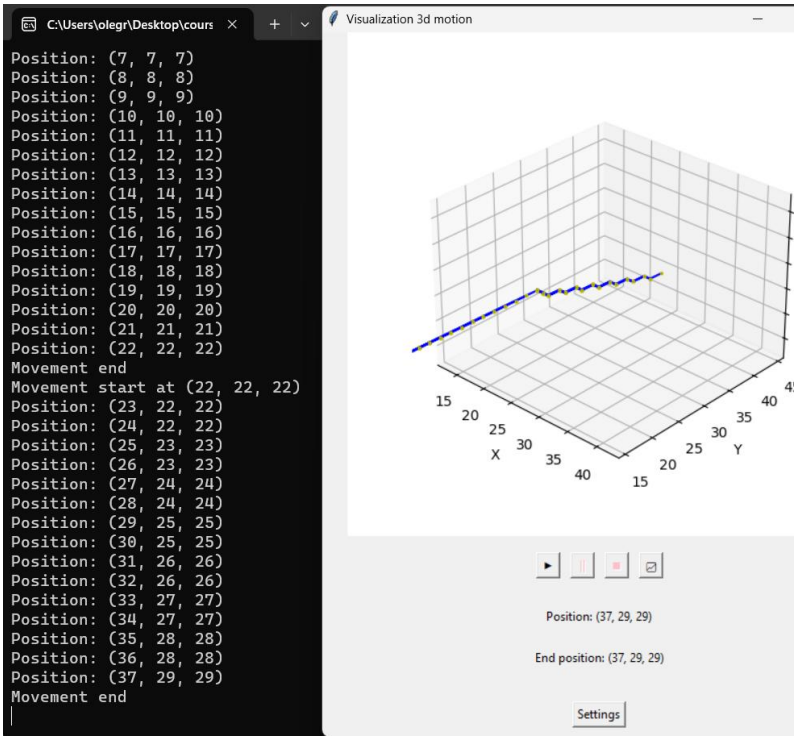

Рисунок 6.2 - Приклад роботи № 2

Для прикладу 3 було обрано координати 7, 7, 29 для більш довгого проходу по одній координаті, рисунок 6.3.

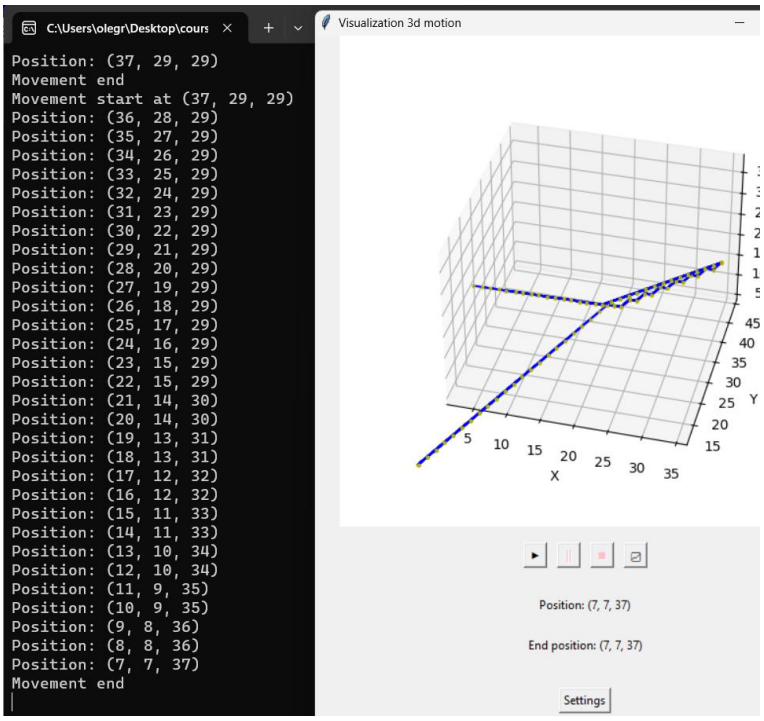

Рисунок 6.3 - Приклад роботи № 3

Щоб показати більш складний шлях на тесті 4 було обрано координати 37, 44, 132, результат цього вибору показано на рисунку 6.4 та на рисунку 6.5

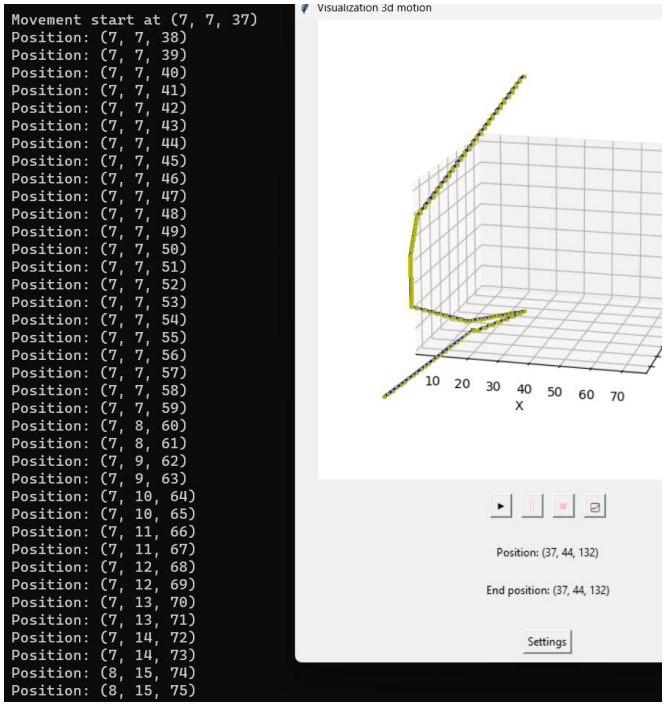

Рисунок 6.4 - Приклад роботи № 4 Частина 1

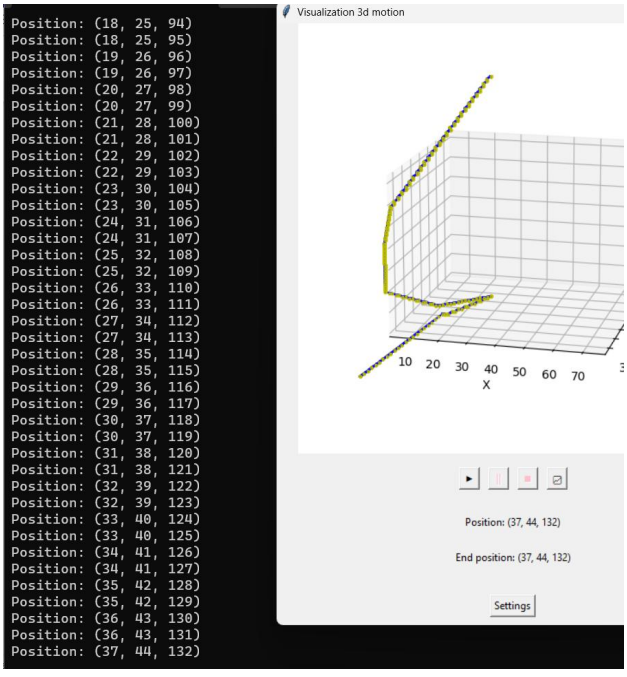

Рисунок 6.5 - Приклад роботи № 4 Частина 2

Для прикладу 5 було обрано рух по одній осі Z до самого кінця, результат цього можна побачити на рисунку 6.6.

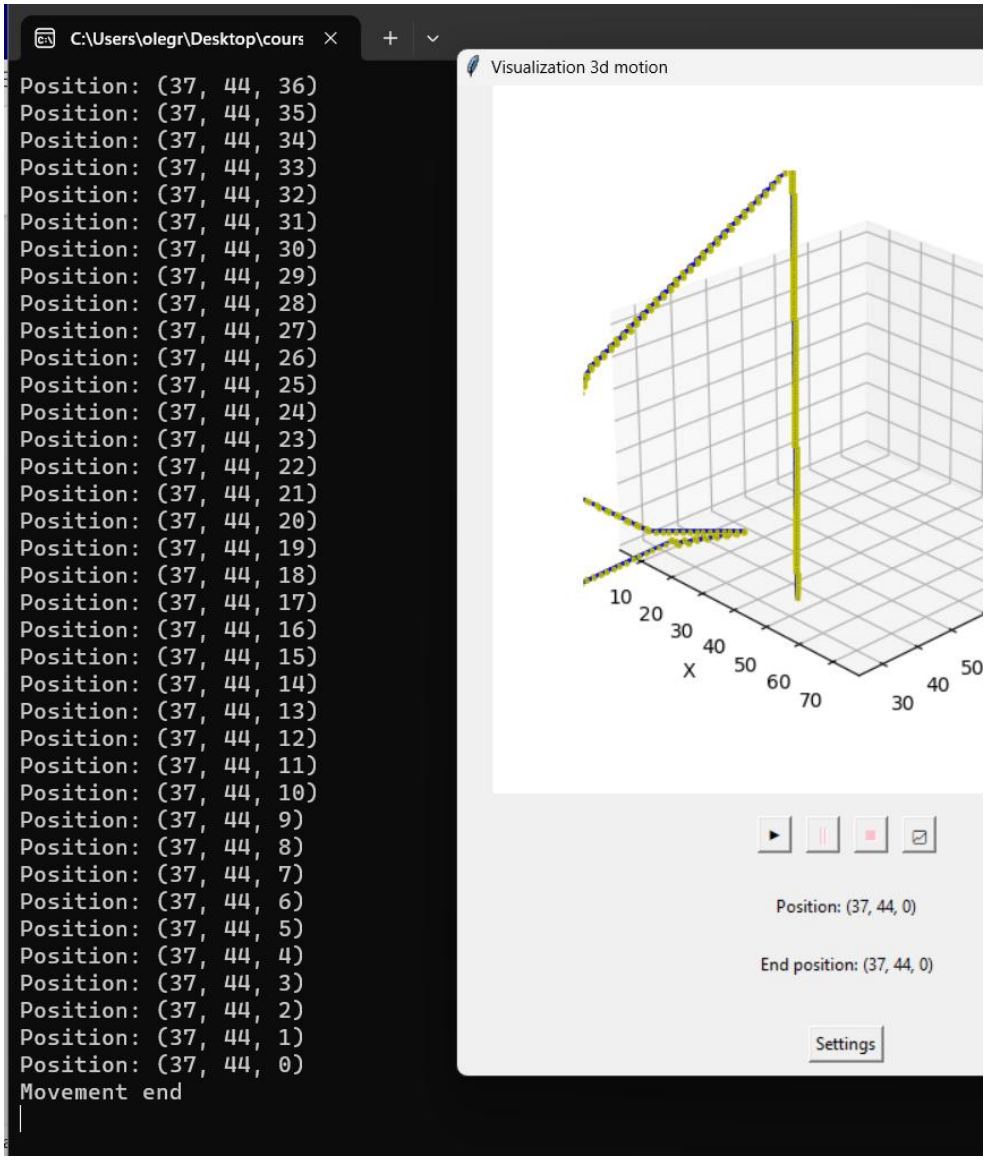

Рисунок 6.6 - Приклад роботи № 5

ВИСНОВКИ

Курсова робота була призначена для створення лінійного робота з лінійним переміщення по 3 координатам. Було створено модель, прописано алгоритм та написана програма.

Для створення моделі було обрано Solid Works в якому було створено просту модель 3д принтера. Модель може переміщуватися по 3 координатам та показати точно положення кінця пензлика.

Для руху робота було обрано алгоритм Брезенхема в його розширено варіанті тобто на три координати.

Для програми було використано Python та додано до функціоналу графічний інтерфейс для кращого наглядного спостереженням за виконанням програми.

БІБЛІОГРАФІЯ

1. Шевель В. В. Конспект лекцій з курсу «Спеціальне ПЗ ІТ» [Електронний ресурс] / В. В. Шевель. – https://drive.google.com/drive/folders/1W5PY76VUzPbyo7-PHyHs5kXM_2JRAYu6?usp=sharing – 13.09.2022.
2. Документація Python 3.11 [Електроний ресурс] - https://docs.python.org/3.11/
3. Draw.io ресурс для створення блок-схем [Електроний ресурс] - https://app.diagrams.net
4. Пояснення роботи алгоритма Брезенхема [Електроний ресурс] - https://digitalbunker.dev/bresenhams-line-algorithm/#:~:text=Bresenham%27s%20algorithm%20-%20a%20fundamental%20method,on%20a%20pixel-based%20display.

ДОДАТОК 1

```
import tkinter as tk
from tkinter import Button, Label, Toplevel, Scale, HORIZONTAL,
OptionMenu, messagebox
from matplotlib.figure import Figure
from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
import time
import threading

# Global variables for storing settings
speed_var = None
color_var = None
color_line_var = None
size_var = None
button_press_time = 0
cur_pos = (0, 0, 0)
end_pos = (35, 50, 65)
is_paused = False
is_running = False
is_force_stopped = False
colors = ['b', 'g', 'r', 'c', 'm', 'y', 'k', 'w']

def step_3d(current_pos, target_pos):
    x0, y0, z0 = current_pos
    x1, y1, z1 = target_pos
    dx = abs(x1 - x0)
    dy = abs(y1 - y0)
    dz = abs(z1 - z0)
```

```
if x0 < x1:
    sx = 1
else:
    sx = -1

if y0 < y1:
    sy = 1
else:
    sy = -1

if z0 < z1:
    sz = 1
else:
    sz = -1

if dx >= dy and dx >= dz:
    err1 = 2 * dy - dx
    err2 = 2 * dz - dx
    x0 += sx
    if err1 > 0:
        y0 += sy
        err1 -= 2 * dx
    if err2 > 0:
        z0 += sz
        err2 -= 2 * dx
    err1 += 2 * dy
    err2 += 2 * dz

elif dy >= dx and dy >= dz:
```

```
            err1 = 2 * dx - dy
            err2 = 2 * dz - dy
            y0 += sy
            if err1 > 0:
                x0 += sx
                err1 -= 2 * dy
            if err2 > 0:
                z0 += sz
                err2 -= 2 * dy
            err1 += 2 * dx
            err2 += 2 * dz

        else:    # dz is the dominant axis
            err1 = 2 * dy - dz
            err2 = 2 * dx - dz
            z0 += sz
            if err1 > 0:
                y0 += sy
                err1 -= 2 * dz
            if err2 > 0:
                x0 += sx
                err2 -= 2 * dz
            err1 += 2 * dy
            err2 += 2 * dx

        return (x0, y0, z0)

def toggle_pause():
    global is_paused
```

```
    is_paused = not is_paused

def stop_visualization():
    global is_running
    is_running = False

def force_stop_visualization():
    global is_force_stopped
    is_force_stopped = True

def visualize_movement():
    global speed_var, color_var, cur_pos, end_pos, canvas, is_paused,
is_running, is_force_stopped, color_line_var, size_var
    positions_x = positions_y = positions_z = None
    positions_x = [cur_pos[0]]
    positions_y = [cur_pos[1]]
    positions_z = [cur_pos[2]]
    is_running = True
    is_force_stopped = False
    print(f"Movement start at {cur_pos}")
    while cur_pos != end_pos and is_running:

        if is_force_stopped:
            break

        if(is_paused):
            time.sleep(0.1)
            continue
```

```
            temp_pos = step_3d(cur_pos, end_pos)
            print(f"Step->    x: { temp_pos[0] - cur_pos[0]}, y:
{temp_pos[1] - cur_pos[1]}, z: {temp_pos[2] - cur_pos[2]}")
            cur_pos = temp_pos
            positions_x.append(cur_pos[0])
            positions_y.append(cur_pos[1])
            positions_z.append(cur_pos[2])
            # Clear the chart before drawing a new line
            ax.plot(positions_x, positions_y, positions_z, linestyle='-',color
= color_var.get(), marker ='s', markersize = size_var.get(),
markeredgecolor=color_line_var.get())
            # Set the current position
            print(f"Position: {cur_pos}")
            coord_label.config(text=f"Position: {cur_pos}")
            canvas.draw()
            canvas.flush_events()
            # Refreshing the Tkinter window
            time.sleep(speed_var.get() / 1000.0)

    print(f"Movement end")
    is_running = False    # Complete the visualization

    # Lock/Unlock the button after the animation is complete
    start_button.config(state=tk.NORMAL)
    pause_button.config(state=tk.DISABLED)
    reset_button.config(state=tk.DISABLED)

# Function to open a new window for setting chart parameters
def open_settings_window():
```

```
    global speed_var, color_var, colors, color_line_var, size_var
    settings_window = Toplevel(root)
    settings_window.geometry("250x350")
    settings_window.title("Settings")
    def set_speed(value):
        speed_var.set(value)
        return
    def set_color(value):
        color_var.set(value)
        return
    def set_line_color(value):
        color_line_var.set(value)
        return
    def set_size(value):
        size_var.set(value)
        return
    # Label and Scale to select the speed between chart steps
    speed_label = Label(settings_window, text="Speed (ms):")
    speed_label.pack()

    speed_scale = Scale(settings_window, from_=1, to=1000,
orient=HORIZONTAL, command=set_speed)
    speed_scale.set(speed_var.get())
    speed_scale.pack(pady=10)

    # Label and menu for selecting the color of chart points from
predefined options
    pcolor_label = Label(settings_window, text="Point color:")
    pcolor_label.pack()
```

```
        pcolor_menu = OptionMenu(settings_window, color_var, *colors,
command = set_color)
        pcolor_menu.pack(pady=10)

        #Label and menu to select the chart line color from predefined
options
        color_label = Label(settings_window, text="Line color:")
        color_label.pack()

        color_menu = OptionMenu(settings_window, color_line_var,
*colors, command = set_line_color)
        color_menu.pack(pady=10)

       # Label and Scale to select the size between chart steps
        size_label = Label(settings_window, text="Size: ")
        size_label.pack()

        size_scale = Scale(settings_window, from_ = 1, to = 10,
orient=HORIZONTAL, command=set_size)
        size_scale.set(size_var.get())
        size_scale.pack(pady=10)

def open_coordinates_window():
        global end_pos
        coordinates_window = Toplevel(root)
        coordinates_window.title("Set coords:")

        coordinates_window.resizable(False, False)
```

```
        coord_frame = tk.Frame(coordinates_window)
        coord_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

        # Label and Entry to set the X coordinate
        x_label = Label(coord_frame, text="X:")
        x_label.grid(row = 0, column = 0)
        x_scale = Scale(coord_frame, from_ = 0, to = 500,
orient=HORIZONTAL)
        x_scale.set(end_pos[0])
        x_scale.grid(row = 1, column = 0)

        # Label and Entry to set the Y coordinate
        y_label = Label(coord_frame, text="Y:")
        y_label.grid(row = 0, column = 1)
        y_scale = Scale(coord_frame, from_ = 0, to = 500,
orient=HORIZONTAL)
        y_scale.set(end_pos[1])
        y_scale.grid(row = 1, column = 1)

        # Label and Entry to set the Z coordinate
        z_label = Label(coord_frame, text="Z:")
        z_label.grid(row = 0, column = 2)
        z_scale = Scale(coord_frame, from_ = 0, to = 500,
orient=HORIZONTAL)
        z_scale.set(end_pos[2])
        z_scale.grid(row = 1, column = 2)

        # Button to confirm the selected coordinates
```

```
    confirm_button = Button(coordinates_window, text="Submit",
command=lambda: set_coordinates(int(x_scale.get()), int(y_scale.get()),
int(z_scale.get()), coordinates_window))
    confirm_button.pack(pady = 10)

def set_coordinates(x, y, z, window):
    global end_pos, is_running
    if(is_running):
        return
    try:
        x = int(x)
        y = int(y)
        z = int(z)
        if 0 <= x <= 500 and 0 <= y <= 500 and 0 <= z <= 500:
            end_pos = (x, y, z)
            coord_label2.config(text=f"End position: {end_pos}")
            window.destroy()
        else:
            messagebox.showerror("Error", "The coordinates must be
between 0 and 500.")
    except ValueError:
        messagebox.showerror("Error", f"Enter the correct numeric
values for the coordinates. {x},{y},{z}")


def start_visualization():
    global cur_pos, end_pos, is_paused, is_running
    if is_running:
```

```
            messagebox.showinfo("Warning", "The visualization is already
up and running.")
            return

        is_running = True
        start_button.config(state=tk.DISABLED)
        pause_button.config(state=tk.NORMAL)
        reset_button.config(state=tk.NORMAL)
        visualization_thread = threading.Thread(target=visualize_movement)
        visualization_thread.start()

# Create a Tkinter window
root = tk.Tk()
root.title("Visualization 3d motion")
root.geometry("550x700")
root.resizable(False, False)

# Creating a canvas for drawing matplotlib
fig = Figure(figsize=(5, 5), dpi=100)
ax = fig.add_subplot(111, projection='3d')

ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Z')

ax.set_xlim(0, 500)
ax.set_ylim(0, 500)
ax.set_zlim(0, 500)
```

```
canvas = FigureCanvasTkAgg(fig, master=root)
canvas.get_tk_widget().pack()

def zoom_zero(event):
    global button_press_time
    if(event.button!=2):
        return
    else:
        current_time = time.time()
        # Check if it's a double-click
        if current_time - button_press_time < 0.5:
            ax.set_xlim(0, 500)
            ax.set_ylim(0, 500)
            ax.set_zlim(0, 500)
            fig.canvas.draw()
        button_press_time = current_time

fig.canvas.mpl_connect('button_press_event', zoom_zero)

def zoom(event):
    # Get the current axis limits
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    zlim = ax.get_zlim()
    # Determine the zoom factor
    zoom_factor = 1.1

    if event.button == 'up':
        # Zoom in by decreasing the axis limits
```

```
            new_xlim = (xlim[0] / zoom_factor, xlim[1] / zoom_factor)
            new_ylim = (ylim[0] / zoom_factor, ylim[1] / zoom_factor)
            new_zlim = (zlim[0] / zoom_factor, zlim[1] / zoom_factor)
        elif event.button == 'down':
            # Zoom out by increasing the axis limits
            new_xlim = (xlim[0] * zoom_factor, xlim[1] * zoom_factor)
            new_ylim = (ylim[0] * zoom_factor, ylim[1] * zoom_factor)
            new_zlim = (zlim[0] * zoom_factor, zlim[1] * zoom_factor)
        else:
            return    # Ignore other mouse events

        # Set the new axis limits to zoom in or out
        ax.set_xlim(new_xlim)
        ax.set_ylim(new_ylim)
        ax.set_zlim(new_zlim)

        # Redraw the plot
        fig.canvas.draw()

fig.canvas.mpl_connect('scroll_event', zoom)

# Create a Frame for placing the buttons
btn_frame = tk.Frame(root)
btn_frame.pack(side=tk.TOP, expand=True, anchor=tk.CENTER)

# Create a button to start drawing
start_button = Button(btn_frame, text="▶ ",
command=start_visualization, width=2, height=1,
disabledforeground="pink")
```

```
start_button.pack(side=tk.LEFT, padx=5, pady =10)

pause_button = Button(btn_frame, text="||", command=toggle_pause,
width=2, height=1, disabledforeground="pink")
pause_button.pack(side=tk.LEFT, padx=5, pady =10)
pause_button.config(state=tk.DISABLED)

reset_button = Button(btn_frame, text="▌  ",
command=force_stop_visualization, width=2, height=1,
disabledforeground="pink")
reset_button.pack(side=tk.LEFT, padx=5, pady =10)
reset_button.config(state=tk.DISABLED)

set_button = Button(btn_frame, text="    ",
command=open_coordinates_window, width=2, height=1,
disabledforeground="pink")
set_button.pack(side=tk.LEFT, padx=5, pady =10)

# Create Frame to place Label with coordinate
coord_frame = tk.Frame(root)
coord_frame.pack(side=tk.TOP, fill=tk.BOTH, expand=True)

# Create Label to display the current coordinate
coord_label = Label(coord_frame, text="Position: (0, 0, 0)")
coord_label.pack(pady=10)
coord_label2 = Label(coord_frame, text=f"End position: {end_pos}")
coord_label2.pack(pady=10)
```

```
# Create a button to open the settings window and automatically apply
settings
settings_button = Button(root, text="Settings",
command=open_settings_window)
settings_button.pack(pady=10)

# Create variables to store the settings
speed_var = tk.IntVar()
speed_var.set(1)

color_var = tk.StringVar()
color_var.set('b')

color_line_var = tk.StringVar()
color_line_var.set('y')

size_var = tk.IntVar()
size_var.set(2)

try:
    icon_path = "icon.ico"
    root.iconbitmap(icon_path)
except:
    print("Problem with icon")

root.mainloop()
```