Bridging CAD and BIM through Automated Construction Product Information for Estimation and Scheduling

Mid-semester Report
CED 444: Major Project-I
(Monsoon 2025)

Submitted by

D Arjun

2210110245

Under Supervision

of

Dr. Gopal Das Singhal

Department of Civil Engineering

SHIV NADAR
INSTITUTION OF EMINENCE DEEMED TO BE UNIVERSITY
DELHI NCR

SCHOOL OF ENGINEERING

Abstract

This project develops a software pipeline that converts 2D CAD floor plans (DWG/DXF) into structured construction information suitable for cost estimation and scheduling, and for export into BIM (Revit) workflows. The system reads an input drawing, identifies geometric primitives (walls, doors, windows), recognizes text room labels, constructs closed room boundaries, computes gross and net areas and wall lengths (deducting openings), and exports a structured dataset (JSON/Excel) that can feed a Dynamo script to create Revit geometry and schedules. A complementary research axis explores using machine learning — notably graph neural networks (GNNs) and symbol classifiers — to automate symbol detection, resolve ambiguous geometry, and generalize to varied drafting conventions. A proof-of-concept implementation using ezdxf, shapely, networkx, and Dynamo for Revit has been developed and validated on sample plans. The midterm deliverable demonstrates feasibility, highlights gaps (robust shared-wall merging, symbol variety), and outlines future work including ML-model training, richer estimate breakdowns, and Revit parameterization.

Keywords: floor plan parsing, CAD-to-BIM, room detection, wall length, graph neural networks, construction estimating

Table of Contents

List of figures

1. Introduction

Modern small-to-medium building projects often start with 2D CAD drawings. Estimating quantities and preparing schedules from such drawings is time-consuming and error-prone when done manually. This project aims to automate, as far as practical, the conversion of 2D CAD floor plans into: (a) per-room quantities (area, wall lengths), (b) material-breakdown estimates (wall volume, paint area, tile counts), and (c) a structured export that can populate a Revit model and schedules via Dynamo.

The scope emphasizes proof-of-concept for typical architectural floor plans: simple rooms, walls drawn as polylines or paired offset lines (wall thickness), block-based doors/windows, and textual room labels. The pipeline is designed to be robust to common drafting variations while remaining implementable within the project timeframe.

2. Literature survey

A focused literature review was undertaken to identify relevant approaches for floor plan parsing and symbol recognition. The key themes are:

- *Graph-based floorplan generalization:* Recent work uses graph neural networks (GNNs) to convert low-level floorplan elements into structured room graphs and to infer semantic relations (adjacency, door/wall connectivity). Such methods generalize across drafting styles and can learn to correct inconsistent lines. These ideas indicate a promising ML path for our project (floorplan -> graph -> labelled building topology).

- *Symbol detection and classification:* Symbol-level CNNs and graph-based classifiers have been applied to detect doors, windows, and other architectural symbols. Combining detection with vector geometry (DXF entities) enables robust assignment of openings to walls.

- *Construction drawing recognition using GNNs:* Papers on applying GNNs directly to CAD entities show that per-entity features (line direction, layer, length, neighborhood) are effective inputs; models then classify entities and predict connectivity.

- *Component sizing via heterogeneous GNNs:* Research in structural component sizing uses hetero-graphs to combine geometric and semantic node types; this is relevant for future work where structural rules (e.g., typical door widths, clear heights) could be inferred or checked automatically.

Overall, the literature supports two complementary tracks: deterministic geometry processing for high-precision computations and ML-driven semantic labelling to increase robustness to real-world variability.

3. Gap analysis

From initial experiments and the literature, key gaps were identified:

1. *Shared-wall deduplication:* Adjacent rooms often define the same wall twice; naive export duplicates walls in BIM. Geometric deduplication needs to merge collinear or overlapping line segments despite small coordinate differences.
2. *Symbol variety:* Door/window blocks differ in naming and insertion points; relying on fixed block names is brittle.
3. *Complex drafting conventions:* Wall thickness sometimes drawn as single centreline, sometimes as two parallel polylines, sometimes as hatch; robust detection must handle all.
4. *Lack of labelled training data:* To train ML models for symbol classification/GNNs, a curated dataset of CAD drawings with annotations is required.

Addressing these gaps is central to making the tool useful on professional plans.

4. Objectives

Primary objectives for the midterm:

- Implement a robust CAD parsing pipeline that extracts rooms, walls, doors, and windows from DXF files.
- Compute per-room gross and net areas and wall lengths after deducting openings.
- Produce initial BOQ's for parsed plans.
- Generate attribute-based output for all objects related to the plan.
- Produce an export (JSON/Excel) compatible with Dynamo to reconstruct geometry in Revit and generate schedules.
- Demonstrate placement of doors/windows and wall creation in Revit via Dynamo.
- Survey ML approaches (GNNs and symbol detectors) and propose a feasible integration plan for subsequent work.

Secondary objectives:

- Provide visualization utilities and debugging plots for verification (matplotlib overlays, labelled points).
- Create project documentation and prepare the midterm report.

5. Methodology

A modular pipeline was implemented. Each module is described below.

5.1 Data ingestion and format conversion

DWG → DXF: Users can export DWG to DXF from AutoCAD or use free conversion utilities. The pipeline assumes the DXF is in readable AutoCAD format (R12/R2000+). Units are checked and normalized (DXF units interpreted in meters where appropriate).

5.2 Reading DXF entities

- *Library:* ezdxf is used to read modelspace entities: LWPOLYLINE, LINE, INSERT (blocks), TEXT, MTEXT.

- *Entity collection:* We query closed LWPOLYLINE for candidate room polygons, LINE segments for unclosed drawings, and INSERT for door/window blocks.

5.3 Room detection

Two strategies were used:

A. Closed-polylines approach: If the drawing contains closed polylines representing rooms (common in neat CAD plans), each LWPOLYLINE with closed=True is converted to a Shapely Polygon. A room label is assigned by testing whether a TEXT/MTEXT insertion point lies within the polygon.

B. Polygonize approach (when single lines are used for walls): When walls are drawn as single-line centerlines, the pipeline: (a) collects all LINE segments, (b) snaps endpoints within a tolerance, (c) builds a planar graph via networkx, and (d) uses shapely.ops.polygonize to recover closed polygons representing rooms. This recovers rooms when closed polylines are absent.

When neither approach yields rooms, a guided manual step in CAD (create closed boundaries) is recommended for the test plan.

5.4 Wall extraction and deduplication

- *Wall segments:* For each room polygon the exterior ring is decomposed into wall segments (LineStrings). A normalize_wall function creates an order-independent key (sorted endpoint pairs) and rounds coordinates to avoid minute floating differences.

- *Advanced deduplication:* If rounding is insufficient, the pipeline performs overlap-based merging: new walls are checked against existing wall geometries with shapely operations (equals_exact, buffer().intersects()). This merges shared walls into a single canonical wall used by both rooms. Several tolerance strategies were implemented (coordinate snapping, buffer-intersection) and tuned experimentally.

5.5 Doors and windows detection

- *Block-based detection:* INSERT entities are inspected for block names containing common door/window substrings (e.g., door, puerta, vent, window). Each block's insert

point is projected perpendicularly to nearby wall segments to find the host wall and measure the opening width.

- *Host assignment:* The nearest wall (by perpendicular distance) is chosen as the host; for shared walls the opening is split/shared appropriately.

- *Robustness:* Because block naming is inconsistent across plans, the ML phase (future) aims to classify block geometry and block attributes rather than rely exclusively on the name.

5.6 Quantity calculations

- *Areas:* For each detected room polygon compute the floor area (polygon.area). Units are normalized to square meters.

- *Gross vs net wall length:* For each wall touching a room we compute gross length (segment length). Deductions for doors/windows assigned to that wall are subtracted (door/window widths), with shared openings split equally between adjacent rooms unless block metadata indicates otherwise.

- *Derived quantities:* With user inputs (wall height, wall thickness, tile coverage factors), we compute wall surface area, wall volume, number of tiles, estimated bricks, plaster volume, and paint quantity using simple engineering formulae and material-specific factors (configurable parameters).

5.7 Export formats & Revit integration

- *JSON/Excel export:* The canonical export includes unique wall list (start/end coordinates, length), room list (label, area, wall references), and openings (type, width, host-wall reference). This file is consumed by Dynamo to build geometry in Revit.

- *Dynamo graph:* A Dynamo graph reads the Excel/JSON, re-creates Point.ByCoordinates for wall endpoints, Line.ByStartPointEndPoint for wall curves, and Wall.ByCurveAndHeight for Revit walls. Doors/windows are placed with FamilyInstance.ByPointAndHost so they are hosted to walls and cut openings correctly.

- *Verification:* Visual checks in Dynamo and Revit are used: overlay of geometry for manual verification, and Revit schedule checks.

5.8 Visualization and debugging

- *Matplotlib overlays:* For each plan we generate debug plots showing polylines/lines (walls), room centroids with labels, detected doors/windows (colored markers), and text showing per-wall lengths. This significantly reduced debug time while developing.

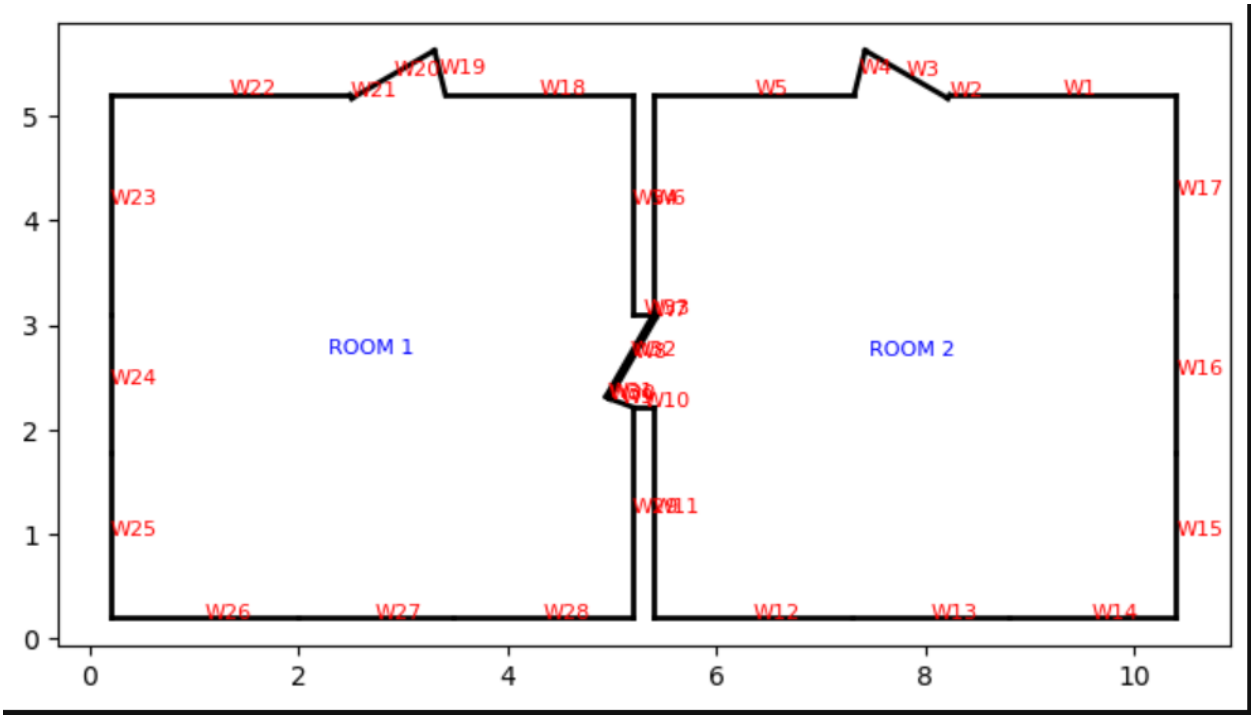

Fig-1 : Matplotlib overlay of imputed autoCAD plan after processing and room detection

5.9 Machine Learning Integration for Construction Drawing Intelligence

5.9.1 Motivation for ML Integration

While traditional CAD parsing enables rule-based extraction of geometric and textual features, it struggles with ambiguities in professional construction drawings such as overlapping entities, inconsistent labeling, or incomplete polylines. Machine Learning (ML), particularly Graph Neural Networks (GNNs) and Deep Learning-based classifiers, provides a scalable approach to:

- Automatically classify drawing elements (walls, doors, windows, furniture, annotations).

- Generalize across heterogeneous floor plan styles.
- Enhance robustness by learning from labeled datasets rather than strict CAD entity assumptions.

This integration bridges the gap between raw 2D CAD files and structured Building Information Models (BIM), enabling automated and intelligent decision-making.

5.9.2 Literature Review on ML in Floor Plan Understanding

- Generalizing Floor Plans Using GNNs (Paper 1): Demonstrates how architectural layouts can be converted into graph structures where nodes represent spaces and edges represent connectivity. The GNN learns generalized representations, enabling automatic zoning and typology recognition.
- Drawing Content Recognition with GNNs (Paper 2): This approach focuses on the recognition of structural and annotation symbols in construction drawings. By embedding CAD elements (lines, arcs, polylines, blocks) as graph nodes, the model learns contextual relationships to correctly classify symbols even when they are partially occluded or inconsistently represented. This paper highlights the potential for ML to go beyond geometry and tap into semantic understanding of drawings.
- Symbols Detection and Classification using GNNs (Paper 3): Here, symbol-level detection is automated using graph-based deep learning, improving efficiency in identifying recurring elements such as windows, doors, and utilities. The key contribution is the demonstration of how rule-based symbol matching (prone to error due to scaling or rotation) can be replaced by learning-based classification.
- Component Size Generation in Reinforced Concrete Structures using Heterogeneous GNNs (Paper 4): Although applied in structural engineering rather than architectural plans, this study shows how heterogeneous graph models can predict dimensional attributes of components. Its relevance lies in the possibility of predicting missing room

attributes (wall thickness, door/window dimensions) directly from data rather than requiring manual annotation.

- Together, these works highlight a trend toward intelligent, data-driven CAD/BIM processing, moving away from purely rule-based approaches.

5.9.3 Application to the Current Project

For this project, the ML integration can proceed in incremental stages:

1. *Data Preparation:*
   - Convert room and wall geometries extracted from DXF into graph structures (nodes = walls, edges = adjacency).
   - Annotate known rooms, walls, and openings as training labels.
2. *Symbol Detection:*
   - Use GNN-based classifiers to identify whether a given block insert represents a door, window, or other furnishing element.
   - This improves accuracy compared to block-name string matching.
3. *Room Classification:*
   - Apply supervised ML to automatically predict room type (bedroom, living, bathroom) based on area, perimeter, and neighboring relationships.
   - Enables estimation workflows without requiring text labels.
4. *Error Recovery:*
   - ML models can learn to infer missing doors/windows based on common architectural design patterns, filling gaps where CAD input is incomplete.
5. *Scalability:*
   - Once trained, the system can generalize to multiple plan formats across projects, making it practical for real-world deployment.

5.9.4 Proposed ML Workflow

The following workflow is proposed:

1. *Input:* DXF floor plan.
2. *Preprocessing:* Extraction of walls, rooms, and opening geometries (as already implemented in the rule-based stage).
3. *Graph Construction:* Represent the floor plan as a graph (rooms/walls as nodes, adjacency relations as edges).
4. *Model Training:* Use GNNs to classify and label entities (room type, wall type, symbol detection).
5. *Output:* Enhanced structured data with semantic labels (e.g., "Living Room – 20 m², 2 windows, 1 door").
6. *BIM Export:* Push this enriched dataset into Revit via Dynamo or IFC for scheduling and estimation.

6. Work done / Results (Midterm)

6.1 Implementation summary

- Implemented DXF parsing with ezdxf.
- Implemented two room detection strategies (closed-polylines and polygonize-from-lines).
- Implemented geometric deduplication for walls using normalized keys and overlap detection.
- Implemented door/window detection from block INSERT entities and host-wall assignment.

- Implemented per-room gross/net length and area computation, with material quantity derivations (wall surface area, wall volume).

- Exported canonical data to Excel/JSON and created a Dynamo graph to import geometry into Revit.

- Verified model reconstruction in Revit (walls, hosted doors) on example plans.

6.2 Sample outputs (example)

```
Found 3 doors, 4 windows
Unnamed_Room_1:
    Floor Area            = 25.36 m²
    Gross wall length     = 20.97 m
    Deducted wall length  = 5.74 m
    Net wall length       = 15.23 m
    Wall Surface area     = 45.69 m²
    Wall Volume           = 9.14 m³
    Doors                 = 2
    Windows               = 2
    Bricks                = 4569
    Plaster vol           = 0.55 m³
    Paint (litres)        = 4.57 L
    Tiles                 = 70

ROOM 1:
    Floor Area            = 25.14 m²
    Gross wall length     = 20.97 m
    Deducted wall length  = 4.37 m
    Net wall length       = 16.60 m
    Wall Surface area     = 49.81 m²
    Wall Volume           = 9.96 m³
    Doors                 = 1
    Windows               = 2
    Bricks                = 4980
    Plaster vol           = 0.60 m³
    Paint (litres)        = 4.98 L
    Tiles                 = 69
```

Fig 2. Room wise BOQ's

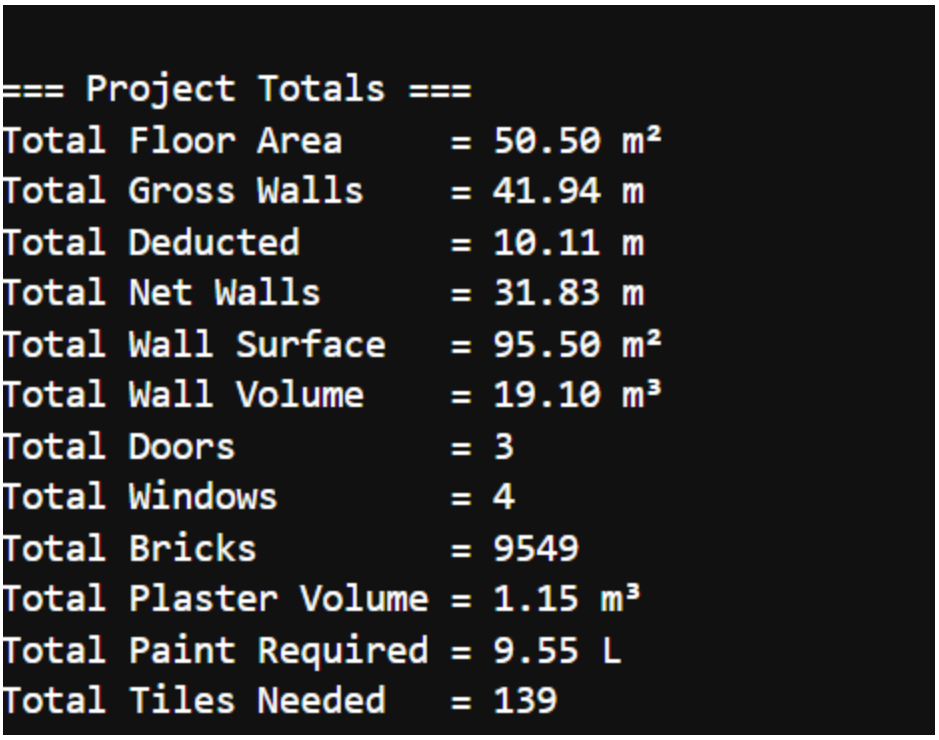

Fig 3. Total plan BOQ

6.3 Key debugging discoveries

- *Units mismatch:* DXF files sometimes lack explicit units. The pipeline checks and normalizes units; manual verification remains necessary.
- *Shared doors handling:* Doors shared by two rooms required logic to avoid double-counting deductions; we implemented wall-centric assignment and split deductions for shared-wall cases.
- *Block naming inconsistencies:* Relying on block names is brittle; plan to move toward shape-based classification.

6.4 Visual verification

Matplotlib-based plots overlaying detected walls/rooms/openings provided fast visual verification. Dynamo previews were used to confirm geometry before committing to Revit.

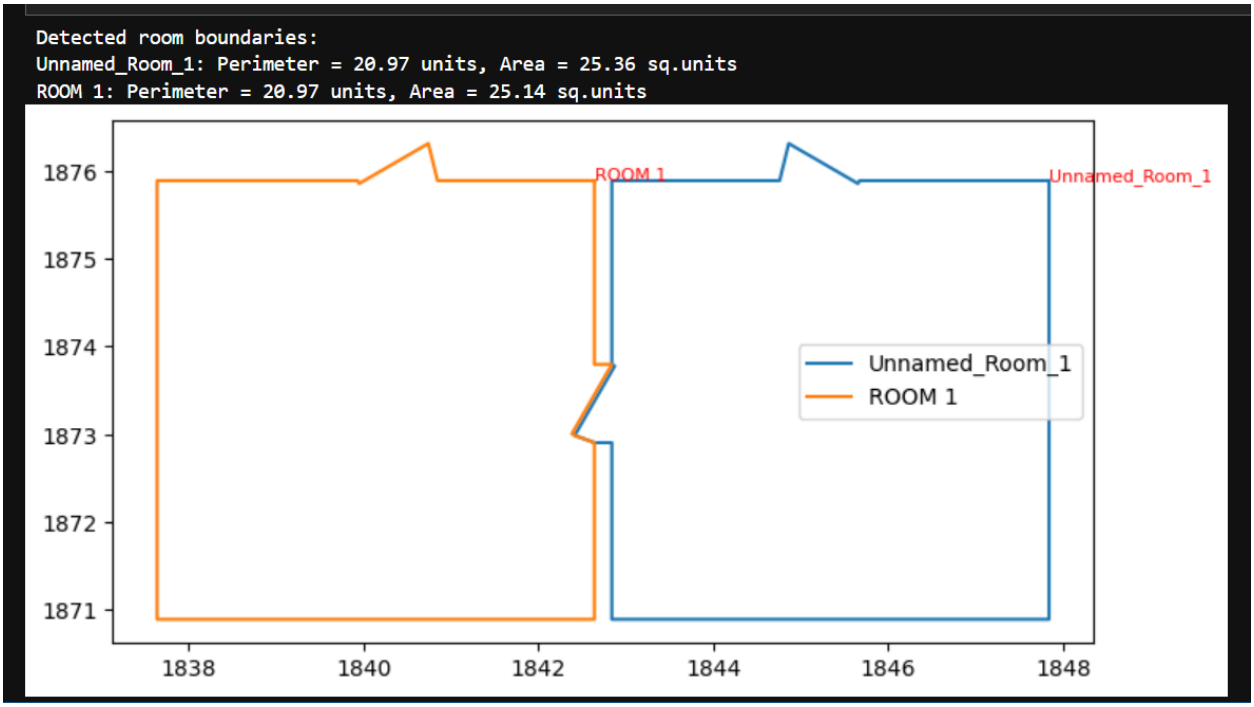

Fig 4. Visual verification

7. Timeline / Monthly progress

Activities	Aug	Sep	Oct	Nov	Dec
Project setup & DWG/DXF pipeline	X				
Room detection (closed poly + polygonize)	X	X			
Wall dedup & opening handling		X			
Export (JSON/Excel) & Dynamo import		X	X		
Revit hosting and schedules			X	X	
ML literature & plan			X	X	
Report and documentation			X		X

(X marks completed/ongoing items; timeline is tentative.)

8. Future work

Short-term (next 4–8 weeks):

1. *Robust wall merging:* finalize geometry-based merging algorithms and test on a larger variety of real plans.

2. *Symbol classifier:* implement a lightweight symbol classifier (rule-based + CNN on small raster patches) to recognize doors/windows and reduce dependence on block names.

3. *Parameterizable estimates* integrate material dictionaries and unit-rate tables for automatic cost estimation and simple schedule templates (WBS mapping).

4. *User interface:* simple CLI or small GUI wrapper so a user can select DXF, choose conversion options, and run export.

Long-term (beyond deadline):

1. *GNN-based* **topology:** train a GNN to predict semantic labels and adjacency on a curated dataset to improve generalization across drawing styles.

2. *Revit family param mapping:* map door/window attributes and wall types to Revit family parameters to enable better schedules.

3. *End-to-end app:* integrate everything into a desktop/web app for handoff to practitioners.

9. References

1. Research papers on Graph Neural Networks applied to floorplans and symbol recognition (reviewed during the project).
2. ezdxf library documentation (reading/writing DXF files).

3. Shapely and networkx documentation (geometry and graph operations).

4. Dynamo and Revit API references for geometry creation and family instance placement.

Appendices

A. Key code excerpts (high-level)

DXF parsing and closed poly room detection (pseudo-code):

```
for e in msp.query("LWPOLYLINE"):
    if e.closed:
        poly = Polygon([(x,y) for x,y in e.get_points()])
        rooms.append(poly)

for e in msp.query("TEXT MTEXT"):
    texts.append((e.plain_text(), e.dxf.insert))

# assign labels by testing if text point in polygon
```

Wall deduplication approach (concept):

```
# normalize endpoints
key = tuple(sorted([round(p1,3), round(p2,3)]))
if key not in walls:
    walls[key] = LineString([p1,p2])
# also check overlap with existing lines using buffer/intersects
```

Export structure (example JSON fields):

```
{
  "walls": [{"x1": ..., "y1":..., "x2":..., "y2":..., "length":...}],
  "rooms": [{"name":"ROOM 1","area":25.36,"walls":[0,1,2,...]}],
  "openings": [{"type":"door","wall_id":3,"width":1.37}]
}
```

B. How to reproduce the midterm demo

1. Open example_plan.dxf in the project folder.

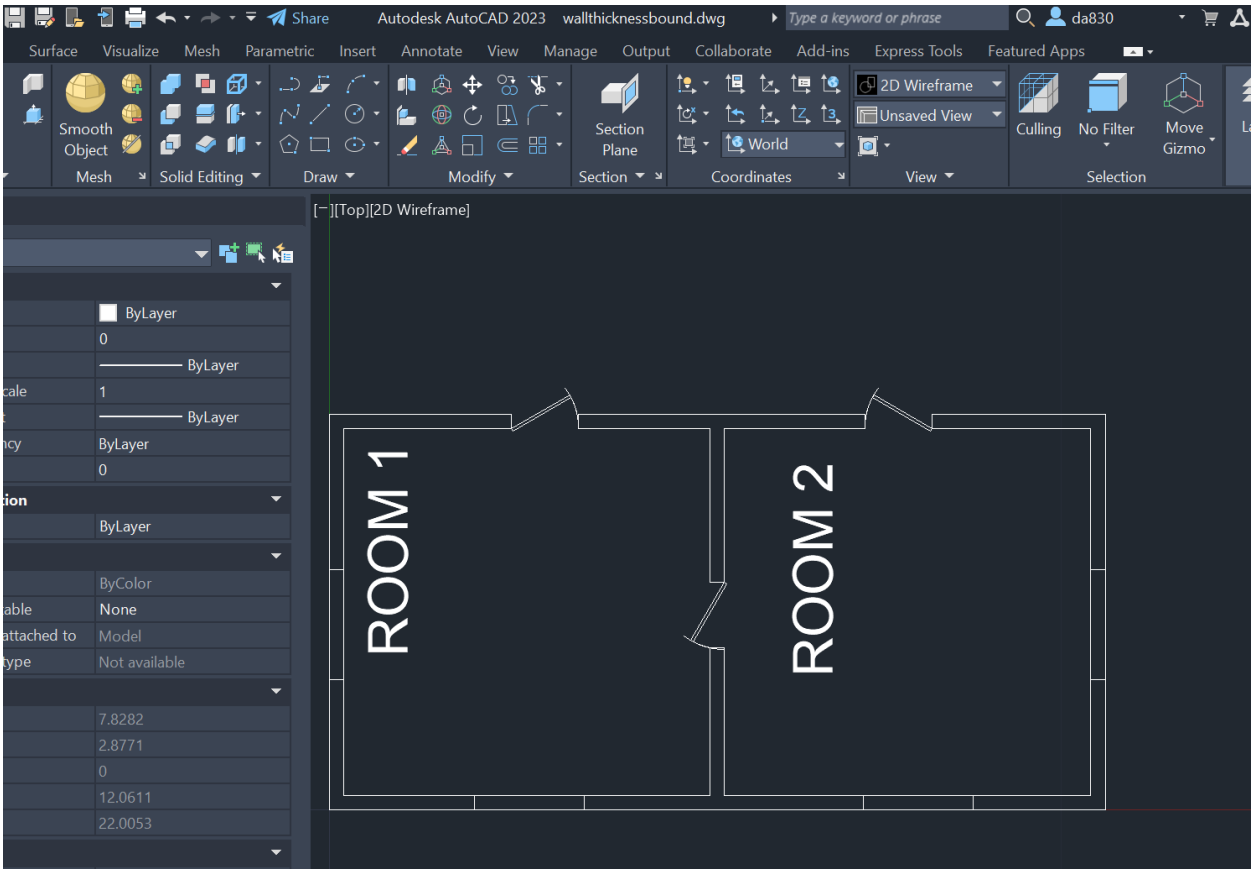

Fig 5. AutoCAD plan

```
# --- Step 1: Load DXF file ---
# Replace with your actual DXF file name
file_path = "wallthicknessbound.dxf"
doc = ezdxf.readfile(r"C:\Users\dhuvo\Documents\university\FYUP\wallthicknessbound.dxf")
msp = doc.modelspace()
```

Fig 6. Code sippet for .dwg file import

2. Run the parsing notebook (parse_floorplan.ipynb).

3. Inspect the generated debug PNGs in debug_plots/.

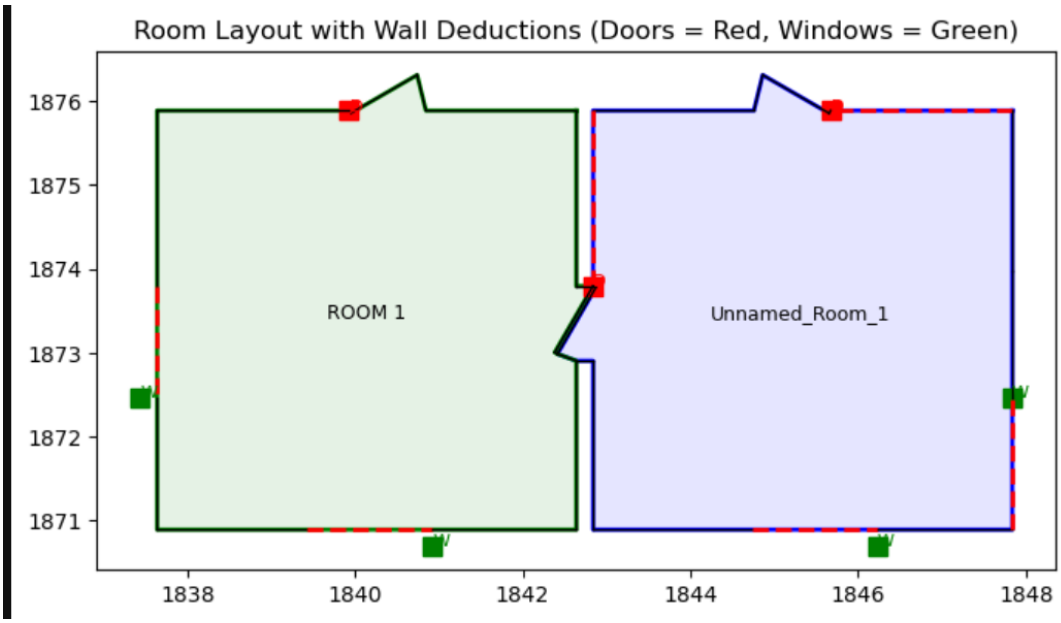

Fig 7. Debug plot

4. Run export_to_excel.py to create rooms_export.xlsx.

A1 | wall_id

	A	B	C	D	E	F	G	H
1	wall_id	x1	y1	x2	y2	length		
2	1	10.40079	5.200005	8.23425	5.200005	2.167		
3	2	8.23425	5.200005	8.2152	5.167009	0.038		
4	3	8.2152	5.167009	7.423994	5.623812	0.914		
5	4	7.423994	5.623812	7.31985	5.200005	0.436		
6	5	7.31985	5.200005	5.400792	5.200005	1.919		
7	6	5.400792	5.200005	5.400792	3.106272	2.094		
8	7	5.400792	3.106272	5.433788	3.087222	0.038		
9	8	5.433788	3.087222	4.976985	2.296016	0.914		
10	9	4.976985	2.296016	5.200792	2.214013	0.238		
11	10	5.200792	2.214013	5.400792	2.214013	0.2		
12	11	5.400792	2.214013	5.400792	0.200005	2.014		
13	12	5.400792	0.200005	7.29417	0.200005	1.893		
14	13	7.29417	0.200005	8.79417	0.200005	1.5		
15	14	8.79417	0.200005	10.40079	0.200005	1.607		
16	15	10.40079	0.200005	10.40079	1.778677	1.579		
17	16	10.40079	1.778677	10.40079	3.278677	1.5		
18	17	10.40079	3.278677	10.40079	5.200005	1.921		
19	18	5.200792	5.200005	3.407124	5.200005	1.794		
20	19	3.407124	5.200005	3.30298	5.623812	0.436		
21	20	3.30298	5.623812	2.511774	5.167009	0.914		
22	21	2.511774	5.167009	2.492724	5.200005	0.038		
23	22	2.492724	5.200005	0.200792	5.200005	2.292		
24	23	0.200792	5.200005	0.200792	3.106272	2.094		
25	24	0.200792	3.106272	0.200792	1.778677	1.328		
26	25	0.200792	1.778677	0.200792	0.200005	1.579		

Fig 8. Sample excel output

5. Open Dynamo in Revit and load the provided graph; supply the Excel file and run to build Revit geometry.

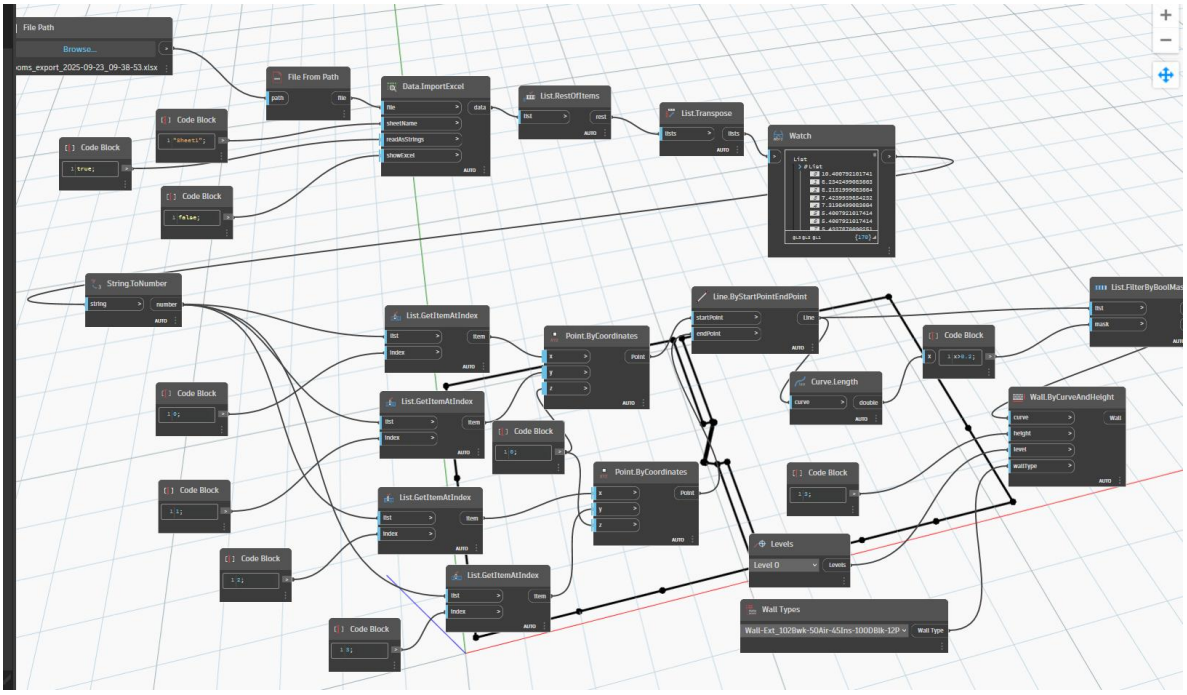

Fig 9. Dynamo Flowchart

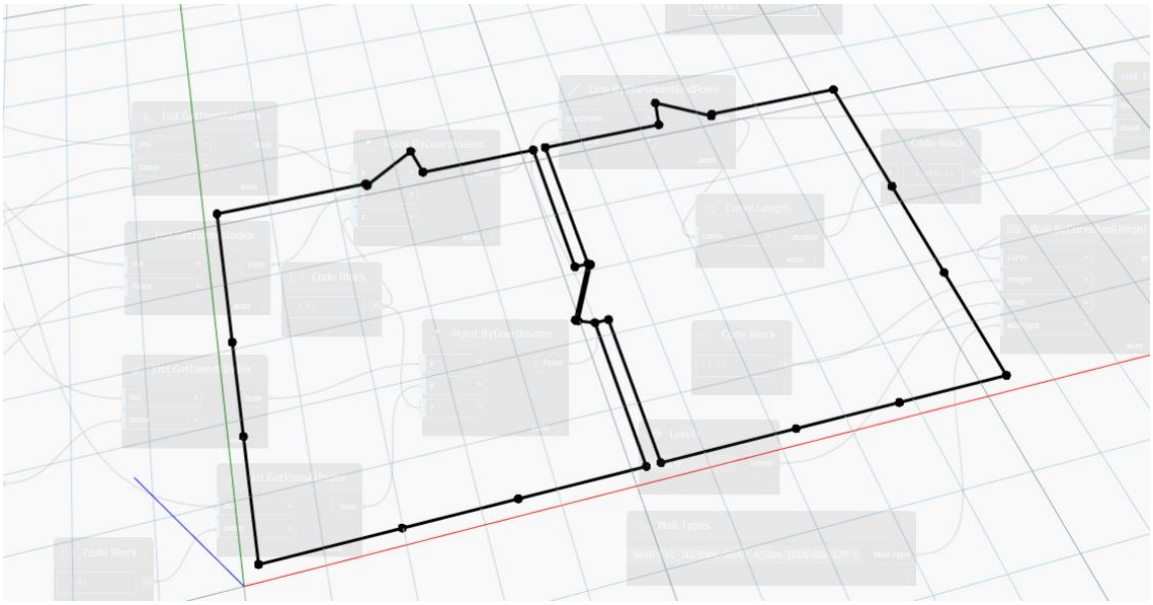

Fig 10. Dynamo output

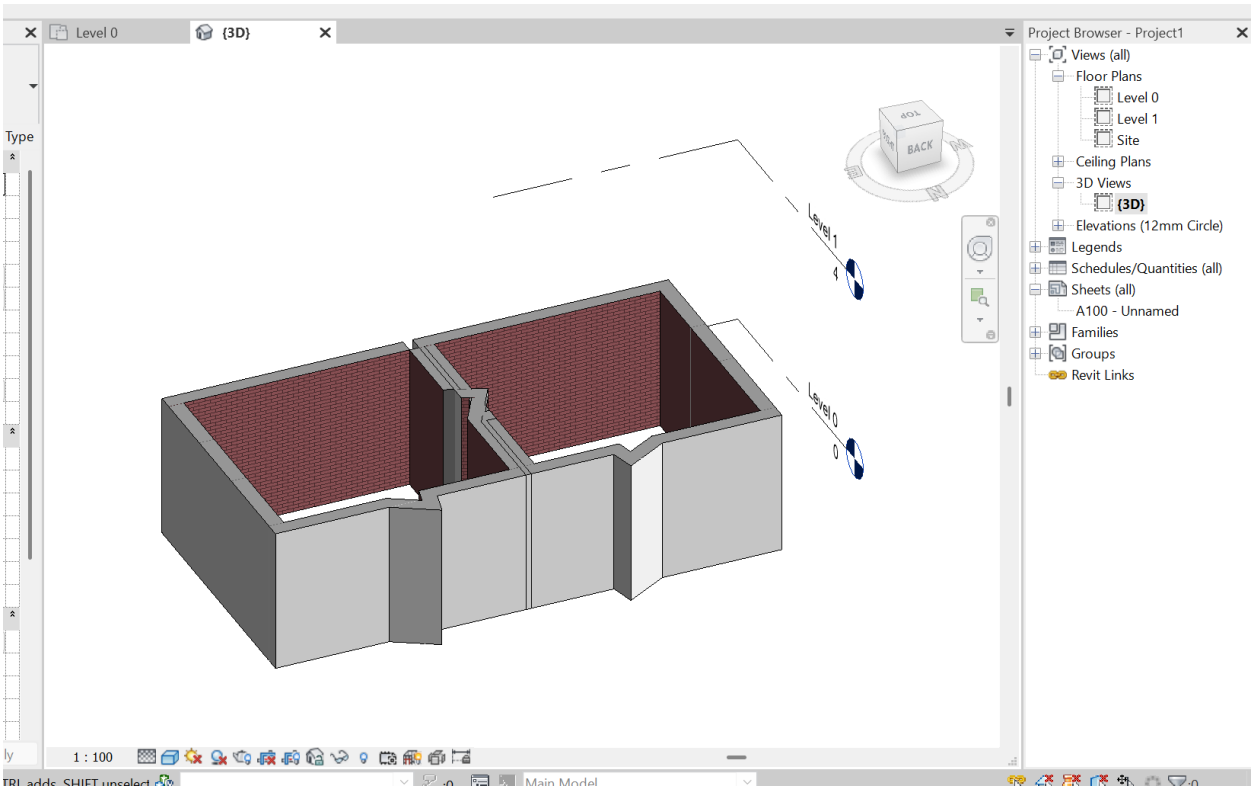

Fig 11. Revit output