T.C.
BAHÇEŞEHİR UNIVERSITY

FACULTY OF ENGINEERING AND NATURAL SCIENCES

CAPSTONE FINAL REPORT

Constructor Robot with a Given Map of Buildings

Şevval Çolak *(Computer Engineering)*
Zeynep Süzen *(Mechatronics Engineering)*
Doğacan Şahin *(Mechatronics Engineering)*
Uğur Tayşi *(Computer Engineering)*

Advisors: Dr. Selin Nacaklı & Dr. Ozan Akdoğan

ISTANBUL, June 2023

STUDENT DECLARATION

By submitting this report, as partial fulfillment of the requirements of the Capstone course, the students promise on penalty of failure of the course that

- they have given credit to and declared (by citation), any work that is not their own (e.g. parts of the report that is copied/pasted from the Internet, design or construction performed by another person, etc.);
- they have not received unpermitted aid for the project design, construction, report or presentation;
  - they have not falsely assigned credit for work to another student in the group, and not take credit for work done by another student in the group.

ABSTRACT

This article presents a project proposal for the development of a robotic crane system that is capable of assembling a scale model of a city from a given map image. The project aims to combine the expertise of computer engineering and mechatronics engineering students to design, develop and implement a robotic crane system that can process an input image of a map of buildings, extract information about the buildings and their locations, and use this information to construct a physical scale model of the city. The system will use image processing algorithms such as pixel reading and optical character recognition (OCR) to extract information from the map image, and 3D modeling libraries such as PyVista to generate a 3D output model. The robotic crane system will be designed and constructed using mechatronics engineering techniques, including robotics, artificial intelligence (AI) and control systems. The project will be divided into two main parts, one for the computer engineering students, and one for the mechatronics engineering students. The project will be completed in 15 weeks, with the first phase focused on team development and defining the scope and objectives, and the second and third phase focused on researching and selecting algorithms, designing, sourcing and assembling the robot, writing code to control its movements and actions and testing and verifying the accuracy of construction. The project is expected to produce a functional robotic crane system that can process an input image of a map of buildings and construct a physical scale model of the city using the extracted information.

Key Words: Image Processing, Map Recognition, Automation, Construction, Robot.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

BAU	Bahçeşehir University
OCR	Optical Character Recognition
MIT	Massachusetts Institute of Technology
CMP	Computer Engineering
MCH	Mechatronics Engineering

1. OVERVIEW

This project is the graduation Capstone project of Şevval Çolak, Uğur Tayşi, Zeynep Süzen and Doğacan Şahin. It is an interdisciplinary project between the computer engineering and mechatronics engineering departments.

The project is called Constructor Robot with a Given Map of Buildings. The task of Computer Engineering students includes the use of image processing in order to evaluate the given map of buildings. This is done using the OpenCV library for processing the map images, and the PyVista library for the 3D visualization of the given maps. The results of the image processing steps are then transferred over to the Mechatronics Engineering sub-team for further use. The task of Mechatronics Engineering students includes building a robot which takes the input given from the Computer Engineering students and uses that data to build the city physically using blocks.

1.1. Identification of the need

Civil engineers have a need to illustrate their projects to their colleagues or supervisors before getting an approval and being able to start on their projects. While they can use drawings and digital simulations, there is also a need to show these plans in a physical form.

1.2. Definition of the problem

While it is easy to generate drawings and digital simulations in an automated or systematic way; conventionally, generating physical scale models often require manual labor and extra effort.

With our project this problem is eliminated by automating the generation of physical scale models so that they are as easy and effortless as producing the digital ones.

1.2.1. Functional requirements

1. The system must be able to evaluate a given map of buildings using image processing techniques.
2. The system must be able to generate instructions for a robot to physically build a scale model of the city represented by the map.
3. The robot must be able to accurately follow the instructions generated by the system to construct the physical model of the city.
4. The system must be able to generate instructions that can be easily interpreted and executed by the robot.
5. The system must be able to handle different heights of buildings accurately.
6. The system must be able to generate instructions that consider the limitations and capabilities of the robot, such as reach.
7. The system must be able to generate instructions in a timely manner.

1.2.2. Performance requirements

1. The system must be able to accurately evaluate maps of buildings using image processing techniques with a high level of precision.
2. The robot must be able to accurately construct the physical model of the city, with minimal deviations from the design specified in the map.
3. The system must be able to generate instructions for the robot in a timely manner, with minimal delay.
4. The system must be able to generate instructions that consider the limitations and capabilities of the robot, such as reach, to avoid collisions and other issues.
5. The system must be able to handle different heights of buildings accurately and generate appropriate instructions for each.

6. The system must be user-friendly and easy to use, with a simple and intuitive interface for inputting and modifying the map and building designs.

1.2.3. Constraints

The main constraints of this project include scheduling, costs, and standards which are explained further below:

Scheduling: Our team consists of four members with skills in computer engineering and mechatronics engineering. This limits our design choices to those that can be implemented with the skills and expertise available in our team. We also have a limited amount of time to complete the project, which may impact our ability to implement certain features or perform certain tasks.

Costs: Our budget for the project is limited, which may restrict our ability to purchase certain materials or equipment. This may impact our design choices and the feasibility of certain features or aspects of the project. The total amount that will be provided by BAU is 2000 Turkish liras.

Standards: For safety standards, if we assume our robot could be classified as an industrial robot, the ISO/TS 15066:2016 safety standard will apply, titled Robots and robotic devices – Collaborative robots. On the other hand, a performance standard that could apply to the project is ISO 9001. This standard for quality management systems includes requirements for a system that can be implemented to consistently meet customer requirements and continuously improve the product or sub-system. It can also be used to guarantee that the project adheres to safety, performance, and reliability requirements. Lastly, for relevant health and safety laws and standards, the product could comply with OSHA regulations for workplace safety and UL

standards for electrical safety. Additionally, the product could also comply with ISO standards for product safety, such as ISO 12100 for machinery safety and ISO 14121 for risk assessment.

Other constraints: In addition to the above, we need to consider business, environmental, health and safety considerations. This may impact our design choices and the implementation of certain features or aspects of the project.

Sustainability considerations and constraints: Our project needs to take into account economic, environmental, and social aspects. This includes the cost and availability of materials, the environmental impact of the project, and the social impact on people's lives and employment. These considerations may impact our design choices and the implementation of certain features or aspects of the project.

1.3. Conceptual solutions

The conceptual solutions for this project involve the use of image processing techniques to evaluate the given map of buildings and generate instructions for a robot to physically construct a scale model of the city. All potential approaches use computer vision algorithms to identify and classify the different buildings in the map, and generate instructions for the robot to place the corresponding blocks in the appropriate locations.

1.3.1. Literature Review

There are several existing products and solutions that aim to automate the process of generating physical scale models of buildings and cities. These include 3D printing technologies and robotics systems that can construct models using blocks or other materials.

One example of such a product is the "BlockBot" developed by researchers at the Massachusetts Institute of Technology (MIT) **[1]**. This system uses a robotic arm and a specialized tool that can pick up and place blocks in a precise manner. The system is able to follow instructions from a computer program, allowing it to construct structures according to a predefined design.
Another example is the "Robot Bricklayer" developed by researchers in France **[2]**. This system guides supplies of bricks and cement along a wall's path by guiding a brick-laying head, allowing it to construct walls and other structures. The system is able to follow instructions from a computer program, allowing it to build complex structures according to a predefined design. A 3D model of the Robot Bricklayer is shown in Figure 1 below.

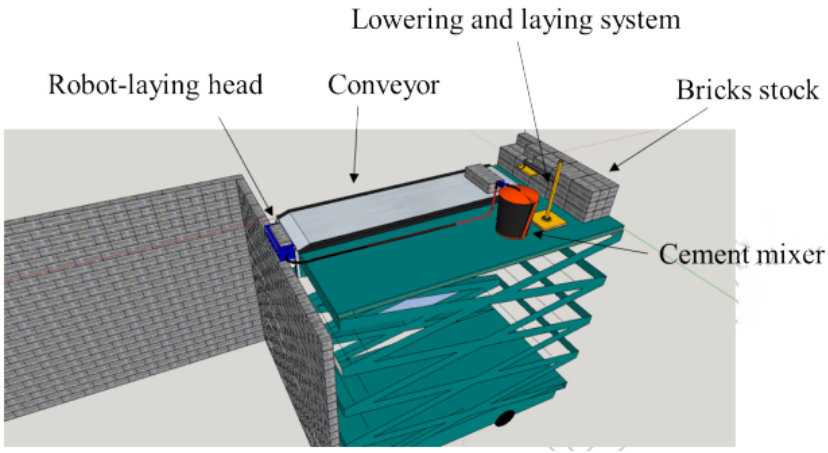


Figure 1: 3D Modeling of the Robot Bricklayer [2].

For image processing we found several different concepts and reviews of those concepts are below:

1. **Pixel reading:** Pixel reading can be used to determine the positions and sizes of buildings by analyzing the pixels that make up the image of the buildings. This can be done by identifying the locations of pixels with a specific color or brightness value that

corresponds to the buildings, or by analyzing patterns in the pixels that are characteristic of the shapes of the buildings.

2. **Feature extraction:** This involves identifying and extracting specific features or characteristics of the buildings in the map, such as edges, corners, or shapes, and using these features to classify the buildings and generate instructions for the robot [3].
3. **Object recognition:** This involves training a machine learning model to recognize and classify different types of buildings in the map, using features extracted from the images. This approach could be more accurate and efficient than manual feature extraction, but may require more time and resources to implement [4].
4. **Optical character recognition (OCR):** If the map includes text labels or other written information, OCR could be used to extract and read this information, potentially providing additional context or details about the buildings and their locations [5].

For 3D output generation we found different python libraries. They are listed below with the explanations.

1. **Matplotlib:** Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. By importing mplot3d toolkit one can create a wide range of 3D plots, including scatter plots, surface plots, wireframe plots, and more. A downside of matplotlib is that it does not utilize GPU hardware acceleration, which can lead to performance issues when working with large numbers of points or faces in 3D plots. These large amounts of data can make the plots slow and unresponsive[6].
2. **Open3D:** One of the most rapidly expanding 3D libraries for Python is a tool that is constructed using C++ but also has all of its features accessible through both C++ and Python. This makes it fast, adaptable, and dependable. The purpose of the library is to become a comprehensive solution for 3D processing, similar to how OpenCV is

considered as an all-in-one solution for computer vision, including all necessary features and easy integration with other libraries[6].

3. **Trimesh:** Trimesh is a library that is completely written in Python and specializes in handling, analyzing, and displaying meshes and point clouds. It's often used to prepare 3D data for statistical analysis and machine learning tasks. Trimesh library doesn't have visualization components as a part of its base library, but it uses pyglet - a library for creating games and interactive applications as an optional dependency. It provides visualization functionality, interaction capabilities, and manipulation tools on top of pyglet. As a result, it provides a wide range of built-in visualization interactions by default[6].
4. **V3do:** Vedo (also known as V3do) is a python library that allows for the scientific analysis and visualization of 3-dimensional objects. It can be used for plotting 1d, 2d, and 3d data, point clouds, meshes, and volumetric visualization. It uses Trimesh as its backend for some of its processing, import/export and generation of mesh and point clouds and provides additional advanced functionality like creating vector illustrations that can be imported into LaTeX, generating physics simulations, deep learning layer overviews and mechanical and CAD-level slices and diagrams. To use Vedo, VTK and numpy should be installed before Vedo is installed[6].
5. **PyVista:** PyVista is a python library that simplifies the process of creating visualizations and analyzing meshes using VTK, a visualization toolkit. It provides an easy to use interface, making it more pythonic, and utilizes OpenGL for smooth visualizations and animations. PyVista supports both point clouds and meshes, and has a wide range of features including slicing, resampling, surface reconstruction, mesh smoothing, and ray-tracing, providing many examples and tutorials that cover simple visualizations to complex analysis[6].

These existing solutions demonstrate the feasibility of using robotics and computer control to automate the process of building physical scale models of buildings and cities. However, our project aims to improve upon these solutions in several ways.

First, our project will use image processing techniques to evaluate the given map of buildings, allowing it to automatically generate instructions for the robot. This will make the system more user-friendly and efficient, as it will not require manual input of instructions.

Second, our project will use advanced robotics and control techniques to enable the robot to construct the physical model with a high level of accuracy and efficiency. This will improve upon existing solutions, which may have limitations in terms of the complexity of structures they can build, or the precision and speed of construction.

Third, our project will consider the limitations and capabilities of the robot, such as its reach, to generate instructions that can be easily executed by the robot. This will improve upon existing solutions, which may not take these factors into account and may result in collisions or other issues.

Overall, our project aims to develop a more advanced and efficient solution for automating the process of generating physical scale models of buildings and cities. This will provide a valuable tool for civil engineers and other professionals who need to illustrate their projects in a physical form.

1.3.2. Concepts

We considered several conceptual solutions for our project. Pixel reading, edge detection, feature detection and optical character recognition for image processing and for 3D modeling technologies PyVista and Matplotlib libraries are considered for CMP part while using a robotic crane with a gripping mechanism to assemble the physical model, or using a mobile robot with a lifting mechanism to move and place blocks, and using a combination of both approaches are considered. After considering the various factors, we decided that the best solution would be to use pixel reading for image processing and PyVista library for 3D output modeling and a combination of a robotic crane and a mobile robot for the robot building.

The Table 1 and Table 2 below compares our three concepts for image processing and two concepts for 3D output modeling with respect to several key considerations:

Consideration	Pixel Reading	Feature Detection	Edge Detection
Cost	*Low*	*High*	*Medium*
Complexity	*Low*	*Low*	*Low*
Performance	*Low*	*High*	*High*
Features	*High*	*High*	*Medium*

Table 1: Strength and Weaknesses of Different Concepts for Image Processing

Consideration	Matplotlib	PyVista
Cost	*Free*	*Free*
Complexity	*Medium*	*High*
Performance	*Medium*	*High*
Features	*High*	*High*

Table 2: Strength and Weaknesses of Different Concepts for 3D Modeling

The Table 3 below compares our three concepts for robot building with respect to several key considerations:

Consideration	Robotic Crane	Mobile Robot	Combination
Cost	*Moderate*	*Moderate*	*High*
Complexity	*Moderate*	*High*	*High*
Performance	*High*	*High*	*Highest*
Features	*High*	*High*	*Highest*

Table 3: Strength and Weaknesses of Different Concepts

The strengths and weaknesses of each concept are as follows:

Robotic Crane: This concept offers high performance in terms of precision and speed, and is capable of handling a wide range of building heights and designs. However, it is limited in its ability to move and place blocks over a large area, and may require additional mechanisms to support this functionality.

Mobile robot: This concept offers high performance in terms of mobility and flexibility, and is capable of moving and placing blocks over a large area. However, it is limited in its ability to handle a wide range of building heights and designs, and may require additional mechanisms to support this functionality.

Combination: This concept offers the highest performance in terms of precision, speed, mobility, and flexibility. It is capable of handling a wide range of building heights and designs, and is able to move and place blocks over a large area. However, it is the most complex and costly of the three concepts.

After considering the strengths and weaknesses of each concept, we decided that the best solution for our project would be to use a combination of a robotic crane and a mobile robot. This solution offers the highest level of performance and features, and is able to handle a wide range of building heights and designs. It also offers flexibility and mobility, allowing the crane (robot) to move and place blocks over a large area. However, it is the most complex and costly of the three concepts, and may require additional mechanisms to support its functionality.

1.4. Physical architecture

This part is split into two parts called “Image Processing” and “Constructor Robot”, where each part represents the work of CMPE and MCH students respectively.

1.4.1. Image Processing

The physical architecture for the CMP students' part of the project, as illustrated in the figure, involves several steps for processing an input image of a map of buildings and generating output data for sending to the robot. Flow chart of the computer engineering side is shown in Figure 2 below.

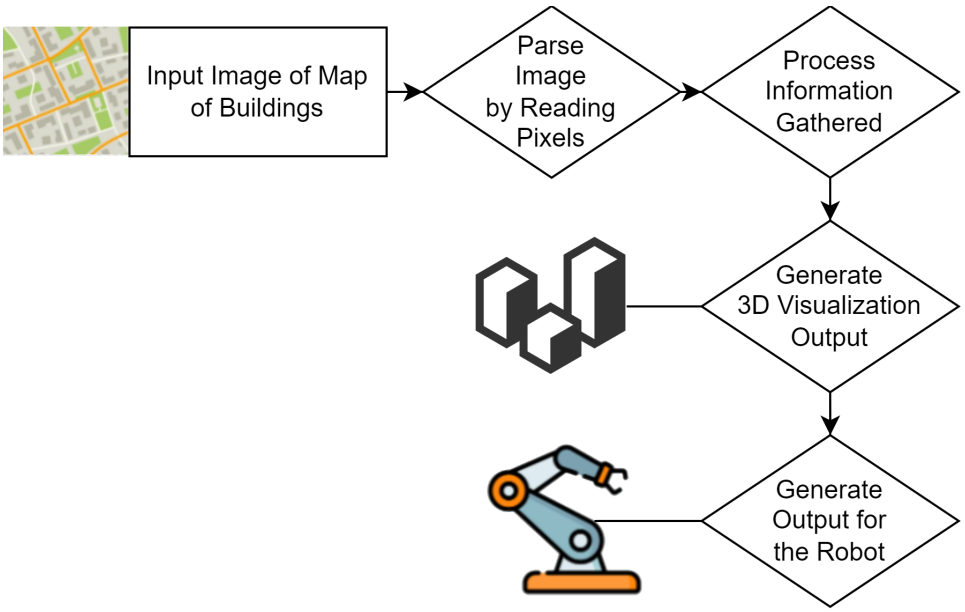


Figure 2: Flow chart of CMP students' image processing tasks

Input Image of Map of Buildings: This is the starting point of the process, where the computer engineering students receive an image of a map of buildings. This image could be in a variety of formats, such as JPEG, PNG, or BMP, and may be generated manually or obtained from a source such as a digital map database.

Parse Image by Reading Pixels: The next step involves parsing the image by reading the individual pixels in the image and extracting information about the buildings and their locations. This could be done using a variety of image processing techniques, such as pixel reading or feature extraction, to identify and classify the different types of buildings in the map.

Process Information Gathered: This step is where the CMPE students process the information gathered from the parsed image, this could include combining the information, applying algorithms to it, making calculations and more. This information will be used in output generation for the 3D visualization and the robot in the next steps.

Generate 3D Visualization Output: This step is where the CMPE students use the processed information to generate a 3D visualization output, this could include a 3D mesh representation of the map and buildings, or other types of 3D models. This step is where the team can use the PyVista library to create 3D models of the buildings. This 3D visualization output could be used for various purposes such as presenting the project results to stakeholders, generating instructions for the robot or testing the system. This step allows the team to be able to visualize and check the accuracy of the parsed image before the output is sent to the robot.

Generate Output for Sending to the Robot: Once the image has been parsed and the relevant information has been extracted, the CMPE students generate output data containing coordinate and height information for each building in the map. This data is formatted in a way that can be understood and used by the robot, which will be responsible for physically constructing the scale model of the city.

Output Data with Coordinate and Heights: The final step in the process involves sending the output data with coordinate and height information to the robot, which will use this data to build the scale model of the city. This data could be transmitted through a variety of means, such as a wired or wireless connection, depending on the specific design and capabilities of the robot.

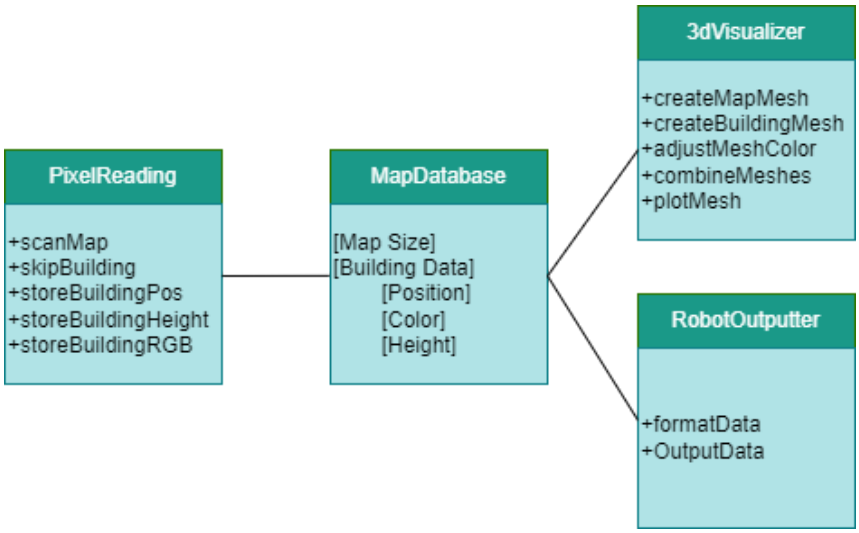


Figure 3: Interface Diagram of the System

The interface diagram of the project that is shown in Figure 3 above shows four main components: PixelReading, MapDatabase, 3DVisualizer, and RobotOutputter. The PixelReading component is responsible for scanning the map and reading the pixel data, including the position, height, and RGB values of each building. This information is then stored in the MapDatabase component, which includes the map size and building data. The 3DVisualizer component takes this data and creates a 3D mesh representation of the map and buildings, adjusting the color and combining the meshes as necessary. Finally, the RobotOutputter component takes the data from the 3DVisualizer and formats it into instructions that the robot can understand and execute. This diagram represents the high-level architecture of the system, and how these components interact with each other to achieve the project's goals.

1.4.2. Constructor Robot

The physical architecture for the MCH students' part of the project aims to analyze the provided building map and provide instructions for a robot to physically create a scale model of the city; the conceptual solutions for this project make use of image processing methods. Firstly we designed the crane, which is the main character of the project and according to the design crane will move in one dimension. Crane head will move to the project area. We chose the mini electric motor to move.

We designed 2 ways to move the chassis. First one is like the train is moving (rail system) another one is the belt system. But this 2 way has disadvantages so we continue to compare the two options even now for the best stable result after that MCH student designed the crane for the 2 options.

2. WORK PLAN

For our project we have used a Work Breakdown Structure (WBS) as a tool used to break down a project into smaller, more manageable pieces. It helped us to organize and define the scope of the project, and provided a clear and detailed outline of the work that needs to be completed.

2.1. Work Breakdown Structure (WBS)

Our WBS is organized into several main categories: Project Initiation, Image Processing, Robot Development, Integration, Testing, Debugging, Construction, and Project Closeout. The WBS is in Figure 4 below, followed by the explanations of each category.

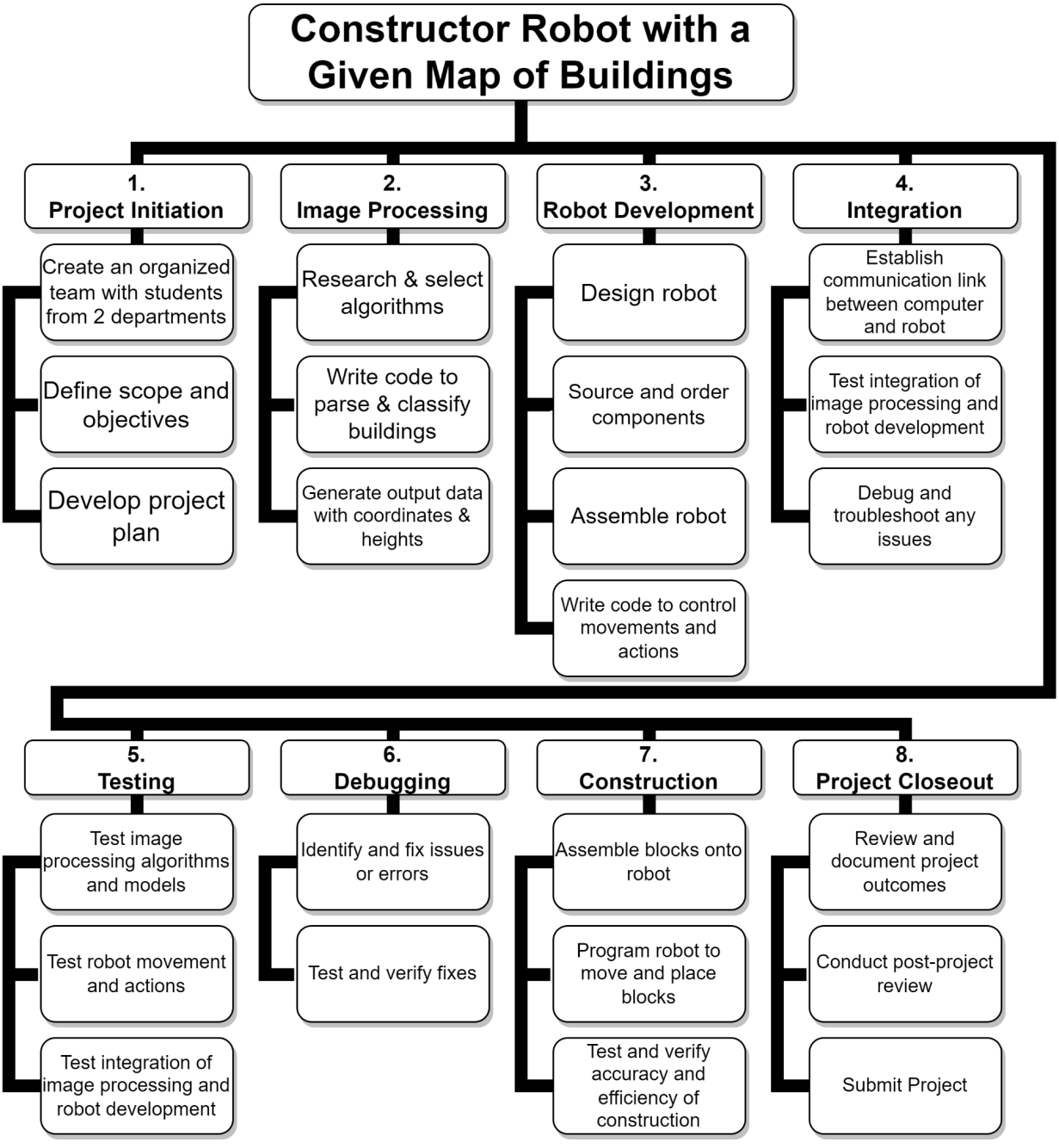


Figure 4: Work Breakdown Structure of the Project

The Project Initiation category includes tasks related to setting up the project, such as developing a project team, defining the scope and objectives, and developing a project plan.

The Image Processing category includes tasks related to developing and implementing the image processing techniques that will be used to read and analyze the input image of the map of buildings. This includes researching and selecting algorithms, and writing code to parse and classify the buildings in the map. It also includes the generation of the outputs regarding height and positions of the buildings.

The Robot Development category includes tasks related to designing, building, and programming the robot that will be used to physically construct the scale model of the city. This includes sourcing and ordering components, assembling the robot, and writing code to control its movements and actions.

The Integration category includes tasks related to coordinating the work of the computer engineering and mechatronics engineering students and integrating the image processing and robot development efforts. This includes establishing a communication link between the computer and the robot, testing the integration of the two systems, and debugging any issues that are found.

The Testing category includes tasks related to testing the various components of the project to ensure that they are working correctly. This includes testing the image processing algorithms, as well as the robot movement and actions.

The Debugging category includes tasks related to identifying and fixing any issues or errors that are found during the testing phase. This includes identifying the root cause of a problem, writing code to fix the issue, and modifying the design of the robot as needed.

The Construction category includes tasks related to physically constructing the scale model of the city using the robot. This includes assembling the blocks onto the robot, programming the robot to move and place the blocks in the correct locations, and testing the accuracy and efficiency of the construction.

Finally, the Project Closeout category includes tasks related to completing the project, such as reviewing and documenting the project outcomes, obtaining customer acceptance, conducting a post-project review, and releasing resources.

2.2. Responsibility Matrix (RM)

The responsibility matrix for our project is as follows:

TASKS	STUDENTS			
	Şevval Çolak	**Uğur Tayşi**	**Zeynep Süzen**	**Doğacan Şahin**
Project Planning	R	R	R	R
Output/Input Interdepartmental Communication	R	R	R	R
Input Integration			R	R
Robot Development			R	R
Pixel Reading	R	S		
Pixel Processing	S	R		
Output Generation for the Robot	R	S		
3D Visualization of the Processed Map	S	R		
Project testing and debugging	R	R	R	R

Table 4: Responsibility Matrix of the Project

According to the matrix above, the tasks of each student depends on whether they are one of the CMPE or MCH students. Each element can be described like the following:

Project Planning: Both CMPE & MCH students are responsible for planning and organizing the project, including establishing goals and objectives, setting a timeline and budget, and gathering necessary resources and materials.

Image processing and analysis: The CMPE students are responsible for developing and implementing image processing techniques to read and analyze the input image of the map of buildings, including tasks such as selecting and testing algorithms, training machine learning models, and writing code to parse and classify the buildings in the map.

Output data generation: The CMPE students are responsible for generating output data with coordinate and height information for each building in the map, which will be used by the robot to build the scale model of the city. This may involve tasks such as formatting the data in a way that can be understood by the robot, or testing and debugging the output data to ensure its accuracy and completeness.

Output/Input Interdepartmental Communication: Both CMPE & MCH students are responsible for communicating with each other and coordinating the integration of the work between the image processing team and the robot development team. This may involve tasks such as establishing a communication link between the computer and the robot, or testing and debugging the integration of the two systems.

Input Integration: The MCH students are responsible for integrating the input data from the CMPE students with the robot development process, including tasks such as parsing and interpreting the output data, and using it to guide the construction of the scale model of the city.

Robot Development: The MCH students are responsible for designing, building, and programming the robot that will be used to physically construct the scale model of the city. This

may involve tasks such as selecting and sourcing components, assembling the robot, and writing code to control its movements and actions.

Project testing and debugging: Both CMPE & MCH students are responsible for testing the overall system to ensure that it is working correctly and meeting the goals and objectives of the project. This may involve tasks such as debugging any issues that arise, or conducting user testing to gather feedback and identify any potential improvements.

2.3. Project Network (PN)

The project network diagram that is shown in Figure 5 below illustrates the various tasks and dependencies involved in the project, as well as the flow of work from start to finish.

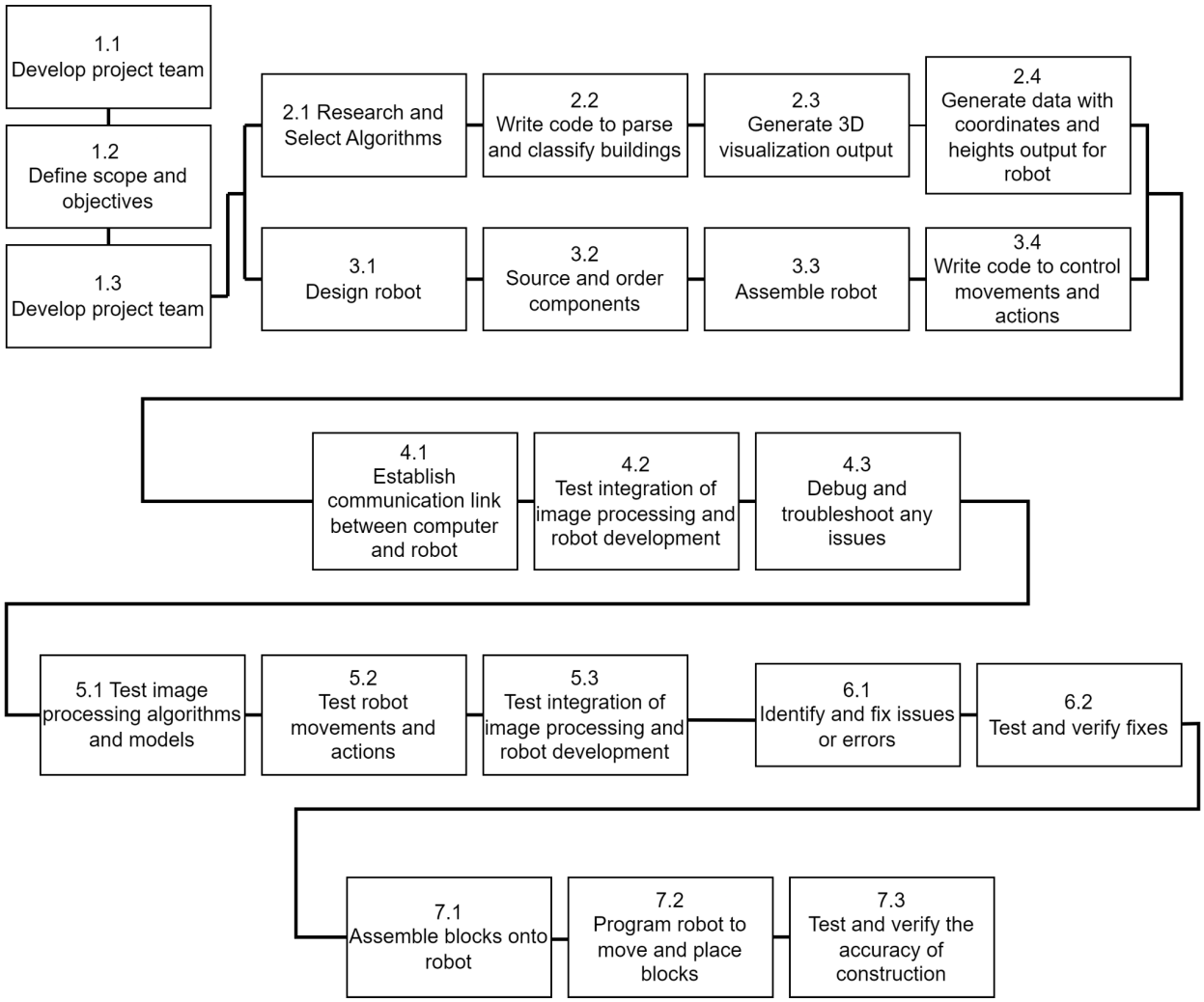


Figure 5: Project Network Diagram

The network diagram of the project shows the various steps involved in the project, organized in a sequential order. The first phase of the project focuses on team development and defining the scope and objectives. Then the second phase is focused on researching and selecting algorithms and writing code for parsing and classifying buildings and 3D visualization outputs, as well as coordinate and heights output for the robot. The third phase focuses on designing, sourcing and

assembling the robot, writing code to control its movements and actions. These processes in phase 2 and 3 are done in parallel. Then the fourth phase is focused on establishing communication link between computer and robot and testing integration of image processing and robot development, with debugging and troubleshooting any issues. The fifth phase is focused on testing the image processing algorithms and models, robot movements, and actions and integration of the two. The sixth phase is focused on identifying and fixing issues or errors, and testing and verifying the fixes. Lastly, the seventh phase is focused on assembling blocks onto the robot, programming it to move and place blocks and testing and verifying the accuracy of construction. This diagram provides an overview of the different steps of the project and the dependencies between them, which helps the team to efficiently organize and manage their work.

2.4. Gantt chart

The Gantt chart for this project that is provided in the next page as Figure 6, covers a period of 15 weeks within the Spring 2023 semester and outlines the various tasks and milestones that need to be completed in each week.

In week 1, the team will initiate the project by developing a project team, defining the scope and objectives, and identifying the stakeholders. They will also develop a project plan.

Between weeks 2-7, the team will focus on image processing. They will research and select the appropriate algorithms that are needed. They will write code to parse and classify the buildings in the map, and generate output data with the coordinates and heights of the buildings.

In parallel, the team will also work on developing the robot. This includes designing the robot, sourcing and ordering components, and assembling the robot. They will also write code to control the movements and actions of the robot.

Between weeks 8-9, the team will focus on integrating image processing and robot development. This includes establishing a communication link between the computer and robot and testing the integration to ensure that everything is working correctly. Any issues or errors will be debugged and troubleshooted.

Between weeks 10-11, the team will test the image processing algorithms and models, the robot movements and actions, and the integration of the two.

Between weeks 12-13, the team will identify any issues or errors and work on debugging and fixing them.

On week 14, the team will focus on assembling the blocks onto the robot and programming it to move and place the blocks accurately and efficiently. They will test and verify the accuracy and efficiency of the construction process.

Overall, the Gantt chart outlines a clear plan for the development of the image processing methods, the robot and the construction of the scale model; with each week dedicated to specific tasks and milestones. It provides a timeline for the project and helps the team stay organized and on track.

	Task Name	Duration	Start	ETA
	Complete project execution	14 weeks	27.02.23	04.06.23
1	**Project Initiation**	1 week	27.02.23	05.03.23
1.1	Develop project team			
1.2	Define scope and objectives			
1.3	Develop Project Plan			
2	Image Processing	6 weeks	06.03.23	16.04.23
2.1	Research and Select Algorithms			
2.2	Write code to parse and classify buildings			
2.3	Generate 3D visualization output			
2.4	Generate data with coordinates and heights output for robot			
3	Robot Development	6 weeks	06.03.23	16.04.23
3.1	Design Robot			
3.2	Source and order components			
3.3	Assemble Robot			
3.4	Write code to control movements and actions			
4	Integration	2 weeks	17.04.23	30.04.23
4.1	Establish communication link between computer and robot			
4.2	Test integration of image processing and robot development			
4.3	Debug and troubleshoot any issues			
5	Testing	2 weeks	01.05.23	14.05.23
5.1	Test image processing algorithms and models			
5.2	Test robot movement and actions			
5.3	Test integration of image processing and robot development			
6.	Debugging	2 weeks	15.05.23	28.05.23
6.1	Identify and fix issues or errors			
6.2	Test and verify fixes			
7.	Construction	1 week	29.05.23	04.06.23
7.1	Assemble blocks onto robot			
7.2	Program robot to move and place blocks			
7.3	Test and verify the accuracy of construction			

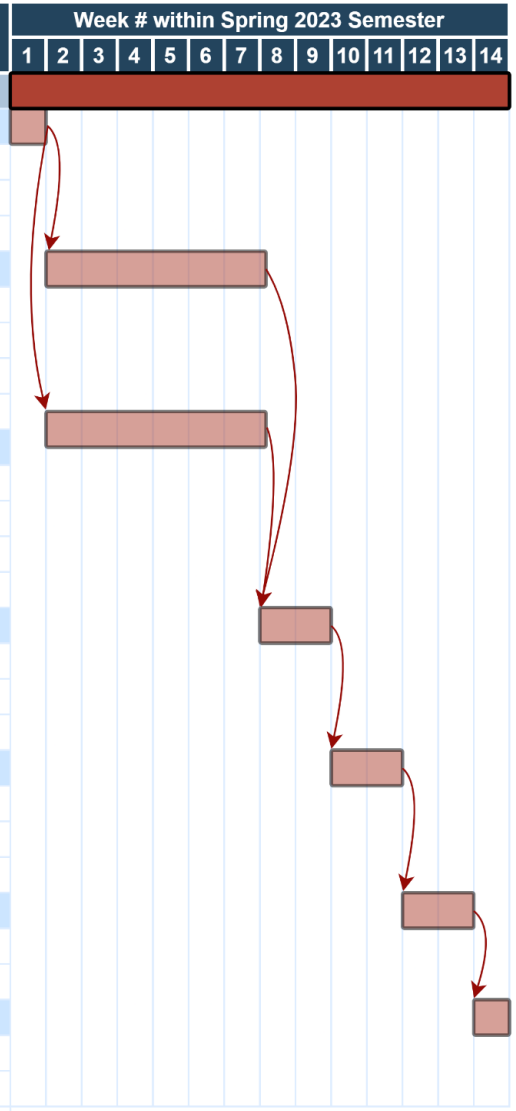


Figure 6: Gantt Chart of the Project

2.5. Costs

Material cost for the building of the crane is calculated as in Table 5 below:

Electro-mechanical	
Geard Motors	373.96₺
L298N Driver Card	54.80₺
Electromagnet	230₺
220V 10A Power Supply	234.70₺
Communications and Control	
Arduino Uno Clone Set	331.55₺
Jumper Cable	54.90₺
Sensor (Ultrasonic distance sensor, Infrared distance sensor)	238.49₺
L298N Driver Card	74.80₺
5V Relay	52.70₺
System Total	**1645.90 ₺**

Table 5. Costs

2.6. Risk assessment

Risk matrix below is used to identify and evaluate potential risks that could impact the success of a project. It involves assigning a probability, impact and priority level to each risk and then plotting them on a matrix to determine the overall level of risk. The higher the probability and impact of a risk, the higher the priority it should be given in terms of addressing and mitigating

it. In the case of our project, the risk matrix was created to identify and evaluate risks related to budget constraints, technical difficulties with image processing algorithms, difficulty in integrating image processing and robot development, and difficulty in controlling the robot.

Afterwards the risk assessment table was used to identify and evaluate potential risks in the project. It provides a more detailed view of the risks, mitigation plans and timeline for addressing them. It also provides a detailed plan of action for addressing each risk, including the steps that need to be taken to mitigate or eliminate the risk, as well as the person or team responsible for implementing the mitigation plan. In the case of our project, the risk assessment table was created to identify and evaluate the same risks as in the risk matrix and provide a detailed plan of action for addressing them.

Risk	Probability	Impact	Priority
Insufficient budget	Medium	High	High
Technical difficulties with image processing algorithms	Medium	High	High
Difficulty in integrating image processing and robot development	Medium	High	High
Difficulty in controlling the robot	Low	High	Medium

Figure 7: Risk Matrix

Risk	Mitigation Plan	Timeline
Insufficient budget	Identify and prioritize essential components and tasks. Seek additional funding through grants and sponsorships. Cut down on non-essential expenses.	Continuously throughout the project
Technical difficulties with image processing algorithms	Research and test multiple algorithms before settling on the final one. Continuously monitor performance and troubleshoot any issues that may arise.	Continuously throughout the project
Difficulty in integrating image processing and robot development	Clearly define the communication protocols and establish a strong communication link between the computer and robot. Regularly test the integration and troubleshoot any issues.	Before week 8
Difficulty in controlling the robot	Use a robust control system and well-established controllers. Regularly test and troubleshoot the control system to ensure it is functioning as intended.	Before week 14

Figure 8: Risk Assessment Table

3. SUB-SYSTEMS

The project is divided into two main sub-systems: the Computer Engineering and Mechatronics Engineering (CMP and MCH) sub-systems. The Computer Engineering (CMP) sub-system is responsible for the image processing and the generation of a 3D model of the map. This sub-system uses image processing algorithms to extract information about the buildings in the map, and generate a 3D model of the city. This sub-system also generates an output log file containing coordinate and height information for each building, which is later used by the robot to construct the scale model of the city.

The Mechatronics Engineering (MCH) sub-system is responsible for the design and production of the robotic crane. This sub-system uses robotics technology to design and control the movement of the robotic crane, including the use of sensors, actuators, and controllers. This sub-system also involves the design and production of the robotic crane's mechanical and electrical components, as well as its software and control systems. The MCH sub-system aims to design and produce a robotic crane that is safe, precise, efficient and cost-effective.

3.1. Computer Engineering (CMPE)

The Computer Engineering (CMPE) sub-system is responsible for developing and implementing image processing techniques to read and analyze the input image of the map of buildings, and generating output data with coordinate and height information for each building in the map. This data is then used by the robot to build the scale model of the city.

3.1.1. Requirements

The functional requirements for the CMPE sub-system include:

- The system must be able to evaluate a given map of buildings using image processing techniques.
- The system must be able to read building positions correctly from the map.
- The system must be able to read building height information correctly from the map.
- The system must be able to generate instructions for a robot to physically build a scale model of the city represented by the map.
- The system must be able to generate instructions that can be easily interpreted and executed by the robot.
- The system must be able to handle different heights of buildings accurately.
- The system must be able to generate instructions in a timely manner.

The performance requirements for the CMPE sub-system include:

- The system must be able to accurately evaluate maps of buildings using image processing techniques with a high level of precision.
- The system must be able to generate instructions for the robot in a timely manner, with minimal delay.
- The system must be able to generate at least 2 buildings next to each other.
- The system must be able to handle different heights of buildings accurately and generate appropriate instructions for each.
- The system must be user-friendly and easy

3.1.2. Technologies and methods

For the CMP students' part of the project, the main technologies and methods that will be used include image processing algorithms and machine learning techniques to parse and classify the building heights in the map, as well as programming languages and libraries to implement the image processing and data analysis tasks.

In terms of image processing algorithms, we will be exploring a range of techniques such as pixel reading, edge detection, feature extraction, template matching, and Optical Character Recognition (OCR) to identify and classify the different heights of buildings in the map. These techniques can be implemented using programming languages such as Python or C++, and may require the use of libraries such as OpenCV or scikit-image. For output generation we have two different tasks: log file generation and 3D modeling. For log file generation we will simply use file operation of chosen programming language. For 3D modeling we considered different libraries of Python such as Matplotlib, Open3, PyVista etc. The named technologies explained below.

3.1.2.1. Image Processing Methods

Pixel reading: Pixel reading can be used to determine the positions and sizes of buildings by analyzing the pixels that make up the image of the buildings. This can be done by identifying the locations of pixels with a specific color or brightness value that corresponds to the buildings, or by analyzing patterns in the pixels that are characteristic of the shapes of the buildings.

Edge detection: In the case of a map, edge detection could be used to identify the outlines of buildings and other structures in the image, which could then be used to determine their positions and shapes.

Feature detection: This involves identifying and extracting specific features from the map image, such as edges, corners, and lines, which can be used to determine the positions and shapes of buildings. One algorithm that can be used is ORB (Oriented FAST and Rotated BRIEF). ORB combines the FAST keypoint detector and BRIEF descriptor, with various modifications to improve performance. It first uses FAST to locate keypoints, then uses the Harris corner measure to select the top N points from those keypoints. Additionally, ORB uses a pyramid to generate multiscale features. As a descriptor ORB uses BRIEF adjusts the BRIEF descriptor based on the orientation of the keypoints, since BRIEF normally has a poor performance with orientation.

Optical character recognition (OCR): If the map includes text labels indicating the names or locations of buildings, OCR can be used to extract this information from the image.To begin an OCR project, you will scan and digitize physical documents, which the OCR software will convert into a binary format. The computer will then examine the scanned images, distinguishing between light and dark areas. Light areas will be identified as the background, while dark areas will be recognized as written characters that need to be deciphered. Afterwards, the computer will analyze the dark areas to locate alphabetic letters, numeric digits, and symbols. OCR programs use various methods to process text, but many focus on identifying one character, word, or block of text at a time. For our purpose OCR can be used to get the height information of buildings from the map.

3.1.2.2. 3D Modelling Technologies

Matplotlib: Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. By importing the mplot3d toolkit one can create a wide range of 3D plots, including scatter plots, surface plots, wireframe plots, and more. A downside of matplotlib is that it does not utilize GPU hardware acceleration, which can lead to

performance issues when working with large numbers of points or faces in 3D plots. These large amounts of data can make the plots slow and unresponsive[6].

Open3D: One of the most rapidly expanding 3D libraries for Python is a tool that is constructed using C++ but also has all of its features accessible through both C++ and Python. This makes it fast, adaptable, and dependable. The purpose of the library is to become a comprehensive solution for 3D processing, similar to how OpenCV is considered as an all-in-one solution for computer vision, including all necessary features and easy integration with other libraries[6].

Trimesh: Trimesh is a library that is completely written in Python and specializes in handling, analyzing, and displaying meshes and point clouds. It's often used to prepare 3D data for statistical analysis and machine learning tasks. Trimesh library doesn't have visualization components as a part of its base library, but it uses pyglet - a library for creating games and interactive applications as an optional dependency. It provides visualization functionality, interaction capabilities, and manipulation tools on top of pyglet. As a result, it provides a wide range of built-in visualization interactions by default[6].

V3do: Vedo (also known as V3do) is a python library that allows for the scientific analysis and visualization of 3-dimensional objects. It can be used for plotting 1d, 2d, and 3d data, point clouds, meshes, and volumetric visualization. It uses Trimesh as its backend for some of its processing, import/export and generation of mesh and point clouds and provides additional advanced functionality like creating vector illustrations that can be imported into LaTeX, generating physics simulations, deep learning layer overviews and mechanical and CAD-level slices and diagrams. To use Vedo, VTK and numpy should be installed before Vedo is installed[6].

PyVista: PyVista is a python library that simplifies the process of creating visualizations and analyzing meshes using VTK, a visualization toolkit. It provides an easy to use interface, making it more pythonic, and utilizes OpenGL for smooth visualizations and animations. PyVista supports both point clouds and meshes, and has a wide range of features including slicing, resampling, surface reconstruction, mesh smoothing, and ray-tracing, providing many examples and tutorials that cover simple visualizations to complex analysis[6].

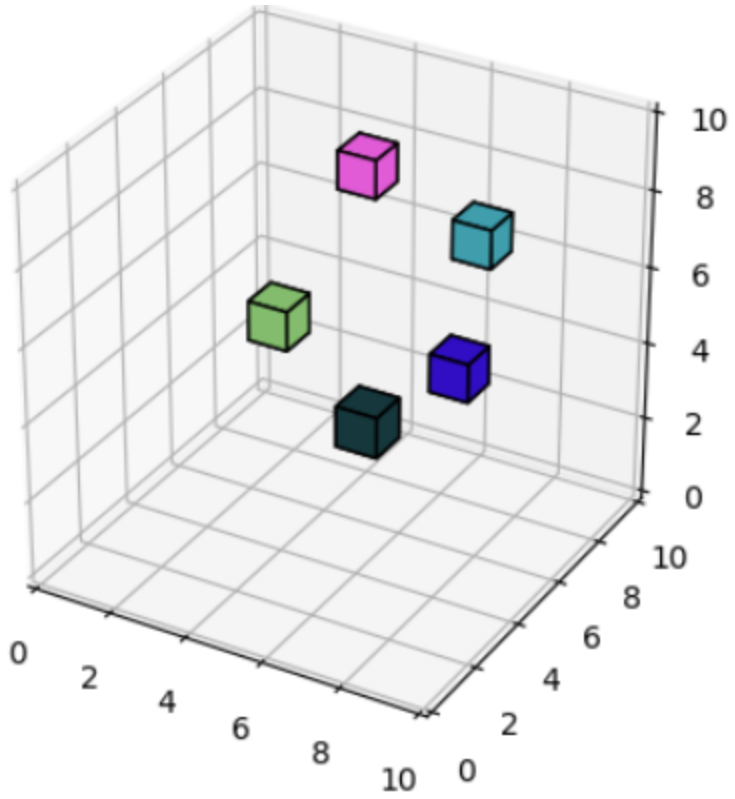


Figure 9: Visual representation of buildings in Matplotlib[8]

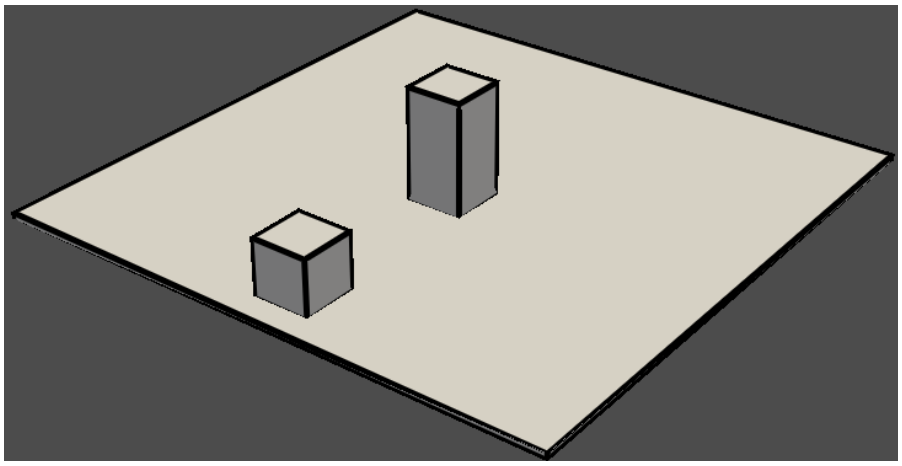


Figure 10: Basic 3D Output We Generated with PyVista [Code included in Appendix A]

In terms of software and hardware, a combination of computer hardware and software tools to implement the image processing tasks can be used. This may include hardware such as computers with CPUs and GPUs with sufficient processing power and memory to handle the data, as well as software such as Python, and libraries such as OpenCV and scikit-image . Database systems to store and manage the data generated by the image processing and machine learning tasks can be used.

3.1.3. Conceptualization

For the CMP students' part of the project, we considered several conceptual solutions for the image processing sub-system. Based on our literature review we considered three algorithms and two Python libraries as conceptual solutions. These considerations are compared in the tables 11 and 12 in the table below:

Image Processing Method	Cost	Complexity	Performance	Features
Pixel Reading Algorithm	Low	Low	Low	High
Feature Detection Algorithm	High	Low	High	High
Edge Detection Algorithm	Medium	Low	High	Medium

Figure 11: Image Processing Algorithms Comparison

Consideration	Matplotlib	PyVista
Cost	*Free*	*Free*
Complexity	*Moderate*	*High*
Performance	*Medium*	*High*
Features	*High*	*High*

Figure 12: 3D Modeling Library Comparison

3.1.3.1. Use Case Modeling

3.1.3.1.1 Actor Glossary

User: The person using the system to process the input image of the map.

Computer: The computer system running the image processing algorithms and generating the output data as 3D model and log file.

Robot: The crane takes the output log file from the computer as an input for itself.

3.1.3.1.2. Use-case Glossary

Add input: The user adds the input image of a map of buildings to the computer.

Process Image: The user selects an input image of a map of buildings and the system processes the image to extract information about the buildings and their locations. For image processing possible algorithms are pixel reading, feature detection or edge detection.

Read height information: The computer reads the height information which means recognition of the numbers given in the input map by using an OCR algorithm.

Generate 3D model: The computer generates a 3D model based on the information gathered by process image stage (Matplotlib or PyVista libraries are the options for this process).

Generate log file: The computer generates output data containing coordinate and height information for each building in the map, which will be used by the robot to build the scale model of the city.

Receive log file: The robot (crane) receives the log file outputted from the computer for using to build the buildings.

For our use-case modeling, we identified three main actor types: the user, the computer, and the robot. The user is responsible for inputting the image of the map of buildings and receiving the output data with coordinate and height information for each building. The computer is responsible for running the image processing algorithms and generating the output data. The robot is responsible for using the output data to physically construct the scale model of the city.

3.1.3.1.3. Use-Case Scenarios

We identified a use-case scenario:

- The user inputs an image of a map of buildings.
- The computer processes the image for building recognition (using either pixel reading, feature detection or edge detection).
- The computer applies OCR algorithm to recognize height information of the buildings.
- The computer generates 3D model of the input map with the information gathered from processing the image (using Matplotlib or PyVista libraries of Python).
- The computer generates output data with coordinate and height information for each building as a log file.
- The robot receives the log file and uses it to construct the scale model of the city.

We created use-case diagram for the corresponding scenario, as shown in Figure 13 below:

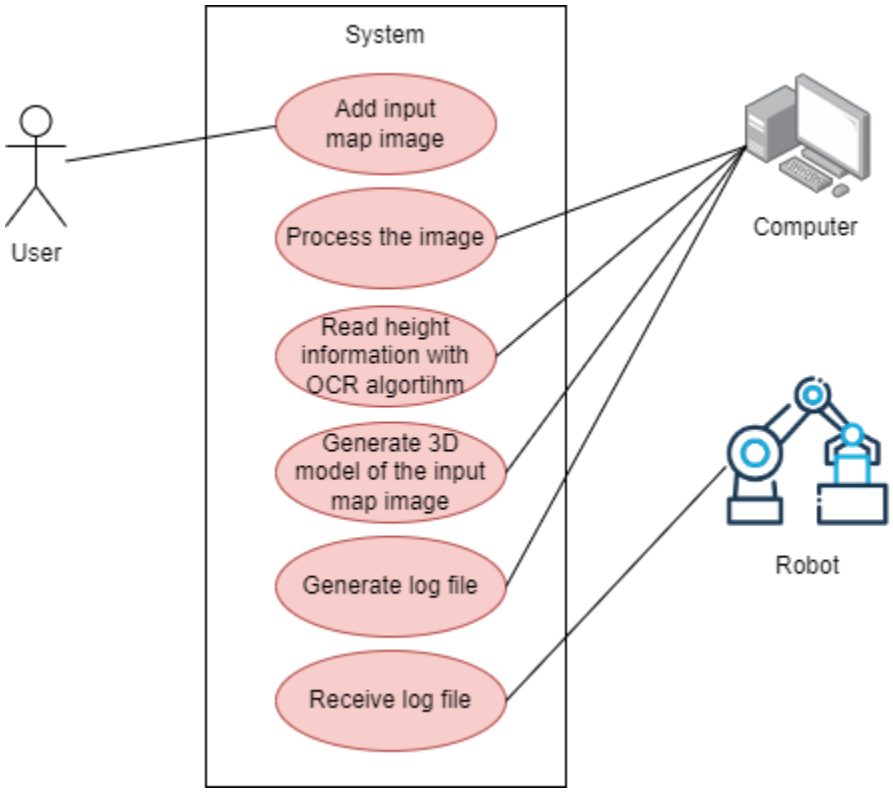


Figure: 13: Use-case Diagram 1

After considering the strengths and weaknesses of each option, we decided that the best solution for our project would be to use pixel reading for image processing and OCR for reading height information. For 3D modeling we decided to use PyVista based on its high performance. While it has the lowest performance, it offers the lowest cost and complexity, and is able to efficiently identify and classify the buildings in the map with more accuracy.

3.1.4. Physical architecture

This software consists of three main parts: image processing, generating a 3D output model for the user and generating an output log file for the robot. Corresponding low-level diagram of the architecture of the computer sub-system is shown in Figure 14 below.

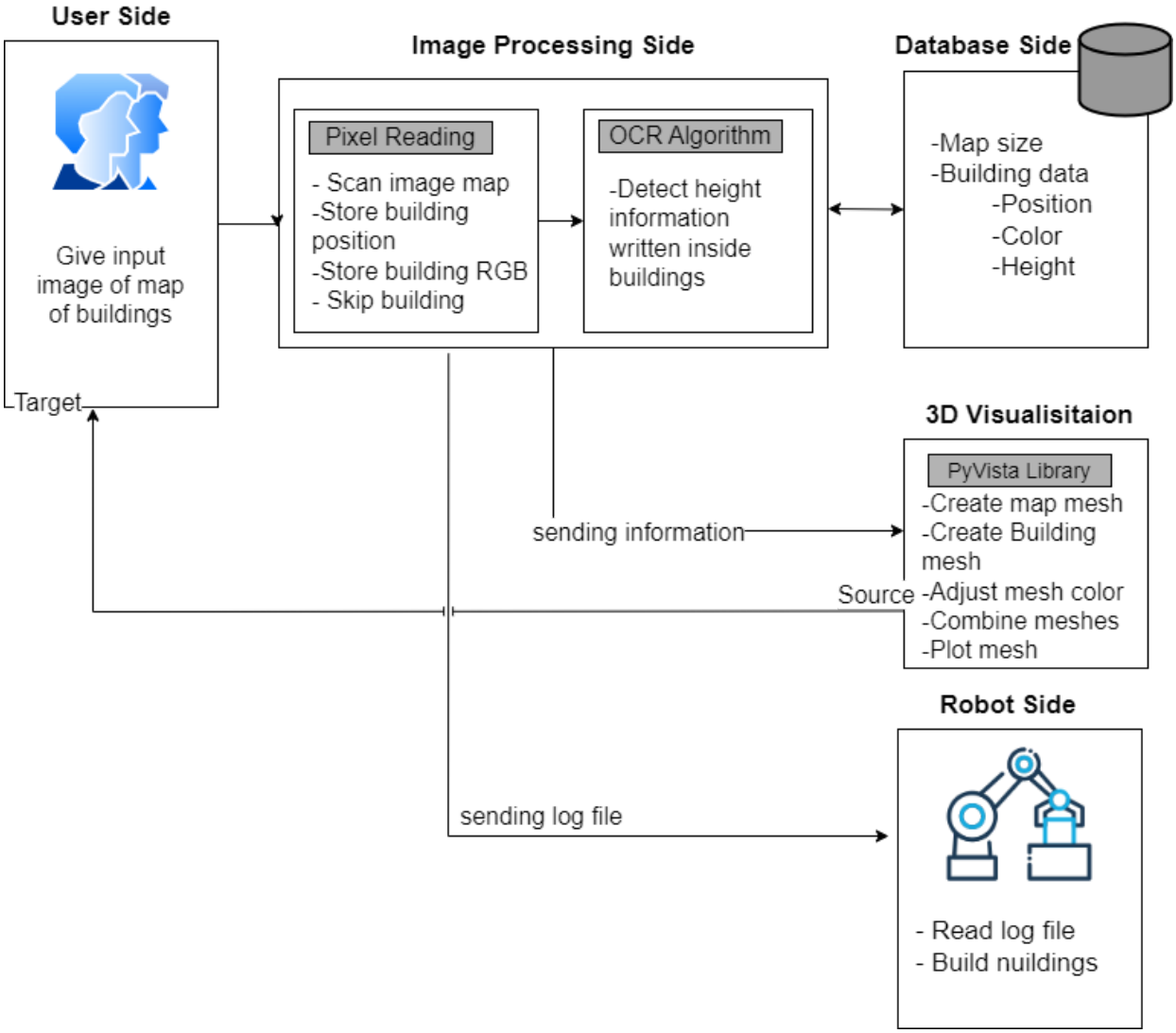


Figure 14: Low-level Software Architecture

User side consists of user operations: providing input to the software and receiving a 3D model of the map. Image processing side consists of application of pixel reading and OCR algorithm.

This part will be explained in the implementation section with more details but basically with pixel reading, position and color information of the building is extracted and stored while with OCR algorithm height information of the building is extracted and stored. Database is to store corresponding data gathered by pixel reading and OCR. Image processing side retrieves the data from the database when necessary such as creating the log file and 3D modeling. Log file is sent to the robot while the 3D model is targeted for the user.

3.1.5. Implementation

Implementation phase will start in the beginning of March. We will start with pixel reading and continue with the OCR algorithm. At the pixel reading stage, we will use the Python Imaging Library (PIL) to read each pixel and save RGB values. Pixel reading is explained step by step:

1. From the selected corner such as top left, we will start reading pixels going to the right for each row of pixels.
2. Once we detect the pixel is different from white, we know there is a building.
3. We will get the RGB value of the pixel to save the color information.
4. Once we know the top left edge position of a building, we will know to skip those X-axises for the next row readings, for the amount of rows equal to the building block's size, to not make duplicate registrations for the same building.
5. This is done through the whole map to find every building.
6. In the end, we have read and stored the position, color and height information for each building.

After starting the implementation stage and experimenting with PIL, we have noticed some irregularities with the expected results. During pixel reading, certain function results were completely accurate and so for those functions we wrongly thought they were complete. However, we later on realized after inputting certain different maps, the results would be corrupt. The problem with PIL was while it was good at differentiating pixel colors, the color

detection results wouldn't be completely accurate. Certain parts of the map background would output as a shade of grayish white. But since we needed better color accuracy with the whites, we tried switching to another library and found out that the OpenCV library worked flawlessly in this regard and therefore made the switch permanently. We also found out that the PNG file format works better since with JPEG there could sometimes be pixelation and defects in color information while we were generating new maps in Adobe Photoshop. Until here we were able to complete the first 3 steps mentioned above.

When we were implementing for the goal in step 4 however, we noticed that instead of skipping pixels in our pixel reading, we should instead save all pixel values which correspond to buildings in an array since the pixels we would skip could potentially be useful for later calculations.

```
for j in range(0, image.shape[1]):
    for i in range (0,image.shape[0]):
        if not np.array_equal(image[j,i], [255,255,255]):
            buildingpos.append((i,j))
```

Figure 15: Finding Colored Pixels and Storing Into an Array

We then needed to process the colored pixels which represented buildings in order to find the starting coordinate of each building. After storing each colored pixel we could first find the building lengths.

```
for za in range (len(buildingpos)-1):
    buildingLen += 1
    if (abs(buildingpos[za+1][0]-buildingpos[za][0]) > 1):
        break
```

Figure 16: Calculating Building Lengths from Buildingpos Array

With the code shown in figure X, we found the widths of the buildings in order to use in the calculation of finding the next starting position of remaining buildings.

```
buildingstartpos = []
buildingstartpos.append(buildingpos[0])
for la in range (0, len(buildingpos)-1, counter-1):
    if (abs(buildingpos[la+1][1]-buildingpos[la][1]) > 1):
        buildingstartpos.append(buildingpos[la])
        print(buildingpos[la])
```

Figure 17: Detecting Unique Building Start Positions

In the code snippet given in Figure X, we detected the differences between each members of the array to find the instances where the member was existing in a new line. This would allow us to know if the member is a pixel of a new building's starting coordinate, or simply a next line of the current building.

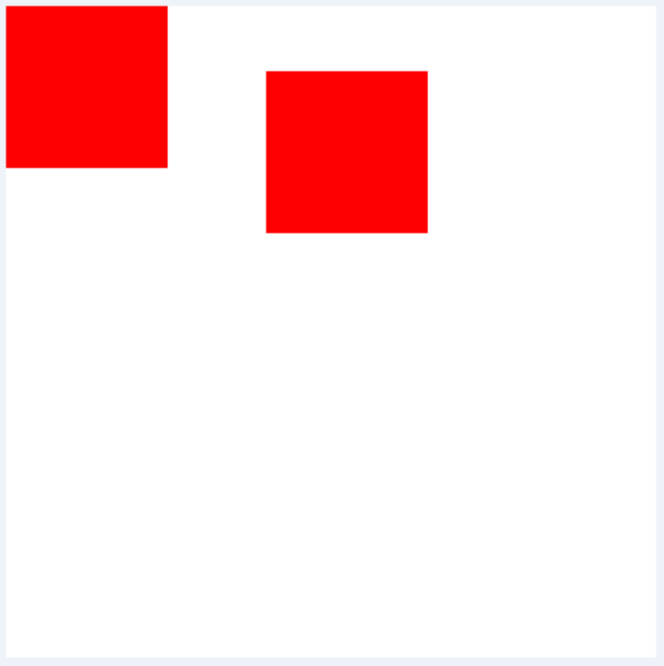


Figure 18: Basic Input Map for Testing with 2 Buildings

Processing the map in Figure X, we could eventually find the output result of (0,0) and (3,1) as building start positions. Note that the reference point of each building is its top left side for the coordinate system we implemented. We later on noticed that our simple check would fail when

the first building from the top would be on the right side of the second building and they were overlapping on the Y axis.

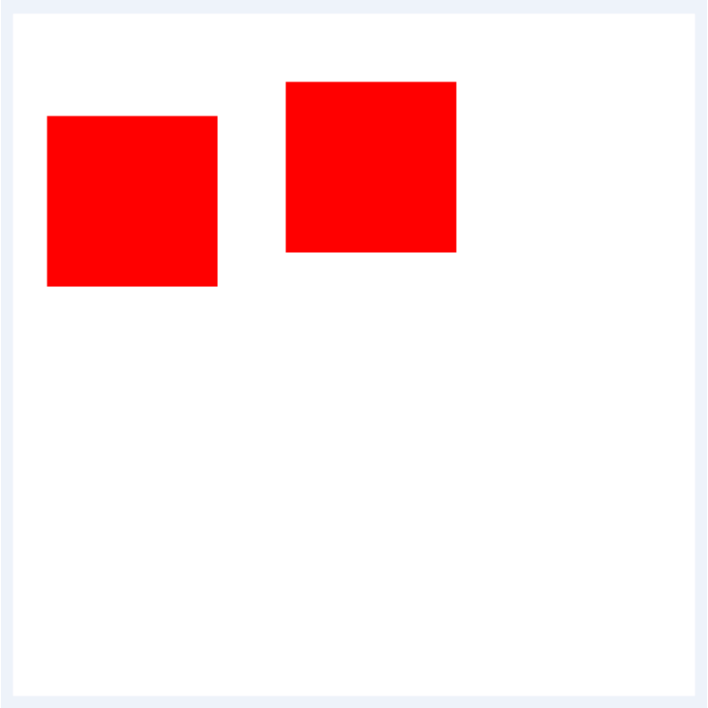


Figure 19: Building Arrangement Causing the Problem

Due to getting errors in output in such cases as Figure X, we checked for additional conditions.

```
buildingstartpos = []
buildingstartpos.append(buildingpos[0])

prevBuilding = buildingpos[buildingLen]
firstRun = True
for la in range (buildingLen, len(buildingpos)-1, buildingLen):
    if (abs(buildingpos[la][1]-buildingpos[la-1][1]) == 0):
        if (firstRun):
            if(buildingpos[la][1]-buildingpos[0][1] != 1):
                buildingstartpos.append(buildingpos[la])
                print(buildingpos[la])
                prevBuilding = buildingpos[la]
                firstRun = False
            else:
                buildingstartpos.append(buildingpos[la-buildingLen])
                print(buildingpos[la-buildingLen])
                prevBuilding = buildingpos[la]
                firstRun = False
        elif(prevBuilding[0]-buildingpos[la][0] !=  0 and prevBuilding[1]
        -buildingpos[la][1] != -1):
            if(buildingpos[la][1]-buildingpos[0][1] != 1):
                buildingstartpos.append(buildingpos[la])
                print(buildingpos[la])
                prevBuilding = buildingpos[la]
            else:
                buildingstartpos.append(buildingpos[la-buildingLen])
                print(buildingpos[la-buildingLen])
                prevBuilding = buildingpos[la]
```

Figure 20: Improvised Method for Detecting Unique Building Start Positions with Extra Conditions

After making these additional checks we were able to detect each building position for more scenarios, however we realized we needed to add too many checks to cover each possible scenario of building combinations. We came to the conclusion that our strategy to find conditions for each pattern is a very low level approach and definitely not a general solution. This is where we decided to stop and change our approach, because the code became more and more specific and impossible to catch every possible situation. We decided to switch to a more high level approach.

```
# Convert the image to grayscale
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

# Threshold the image to create a binary image
_, binary = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY_INV)

# Find contours in the binary image
contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL, cv2.
CHAIN_APPROX_SIMPLE)

# Iterate contours, append into building_info
building_info = []

for i in range (len(contours)-1,-1,-1):
    # Get the bounding rectangle for the contour
    x, y, w, h = cv2.boundingRect(contours[i])
    print('test1')
    print('')
    # Append the top-left corner of the bounding rectangle to the list
    building_info.append((x, y, w, h))
```

Figure 21: New High Level Method Using OpenCV Functions

After research and brainstorming we decided we could use the higher level functions of the OpenCV library to convert the image to a binary image. After doing this, we were able to find the contours in the image with OpenCV. With the contour results we could find each bounding rectangle for a given contour and find its starting coordinate as our result.

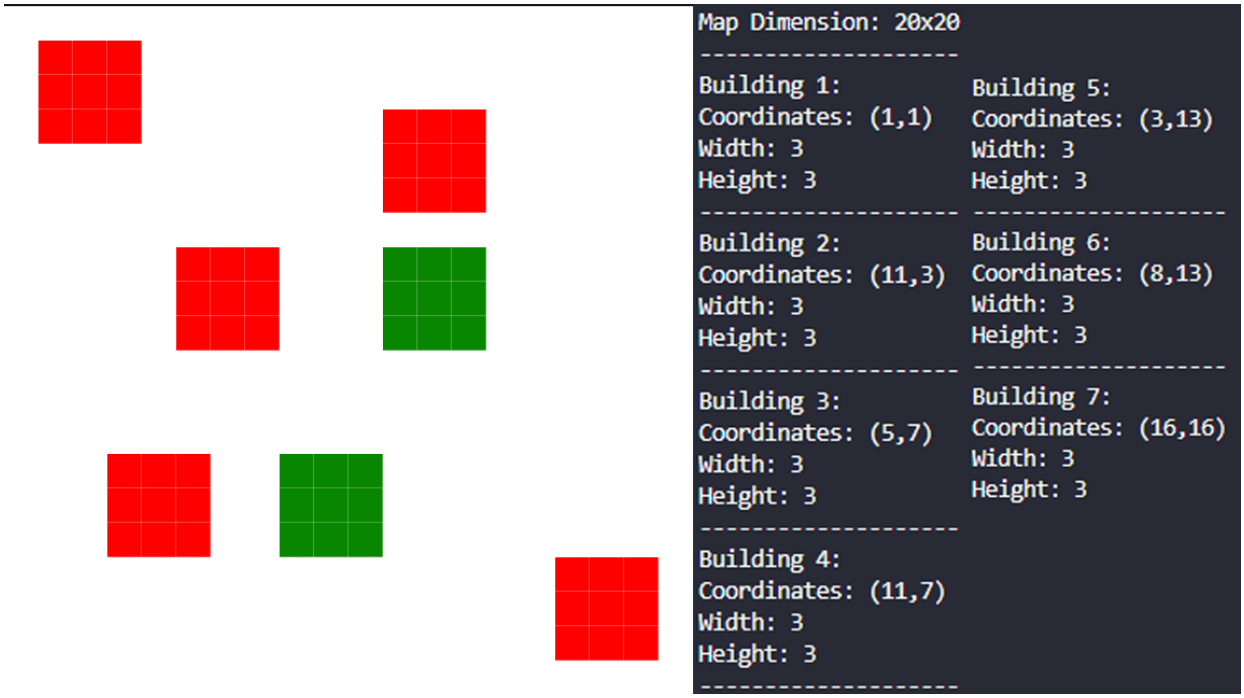


Figure 22: Desired Complexity Input and Successful Output

With the new improved method we have switched to, after inputting a map with the desired complexity, we have confirmed that we get correct results for each scenario.

OCR algorithm will use the position information obtained by pixel reading. Since we know all positions, we will simply go to each building position and read the corresponding number values in the center for each building to store their height information using OCR algorithm. Height information obtained by OCR will be saved as integer value in the database for later use. Output generation is less computational work compared to parsing image processes. First 3D model output will be generated by using PyVista library of Python. Necessary data will be retrieved from the database and created map and building mesh. Mesh color will be adjusted next and meshes will be combined. As a last step mesh will be plotted. Log file generation is the easiest stage among all. We will create a .txt file with position and height information. Possible outputs of each output generation stage are shown in Figure 15 and 16 below.

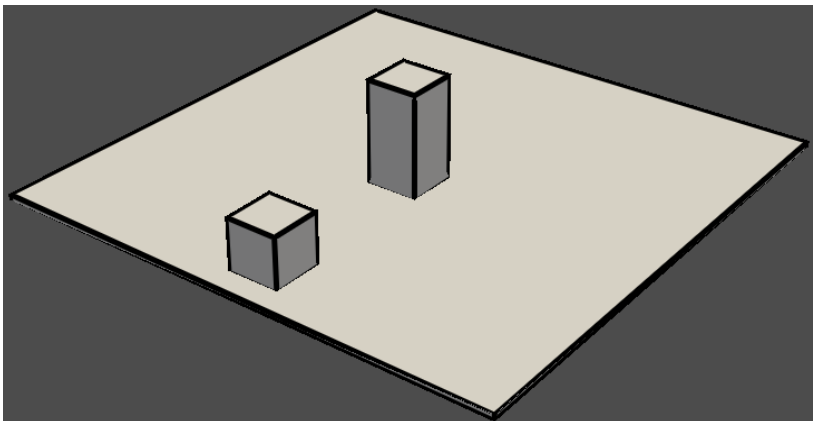


Figure 23: Sample of Output 3D Model

```
Building 1 - (50,255), 3
Building 2 - (45,70), 2
Building 3 - (65,70), 3
        .
        .
        .
```

Figure 24: Sample of Output Log File

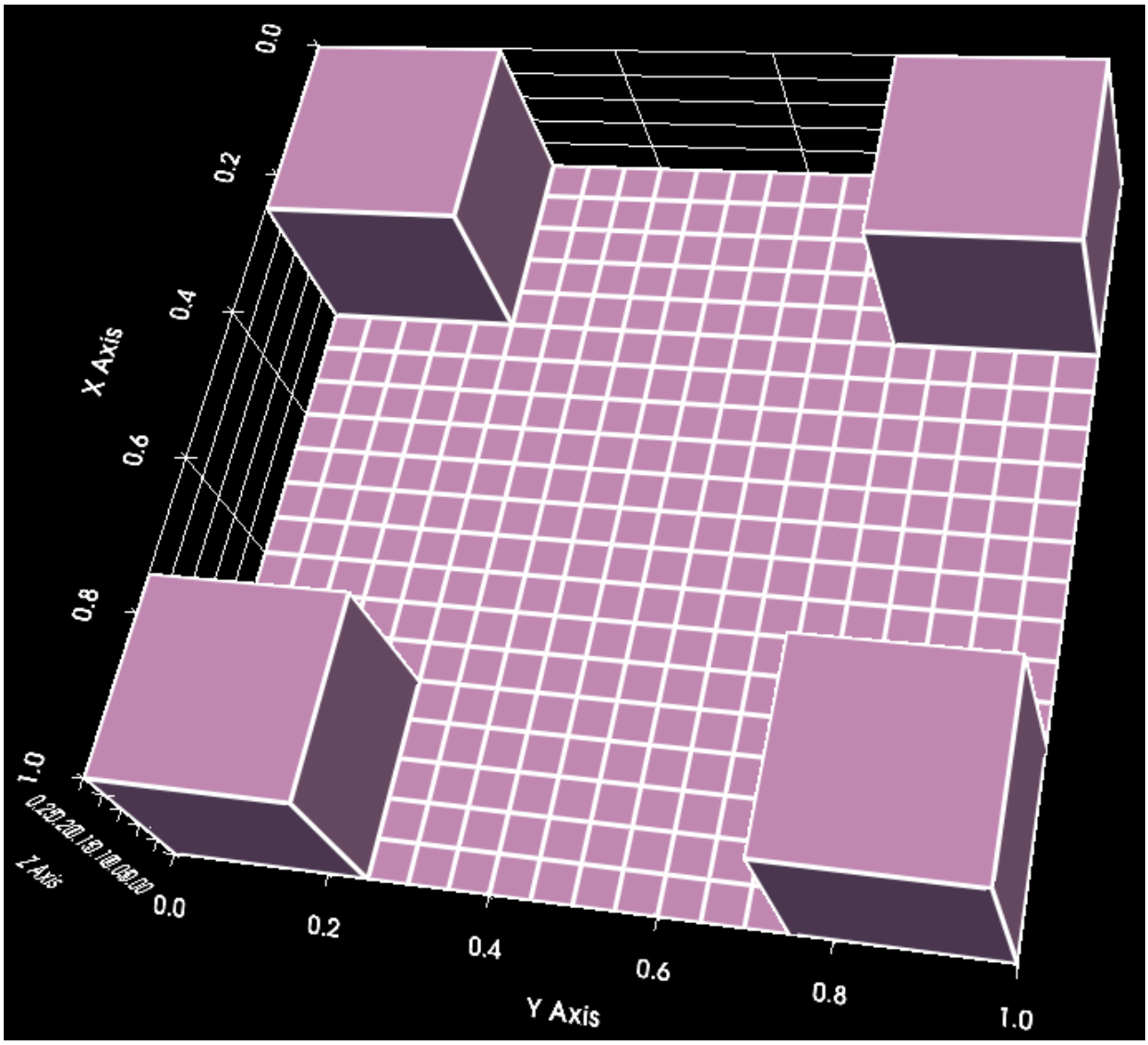


Figure 25: Earlier stage of 3D Visualization with Cubic Outputs

The screenshot above shows the earlier stages of 3D visualization. In this stage all the functions of the visualization were implemented and the only problem which needed solving was fixing a scaling issue which caused the buildings to misalign along the map, which was eventually fixed. Later on there was a switch from cubes to cylinders due to the new requirements of the crane robot mentioned by the Mechatronics sub-team.

Focusing on the implementation of the 3D implementation; for each building information stored in the building_info array, a cylinder mesh is created within PyVista with the appropriate position, color and height attributes. A label with text containing building information and a line reaching from each cylinder's top surface is appended on top of a cylinder in a hovering state.

```
building = pyvista.Cylinder(radius = building_width/2, height= building_height, direction
= (0,0,1), center=(x_coord,y_coord, building_height/2))
color = pos[4]
normalized_color = tuple([c/255 for c in color])  # normalize color values
# Add the building to the plotter with its color
plotter.add_mesh(building, color=normalized_color, show_edges=True, line_width=1,
smooth_shading= True, show_scalar_bar=False)

label_text = f"Building {counter}: \n({int(y_coord)}, {int(x_coord)}), \nH: {pos[3]}"
labels.append((label_text))
points.append((x_coord-building_width/6, y_coord+building_length/6, building_height*1.5))
line = pyvista.Line(pointa=(x_coord, y_coord, building_height), pointb=
(x_coord-building_width/6, y_coord+building_length/6, building_height*1.5), resolution=1)
plotter.add_mesh(line, show_scalar_bar=False)
```

Figure 26: Cylinder Mesh & Label Creation Implementation

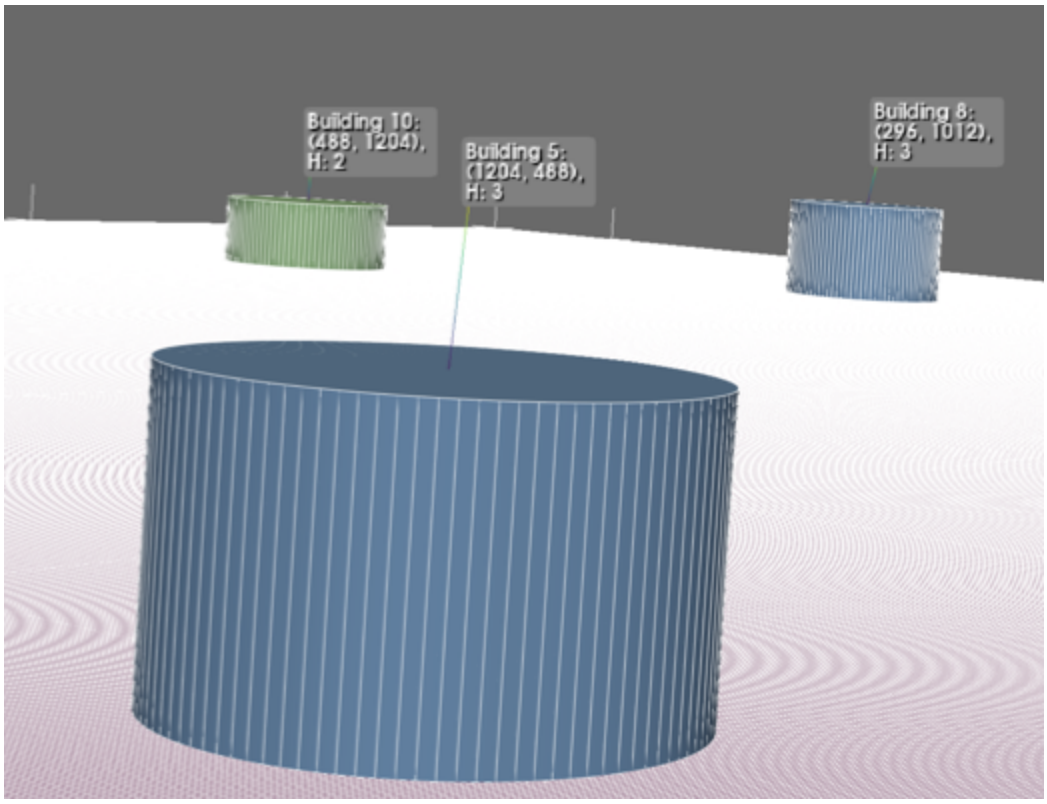


Figure 27: Cylinder Mesh & Labels in 3D Visualization

In the end the user is given two choices for 3D visualization type. They can choose between “Freeview" and “Rotation GIF”. In Freeview, a PyVista plotter window is opened and the user is able to dynamically view the map with the buildings, pan around, and zoom according to their preference. In Rotation GIF, a GIF which automatically revolves around the map in a full 360 degrees angle is created and saved to the project folder.

In order to get height information drawn inside buildings, we use OCR (Optical Character Recognition) and to perform this we use Tesseract. After downloading and installing Tesseract to our computer, we added full path of our Tesseract executable as shown in Figure below.

```
pytesseract.pytesseract.tesseract_cmd = r'C:\Program Files\Tesseract-OCR\tesseract.exe'
```

Figure 27: Adding Tesseract Executable To The Code

To make things easier for our calculation, we crop our input image at each building's position and then apply OCR. We add the result of OCR into the tuples that we store in building_info array. We repeat these steps for every building in the input map. The code is shown in Figure below:

```
for i in range (0,len(building_info),1):
    print('--------------------')
    print('Building ' + str(i+1) + ': ')
    print('Coordinates: (' + str(int(building_info[i][0])) + ',' + str(int
        (building_info[i][1])) +')')
    # Crop the image to the area of the building
    cropped = image[int(building_info[i][1]-building_info[i][2]/2):int
        (building_info[i][1]+building_info[i][2]/2), int(building_info[i][0]
        -building_info[i][2]/2):int(building_info[i][0]+building_info[i][2]/2)]
    #resized = cv2.resize(cropped, None, fx=20, fy=20, interpolation=cv2
        .INTER_CUBIC)
    #Recognize the text from the cropped image
    text = pytesseract.image_to_string(cropped, config='--psm 7 digits')
    #Add found text with OCR to building_info
    try:
        building_info[i].append(int(text))
    except ValueError:
        building_info[i].append(0)
```

Figure 28: OCR Implementation

As you see in the code psm is chosen as 7. PSM stands for Page Segmentation Mode. It is important for the accurate result of OCR. Since we try to OCR a single character at a time, psm 7 works well for us.

As mentioned before, to comply with the MCH sub-team's requirements, we switched to cylinders from boxes and made our inputs only have building's on the circumference of the circle that the crane's arm creates when rotated 365 degrees, so that the crane will be able to pick them up. The problem of the MCH sub-team was that their crane was not being able to properly carry the cubes. The cubes wobbled while being transported by crane, which could lead to collapse when adding new floors to the building. They tried with cylinders and the problem was no longer happening. As a result we needed to change cubes in our input images

and 3D visualization with cylinders. Also we were giving them the top left corner position of buildings but since cylinders do not have edges, we switched to center information.
Below in Figures, what we explained above is represented better.

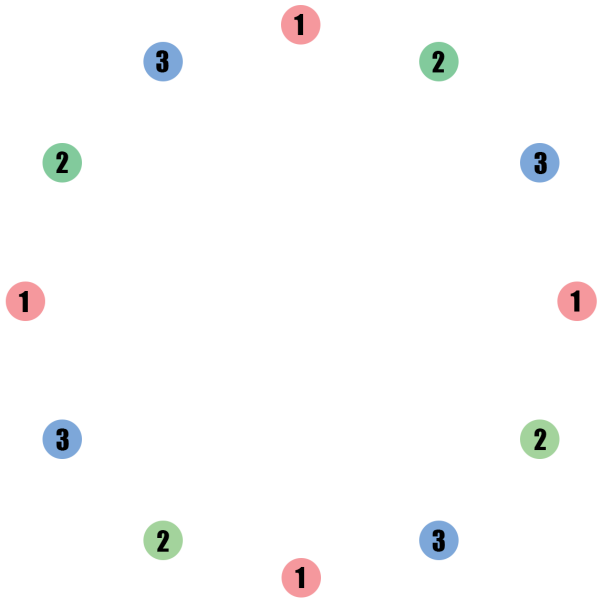


Figure 29: Buildings on Circle that is Created by 365 Degree Rotation of Crane's Arm

Since the crane is only able to construct a few buildings due to lack of enough cylinders a more realistic input map is shown in Figure below.

Figure 30: Another Circler Input Map Example

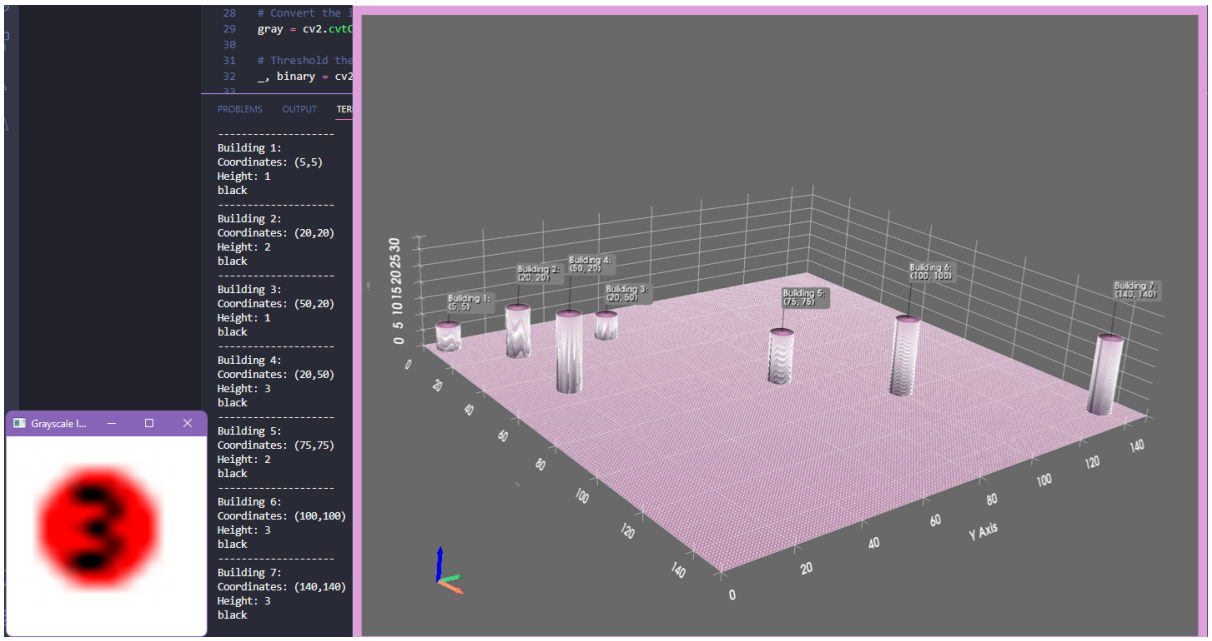


Figure 31: Example Text, 3D and OCR Outputs Side by Side

For getting the color information we move half of the diameter pixels from the center of each building. We were receiving the color information wrong however, we later found the reason and solved this issue. OpenCV uses BGR as color model but we did not know this and used the result as RGB color. As shown in Figure below, we manually convert the BGR value to RGB.

```
pixel_color_BGR = (image[int(building_info[i][1]), int(building_info[i][0]-building_info[i][2]/2.5)])
pixel_color = pixel_color_BGR[::-1] # reverse the order of channels, making it RGB
```

Figure 32: Converting Color Result From BGR to RGB

Moreover, the full code of the implementation of the CMP sub-system can we found in Appendix B at the end of the report.

3.1.6. Evaluation

In the evaluation step we will test our system with many different images of maps. In the 3D model we will also plot the corresponding numbers so that later we can also use this 3D model to verify that our image processing is done successfully. For validation of the model we will use accuracy. Corresponding equation is given in Figure 17 below.

$$Accuracy = \frac{Number\ of\ images\ read\ correctly}{Total\ number\ of\ images\ tested}$$

Figure 33: Accuracy Equation for our system

Functional Requirements	
The system must be able to evaluate a given map of buildings using image processing techniques.	Met
The system must be able to read building positions correctly from the map.	Met
The system must be able to read building height information correctly from the map.	Met
The system must be able to generate instructions for a robot to physically build a scale model of the city represented by the map.	Met
The system must be able to generate instructions that can be easily interpreted and executed by the robot.	Met
The system must be able to handle different heights of buildings accurately.	Met
The system must be able to generate instructions in a timely manner.	Met

Non-Functional Requirements	
The system must be able to accurately evaluate maps of buildings using image processing techniques with a high level	Met

of precision.	
The system must be able to generate instructions for the robot in a timely manner, with minimal delay.	Met
The system must be able to generate at least 2 buildings next to each other.	Met
The system must be able to handle different heights of buildings accurately and generate appropriate instructions for each.	Met
The system must be user-friendly and easy	Met

Figure 34: Requirements Table

3.2. Mechatronics Engineering (MCH)

The Mechatronics Engineering (MCH) sub-system is responsible for robotic crane design and production. Firstly, we drew the technical design for the robotic crane and made some corrections for the feedback and we placed 2 geared motors on technical design. The robotic crane has one main part and this part is on. Second part is the upper part and it moves specifics with (second) geared motors.

The most significant challenge we encountered during the 3D design process was the inability to find a suitable crane-like product for our drawings. However, once we moved past this stage and transitioned into product-based project design, our project started progressing. If we had attempted to proceed with designing and producing it using a 3D printer, the cost would have increased significantly. Therefore, once we transitioned into the product-based project phase,

our drawings remained purely representational. Apart from the drawings, we have acquired a prototype crane along with the motor and driver, and we are currently in the testing phase.

3.2.1. Requirements

The requirements for a robotic crane would depend on the specific application and task it is being used for. However, some common requirements for a robotic crane include:

1. **High precision and accuracy:** The robotic crane must be able to accurately position and move loads.
2. **Strong and durable construction:** The crane must be able to handle loads and operate in harsh environments.
3. **Safe operation:** The crane must be designed to operate safely and prevent accidents and injuries.
4. **Versatility:** The crane should be able to perform a variety of tasks, such as lifting, moving, and manipulating loads.
5. **Remote control and monitoring capability:** The crane should be able to be controlled remotely, and have monitoring capability to check the status of the crane, load and the surrounding environment.
6. **Compatibility with other equipment:** The crane should be able to be integrated with other equipment, such as other robots or automated systems, to improve efficiency and productivity.
7. **Scalability:** The crane should be able to be scaled up or down to meet different job requirements.
8. **Cost-effective:** The crane must not exceed the specified budget.

The performance requirements for a robotic crane will depend on the specific application and task it is being used for, but some common requirements include:

1. **Load capacity:** The crane should be able to lift and move loads safely and efficiently.
2. **Speed and precision:** The crane should be able to move quickly and precisely to position loads accurately.
3. **Repeatability:** The crane should be able to perform the same task multiple times with consistent results.
4. **Reliability:** The crane should be able to operate continuously and consistently without breaking down or malfunctioning.
5. **Flexibility:** The crane should be able to handle a variety of tasks, such as lifting, moving, and manipulating heavy loads.
6. **Autonomy:** The crane should be able to operate autonomously with minimal human supervision.

3.2.2. Technologies and methods

There are several technologies and methods that are commonly used in the development of robotic cranes.Robotics technology is used to design and control the movement of the robotic crane. This includes the use of sensors, actuators, and controllers to control the crane's movement and positioning. Artificial Intelligence (AI) techniques such as machine learning, deep learning and computer vision are used to enhance the crane's ability to sense and understand its environment, to adapt to different scenarios and to improve its performance.

3.2.3. Conceptualization

Conceptualization is the process of taking an idea and turning it into a concrete concept or plan. In the case of robotic cranes, conceptualization would involve taking the idea of a crane that is operated by robots and turning it into a detailed plan of what the crane will look like, how it will function, and what features and capabilities it will have.

This process would likely involve research and development in various areas such as robotics, computer vision, control systems, and power management. MCH Students and CMP Students would work together to create a detailed design of the crane, including its mechanical and electrical components, as well as its software and control systems.

The conceptualization process would also involve determining the specific requirements and specifications for the crane, such as its load capacity, range of motion, and safety features. This would involve analyzing the specific tasks and environments in which the crane will be used

and determining what capabilities and features are necessary for it to perform those tasks effectively.

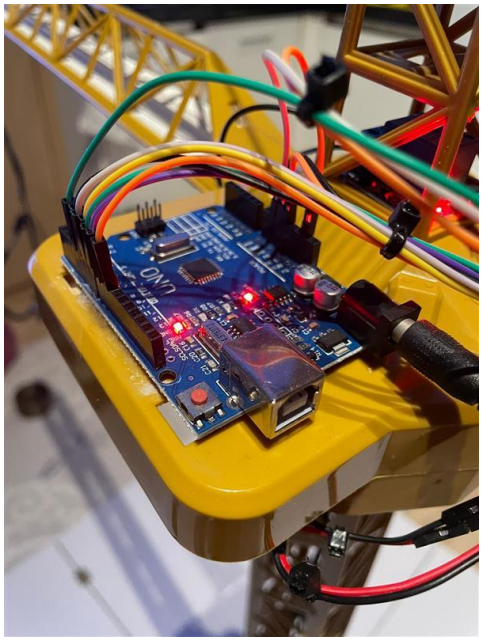
Once the conceptualization process is complete, the crane would then be built and tested, and any necessary adjustments would be made.

3.2.4. Physical architecture

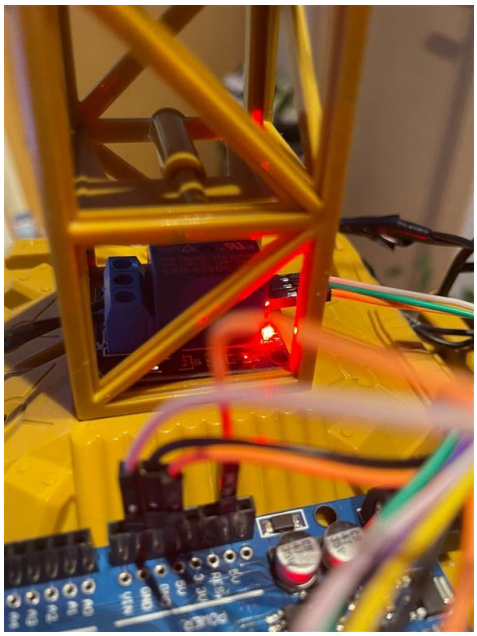
The physical architecture of a robotic crane refers to the design and layout of its mechanical and electrical components.

A typical robotic crane would have the following components:

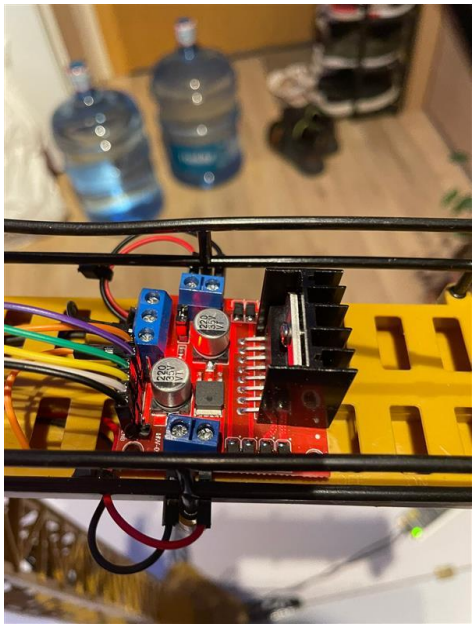
1. **Base:** The base of the crane is the foundation upon which the rest of the crane is built. It is typically a large, heavy-duty platform that provides stability and support for the crane's other components.
2. **Boom:** The boom is the arm of the crane that extends out from the base. It is typically made of strong and lightweight materials, such as aluminum or other metal, and is able to rotate and extend to reach different areas.
3. **Hoist:** The hoist is the mechanism that is used to lift and lower the load. It is typically made of steel or other strong materials.
4. **Control system:** The control system is the brain of the crane, it is responsible for controlling the movement and operation of the crane's various components. It includes sensors and controllers that work together to ensure the crane's movement is precise and accurate.
5. **Power source:** The power source is responsible for providing energy to the crane's various components. It can be an internal electric motor.
6. **Sensor:** An infrared distance sensor and an ultrasonic distance sensor help our crane operate precisely by providing the necessary measurements and alignment.
7. **Code:** The code we wrote to operate the Arduino and utilize the control systems of our designed crane.



Arduino and Arduino Connections



Relay



Motor Driver

DC Motor

Crane

3.2.5. Materialization

Materialization is the process of taking a concept or design and turning it into a physical product. In the case of robotic cranes, materialization would involve building the crane according to the design and plans developed during the conceptualization stage.

This process would involve sourcing and procuring the materials and components needed to build the crane, such as the boom, trolley, hoist, and control systems. The materials used in the construction of the crane would depend on the specific requirements and specifications of the crane, such as its load capacity and the environment in which it will be used.

Once the materials and components have been procured, the crane would be assembled and tested. The assembly process would involve putting together the various components of the crane, such as the boom, trolley, and hoist, according to the design and plans. The crane would then be tested to ensure that it meets the performance requirements and specifications.

After the crane is assembled and tested, it would go through a commissioning process, in which the crane's settings, parameters, and control systems are configured and tested to ensure that it is ready for operation.

3.2.6. Evaluation

Evaluation of a robotic crane is the process of assessing its performance and determining whether it meets the requirements and specifications set out during the conceptualization and materialization stages.

There are several key areas that should be evaluated when assessing a robotic crane:

1. **Load capacity:** The crane's ability to lift and move loads safely and efficiently should be evaluated.
2. **Speed and precision:** The crane's ability to move quickly and precisely to position loads accurately should be evaluated.
3. **Repeatability:** The crane's ability to perform the same task multiple times with consistent results should be evaluated.
4. **Range of motion:** The crane's ability to reach and move in a wide range of positions to access difficult-to-reach areas should be evaluated.
5. **Durability:** The crane's ability to operate in harsh environments and withstand heavy use should be evaluated.
6. **Reliability:** The crane's ability to operate continuously and consistently without breaking down or malfunctioning should be evaluated.
7. **Flexibility:** The crane's ability to handle a variety of tasks, such as lifting, moving, and manipulating loads should be evaluated.

8. **Autonomy:** The crane's ability to operate autonomously with minimal human supervision should be evaluated.

The Automatic Crane System aims to automate cranes, which play a significant role in the industrial transportation and logistics sector. In traditional crane systems, loads need to be manually handled or operated by an operator, resulting in a time-consuming, costly, and potentially risky process. The Automatic Crane System is designed to solve these issues by making transportation operations more efficient and safe. The system consists of components such as Arduino Uno microcontroller, HC-SR04 ultrasonic distance sensor, DC motors and motor driver, relay, and electromagnet. Python programming language and related libraries are utilized for tasks such as image processing, map creation, data communication, and more. Through image processing techniques, the system detects the position, size, and color of buildings in a designated area and generates a 3D map that aids in determining the crane's movement sequence and target locations. The HC-SR04 sensor is used for position determination, while the electromagnet and relay enable the system to pick up and place objects in specified positions. By automating the transportation process, the Automatic Crane System minimizes human error and time loss. Additionally, safety measures, including the HC-SR04 sensor and others, ensure safe working conditions and prevent potential hazards. The system is also highly customizable, allowing it to be tailored to different transportation requirements. The Automatic Crane System presents a significant potential in the industrial transportation and logistics sector, offering efficiency, safety, and flexibility by automating manual transportation processes. Through the seamless integration of components, the system aims to achieve the desired outcomes, highlighting its advantages and potential.

```
#include <Wire.h>

// Motor pins
#define ENA 10 // Motor 1
#define IN1 9
#define IN2 8
#define ENB 5 // Motor 2
#define IN3 7
#define IN4 6

#define IR1 2 // IR sensor 1 pin
#define IR2 3 // IR sensor 2 pin

#define OBJECT_HEIGHT 1
#define PICK_UP_TIME 7.75 // pick up time in seconds

#define ROTATION_TIME_LEFT 10.0 // left rotation time in seconds
#define ROTATION_TIME_RIGHT 11.0 // right rotation time in seconds

#define FULL_ROTATION 360 // Full rotation in degrees
#define SECTOR_SIZE 10 // Size of one sector in degrees

int motorSpeed1 = 150; // Motor 1 speed
int motorSpeed2 = 255; // Motor 2 speed

void setup() {
  pinMode(ENA, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);

  pinMode(ENB, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);

  pinMode(IR1, INPUT);
  pinMode(IR2, INPUT);

  // Initialize motor 2
  calibrateMotor2();
}
```

```
void loop() {
  // Define the target sectors
  int targets[3] = {12, 3, 3};
  // Define the number of objects at each target
  int objects[3] = {3, 1, 2};

  // Loop through the targets
  for (int i = 0; i < 3; i++) {
    moveToSector(targets[i]);
    for (int j = 0; j < objects[i]; j++) {
      pickUpObject();
      moveToSector(targets[i]);
      dropObject();
    }
  }
}

void moveToSector(int sector) {
  // Calculate the time it takes to move to the target sector
  float rotationTime = (ROTATION_TIME_RIGHT / FULL_ROTATION) * SECTOR_SIZE * (sector - 1);
  rotateRight(rotationTime);
}

void rotateRight(float duration) {
  analogWrite(ENA, motorSpeed1);
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  delay(duration * 1000);  // delay for the duration in ms
  analogWrite(ENA, 0);
}

void pickUpObject() {
  // Move the second motor down to pick up the object
  analogWrite(ENB, motorSpeed2);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  delay(PICK_UP_TIME * 1000); // delay for the duration in ms
  analogWrite(ENB, 0);
}
```

**We have added a part of the code that we wrote here, and we are still working on it.

4. INTEGRATION AND EVALUATION

Our product will consist of two main parts: software and the crane. The crane will build the buildings according to data it received from the software part. This is where our two sub-systems will get integrated. The crane must successfully receive the log file from the software and process the data accordingly.

As mentioned in the Gantt Chart in Figure 5, our integration phase will start in mid-april. It is foreseen that this phase along with its evaluation will take 2 weeks time.

4.1. Integration

The integration part comprises file sharing. The software must send the corresponding log file to the crane's microprocessor. And the crane's micro-processor, which is Arduino Uno, must properly receive the log file that contains building information.

File sharing will be done through serial communication on Arduino's serial port. The computer with the built python software will start a serial connection with Arduino. To do that we use the PySerial library. The .txt file that contains building information for the crane, is open and strings are encoded as utf-8 and sent through serial connection to Arduino. Received information will be read and the microprocessor will control the crane according to those received data.
For communicating our python code with the Arduino we wrote simple codes both in Python and Arduino as shown below.

```python
# create a serial object
ser = serial.Serial('COM5', 9600) # substitute 'COM3' with your Arduino's port
#time.sleep(2)  # give the connection a second or two to establish                #ENABLE AFTER DEVELOPING

# open text file in write mode to clear it
with open("output.txt", "w") as f:
    pass

# open text file in append mode
with open("output.txt", "a") as f:
    for building in robot_output:
        # convert building info to a comma-separated string
        str_out = ','.join(map(str, building))
        if ser.is_open:
            # encode the string to bytes and send it over serial
            ser.write((str_out + '\n').encode())
            # write the same data to the text file
            f.write(str_out + '\n')
        else:
            print("Serial connection is not open!")

# close the serial connection
ser.close()
```

Figure 35: Python Code for Serial Communication

```arduino
void setup() {
  Serial.begin(9600);
}

void loop(){
  if (Serial.available()) {
    Serial.println("data has been entered");
    string x = Serial.readString();

  }
  Serial.println(x +" ");
}
```

Figure 36: Arduino Code for Serial Communication Trial

4.2. Evaluation

For evaluation of the finished product we will check if each functional and performance requirements we mentioned in section 1.2.1. is met. For functional and performance requirements validation we will apply the methods that we mentioned in details in 3.1.6. and 3.2.6. Sections. We apply tests after completion of each stage, so that we ensure we step forward without any bugs.

For evaluation of the integration we will use the 3D model output of the software. If the crane is not building the building to corresponding positions or building them in the wrong height we will know there is a problem. Since we evaluated that the system creates the log file properly and ensured that the system works before the integration, it only means something went wrong with the integration. If we do not complete necessary evaluation steps before integration we would not be able to catch if the problem occurred during integration or not.

The performance of the system will be calculated as in the equation given in Figure 1 below:

$$Accuracy = \frac{Number\ of\ buildings\ that\ built\ as\ desired}{Total\ number\ of\ building\ must\ be\ built}$$

Figure 37: Performance Equation of the System

5. SUMMARY AND CONCLUSION

In summary, this project aims to develop a system that can accurately construct a scale model of a city using a combination of image processing and robotics. The project is divided into two main sub-systems, one handled by computer engineering students, who focus on image

processing, and the other handled by Mechatronics engineering students, who focus on the design and production of the robotic crane. The project aims to use pixel reading for image processing, PyVista for 3D output modeling, and a combination of robotic crane and mobile robot for the constructor robot. The project also aims to integrate the two sub-systems for a comprehensive solution to construct a scale model of a city.

Overall, this project demonstrates the potential of using image processing and robotic technologies for building physical models of cities and the importance of interdisciplinary collaboration in the design and implementation of such projects. It also serves as a proof of concept for future projects that aim to use image processing and robotic technologies in the field of urban planning and architecture.

In conclusion, this project will be a challenging but rewarding endeavor, requiring a high level of collaboration between computer engineering and Mechatronics engineering students. The project will be a great opportunity for students to apply their knowledge and skills in real-world scenarios, and to develop new ones through research and development. The end product will be a functional, cost-effective and high-performance system that can construct a scale model of a city. The project will be an excellent opportunity for the students to learn and work with the latest technologies and methods, as well as to gain valuable experience in project management and teamwork.

ACKNOWLEDGEMENTS

We wish to thank our advisors Dr. Selin Nacaklı & Dr. Ozan Akdoğan for their contributions throughout the semester.
This work was partially funded by Bahçeşehir University.

REFERENCES

1. Slocum, Alexander & Schena, Bruce. (1988). Blockbot: A robot to automate construction of cement block walls. Robotics and Autonomous Systems. 4. 111-129. DOI: 10.1016/0921-8890(88)90020-6.
2. Zakaria Dakhli & Zoubeir Lafhaj | Sanjay Kumar Shukla (Reviewing Editor) (2017) Robotic mechanical design for brick-laying automation, Cogent Engineering, 4:1, DOI: 10.1080/23311916.2017.1361600
3. Guyon, I. (2006). *Feature Extraction Foundations and applications*. Springer-Verlag.
4. Grauman, K., & Leibe, B. (2011). Visual object recognition. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, *5*(2), 1–181. DOI: 10.2200/s00332ed1v01y201103aim011
5. Chaudhuri, A., Mandaviya, K., Badelia, P., & Ghosh, S. K. (2016). Optical Character Recognition Systems. *Optical Character Recognition Systems for Different Languages with Soft Computing*, 9–41. DOI: 10.1007/978-3-319-50252-6_2
6. Nikolov, I. (2022, May 5). *Python libraries for mesh, Point Cloud, and Data Visualization (part 1)*. Medium. Retrieved January 11, 2023, from https://towardsdatascience.com/python-libraries-for-mesh-and-point-cloud-visualization-part-1-daa2af36de30
7. Hackeloeer, A., Klasing, K., Krisp, J. M., & Meng, L. (2014). Georeferencing: A review of methods and applications. *Annals of GIS*, *20*(1), 61–69. DOI: 10.1080/19475683.2013.868826
8. I. O. B. E. (2017, March 5). Representing voxels with Matplotlib. Stack Overflow. Retrieved January 11, 2023, from

https://stackoverflow.com/questions/42611342/representing-voxels-with-matplotlib

APPENDIX A

Code running the basic sample 3D Visualization using the PyVista library:

```python
import pyvista
import numpy as np


map = pyvista.Cube(x_length = 100, y_length= 100, z_length= 1)

building1 = pyvista.Cube(x_length = 10, y_length= 10, z_length= 20, center= (5,1,10))
building2 = pyvista.Cube(x_length = 10, y_length= 10, z_length= 10, center= (40,1,5))

merged = map.merge([building1,building2])
merged.plot(show_edges=True, line_width=5)
```

APPENDIX B

Full code of the CMP sub-system. You can reach all project files on the Github repository at https://github.com/utaysi/MapImage2Robot.

```
import os

import pyvista

import cv2

import numpy as np

import pytesseract

import webcolors
```

```
import serial

import time




pytesseract.pytesseract.tesseract_cmd = r'C:\Program
Files\Tesseract-OCR\tesseract.exe'


def closest_color(requested_color):

    min_colors = {}

    for key, name in webcolors.CSS3_HEX_TO_NAMES.items():

        r_c, g_c, b_c = webcolors.hex_to_rgb(key)

        rd = (r_c - requested_color[0]) ** 2

        gd = (g_c - requested_color[1]) ** 2

        bd = (b_c - requested_color[2]) ** 2

        min_colors[(rd + gd + bd)] = name

    return min_colors[min(min_colors.keys())]




# Load map image

project_folder = os.path.abspath(os.path.dirname(__file__))

output_folder = os.path.join(project_folder, 'output_images')

os.makedirs(output_folder, exist_ok=True)
```

```
image_path = os.path.join(project_folder, 'maps', 'map1.png')

image = cv2.imread(image_path)


# Convert the image to grayscale

gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


# Threshold the image to create a binary image. Ignore first return value
"threshold value" with _ placeholder, store second return value
"thresholded image" in "binary" variable.

_, binary = cv2.threshold(gray, 240, 255, cv2.THRESH_BINARY_INV)


# Find contours in the binary image

contours, _ = cv2.findContours(binary, cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)



# Iterate contours, append into building_info

building_info = []


for i in range (len(contours)-1,-1,-1):

    # Get the bounding rectangle for the contour

    x, y, w, h = cv2.boundingRect(contours[i])

    center_x = x + w/2

    center_y = y + h/2
```

```
    # Append the top-left corner of the bounding rectangle to the list
    building_info.append([int(round(center_x)), int(round(center_y)), w])



# Print the starting positions of buildings
print('-------------------------------------------------------------------
-----------------------------------------\n')
print('Map Dimension: ' + str(image.shape[0]) + 'x'+ str(image.shape[1]))
for i in range (0,len(building_info),1):
    print('--------------------')
    print('Building ' + str(i+1) + ': ')
    print('Coordinates: (' + str(int(building_info[i][0])) + ',' +
str(int(building_info[i][1])) +')')
    # Crop the image to the area of the building
    cropped =
image[int(building_info[i][1]-building_info[i][2]/2):int(building_info[i]
[1]+building_info[i][2]/2),
int(building_info[i][0]-building_info[i][2]/2):int(building_info[i][0]+bu
ilding_info[i][2]/2)]
    #Recognize the text from the cropped image
    text = pytesseract.image_to_string(cropped, config='--psm 7 digits')
    #Add found text with OCR to building_info
    try:
        building_info[i].append(int(text))
    except ValueError:
```

```
        building_info[i].append(0)


    print(f"Height: {building_info[i][3]}")


    filename = os.path.join(output_folder, f'building_{i+1}.png')
    cv2.imwrite(filename, cropped)


    pixel_color_BGR = (image[int(building_info[i][1]),
int(building_info[i][0]-building_info[i][2]/2.5)])
    pixel_color = pixel_color_BGR[::-1] # reverse the order of channels,
making it RGB
    building_info[i].append(pixel_color)
    print(pixel_color)
    print(closest_color(pixel_color))


#Create simplified robot output array
robot_output = []
for i in building_info:
    robot_output.append((int(i[0]), int(i[1]), i[3]))


# -------OUTPUT: TEXT FILE----------


# open text file in write mode to clear it
```

```
with open("output.txt", "w") as f:
    pass
# open text file in append mode
with open("output.txt", "a") as f:
    for building in robot_output:
        # convert building info to a comma-separated string
        str_out = ','.join(map(str, building))
        # write the same data to the text file
        f.write(str_out + '\n')

# -------OUTPUT: ARDUINO SERIAL CONNECTION----------
# Remove comments when Arduino Device is connected to the PC

# # create a serial object
# ser = serial.Serial('COM5', 9600)
# time.sleep(2)  # give the connection a second or two to establish.
# for building in robot_output:
#     # convert building info to a comma-separated string
#     str_out = ','.join(map(str, building))
#     if ser.is_open:
#         # encode the string to bytes and send it over serial
#         ser.write((str_out + '\n').encode())
```

```
#     else:
#         print("Serial connection is not open!")
# # close the serial connection
# ser.close()




print('-------------------------------------------------------------------
-----------------------------------------\n' + str(building_info) +
'\n')
print('-------------------------------------------------------------------
-----------------------------------------\n' + str(robot_output) +
'\n')




### PyVista 3D Visualization
theme = pyvista.themes.DefaultTheme()
theme.background = 'dimgrey'
theme.color = 'plum'
theme.edge_color = 'white'
theme.render_points_as_spheres = True
```

```
plotter = pyvista.Plotter(border= True, border_width= 50, border_color=
'plum', line_smoothing= True, polygon_smoothing= True, lighting= 'light
kit', theme= theme)

map = pyvista.Plane(center=(image.shape[0]/2, image.shape[1]/2, 0),
i_size=image.shape[0], j_size=image.shape[1],i_resolution=image.shape[0],
j_resolution=image.shape[1])



modelBuildings = []

labels = []

points = []

counter = 0

colors = []

for pos in building_info:

    counter += 1

    x_coord = pos[1]

    y_coord = pos[0]

    building_height = pos[3]*15

    building_width = pos[2]

    building_length = pos[2]



    building = pyvista.Cylinder(radius = building_width/2, height=
building_height, direction = (0,0,1), center=(x_coord,y_coord,
building_height/2))
```

```
    color = pos[4]

    normalized_color = tuple([c/255 for c in color])  # normalize color
values

    # Add the building to the plotter with its color

    plotter.add_mesh(building, color=normalized_color, show_edges=False,
line_width=1, smooth_shading= True, show_scalar_bar=False)



    label_text = f"Building {counter}: \n({int(y_coord)},
{int(x_coord)}), \nH: {pos[3]}"

    labels.append((label_text))

    points.append((x_coord-building_width/6, y_coord+building_length/6,
building_height*1.5))

    line = pyvista.Line(pointa=(x_coord, y_coord, building_height),
pointb=(x_coord-building_width/6, y_coord+building_length/6,
building_height*1.5), resolution=1)

    plotter.add_mesh(line, show_scalar_bar=False)







merged = map.merge(modelBuildings)



actor =
plotter.add_point_labels(points,labels,italic=False,font_size=10,point_co
lor='red',point_size=1,render_points_as_spheres=True,always_visible=True,
shadow=True)
```

```
plotter.enable_anti_aliasing('ssaa')

_ = plotter.add_axes(line_width=10, labels_off=True)

plotter.add_mesh(merged, show_edges=False, line_width=1, smooth_shading=
True, show_scalar_bar=False)

plotter.show_grid()


print('\nChoose 3D Visualization Type: \n1- Freeview \n2- Rotation GIF
(This takes around 30 seconds to render)')

userchoice = input()

if userchoice == '1':

    plotter.show(auto_close= True)

if userchoice == '2':

    # Open a gif

    plotter.open_gif("rotation.gif", fps=20)

    # Determine the center of your map for the camera to orbit

    orbit_center = map.center

    # Define a vector for the camera position. This will be updated in
the loop.

    orbit_radius = 2*max(image.shape[0], image.shape[1])  # use the
larger dimension of the image as the orbit radius

    camera_position = [orbit_center[0], orbit_center[1], orbit_center[2]
+ orbit_radius * np.sqrt(2) / 2]

    # Define the number of frames
```

```
    nframe = 360  # 36 frames will result in 10 degrees per frame

    # Rotate the camera and write a frame for each updated position

    for i in range(nframe):

        plotter.camera_position = [camera_position, orbit_center, [0, 0,
1]]  # sets camera position, focal point, and view up

        plotter.render()  # render the scene

        plotter.write_frame()  # write the frame

        # Update the camera position for the next frame. This creates a
10 degree rotation per frame.

        camera_position = [orbit_center[0] + orbit_radius *
np.sin(np.radians(i * 1)),

                           orbit_center[1] - orbit_radius * np.cos(np.radians(i
* 1)),

                           orbit_center[2] + orbit_radius * np.sqrt(2) / 2]

    # Close and finalize the gif

    plotter.close()
```