

PROJECT REPORT

(Project Semester Jan-July 2021)

ART GALLERY THEOREMS AND ALGORITHMS FOR ROBOTICS AND
LiDAR MAPPING APPLICATIONS (NTU Singapore Project)

&

DESIGN OF A HIGH EFFICACY FULLY AUTOMATIC FISH DEBONING
MACHINE(IIT Ropar iHub- AWaDH Project)

Submitted by-

Shivu Chauhan

101809031

3MTX-2

Under the guidance of-

Dr. R.K. Duvedi

Associate Professor

Dept of MEE Engg,

TIET,Patiala

Dr Cai Yiyu

Associate Professor

School of MAE,NTU

Dr.Dhiraj K. Mahajan

Associate Professor

Dept of MEE Engg,

IIT,Ropar



DEPARTMENT OF MECHANICAL ENGINEERING THAPAR INSTITUTE OF
ENGINEERING & TECHNOLOGY, PATIALA

(Declared as Deemed-to-be-University u/s 3 of the UGC Act, 1956)

DECLARATION
(NTU Singapore Project)

I hereby declare that work detailed here is an authentic record of my own work carried out at the School of Mechanical and Aerospace Engineering ,NTU as per the requirements of the project semester for the award of degree of B.E. (Mechatronics Engineering), Thapar Institute of Engineering and Technology, Patiala, under the guidance of Dr. Cai Yiyu, Associate Professor and Head of VR Lab, School of Mechanical and Aerospace Engineering , NTU and Dr R.K. Duvedi, Thapar Institute of Engineering and Technology

Shivu Chauhan

Shivu Chauhan

101809031

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Dr. R.K. Duvedi

Associate Professor

Dept of MEE Engg,

TIET Patiala

Dr Cai Yiyu

Associate Professor

School of MAE,NTU

DECLARATION

(IIT Ropar iHub- AWaDH Project)

I hereby declare that work detailed here is an authentic record of my own work carried out at Agriculture and water technology Development Hub (iHub-AWaDH) , IIT Ropar as per the requirements of the project semester for the award of degree of B.E. (Mechatronics Engineering), Thapar Institute of Engineering and Technology, Patiala, under the guidance of Dr Dhiraj K Mahajan , Indian Institute of Technology Ropar and Dr R.K. Duvedi, Thapar Institute of Engineering and Technology

Shivu Chauhan

Shivu Chauhan

101809031

Certified that the above statement made by the student is correct to the best of our knowledge and belief.

Dr. R.K. Duvedi

Associate Professor

Dept of MEE Engg,

TIET, Patiala

Dr.Dhiraj K. Mahajan

Associate Professor

Dept of MEE Engg,

IIT,Ropar

ACKNOWLEDGEMENT

I would like to express my sincerest and deepest thanks to Dr T.P. Singh, Professor and HOD of Mechanical department and my faculty mentor Dr R.K. Duvedi, Associate Professor, Thapar Institute of Engineering and Technology (TIET), Patiala, for providing me with an opportunity to undergo Industrial/Research training as a part of the curriculum.

The internship opportunity I had with NTU Singapore and at IIT Ropar was a great chance for learning and professional development which helped me learn new skills. Therefore, I am grateful that I got an opportunity to be a part of it, I am also grateful for having a chance to meet many wonderful people and professionals who led me through this internship period.

I am grateful to my Research mentor Dr Cai Yiyu, Associate Professor, School of MAE, NTU Singapore, for his supervision and critical assessment from time to time. I am indebted to him for providing with me all the necessary facilities and helping me in the project report and I also want to express my deepest gratitude to Dr. Dhiraj K Mahajan, Associate Professor, Dept of Mechanical Engineering and Overall in charge of iHub-AWaDH, IIT Ropar, for permitting me to undergo the internship under his guidance and his for constant guidance throughout the internship period.

ABSTRACT

NTU Singapore Project:

The tasks and projects assigned to me during my internship tenure included majorly developing opensource software packages and libraries in Python and C++ language in field of computational geometry specifically the famous Art Gallery Problem and its extensions.

Very less research has been done in these problems since its conception in 1972. I Initially spent first few weeks in learning python from basics syntaxes to Advanced level Data structures and Algorithms and Debugging through many online resources including the official documentation of the language.

I read several research papers , PhD thesis of NTU and NUS Alumni relating to Computational geometry, and conference proceedings of Computational geometry.

After several iterations I was guided by Dr. Yiyu to reduce the time and space complexity of the program developed to make the package adaptable across different machines.

I developed software packages for Art gallery theorems and its extension both in Python and C++ with a time complexity of $O(n \log n)$ and space complexity of $O(n)$. The software is available on my GitHub profile :

ABSTRACT

IIT Ropar iHub- AWaDH Project

In this internship I was part of a team of five students , from different colleges of India. Together we were assigned to Research and Develop a digital twin of a high efficiency and automatic fish deboning machine, a project relating the fisheries industry of India

Being a student of Mechatronics Engineering , I was assigned to design the electrical model of the machine that would run on 230V single phase AC Supply in fish farms, the Control system for speed control of the motors to be employed in the machines , a Human Machine interface that could specify whether the user wants Minced meat of fish or filleted fish as the output and an IoT enabled system which can inform the farmers of the output produced through a website on their smartphones.

I started by learning MATLAB Simulink and Simscape Electrical for designing the mechatronic systems of the machine, I completed several online tutorials available on MATLAB's documentations followed by YouTube lecture series.

I was able to design a stable 2 pole control system with positive gain and phase margins in Simulink for the speed control of Motors and I was able to replicate the electrical system in Simscape electrical and the results of Current consumption by specific motors was studied to determine the watt rating of the machine and the cost that would incur by running it for 6 hours a day.

TABLE OF CONTENTS

S. No.	Title	Pg. No.
1	Declaration	2 , 3
2	Acknowledgement	4
3	Abstract	5 , 6
4	PART A	
5	Company Profile	11
6	CHAPTER 1	12
	1.1 Objectives of Training	12
	1.2 Project Overview	12
6	CHAPTER 2 – Prerequisites & Literature Review	13
	2.1 Introduction	13
	2.2 Python, C++	13
	2.2.1 Data Structures	14,15
	2.2.2 Algorithms	15
	2.2.3 Complexity	16,17
	2.3 Literature Review of <i>Art Gallery theorems</i>	18,19
7	CHAPTER 3 – Programming the <i>Art Gallery Problem</i>	20
	3.1 Polygon Triangulations	20
	3.1.1 Ear Clipping Method	20,21
	3.1.2 Monotone Polygon Method	21,22
	3.2 Doubly Connected Edge List	22,23,24,25,26,27
	3.3 Fisk’s Approach with Three coloring theorem	28,29,30
	3.4 Solving the traditional <i>Art Gallery Problem</i>	30,31
8	CHAPTER 4 – <i>Art Gallery Problem with Holes</i>	32

	4.1 Literature Review	32
	4.2 Bridge Approach	32,33
	4.3 Algorithm for Holes	33,34,35
	4.4 Results	35,36
9	CHAPTER 5 - Conclusion	36
	5.1 Scope for Future Work	36
	5.2 Conclusion	36,37
	5.3 References	37,38,39
10	PART B	40
11	Company Profile	41
12	CHAPTER 1	42
13	1.1 Objective of Training	42
	1.2 Project Overview	42
14	CHAPTER 2 - Simscape	43
15	2.1 Simscape Electrical Design	43
	2.1.1 Full Wave Rectifier	43,44
	2.1.2 PWM & H Bridge Motor Controller	45,46,47,48
	2.1.3 Rating of Machine	48
16	CHAPTER 3 - Simulink Control System	49
17	3.1 Control system design	49,50
	3.2 Stability	50,51
	3.2.1 Root Locus	51,52
	3.2.2 Bode Plot	52,53
18	CHAPTER 4 - IoT & Human Machine Interface	54
19	4.1 IoT Circuit and Code	54,55
	4.2 Human Machine Interface	55
20	CHAPTER 5 - Conclusion	56
21	5.1 Conclusion	56
	5.2 References	56
22	APPENDIX	57-69

TABLE OF FIGURES

Fig. No.	Description	Page No.
1	Polygon and its Triangulation	18
2	Triangulation	20
3	Polygon Ear	21
4	Monotone Triangulation	22
5	Different types of Vertices	23
6	Helper Vertex	23
7	Monotone Partitioning	26
8	3-Colorability Matrix	27
9	Polygon Ear	28
10	Polygon Ear	28
11	3-Colorability Matrix	30
12	Python Implementations	31
13	Polygon with holes	32
14	Degenerate Polygon	33
15	Triangulated Polygon with holes	33
16	Bridge Implementation	35
17	Full wave Rectifier	43
18	230V AC Supply	44
19	DC Output	44
20	PWM & H-Bridge	45
21	Motor for Fin removal	46
22	Descaling, Head and tail removal	46
23	Gutting	46
24	Filleting	47
25	Deboning	47

26	Conveyor Belt motors	48
27	Control System	49
28	Controller Parameters	50
29	Root Locus Plot	51
30	Bode Plot	52
31	IoT Circuit	54
32	Human Machine Interface	55

COMPANY PROFILE

The School of Mechanical and Aerospace Engineering (MAE)) is ranked 5th in the world (QS Subject ranking 2021) and boasts well-established industry partnerships with prominent companies such as Rolls-Royce, General Electric, HP Inc, Pratt & Whitney and Singapore Airlines. With a faculty comprising more than 90 professors, it is one of the largest mechanical engineering schools in the world. Faculty members are drawn from renowned universities worldwide, providing a wealth of collective expertise in traditional and emerging mechanical and aerospace engineering, and in specialities including advanced manufacturing, mechatronics, innovative design, nanotechnology, and biomedical and computational applications.

MAE prides itself in its strong research capabilities in many areas including, advanced manufacturing, aerospace, biomedical, industrial engineering, maritime engineering, and robotics etc. In alignment with Singapore's national strategy (RIE 2020), the school of MAE is also rigorous in developing new competencies in manufacturing, aimed at supporting the competitiveness and growth of manufacturing and engineering sectors in the Singapore landscape.

CHAPTER 1

1.1 OBJECTIVES OF TRAINING

- To develop new research in the field of computational geometry by working on the existing techniques and enhancing their capabilities
- To opensource the software packages developed in the ‘Art Gallery theorems’ so that they can be used worldwide to carry out research
- To understand the role of an engineer in an engineering research.
- To gain experience of working in a professional environment and interacting with researchers.

1.2 PROJECT OVERVIEW

The project is based on creating packages and modules in python and C++ on the famous Art Gallery Problems and its extensions so as to develop novel techniques for their computation and analysis, Art gallery theorems are a set of geometrical mathematical algorithms based on the problem “*what is minimum number of guards that are required to be placed and at what locations so as to guard an art gallery*” “These problems are also used in Robotics engineering for motion planning of a robot , they are also used in Virtual reality application for placing of cameras or LiDAR sensors to develop a virtual environment by scanning cameras and its surroundings, and in Building infrastructure management techniques (BIM) for placing security guards in a building/floor/room or security cameras etc. This project starts by analysing the work done in art gallery them rems followed by the algorithmic complexity of the same, by the end of the project I developed python and C++ packages for art gallery problems and its extensions.

CHAPTER 2

2.1 INTRODUCTION

Computational geometry emerged from the field of algorithms design and analysis in the late 1970s. It has grown into a recognized discipline with its own journals, conferences, and a large community of active researchers. The success of the field as a research discipline can on the one hand be explained from the beauty of the problems studied and the solutions obtained, and, on the other hand, by the many application domains—computer graphics, geographic information systems (GIS), robotics, and others—in which geometric algorithms play a fundamental role. For many geometric problems the early algorithmic solutions were either slow or difficult to understand and implement. In recent years a number of new algorithmic techniques have been developed that improved and simplified many of the previous approaches. Programming languages like C++ has a huge support for computational geometry with its native library CGAL which has a huge number of inbuilt functions and header files for this field. In the recent years Python has also been used for computational geometry application through is Scikit-geom library and SciPy library.

2.2 Python , C++

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation. Its language constructs as well as its object-oriented approach aim to help programmers write clear, logical code for small and large-scale projects.

Python is dynamically-typed and garbage-collected. It supports multiple programming paradigms, including structured (particularly, procedural), object-oriented and functional programming. Python is often described as a "batteries included" language due to its comprehensive standard library.

Guido van Rossum began working on Python in the late 1980s, as a successor to the ABC programming language, and first released it in 1991 as Python 0.9.0. Python 2.0 was released in 2000 and introduced new features, such as list comprehensions and a garbage collection system using reference counting. Python 3.0 was released in 2008 and was a major revision of the language that is not

completely backward-compatible and much Python 2 code does not run unmodified on Python 3. Python 2 was discontinued with version 2.7.18 in 2020.

Python consistently ranks as one of the most popular programming languages

C++ is a general-purpose programming language created by Bjarne Stroustrup as an extension of the C programming language, or "C with Classes". The language has expanded significantly over time, and modern C++ now has object-oriented, generic, and functional features in addition to facilities for low-level memory manipulation.

C++ was designed with an orientation toward system programming and embedded, resource-constrained software and large systems, with performance, efficiency, and flexibility of use as its design highlights. C++ has also been found useful in many other contexts, with key strengths being software infrastructure and resource-constrained applications.

C++ is standardized by the International Organization for Standardization (ISO), with the latest standard version ratified and published by ISO in December 2020 as *ISO/IEC 14882:2020* (informally known as C++20). C++ was developed by Danish computer scientist Bjarne Stroustrup at Bell Labs since 1979 as an extension of the C language; he wanted an efficient and flexible language similar to C that also provided high-level features for program organization. Since 2012, C++ has been on a three-year release schedule, with C++23 as the next planned standard.

2.2.1 Data Structures

A data structure is a specialized format for organizing, processing, retrieving and storing data. There are several basic and advanced types of data structures, all designed to arrange data to suit a specific purpose. Data structures make it easy for users to access and work with the data they need in appropriate ways. Most importantly, data structures frame the organization of information so that machines and humans can better understand it.

In computer science and computer programming, a data structure may be selected or designed to store data for the purpose of using it with various algorithms. In some cases,

the algorithm's basic operations are tightly coupled to the data structure's design. Each data structure contains information about the data values, relationships between the data and -- in some cases -- functions that can be applied to the data.

For instance, in an object-oriented programming language, the data structure and its associated methods are bound together as part of a class definition. In non-object-oriented languages, there may be functions defined to work with the data structure, but they are not technically part of the data structure.

Typical base data types, such as integers or floating-point values, that are available in most computer programming languages are generally insufficient to capture the logical intent for data processing and use. Yet applications that ingest, manipulate and produce information must understand how data should be organized to simplify processing. Data structures bring together the data elements in a logical way and facilitate the effective use, persistence and sharing of data. They provide a formal model that describes the way the data elements are organized.

Data structures are the building blocks for more sophisticated applications. They are designed by composing data elements into a logical unit representing an abstract data type that has relevance to the algorithm or application. An example of an abstract data type is a "customer name" that is composed of the character strings for "first name," "middle name" and "last name."

It is not only important to use data structures, but it is also important to choose the proper data structure for each task. Choosing an ill-suited data structure could result in slow runtimes or unresponsive code. Five factors to consider when picking a data structure include the following:

1. What kind of information will be stored?
2. How will that information be used?
3. Where should data persist, or be kept, after it is created?
4. What is the best way to organize the data?
5. What aspects of memory and storage reservation management should be considered?

2.2.2 Algorithms

An algorithm is a set of instructions designed to perform a specific task. This can be a simple process, such as multiplying two numbers, or a complex operation, such as playing a compressed video file. Search engines use proprietary algorithms to display the most relevant results from their search index for specific queries.

In computer programming, algorithms are often created as functions. These functions serve as small programs that can be referenced by a larger program. For example, an image viewing application may include a library of functions that each use a custom algorithm to render different image file formats. An image editing program may contain algorithms designed to process image data. Examples of image processing algorithms include cropping, resizing, sharpening, blurring, red-eye reduction, and color enhancement.

In many cases, there are multiple ways to perform a specific operation within a software program. Therefore, programmers usually seek to create the most efficient algorithms possible. By using highly-efficient algorithms, developers can ensure their programs run as fast as possible and use minimal system resources. When we say a new version of a software program that has been "optimized" or has "faster performance," it most means the new version includes more efficient algorithms.

2.2.3 Complexity

We can express algorithmic complexity using the big-O notation. For a problem of size N :

- A constant-time function/method is “order 1” : $O(1)$
- A linear-time function/method is “order N ” : $O(N)$
- A quadratic-time function/method is “order N squared” : $O(N^2)$

Definition: Let g and f be functions from the set of natural numbers to itself. The function f is said to be $O(g)$ (read big-oh of g), if there is a constant $c > 0$ and a natural number n_0 such that $f(n) \leq cg(n)$ for all $n \geq n_0$

Abuse of notation: $f = O(g)$ does not mean $f \in O(g)$.

The Big-O Asymptotic Notation gives us the Upper Bound Idea, mathematically described below:

$f(n) = O(g(n))$ if there exists a positive integer n_0 and a positive constant c , such that $f(n) \leq c \cdot g(n) \quad \forall n \geq n_0$

The general step wise procedure for Big-O runtime analysis is as follows:

1. Figure out what the input is and what n represents.
2. Express the maximum number of operations, the algorithm performs in terms of n .
3. Eliminate all excluding the highest order terms.
4. Remove all the constant factors.

Some of the useful properties of Big-O notation analysis are as follow:

Constant Multiplication:

If $f(n) = c \cdot g(n)$, then $O(f(n)) = O(g(n))$; where c is a nonzero constant.

Polynomial Function:

If $f(n) = a_0 + a_1 \cdot n + a_2 \cdot n^2 + \dots + a_m \cdot n^m$, then $O(f(n)) = O(n^m)$.

Summation Function:

If $f(n) = f_1(n) + f_2(n) + \dots + f_m(n)$ and $f_i(n) \leq f_{i+1}(n) \quad \forall i=1, 2, \dots, m$.
then $O(f(n)) = O(\max(f_1(n), f_2(n), \dots, f_m(n)))$

Logarithmic Function:

If $f(n) = \log_a n$ and $g(n) = \log_b n$, then $O(f(n)) = O(g(n))$
; all log functions grow in the same manner in terms of Big-O.

Basically, this asymptotic notation is used to measure and compare the worst-case scenarios of algorithms theoretically. For any algorithm, the Big-O analysis should be straightforward as long as we correctly identify the operations that are dependent on n , the input size.

2.3 Literature Review of Art Gallery theorems

In 1973, Victor Klee posed the problem of determining the minimum number of guards sufficient to cover the interior of a wall art gallery room (Honsberger 1976). He posed this question extemporaneously in response to a request from Vasek Chvatal (at a conference at Stanford in August) for an interesting geometric problem, and Chvatal soon established what has become known as "Chvatal's Art Gallery Theorem" (or sometimes, "watchman theorem"): $\lfloor n/3 \rfloor$ guards are occasionally necessary and always sufficient to cover a polygon of n vertices (Chvatal 1975). This simple and beautiful theorem has since been extended by mathematicians in several directions, and has been further developed by computer scientists studying partitioning algorithms. Let P be a simple polygon with n vertices. Because P may be a complicated shape, it seems difficult to say anything about the number of cameras we need to guard P . Hence, we first decompose P into pieces that are easy to guard, namely triangles. We do this by drawing diagonals between pairs of vertices.

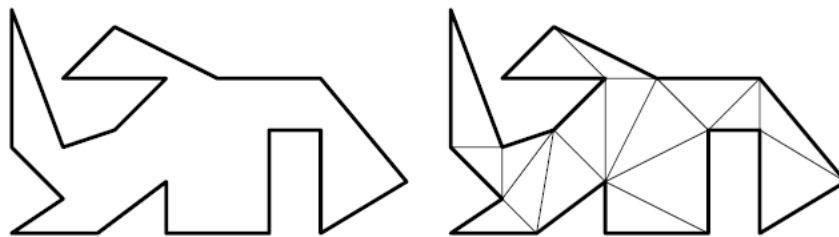


Figure 1: Polygon and its Triangulation

After triangulating the polygon, we assign each of the vertices of the triangle with a unique color this can be 3 coloring theorem. In a 3-coloring of a triangulated polygon, every triangle has a white, a Gray, and a black vertex. Hence, if we place cameras at all gray vertices, say, we have guarded the whole polygon. By choosing the smallest color class to place the cameras, we can guard P using at most $\lfloor n/3 \rfloor$ cameras. We can find a 3-coloring using a simple graph traversal, such as depth first search. While we do the depth first search, we maintain the following invariant: all vertices of the already encountered triangles have been colored white, gray, or black, and no two connected vertices have

received the same color. The invariant implies that we have computed a valid 3-coloring when all triangles have been encountered.

CHAPTER 3

3.1 Polygon Triangulations

In computational geometry, polygon triangulation is the decomposition of a polygonal area (simple polygon) P into a set of triangles, i.e., finding a set of triangles with pairwise non-intersecting interiors whose union is P .

Triangulations may be viewed as special cases of planar straight-line graphs. When there are no holes or added points, triangulations form maximal outerplanar graphs.

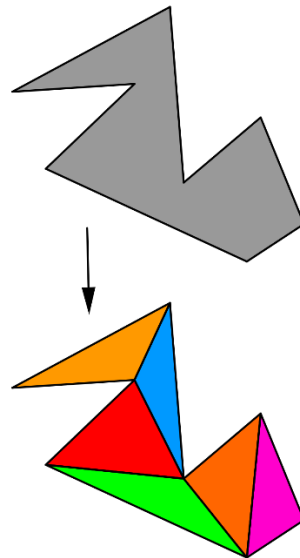


Figure 2: Triangulation

3.1.1 Ear Clipping Method

One way to triangulate a simple polygon is based on the two ears theorem, as the fact that any simple polygon with at least 4 vertices without holes has at least two 'ears', which are triangles with two sides being the edges of the polygon and the third one completely inside it. The algorithm then consists of finding such an ear, removing it from the polygon

(which results in a new polygon that still meets the conditions) and repeating until there is only one triangle left.

This algorithm is easy to implement, but slower than some other algorithms, and it only works on polygons without holes. This method is known as *ear clipping* and sometimes *ear trimming*. An efficient algorithm for cutting off ears was discovered by Hossam ElGindy, Hazel Everett, and Godfried Toussaint.

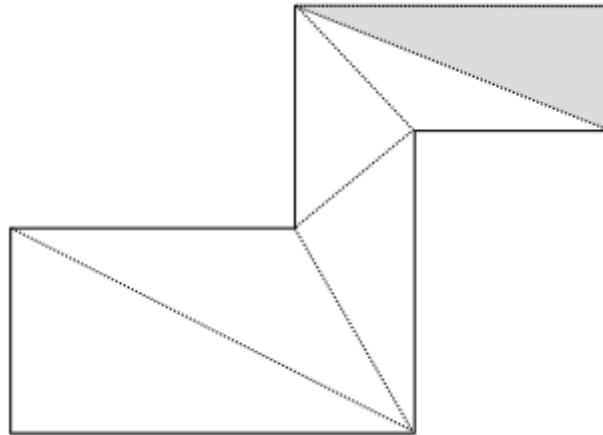


Figure 3: Polygon Ear

3.1.2 Monotone Polygon Triangulation Method

A simple polygon is monotone with respect to a line L , if any line orthogonal to L intersects the polygon at most twice. A monotone polygon can be split into two monotone *chains*. A polygon that is monotone with respect to the y -axis is called *y-monotone*. A monotone polygon with n vertices can be triangulated in $O(n)$ time. Assuming a given polygon is y -monotone, the greedy algorithm begins by walking on one chain of the polygon from top to bottom while adding diagonals whenever it is possible.^[1] It is easy to see that the algorithm can be applied to any monotone polygon.

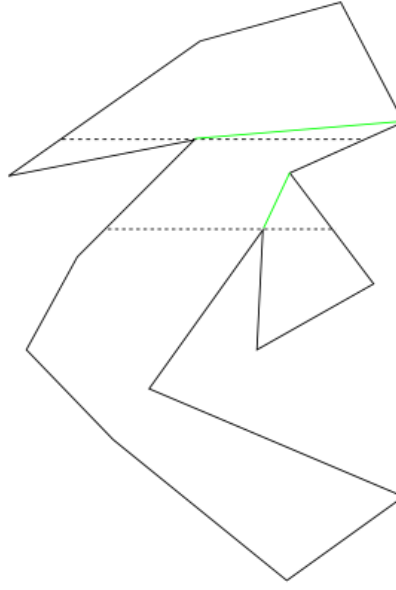


Figure 4: Monotone Triangulation

3.2 Doubly Connected Edge List

We solve the art gallery problem by first entering the information of the polygon in a DCEL structure. The polygon once entered will be partitioned into y-monotone pieces using plane sweep method. Then we will triangulate the monotone pieces obtained followed by 3 coloring to find the least number of guards in any arbitrary polygon. While implementing the plane sweep algorithm from top to bottom direction, we come across various kinds of vertices concave, convex, split. We need to turn specifically for all of them hence we use the algorithms defined for these turns.

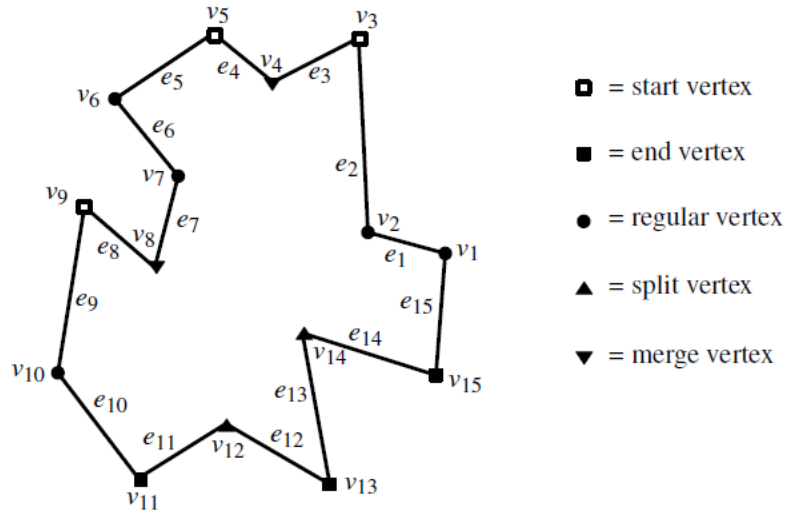


Figure 5: Different types of Vertices

We also define a helper(e_j) as the : lowest vertex v_i above the sweep line such that the horizontal segment connecting the vertex to e_j lies inside P . Here Let e_j is the edge immediately to the left of v_i .

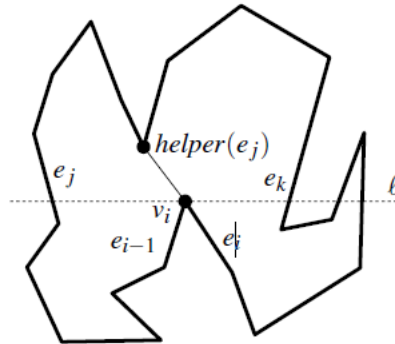


Figure 6: helper vertex

Algorithm MAKEMONOTONE(P):

Input. A simple polygon P stored in a doubly-connected edge list D.

Output. A partitioning of P into monotone subpolygons, stored in D.

1. Construct a priority queue Q on the vertices of P, using their y-coordinates as priority. If two points have the same y-coordinate, the one with smaller x-coordinate has higher priority.
2. Initialize an empty binary search tree T.
3. while Q is not empty
4. do Remove the vertex v_i with the highest priority from Q.
5. Call the appropriate procedure to handle the vertex, depending on its type.

HANDLESTARTVERTEX(v_i)

Insert e_i in T and set $\text{helper}(e_i)$ to v_i .

At the start vertex v_5 in the example figure, for instance, we insert e_5 into the tree T.

HANDLEENDVERTEX(v_i)

1. if $\text{helper}(e_{i-1})$ is a merge vertex
2. then Insert the diagonal connecting v_i to $\text{helper}(e_{i-1})$ in D.
3. Delete e_{i-1} from T.

HANDLESPLITVERTEX(v_i)

1. Search in T to find the edge e_j directly left of v_i .
2. Insert the diagonal connecting v_i to $\text{helper}(e_j)$ in D.
3. $\text{helper}(e_j) \leftarrow v_i$
4. Insert e_i in T and set $\text{helper}(e_i)$ to v_i .

HANDLEMERGEVERTEX(v_i)

1. if $\text{helper}(e_{i-1})$ is a merge vertex
2. then Insert the diagonal connecting v_i to $\text{helper}(e_{i-1})$ in D.
3. Delete e_{i-1} from T.
4. Search in T to find the edge e_j directly left of v_i .
5. if $\text{helper}(e_j)$ is a merge vertex

6. then Insert the diagonal connecting v_i to $\text{helper}(e_j)$ in D
7. $\text{helper}(e_j) \leftarrow v_i$

HANDLEREGULARVERTEX(v_i)

1. if the interior of P lies to the right of v_i
2. then if $\text{helper}(e_{i-1})$ is a merge vertex
3. then Insert the diagonal connecting v_i to $\text{helper}(e_{i-1})$ in D .
4. Delete e_{i-1} from T .
5. Insert e_i in T and set $\text{helper}(e_i)$ to v_i .
6. else Search in T to find the edge e_j directly left of v_i .
7. if $\text{helper}(e_j)$ is a merge vertex
8. then Insert the diagonal connecting v_i to $\text{helper}(e_j)$ in D .
9. $\text{helper}(e_j) \leftarrow v_i$

After partitioning the polygon into y-monotone pieces, we obtain a DCEL of y-monotone polygon, now we can triangulate this using a greedy algorithm.

Algorithm TRIANGULATEMONOTONEPOLYGON(P):

Input. A strictly y-monotone polygon P stored in a doubly-connected edge list D .

Output. A triangulation of P stored in the doubly-connected edge list D .

1. Merge the vertices on the left chain and the vertices on the right chain of P into one sequence, sorted on decreasing y-coordinate. If two vertices have the same y-coordinate, then the leftmost one comes first. Let u_1, \dots, u_n denote the sorted sequence.
2. Initialize an empty stack S , and push u_1 and u_2 onto it.
3. for $j \leftarrow 3$ to $n-1$
4. do if u_j and the vertex on top of S are on different chains
5. then Pop all vertices from S .
6. Insert into D a diagonal from u_j to each popped vertex, except the last one.
7. Push u_{j-1} and u_j onto S .
8. else Pop one vertex from S .

9. Pop the other vertices from S as long as the diagonals from u_j to them are inside P. Insert these diagonals into D. Push the last vertex that has been popped back onto S. 10. Push u_j onto S. 11. Add diagonals from u_n to all stack vertices except the first and the last one.

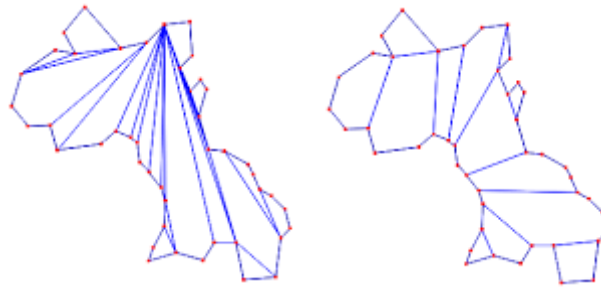


Figure 7: Monotone Partitioning

After triangulating the monotone polygon we use 3 coloring method of Computational graph theory on the vertices made available from which the 3-colorability of the graph obtained that is a triangulation of a simple polygon follows easily.

The graph 3-colorability problem is a decision problem (NP-Complete) in graph theory which asks if it is possible to assign a colour to each vertex of a given graph using at most three colors, satisfying the condition that every two adjacent vertices have different colors
Three coloring theorem:

The graph 3-colorability problem is a decision problem in graph theory which asks if it is possible to assign a color to each vertex of a given graph using at most three colors, satisfying the condition that every two adjacent vertices have different colors.

The Algorithm Used:

Require : $G = (V, E)$ an undirected Graph

$n \leftarrow |V|$

Give all vertices an index $1 \leq i \leq n$ that defines an order

for $i \in 1, \dots, n$ do

```

     $v_i$ .color  $\leftarrow$  0
end for
if  $n == 1$  then
    return
else
    for maxColors  $\in$  2..... $n$  do
        while G is not properly colored and not all vertices have color (maxcolors - 1)
        do
             $(v_1v_2....v_n) \leftarrow (v_1v_2....v_n) + 1$  (count up in base maxColor)
            end while
        end for
    end i

```

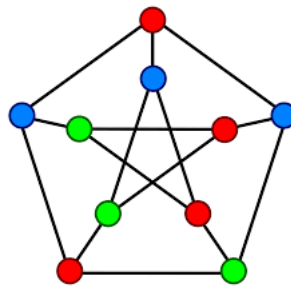


Figure 8:3 colorability

Through the minimum colors we get the minimum number of guards required for the gallery.

The complete implementation of the above Algorithm(DCEL):

<https://github.com/shivu926>

3.3 Fisk's Approach with Three coloring theorem

This approach uses Two ears theorem followed by 3 colourability to find the minimum number of guards for the gallery.

An ear is formed when: Three consecutive vertices v_{i-1}, v_i, v_{i+1} of P does form an ear if

1. v_i is a convex vertex;
2. the closure $C(v_{i-1}, v_i, v_{i+1})$ of the triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ does not contain any reflex vertex of P (polygon) (except possibly v_{i-1}, v_{i+1}). The closure of a triangle is formed by the union of its interior and its three boundary edges.

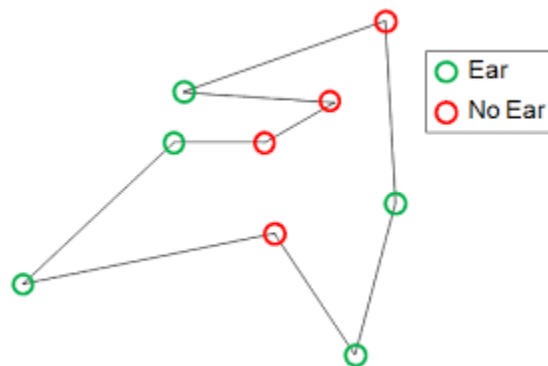


Figure 9: Polygon Ear

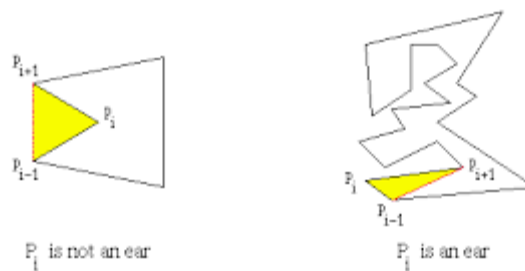


Figure 10: Polygon Ear

We use the following algorithm for triangulating by two ear theorem:

Algorithm: Triangulate Simple Polygon (P):

Input: A simple polygon P with n vertices $V(v_0, v_1, \dots, v_{n-1})$

Output: A triangulation T with $n-2$ triangles

- 1: Compute interior angles of each vertex in P.
- 2: Identify each vertex whether it is an ear tip or not.
- 3: while number of triangles in T < $n-2$ do
- 4: Find the ear tip v_i which has the smallest interior angle.
- 5: Construct a triangle $\Delta(v_{i-1}, v_i, v_{i+1})$ and add it onto T.
- 6: Let v_i be no longer an ear tip.
- 7: Update connection relationship of v_{i-1} and v_i , v_i and v_{i+1} , v_{i-1} and v_{i+1} .
- 8: Compute the interior angles of v_{i-1} and v_{i+1} .
- 9: Refresh the ear tip status of v_{i-1} and v_{i+1} .
- 10: end while

Three coloring theorem:

The graph 3-colorability problem is a decision problem in graph theory which asks if it is possible to assign a color to each vertex of a given graph using at most three colors, satisfying the condition that every two adjacent vertices have different colors.

The Algorithm Used:

Require : $G = (V, E)$ an undirected Graph

$n \leftarrow |V|$

Give all vertices an index $1 \leq i \leq n$ that defines an order

for $i \in 1, \dots, n$ do

$v_i.\text{color} \leftarrow 0$

end for

if $n == 1$ then

return

```

else
for maxColors  $\in$  2.....n do
    while G is not properly colored and not all vertices have color (maxcolors - 1)
    do
         $(v_1v_2....v_n) \leftarrow (v_1v_2....v_n) + 1$  (count up in base maxColor)
        end while
    end for
end i

```

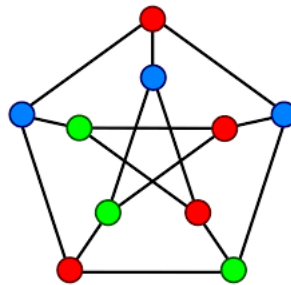


Figure 11: 3 colourability problem

3.4 Solving the traditional *Art Gallery Problem*

Fisk's approach with 3 colorability gives us the minimum number of guards required to guard any arbitrary shaped polygon.

```

1 from polygon_triangulate import *
2 from Graphcoloring import *
3
4
5 # Cartesian coordinates for the vertices of a CLOSED polygon, in either clockwise or counter-clockwise order
6 polygon = [(0,0),(1,0),(1,1),(2,4),(3,1),(4,1),(5,4),(6,1),(7,1),(8,4),(9,1),(10,1),(10,0)] * #n:=5
7
8 print(polygon) # the input
9
10 # Triangulate the polygon
11 triangles = Earclip(polygon) # Returns 2D list of triangle vertex coordinates
12
13 # Add ^those vertices to a dict, without repeating any, mapping each to a index from 0 to n-1,
14 vertices = {}
15 for x, y, z in triangles:
16     if x not in vertices: vertices[x] = len(vertices)
17     if y not in vertices: vertices[y] = len(vertices)
18     if z not in vertices: vertices[z] = len(vertices)
19
20 # initialize adjacency matrix of the triangle mesh to zeroes
21 num_vertices = len(vertices)
22
23 adjacency_matrix = [[0 for i in range(num_vertices)] for j in range(num_vertices)]
24
25 # populate adjacency matrix
26 for x, y, z in triangles:
27     adjacency_matrix[vertices.get(x)][vertices.get(y)] = 1
28     adjacency_matrix[vertices.get(x)][vertices.get(z)] = 1
29
30
31 [1, 0, 0, 0, 1, 0, 1, 1, 0, 0, 0, 1]
32 [0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0, 0]
33 [0, 0, 0, 0, 0, 1, 1, 0, 1, 0, 0, 1]
34 [0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 1, 1]
35 [0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0]
36 [0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1]
37 [1, 0, 0, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0]
38
39 Guards to placed at [(9, 1), (6, 1), (3, 1)]
40 [Finished in 0.5s]

```

Figure 12: Python Implementation

The complete implementation of the above Algorithm: <https://github.com/shivu926>

For any polygon with the given vertices entered in any order. Polygon = [(1,0) , (1,1) , (2,4) , (3,1) , (4,1) , (5,4) , (6,1) , (7,1) , (8,4) , (9,1) , (10,1) , (10,0)]

the guards will be placed at = [(1, 0), (8, 4), (5, 4), (2, 4)]

The Fisk's proof takes $O(n^2)$ time as compared to Monotone method that takes $O(n \log n)$ time.

CHAPTER 4

4.1 Literature Review

The basic algorithms(Two ears & 3-coloring) have been described above have been used .However, the two algorithms are only valid for simple polygons and cannot be directly applied on polygons with holes. Thus, a pre-processing of creating ‘Bridge’ edges must be done to transform the polygon with holes into a single polygon

4.2 Bridge Approach

Creating ‘Bridge’ edges is widely used to divide a general closed domain into several simply connected, convex sub domains, such as generating Delaunay triangulations or Voronoi diagram in multi-domain polygons. Different from dividing a closed domain, ‘Bridge’ edges to transform a multiply-connected polygonal area into a single polygon. The resulting polygon is not a simple polygon since each ‘Bridge’ edge appears twice with opposite orientations. These polygons are defined as ‘degenerate’ polygons. The algorithm of triangulating polygon with holes can be divided into two stages.

Stage 1: Create several ‘Bridge’ edges to transform a polygon with holes to a degenerate polygon without holes.

Stage 2: Triangulate the resulting polygon in Step 1 by ear clipping.

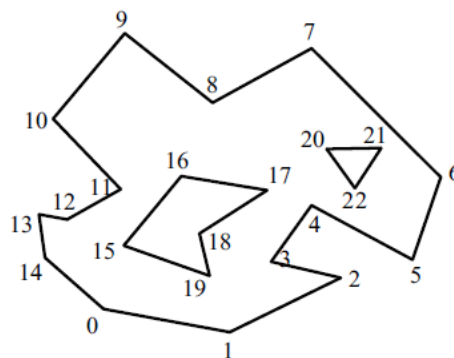


Figure 13: Polygon with Holes

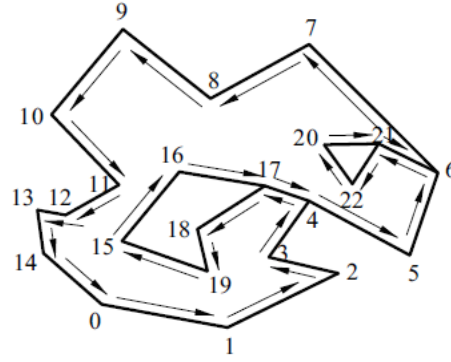


Figure 14: Degenerate Polygon

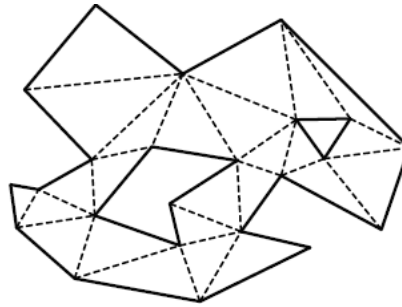


Figure 15: Triangulated Polygon with holes

4.3 Algorithm for Holes

Supposing there has a polygon P with m vertices $PV(pv_0, pv_1, \dots, pv_{m-1})$ which orientate in count-clockwise and a hole H with n vertices $HV(hv_0, hv_1, \dots, hv_{n-1})$ orientated in clockwise, the aim is to create 'Bridge' edges to combine polygon P and hole H into a new polygon P_{new} .

Firstly, we create $m \times n$ segments by a pair of vertices. In the pair of vertices, one is a vertex pvi from $PV(pv_0, pv_1, \dots, pv_{m-1})$ and the other is a vertex hvi from HV .

Secondly, we compute the length of each segment, and then select the shortest segment as the candidate 'Bridge' edge temporarily. We check whether there has been any edge

from either the polygon P or the hole H intersects the ‘Bridge’ edge, if not, the ‘Bridge’ edge is valid; otherwise, invalid and then test the next shortest segment.

These selecting and checking will repeat until a valid ‘Bridge’ is found.

Finally, we combine polygon P and hole H into a new polygon Pnew. Because the two vertices of the ‘Bridge’ edge are added twice, the new polygon Pnew has $(m+n+2)$ vertices.

After transforming a polygon with holes to a degenerate polygon, the proposed algorithms described above will be accepted to generate the final triangulation.

AlgorithmFindBridge:

- Pick an arbitrary vertex of the graph root and run depth first search from it.
- In the DFS, look through the edges starting from vertex v. The current edge (v,to) is a bridge if and only if none of the vertices to and its descendants in the DFS traversal tree has a back-edge to vertex v or any of its ancestors. Indeed, this condition means that there is no other way from v to to except for edge (v,to).
- We’ll use “time of entry into node” computed by the depth first search.
- Let $tin[v]$ denote entry time for node v. We introduce an array low which will let us check the fact for each vertex v. $low[v]$ is the minimum of $tin[v]$, the entry times $tin[p]$ for each node p that is connected to node v via a backedge(v,p) and the values of $low[to]$ for each vertex to which is a direct descendant of v in the DFS tree.
- Now, there is a back edge from vertex v or one of its descendants to one of its ancestors if and only if vertex v has a child to for which $low[to] \leq tin[v]$. If $low[to] == tin[v]$, the back edge comes directly to v otherwise it comes to one of the ancestors of v.
- Thus, the current edge (v,to) in the DFS tree is a bridge if and only if $low[to] > tin[v]$.
- $low[v] = \min(tin[v], tin[p], low[to])$

for all p for which (v,p) is a back-edge.

for all to for which (v,to) is a tree-edge.

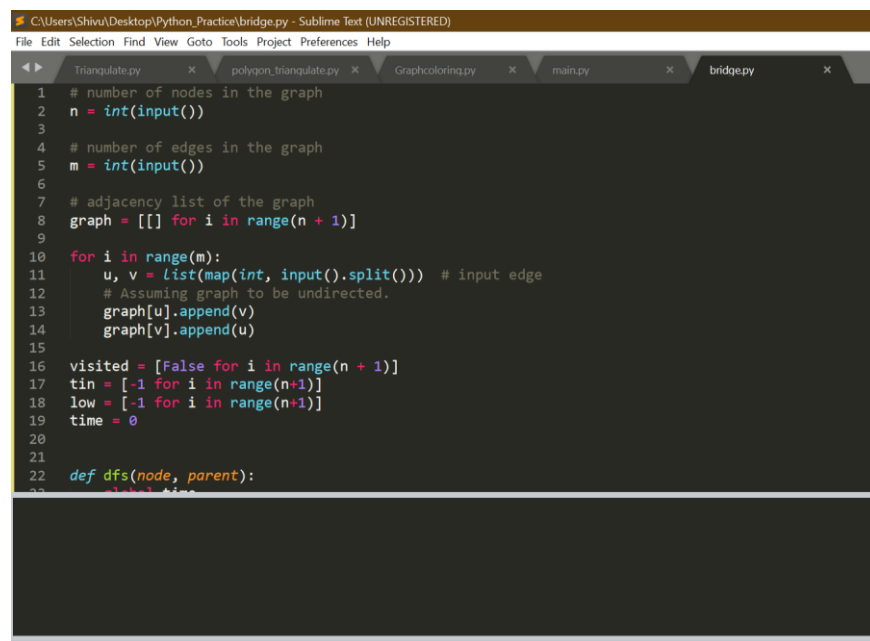
➤ The implementation needs to distinguish three cases: when we go down the edge in DFS tree, when we find a back edge to an ancestor of the vertex and when we return to a parent of the vertex.

➤ These are the cases:-

1. `visited[to] = False` - the edge is part of DFS tree.
2. `visited[to] = True` and `to != parent` - the edge is back edge to one of the ancestors.
3. `to = parent` - the edge leads back to parent.

4.4 Results

In this algorithm, the computation of interior angle for all vertices cost $O(n)$ time. And finding all ear tips need $O(n^2)$ time according to Condition 1, although this complexity can be decreased. In the while loop, the most expensive operation is to search the ear tip which has the smallest angle runs in $O(n)$ time. Thus, the while loop spends $O(n^2)$ time on cutting off all ears recursively. Considering this algorithm in whole, it runs in $O(n^2)$ time and $O(n)$ space



```
C:\Users\Shiva\Desktop\Python_Practice\bridge.py - Sublime Text (UNREGISTERED)
File Edit Selection Find View Goto Tools Project Preferences Help

1 # number of nodes in the graph
2 n = int(input())
3
4 # number of edges in the graph
5 m = int(input())
6
7 # adjacency list of the graph
8 graph = [[] for i in range(n + 1)]
9
10 for i in range(m):
11     u, v = list(map(int, input().split())) # input edge
12     # Assuming graph to be undirected.
13     graph[u].append(v)
14     graph[v].append(u)
15
16 visited = [False for i in range(n + 1)]
17 tin = [-1 for i in range(n+1)]
18 low = [-1 for i in range(n+1)]
19 time = 0
20
21
22 def dfs(node, parent):
23     # global time
24     time += 1
25     visited[node] = True
26     tin[node] = low[node] = time
27     for v in graph[node]:
28         if v == parent:
29             continue
30         if not visited[v]:
31             dfs(v, node)
32         low[node] = min(low[node], low[v])
33     if low[node] == tin[node]:
34         # node is an ear tip
35         # remove edges from node to its children
36         for v in graph[node]:
37             if v != parent:
38                 graph[v].remove(node)
39                 graph[node].remove(v)
```

Figure 16: Bridge Implementation

The complete implementation of the above Algorithm: <https://github.com/shivu926>

CHAPTER 5

5.1 Scope for future Work

Currently the packages developed does not offer transparency to the user, the computation is done own its on without informing the user of any steps involved, In future versions I can offer transparency for the same, Also the art gallery is not Graphically supported, the user will only know the coordinates of where to place the guards or camera, there is no GUI, I will be including graphical support to it using the Tkinter Python module and Graphics.h library for C++.

5.2 Conclusion

- Monotone Partitioning Method = $O(n \log n)$ time complexity
- Two Ears theorem/Fisk's approach = $O(n^2)$ time complexity

Which method to use?

- Art Gallery has no 'Perfect Solution!' so it depends on the application of theorems
- Monotone take less time but is not that accurate in results
- Two Ears/Fisk's theorem takes more time but is more accurate
 - For Robot Motion planning we want accuracy more hence Two Ears theorem/Fisk's Approach should be used
 - For LiDAR scanning for VR/AR Projects we want speed as latency is not desirable in these VR/AR Games and other application that use LiDAR hence we use Monotone Partitioning method for more speed over accuracy.
 - It's a trade-off between the two

I had a great experience of interning at this prestigious institution. Coming from a non-programmer background, I learned a lot about computer programming in the python language and the versatility this language has in domains like Artificial Intelligence,

Robotics, Finance, etc. The projects also helped me to learn to do a literature survey of prior work done in the field. My internship has allowed me to see so much more about the Mechatronics field than I can learn in the classroom. I have developed many skills and have a much greater concept of what to expect after college. My internship has given me a greater understanding of what it is I have learned in the classroom and allowed me to apply it to real situations. One of the greatest benefits that I have received is the knowledge that I enjoy doing what it is I have been studying for. Now I can comprehend Algorithms and program them in python using various data structures. My learnings throughout this project have been unprecedented where I was given an opportunity to explore a domain of Computer science that I had never previously worked on. The field of Computational geometry is relatively new and I feel lucky enough to be given exposure to this field during this internship. I will carry forward the knowledge and learning of Computational geometry that I gained for future projects. Publication of the above research are expected.

5.3 References

1. Berg, M., Cheong, O., Kreveld, M., & Mark Overmars, M. (2008). Computational Geometry Algorithms and Applications. Berlin Heidelberg: Springer-Verlag.
2. Chazelle, B. (1982). A theorem on polygon cutting with applications. 23rd Annual Symposium on Foundations of Computer Science, 339–349.
3. Chazelle, B. (1991). Triangulating a simple polygon in linear time. Discrete Comput. Geom. 6(5), 485–524.
4. ElGindy, H., Everett, H., & Toussaint, G. (1993). Slicing an ear using prune-and-search. Pattern Recogn. Lett. 14(9), 719–722.

5. Feng, H. -Y. F., & Pavlidis, T. (1975). Decomposition of polygons into simpler components: feature generation for syntactic pattern recognition. *IEEE Trans. Comput.* 24(6), 636–650.
6. <ftp://ftp.cis.uab.edu/pub/sloan/Software/triangulation/src/>
7. Garey, M. R., Johnson, D. S., Preparata, F. P., & Tarjan, R. E. (1978). Triangulating a simple polygon. *Inform. Process. Lett.* 7, 175–179.
8. Held, M. (2001). FIST: Fast industrial-strength triangulation of polygons. *Algorithmica*, 30, 563–596.
9. Kong, X., Everett, H., & Toussaint, G. (1990). The graham scan triangulates simple polygons. *Pattern Recogn. Lett.* 11(11), 713–716.
10. Meisters, G. H. (1975). Polygons have ears. *Amer. Math.* 82(6), 648–651.
11. O'Rourke, J. (1998). *Computational Geometry in C*. Cambridge: Cambridge University Press.
12. Seidel, R. (1991). A simple and fast incremental randomized algorithm for computing trapezoidal decompositions and for triangulating polygons. *Comput. Geom. Theory Appl.* 1(1), 51–64.

13. Tipper, J. C. (1991). FORTRAN programs to construct the planar Voronoi diagram. *Comput. Geosci.* 17(5), 597–632.
14. Toussaint, G. T. (1991). Efficient triangulation of simple polygons. *Vis. Comput.* 7, 280– 295.
15. Bjorlig-Sachs and D. Souvaine. An efcient algorithm for guard placement in polygons with holes.
16. F. Hoffmann, M. Kaufmann, and K. IGiegel. The art gallery theorem for polygons with holes. *Pmc. Of the Symposium of Foundations of Computer Science*, pages 39-84,1991.
17. M. Garey, D. Johonson, F. Preparata, and R. Tarjan. Triangulating a simple polygon. *Inform. Process. Lett*,7:175-179, 1978.
18. A. Elnagar and L. Lulu. A robust and fast algorithm for guarding simple polygons. 16th Informational Canadian Conference on Computational Geometry, 2004.
19. L. Lnlu and A. Elnagar. Perfomance evaluation of robot motion planning algorithms: Visprm vs. a m . *IEEE Intematianal Conference on Systems, Man and Cybernetics*, 2004.
20. Ear-clipping Based Algorithms of Generating High-quality Polygon Triangulation by Gang Mei, John C.Tipper and Nengxiong Xu

PART B

(IIT Ropar iHub- AWaDH Project)

COMPANY PROFILE

A Department of Science & Technology (DST), Government of India, funded Technology Innovation Hub (TIH) for Agriculture and Water technology development under the National Mission on Interdisciplinary Cyber-Physical Systems (NM - ICPS) at the Indian Institute of Technology Ropar.

In order to support the activities of the Agriculture and Water Technology Development Hub (AWaDH), a section - 8 company, IIT Ropar - Technology & Innovation Foundation, was founded. They are primarily working on (i) Water and Soil Quality Assessment Processes, (ii) Water Treatment and Management, (iii) Agriculture Automation and Information Systems, (iv) Stubble Management and Urban Farming, (v) IoT Systems, and (vi) Nuclear Instrumentation for Agriculture & Water, towards eco-friendly farming practices and to make farming more profitable for the grower.

The mission of the Hub is to develop 30 Agritech, support 35 startups/spin-off companies, train more than 1000 professionals in Cyber-Physical Systems, give more than 8000 employments through different Agritech innovations, etc. The technology developed at Hub would advance the (i) Environment, Forest and Climate, (ii) Fisheries (iii) Food Processing and Public Distribution, (iv) Rural and Women Empowerment by Skill Development and Entrepreneurship, (v) Land Resources, (vi) Electronics and Information Technology, (vii) Fertilizers, to name a few.

CHAPTER 1

1.1 Objectives of Training

- The 8-week training period was aimed at developing novel designs and mechanisms of the project allotted and filing patents for the same.
- To understand the role of an engineer in project management.
- To gain experience of working in a professional environment and interacting with industry leaders

1.2 Project Overview

Fish Deboning Machine is mainly used to separate fish meat from fish bone, fish skin, fish tendon in order to raise the fish use rate and save the manpower. Existing machines are of very poor efficacy. We were expected to design high efficiency machines so that the wastage of the produce can be minimized. In this machine, the food touching places were made out of high-quality stainless steel and met the requirements of national food hygienic standards. We also included an IoT system in the project for informing the farmers about the produce obtained and efficiency from a website accessible via smartphones.

CHAPTER 2

2.1 Simscape Electrical Design

The entire electrical circuitry of the deboning machine is implemented in Simscape Electrical. The machine will work on single phase AC input of 230 Volts at 50 Hz Frequency. Through the software I was able to calculate the current required by each motor in the machine and the power consumed, hence the electrical rating of the machine.

2.1.1 Full wave Rectifier

A convertor circuit is implemented to convert the $230(\sin 100t)$ input to constant 230V dc output to be given to the motors.

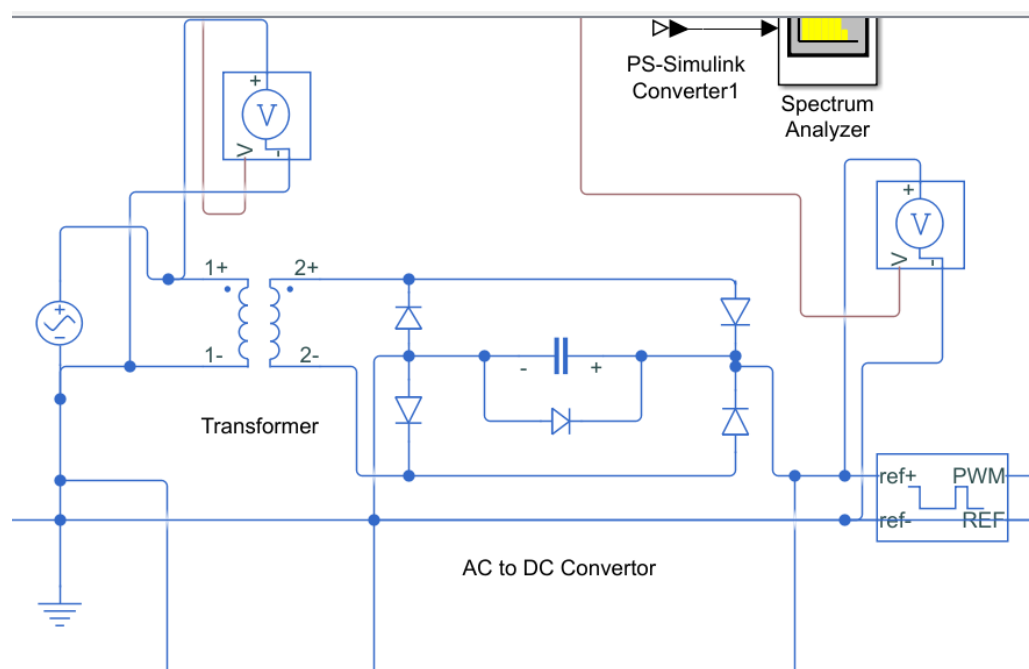


Figure 17: Full wave Rectifier

The bridge rectifier, in this case a full-wave rectifier. It is made out of individual discrete diodes 1N4007 with forward voltage of 1.1V. The idea is that we switch the negative AC

pulses to positive pulses, and leave the already positive pulses there. There is some voltage loss due to the voltage requirements of the diodes, but it is minimal. The end result is a pulsed DC voltage, going from 0 to maximum voltage at 50Hz. We use a capacitor of 47uF across the '+' and '-' terminals to smooth out the ripples. As the voltage rises from 0 to max, the capacitor charges. When the voltage starts to drop, the capacitor discharges through the circuit but at a much slower rate, in effect holding the voltage up while the supply drops to 0 and then rises again. Once the voltage rises to where the capacitor voltage is, it recharges the capacitor and surges back to max again. As long as the ripple doesn't get below a certain value. We can use that to power a voltage regulator, which simply stabilizes the wobbly input voltage to a specific output voltage. Full-wave rectifiers are better here than half-wave, since there is less time between the high and low pulses, resulting in a more stable output. Here we are getting a constant +230V dc output after the rectification.

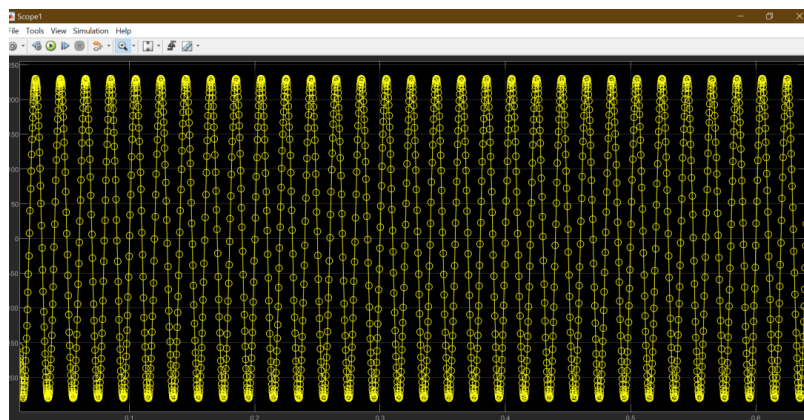


Figure 18: 230V AC Supply



Figure 19: DC Output

2.1.2 PWM & H Bridge Motor Controller

The DC Voltage output is fed into a PWM controller with a switching frequency of 1000Hz of successive pulse.

The output of the PWM controller is set at 5V which is to be fed to the H Bridge Industrial Motor controller rated at 5V DC input.

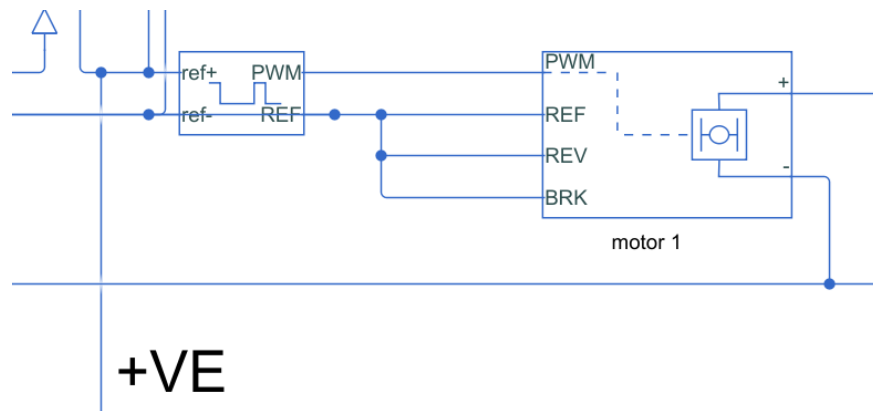


Figure 20: PWM & H Bridge

For the purpose of Fin Removal, Descaling, Head& tail removal, Head collection, Filleting, Gutting, and to run Conveyor belts we have used industrial motors rated at 100W , 100 V , 5N-mm Torque. The voltage output from the H bridge is amplified to 100 V to run these motors.

- Fin Removal: Two motors are used for Fin removal purposes. The current characteristics of these motors are plotted in the graphs. For Fin removal the motors use 1.0078 Amps of Current in its operation. Total power consumed = $(2 \times 100 \times 1.0078) = 208 \text{ W}$

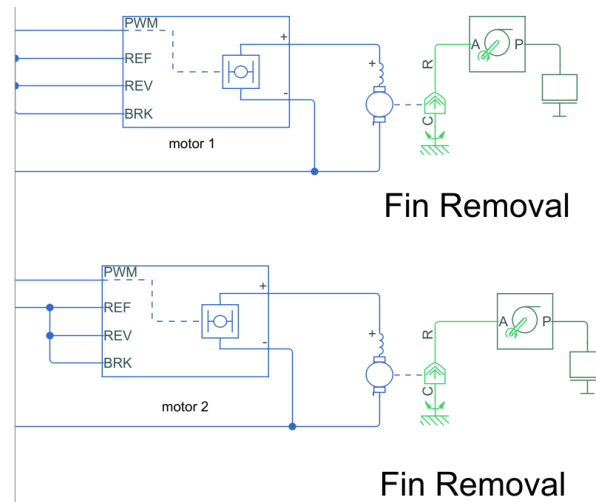


Figure 21:Motors for Fin removal

- Descaling & Head, Tail Removal : 1 motor each is used for these two purposes The current characteristics of these motors are plotted in the graphs. For these the motors use 1.0078 Amps of Current in its operation. Total power consumed = $(2 \times 100 \times 1.0078) = 208 \text{ W}$

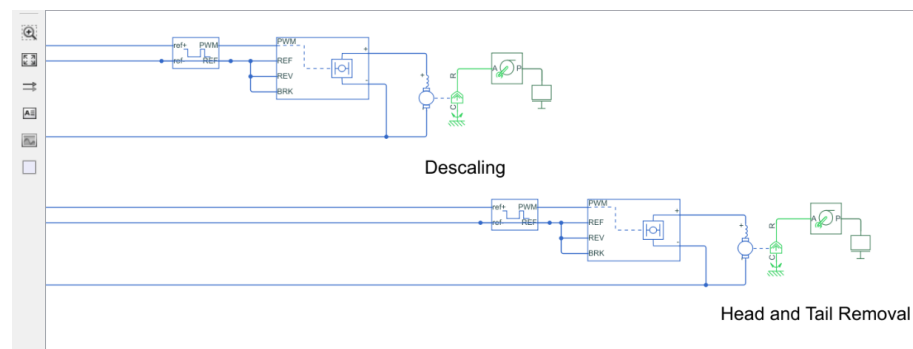


Figure 22:Descaling, Head & Tail removal

- Gutting: The fish is Gut to separate the organs for this the motor use 1.04 Amps of Current in its operation. Total power consumed = $(100 \times 1.04) = 104 \text{ W}$

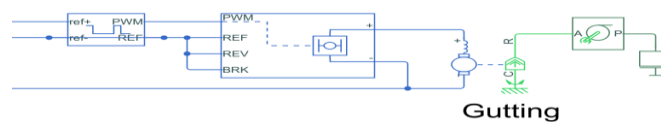


Figure 23:Gutting

- Filleting: The fish is filleted during this process according to the used demands. For this the motors used 1.04 Amps of Current in its operation. Total power consumed = $(2*100*1.04) = 208W$

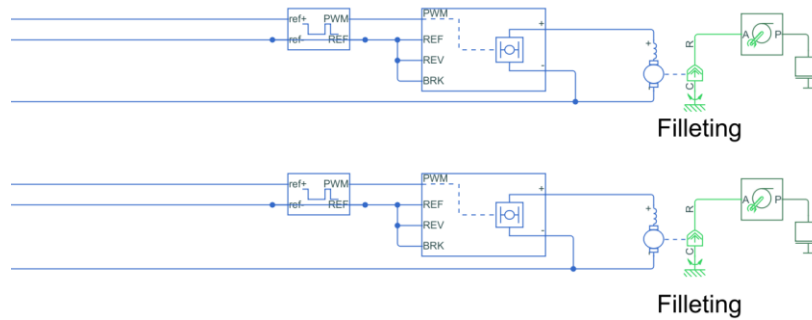


Figure 24: Filleting

- Deboning: This motor runs the perforated drum for deboning this motor is rated at 230V. . For this the motor use 0.25 Amps of Current in its operation. Total power consumed = $(230*0.25) = 57.5W$

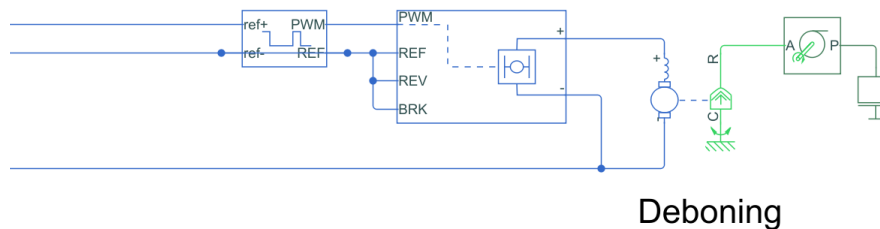


Figure 25: Deboning

- Motors for running Conveyor belts: In total 3 motors are required for running the conveyor belt at 60 Rpm. . For this the motor use 1.04 Amps of Current in its operation. Total power consumed = $(3*100*1.04) = 312W$

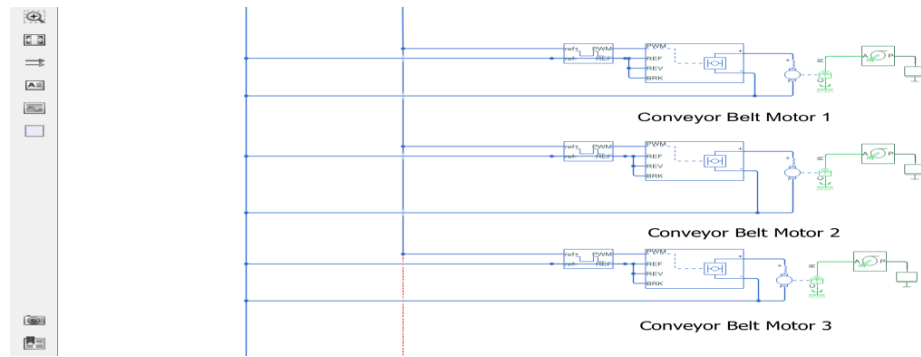


Figure 26: Conveyor Belts Motors

2.1.3 Rating of Machine

Total power consumed by all motors : 1293 Watts

Power rating of Machine : 1293-Watt hour

CHAPTER 3

3.1 Control System Design

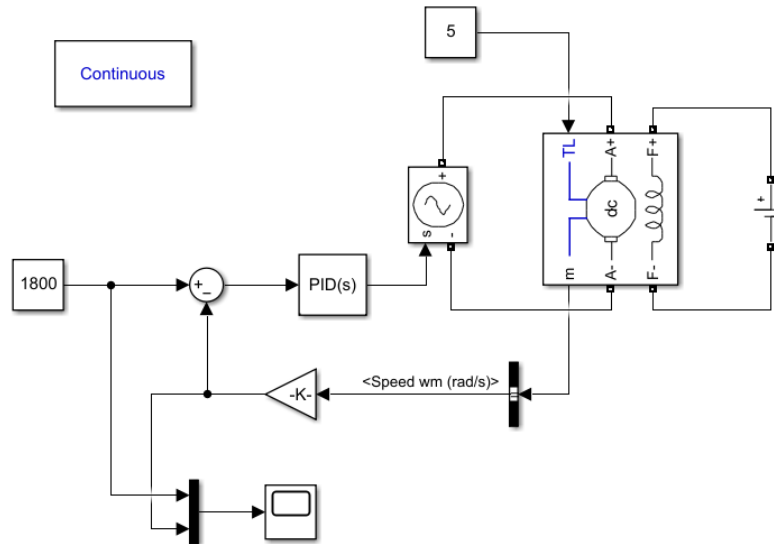


Figure 27:Control System

A closed loop PID based control system is designed for the optimal speed control of the Motor. The constant is set at 1800 Rpm which is required for the deboning process. Here the Dc machine is fed with variable voltage supply its RPM output is obtained and is multiplied by specific gain constant to convert it to RPM. This acts as a feedback signal to the control system this is subtracted from the constant RPM we want i.e. 1800 RPM which produces the error signal , this error signal is fed to PID block where the gains K_p , K_d , K_i are tuned accordingly to reduce the settling time (T_s) and minimize the maximum overshoot (M_p). The actuating signal obtained produced is given to the variable voltage supply. The results are plotted using a scope.

$$K_p = 0.851320759653448$$

$$K_i = 11.6736558254255$$

$$K_d = 0.014231201926504$$

The stable control system parameters obtained:

Rise Time = 0.0615 secs

Settling Time = 0.201 secs

Overshoot = 6.8%

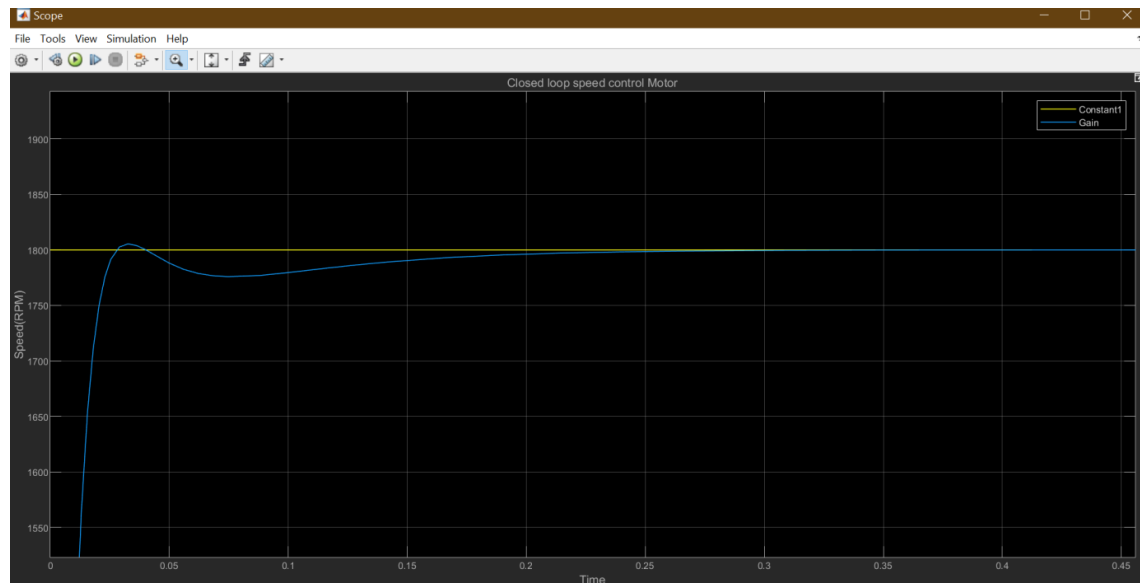


Figure 28: Controller Parameters

3.2 Stability

For stability analysis I checked Absolute stability of the control system using Root Locus Approach while the relative stability was checked using Bode Plot and the values of Gain and Phase margins obtained from the plot . Stability analysis was done using MATLAB's Control System Designer.

A live script is written in MATLAB with certain motor constants , through this we calculated the continuous time transfer function of the motor

$$J = 0.01;$$

$$b = 0.1;$$

$$K = 0.01;$$

$$R = 1;$$

$$L = 0.5;$$

$$s = tf('s');$$

$$P_motor = K/((J*s+b)*(L*s+R)+K^2)$$

P_motor =

0.01

$$0.005 s^2 + 0.06 s + 0.1001$$

Continuous-time transfer function.

3.2.1 Root Locus

Root Locus for the above transfer function was obtained using MATLAB's Control system designer.

Root Locus of the open loop transfer function is obtained for learning about absolute stability of the system.

%Root Locus%

controlSystemDesigner('rlocus', P_motor)

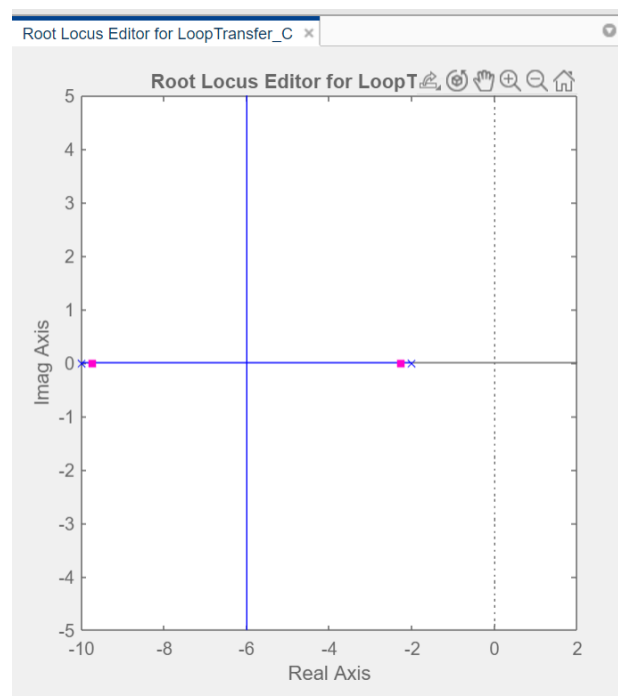


Figure 29:Root Locus Plot

Since the Root locus is towards the left of the imaginary axis hence the system is stable

3.2.2 Bode Plot

Bode Plot is also obtained for the system for information regarding relative stability of the system. The gain margin and phase margins are also calculated

```
bode(P_motor)
grid
title('Bode Plot of the Original Plant')
```

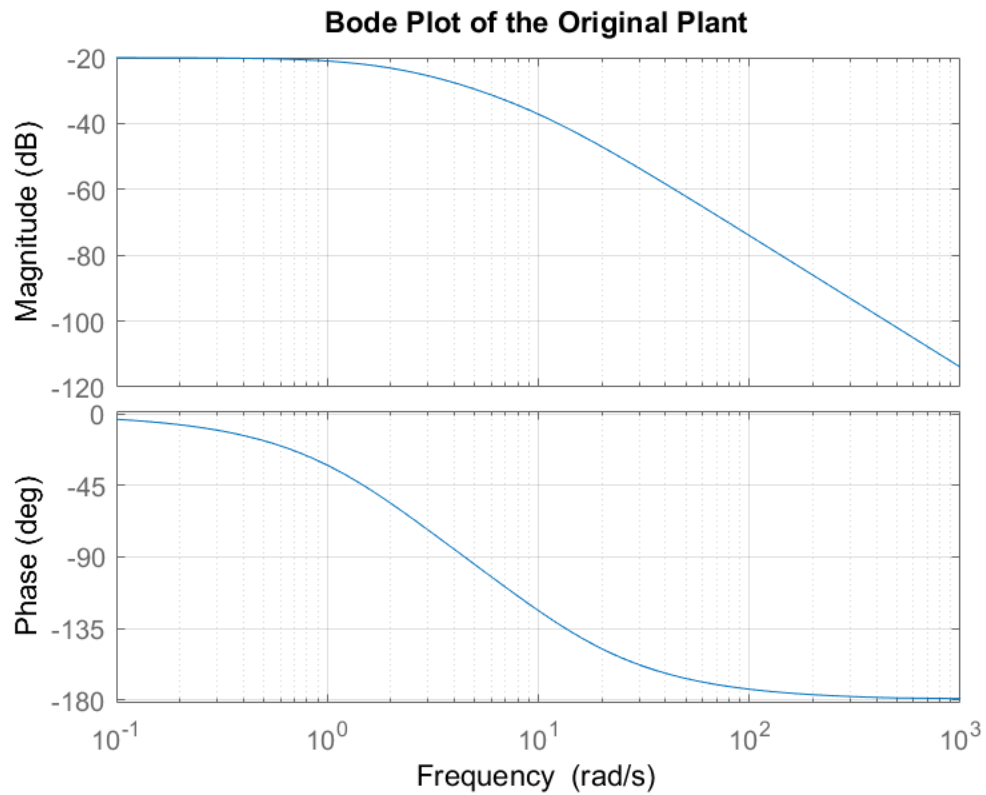


Figure 30: Bode Plots

```
margin(P_motor)
[Gm,Pm,Wcg,Wcp] = margin(P_motor)
```

Gain and Phase margins:

$$G_m = \text{Inf}$$

$$P_m = \text{Inf}$$

$$W_{cg} = \text{Inf}$$

$$W_{cp} = \text{NaN}$$

Since there are coming out to be Infinity so our motor will always be stable throughout its functioning in the machine.

CHAPTER 4

4.1 IoT Circuit and Code

The amount of fish weight obtained is sensed thorough a load cell controlled by an ESP8266 WIFI Module, this module will send the real time weight data to MATLAB ThingSpeak an online IoT analytics platform, The farmers can know about the amount of fish meat produced and compare their produce. It will also have a temperature sensor LM35 to keep track of the temp of fish obtained.

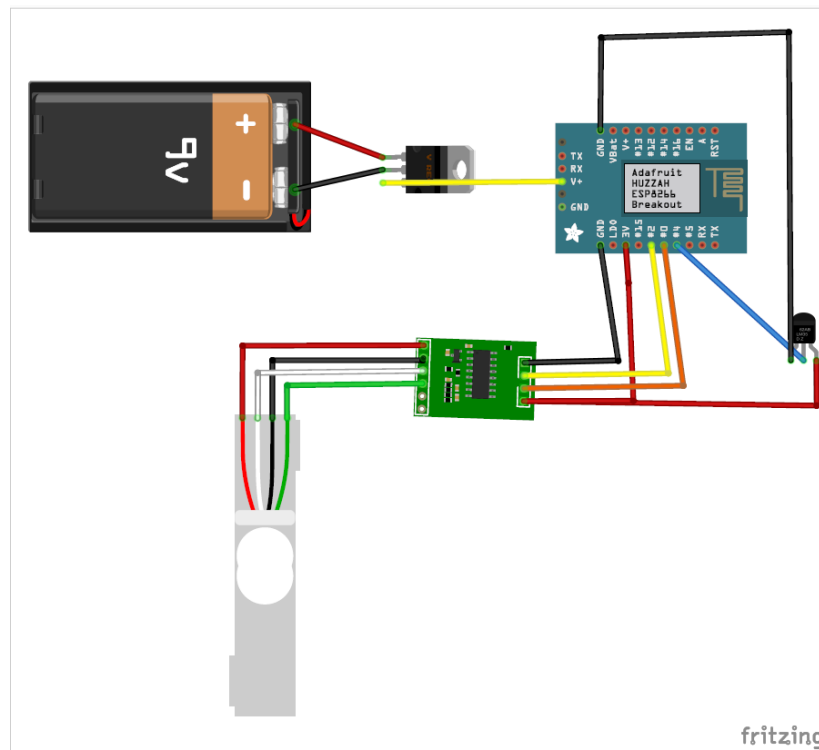


Figure 31:IoT circuit



The program for sending the data to ThingSpeak platform (IoT Code):

GitHub Profile : https://github.com/shivu926/IIT_Ropar_FishDeboning

4.2 Human Machine Interface

The HMI is placed at the starting of the machine for the operator to decide whether the desired output is fillet or minced fish meat.



Figure 32:HMI

Depending upon the choice the output will be produced.

Connected of STM32 Controller

```
int x;
printf("%s", "Enter the choice 1. Fillet 2.Mush")
scanf("%d",x);
switch(x):
{
case 1:
GPIO.output(Motor9.HIGH);
break;
case 2:
GPIO.output(Motor9.LOW);
break;
Default
GPIO.output(Motor9.LOW);

break;
}
```

CHAPTER 5

5.1 Conclusion

My internship has allowed me to see so much more about the Mechatronics field than I can learn in the classroom. I have developed many skills and have a much greater concept of what to expect after college. My internship has given me a greater understanding of what it is I have learned in the classroom and allowed me to apply it to real situations. One of the greatest benefits that I have received is the knowledge that I enjoy doing what it is I have been studying for. Now I can comprehend Control Systems and program them . My learnings throughout this project have been unprecedented where I was given an opportunity to explore a domain of Electronics that I had never previously worked on. Being a student of Mechatronics engineering knowledge of Control systems and Cyber physical systems like IoT is of utmost importance and I was able to learn them during the internship period.

Patents are expected of the CAD Model and Mechanisms developed

5.2 References

- MATLAB Documentation- <https://in.mathworks.com/help/matlab/>
- Simscape Documentation- <https://in.mathworks.com/help/physmod/simscape/>
- FritZing Documentation - <https://fritzing.org/services/>
- K Ogata, Modern Control Engineering, Pearson, 2004

APPENDIX

March 2021

S.no.	Date	Work Done	Reflection
1	4 th January - 31 st March, 2021	<ul style="list-style-type: none"> • Programming Languages <ul style="list-style-type: none"> ○ Python <ul style="list-style-type: none"> ▪ Advanced Data Structures ▪ Theory of Computation ▪ Complexity Analysis ▪ Geometric Algorithms ▪ Open-Source Libraries like SciPy, SciKit Geom ○ C++ <ul style="list-style-type: none"> ▪ Advanced Data Structures ▪ CGAL Library 5.0 (A widely used Massive library solely develop for Computational Geometry) ▪ Complexity 	<p>I learned Python and C++ right from its basics to advanced data structures. Learned the following:</p> <ul style="list-style-type: none"> • Comprehend Algorithms • Write Procedural Programs • Design & Debug Algorithms for Robotic application • Computation complexity of programs

April 2021

S.No.	Date	Work Done	Reflection
1	1/04/21	Literature Survey of Art gallery Problems: Survey through various online journals, research paper, conferences on computational geometry, IEEE Platforms, NTU/NUS e-Library resources, TIET's Nava Nalanda Library resources.	Learned how to conduct literature surveys of prior research done in the specified topics.

2	2/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
3	3/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
4	4/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
5	5/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
6	6/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
7	7/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics

8	8/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
9	9/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
10	10/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
11	11/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
12	12/04/21	Literature Survey	Learned how to conduct literature surveys of prior research done in the specified topics
13	13/04/21	Literature Survey	Learned how to conduct literature surveys.

14	14/04/21	Literature Survey	Learned how to conduct literature surveys.
15	15/04/21	Literature Survey	Learned how to conduct literature surveys.
16	16/04/21	Literature Survey	Learned how to conduct literature surveys.
17	17/04/21	Literature Survey	Learned how to conduct literature surveys.
18	18/04/21	Literature Survey	Learned how to conduct literature surveys.
19	19/04/21	Literature Survey	Learned how to conduct literature surveys.
20	20/04/21	Literature Survey	Learned how to conduct literature surveys.
21	21/04/21	Literature Survey	Learned how to conduct literature surveys.
22	22/04/21	Literature Survey	Learned how to conduct literature surveys.
23	23/04/21	Literature Survey	Learned how to conduct literature surveys.

24	24/04/21	Started programming in Python 3.9 using Sublime text editor.	Learned the properties of the sublime text editor for python programming
25	25/04/21	Programming the doubly connected edge list data structure for storing polygon information	Learned how to use the DCEL Structure
26	26/04/21	Debugged the DCEL code as there were some errors	Debugging techniques
27	27/04/21	Debugged the DCEL code as there were some errors	Debugging techniques
28	28/04/21	Debugged the DCEL code as there were some errors	Debugging techniques
29	29/04/21	Debugged the DCEL code as there were some errors	Debugging techniques
30	30/04/21	Debugged the DCEL code as there were some errors	Debugging techniques

May 2021

S.No.	Date	Work Done	Reflection
1	1/05/21	Started the Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
2	2/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
3	3/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
4	4/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
5	5/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
6	6/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
7	7/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
8	8/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming
9	9/05/21	Monotone Partitioning Method for Triangulation	Learned about geometric algorithms programming

10	10/05/21	Completed Monotone triangulation	Learned about geometric algorithms programming
11	11/05/21	Started with Two ears theorem	Learned about geometric algorithms programming
12	12/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
13	13/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
14	14/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
15	15/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
16	16/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
17	17/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
18	18/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
19	19/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
20	20/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
21	21/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
22	22/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
23	23/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming
24	24/05/21	Working on Two Ears Theorem and 3-Colorability	Learned about geometric algorithms programming

25	25/05/21	Completed Two ears theorem triangulations	Learned about geometric algorithms programming
26	26/05/21	Complexity analysis of the algorithms performed	Learned about complexity analysis and comparing Algorithmic efficiency
27	27/05/21	Complexity analysis of the algorithms performed	Learned about complexity analysis and comparing Algorithmic efficiency
28	28/05/21	Complexity analysis of the algorithms performed	Learned about complexity analysis and comparing Algorithmic efficiency
29	29/05/21	Complexity analysis of the algorithms performed	Learned about complexity analysis and comparing Algorithmic efficiency
30	30/05/21	Complexity analysis of the algorithms performed	Learned about complexity analysis and comparing Algorithmic efficiency
31	31/05/21	Complexity analysis of the algorithms performed	Learned about complexity analysis and comparing Algorithmic efficiency

June 2021

S.No.	Date	Work Done	Reflection
1	1/06/21	Compiling results and making report for NTU Submission	Learned about scientific documentation
2	2/06/21	Compiling results and making report for NTU Submission	Learned about scientific documentation
3	3/06/21	Compiling results and making report for NTU Submission	Learned about scientific documentation
4	4/06/21	Compiling results and making report for NTU Submission	Learned about scientific documentation
5	5/06/21	Started Working for fish deboning machine Doing patent search for previously developed fish deboning mechanisms	Learned about Patent search
6	6/06/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
7	7/06/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
8	8/06/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
9	9/06/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape

10	10/6/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
11	11/6/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
12	12/6/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
13	13/6/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
14	14/6/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
15	15/6/21	Simscape Electrical design for the machine	Learned about electrical principles that go in designing mechatronics systems on Simscape
16	16/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications

17	17/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
18	18/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
19	19/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
20	20/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
21	21/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
22	22/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications

23	23/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
24	24/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
25	25/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
26	26/6/21	Simulink Control systems	Learned how to use theoretical concepts of control systems to develop them in software applications
27	27/6/21	IoT code and Circuit using ESP8266, Fritzing, Load Cell , HX711-Amplifier,	Learned how to develop IoT applications and testing them in cyberspace
28	28/6/21	IoT code and Circuit using ESP8266, Fritzing, Load Cell , HX711-Amplifier,	Learned how to develop IoT applications and testing them in cyberspace
29	29/6/21	IoT code and Circuit using ESP8266, Fritzing, Load Cell , HX711-Amplifier,	Learned how to develop IoT applications and testing them in cyberspace

30	30/06/21	IoT code and Circuit using ESP8266, Fritzing, Load Cell , HX711-Amplifier,	Learned how to develop IoT applications and testing them in cyberspace
----	----------	--	--

July 2021

S.No.	Date	Work Done	Reflection
1	1/07/21	Programming a human machine interface in C++	Learned about embedded programming in C++
2	2/07/21	Programming a human machine interface in C++	Learned about embedded programming in C++
3	3/07/21	Programming a human machine interface in C++	Learned about embedded programming in C++
4	4/07/21	Programming a human machine interface in C++	Learned about embedded programming in C++
5	5/07/21	Programming a human machine interface in C++	Learned about embedded programming in C++
6	6/07/21	Made the Electronics report of the project	Learned about electronic documentation process
7	7/07/21	Made the Electronics report of the project	Learned about electronic documentation process
8	8/07/21	Made the Electronics report of the project	Learned about electronic documentation process
9	9/07/21	Made the Electronics report of the project	Learned about electronic documentation process
10	10/07/21	Made the Electronics report of the project	Learned about electronic documentation process
11	11/07/21	Made the Electronics report of the project	Learned about electronic documentation process
12	12/07/21	Made the Electronics report of the project	Learned about electronic documentation process