# Weekly Report – February 20th, 2023

# Code

## TSP Model: SOM (Self-Organizing Map)

In order to solve the TSP portion of the problem, I have selected a Self-Organizing Map approach derived from: <https://github.com/diego-vicente/som-tsp.> The idea behind this is that we can coerce a network of points towards nearby way points by applying a Gaussian filter to the network based off random sampling of our way points. I like to think of it as dips in a flat sheet of material with sand on it, and by gently “shaking” it, the sand naturally moves towards the closest dip. This overall produces a “band” of network points which eventually connects the various way points. Leveraging these network points we can quickly identify a reasonable solution to the TSP. My code thus far is available [here](https://github.com/KKonaOG/ECE8833_FinalProject). At the moment, I am investigating a slight issue (does not affect output), where my code executes additional iterations in comparison to the source code for the same input. This is considered non-breaking but does increase overall execution time. Outputs from execution are shown below:

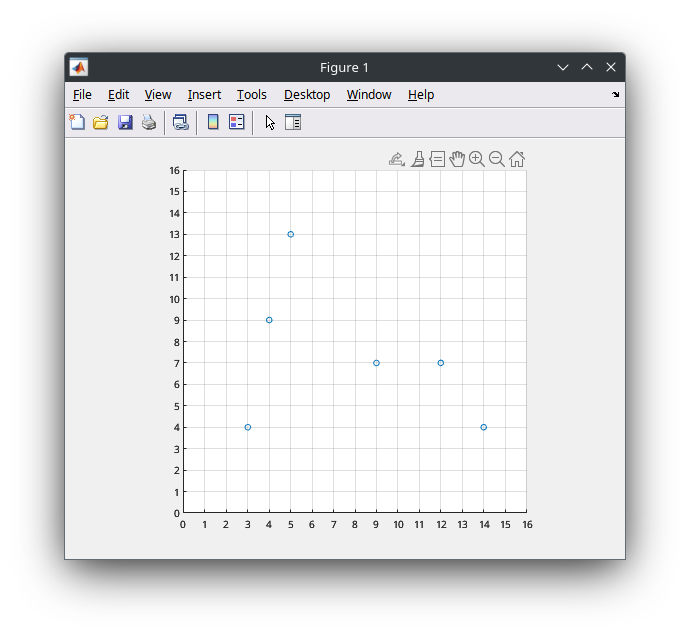


Figure 1: Map with Way Points

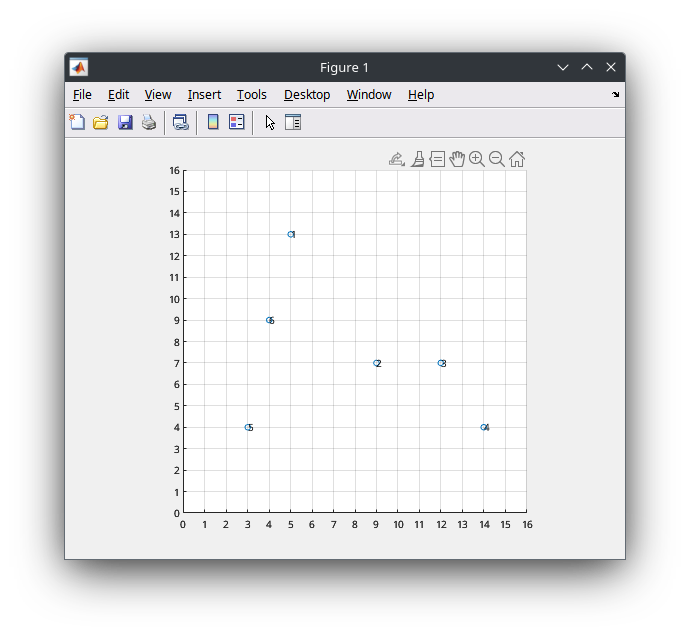


Figure 2: Map with Way Points Post-TSP

Example Variable Outputs:



The network size is a correlation of the number of “cities” or way points per the desired amount neurons. Additional neurons can lead to higher fidelity, however, a simple number of 8 has been chosen for now. Each network consists of many “neurons” which are simply locations on the normalized map.

## Code Flow Model

In order to facilitate a better understanding of my code’s flow, I have created the following breakdown and flowchart which models the intended functionality. It is considerably similar flow model to that of my final project for Advanced Robotics.

### Project Structure:

* main.m – Entry Point
* Algorithms/
  + TSP/
    - Helpers/
      * GetClosestPoint.m
    - self\_organizing\_map.m
  + PTP/
* Maps/
  + Helpers
    - CreateWaypoint.m
    - CreateObstacle.m
  + Map1.m
  + Map2.m
  + Map3.m

### Algorithms

Each algorithm is classified by its use-case. I.e, for TSP algorithms, they are placed into the TSP folder. Any helper functions for the TSP algorithm would be placed in a folder within TSP labeled Helpers. This is intended to make an easy, clean, and organized method for identifying related code files. Each algorithm itself is a function which is called from the main.m (entry) file, in the case of TSP it calls the self\_organizing\_map function.

### Maps

Maps are a critical piece of the puzzle for simulating all algorithms. Each Map is implemented as a standalone MATLAB file which is intended to be run in order to populate the workspace with related map variables. This enables the possibility to move maps from one project to another in the future. Importantly, the variables exposed by running a map are the map figure, way points, obstacles, and other related information.

### Entry Point (Main.m)

Main.m is intended for the user to select a Map to run through the simulation. It runs the selected Map file and executes each step of the model in order. Additionally, it handles any plotting of the outputs between each stage of the model.

## Flowchart

Below (and attached) is a minimal flowchart representing the current intended code flow. As work proceeds, each flow chart will become more detailed and more encompassing, this is just an initial flowchart as a starting point.

