Robotics Assignment 3 Report

Theodor Novitsky (tn347), Jad Ziab (jgz22)

IMPORTANT MODULES IN CONDA ENVIRONMENT:

conda install conda-forge::matplotlib

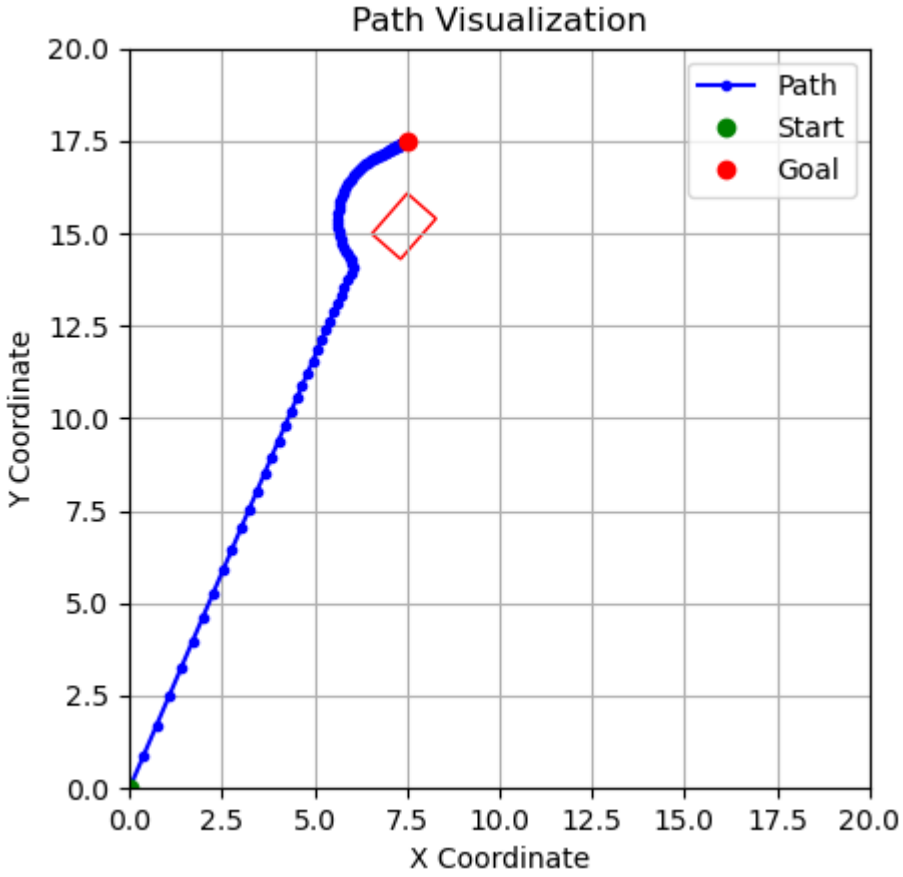
conda install anaconda::networkx

conda install anaconda::scipy

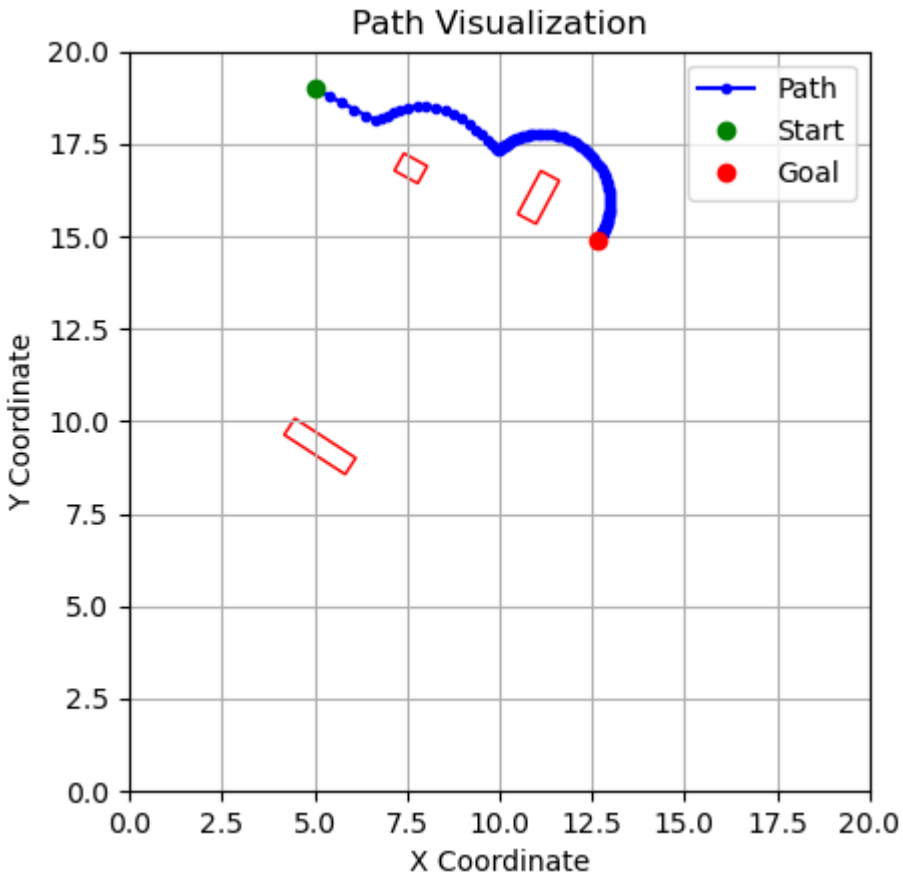
1)**Potential Function:**

We implement the gradient descent Algorithm so that the robot can navigate through the obstacles. Attractive potential pulls the robot towards the goal, and repulsive potential pushes the robot away from obstacles. The total gradient potential is calculated by summing both the repulsive and repulsive. Then the gradient is taken and the robot navigates, this process repeats until the gradient gets close enough to 0 within a certain tolerance. We use the center of the obstacles to help measure the repulsive potential. Certain coefficients can be altered to slightly tweak the robots movement. Within the images folder, you can find 5 subfolders for each environment which contain each image and animation of unique start and goals we tested. Below you can see one example from each environment:

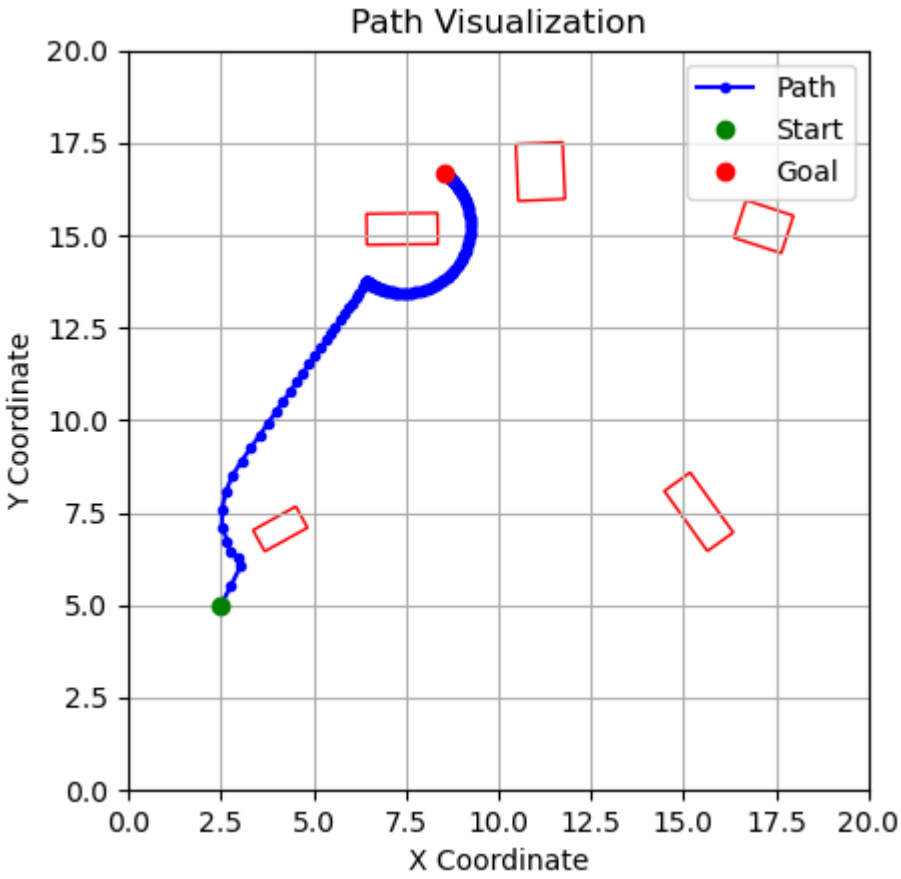
Environment 1:



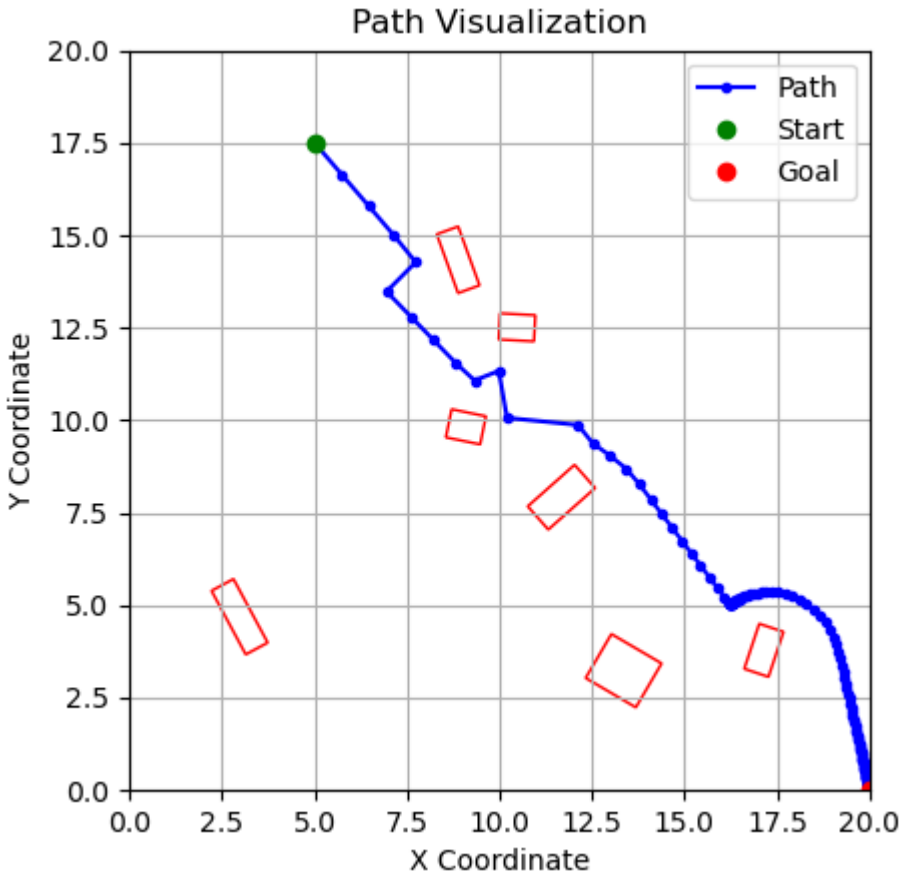
Environment 2:



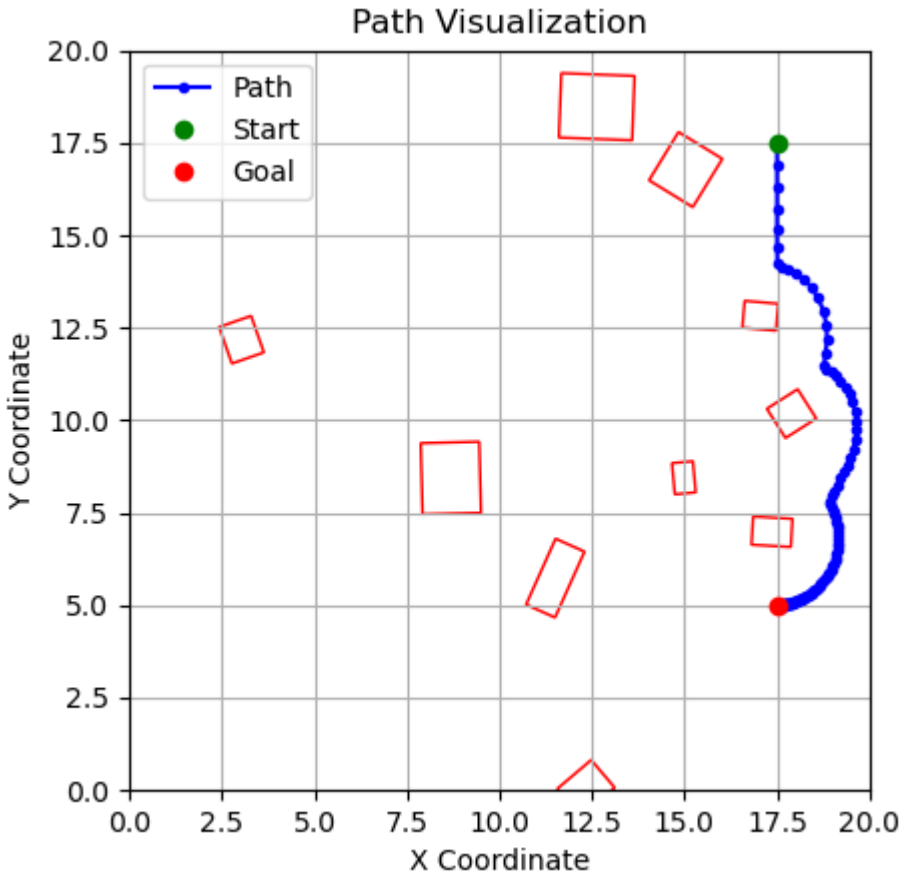
Environment 3:



Environment 4:



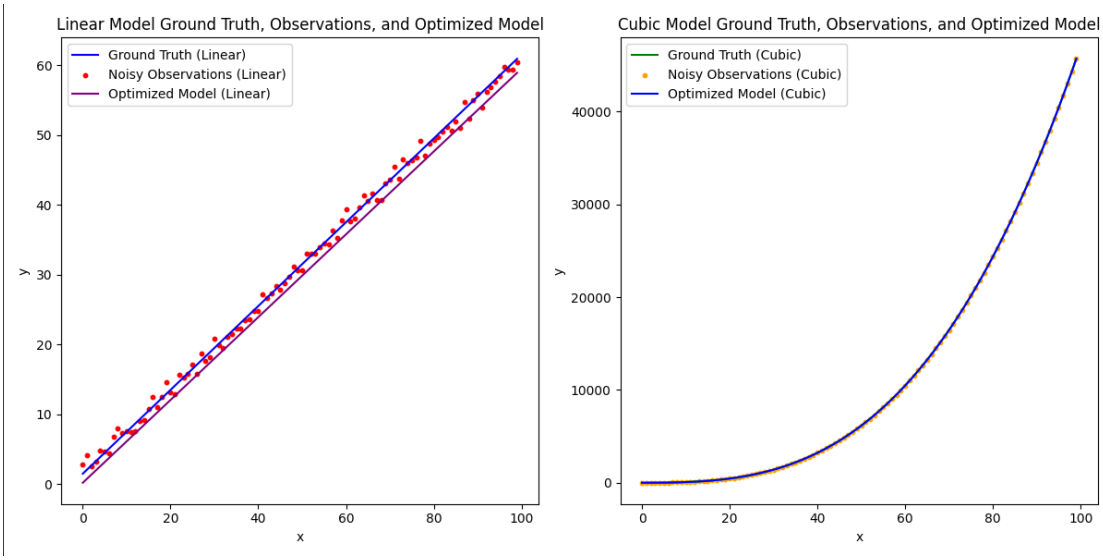
Environment 5:



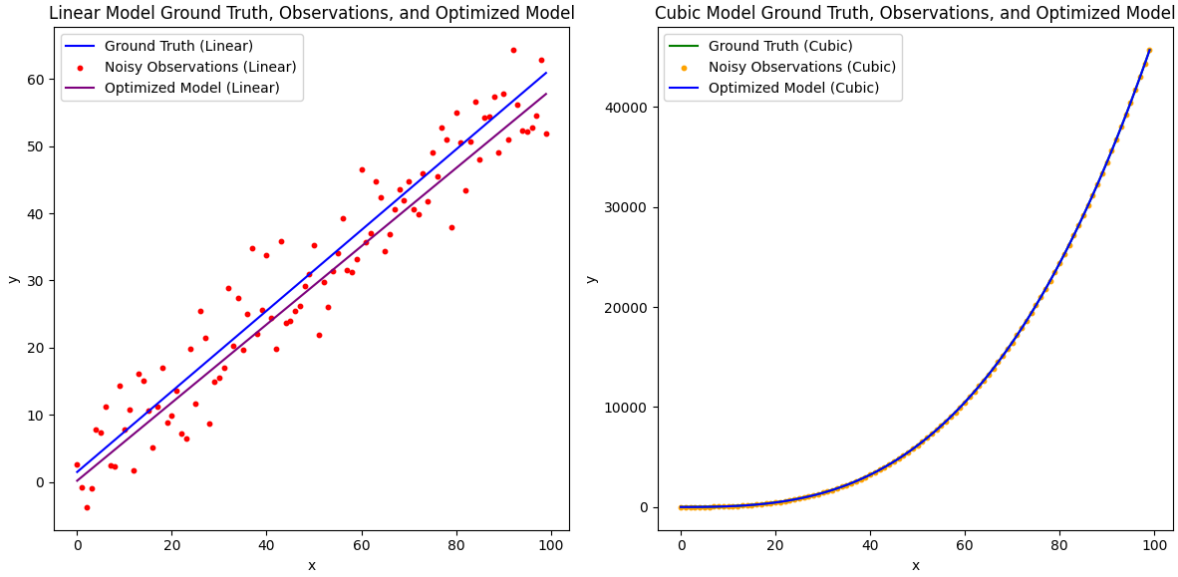
2) Gtsam: Factor Graphs:

The program implements a polynomial optimization problem using third-order polynomials, fitting a cubic function (f(x) = 0.045xˆ3 + 0.2xˆ2 + 0.7x + 4.86) to noisy observations generated by adding Gaussian noise (= {1, 5, 10}) to the ground-truth values. The optimization process is conducted using GTSAM's Levenberg-Marquardt optimizer, analyzing both linear and cubic models. The results show that the optimized models effectively approximate the true underlying functions, demonstrating the utility of non-linear optimization for recovering model parameters in noisy data. The screenshots provided show the program running for sigma/noise level 1, 5, and 10.

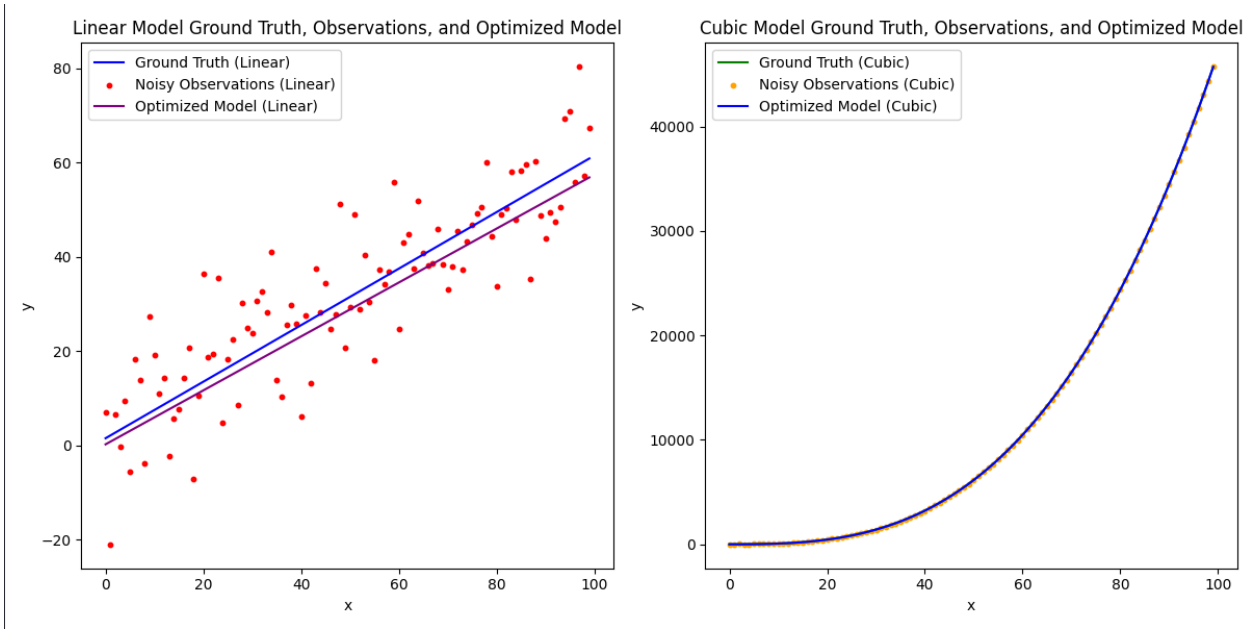
Sigma = 1



Sigma =5



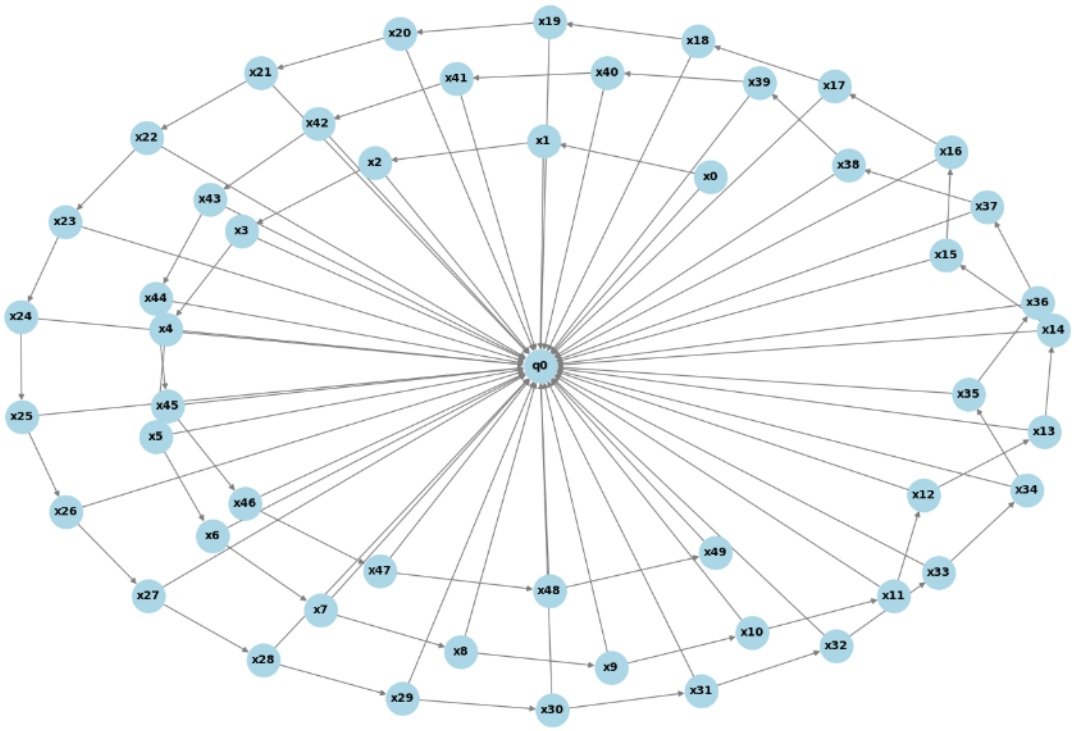
Sigma =10



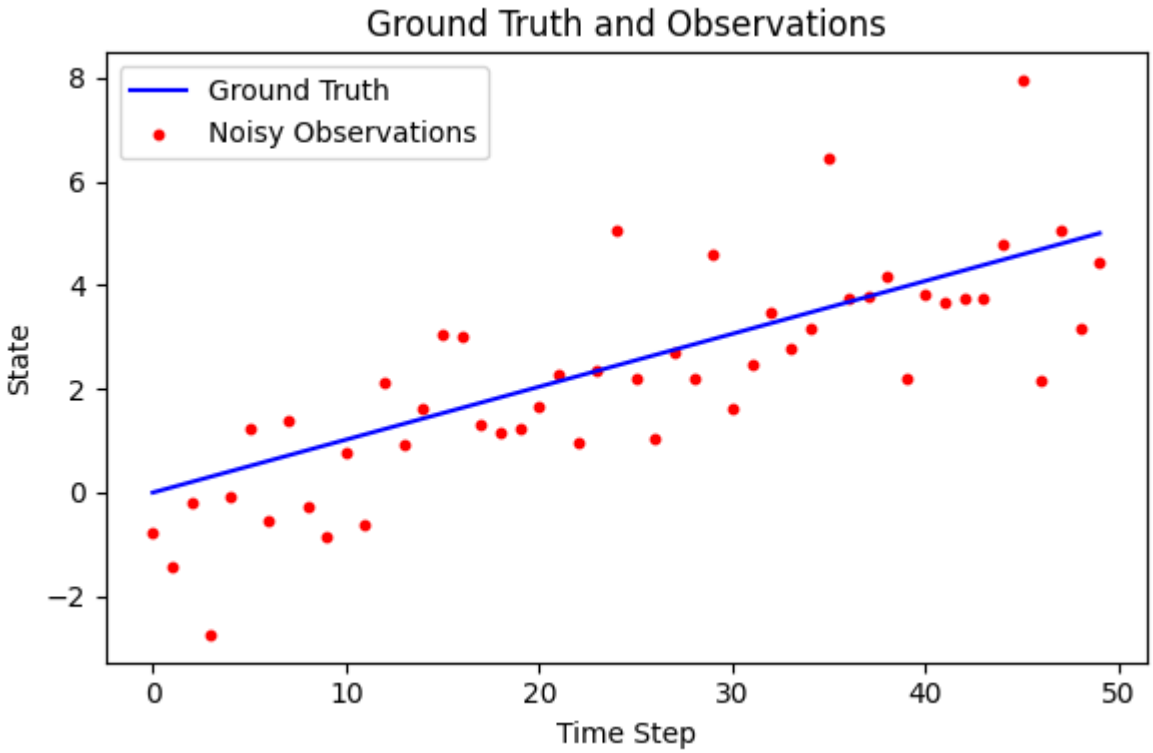
Simple Trajectory:

This program implements a trajectory optimization via factor graphs, specifically for a first-order point system moving from a start state to a goal state over a given number of time steps. The optimization is performed using GTSAM to minimize the error between predicted and observed states, given noisy observations. The factor graph model captures the dynamics between successive states and incorporates an initial velocity estimate. The script visualizes the factor graph structure and outputs both a factor graph diagram and a plot of the optimized trajectory compared to ground truth and noisy observations.

Factor Graph:


x19
x20
x18
x21
x41
x40
x39
x17
x42
x22
x1
x16
x2
x0
x38
x43
x37
x23
x3
x15
x44
x36
x24
x4
x14
q0
x35
x45
x25
x13
x5
x12
x34
x26
x46
x6
x49
x47
x33
x27
x48
x11
x7
x10
x8
x32
x28
x9
x29
x31
x30

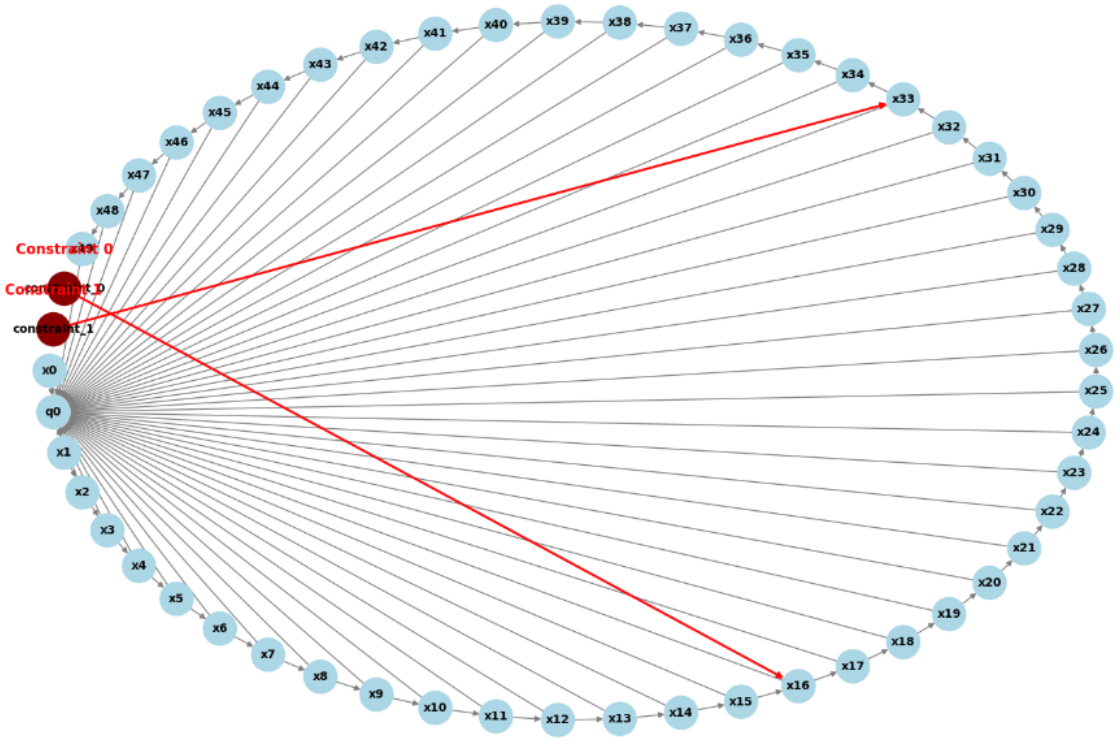
Trajectory:



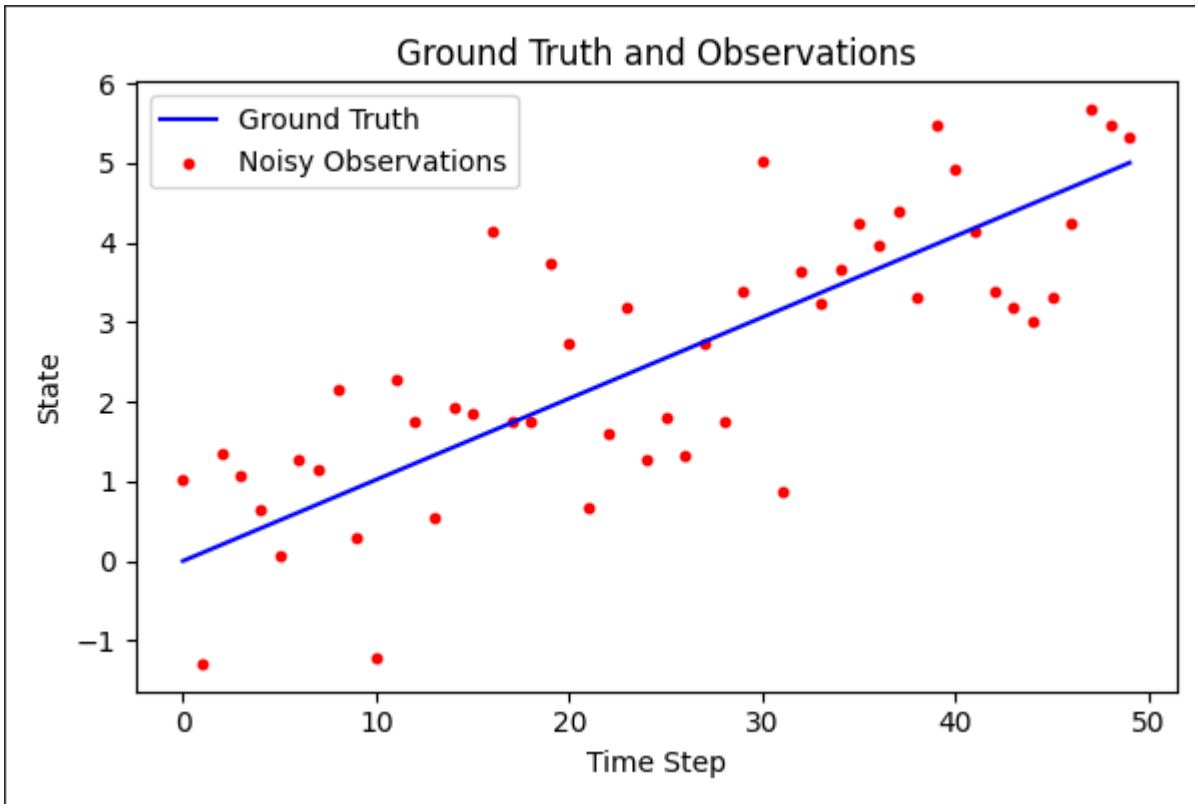
Extra Constraints:

The program performs trajectory optimization by constructing a nonlinear factor graph to estimate an optimal path between a start and a goal state while ensuring the path visits two specific intermediate points. The optimization uses a first-order dynamics model and incorporates extra constraints at the positions T/3T/3 and 2T/32T/3, specified as x0inx0in and x1inx1in. The GTSAM library is utilized to create factors that define relationships between states and intermediate constraints, and a Levenberg-Marquardt optimizer is used to estimate the optimized trajectory. The program provides visualizations of the factor graph structure and the trajectory, emphasizing the added constraints, which significantly influence the trajectory to ensure it passes near the predefined points.

Factor Graph with constraints:

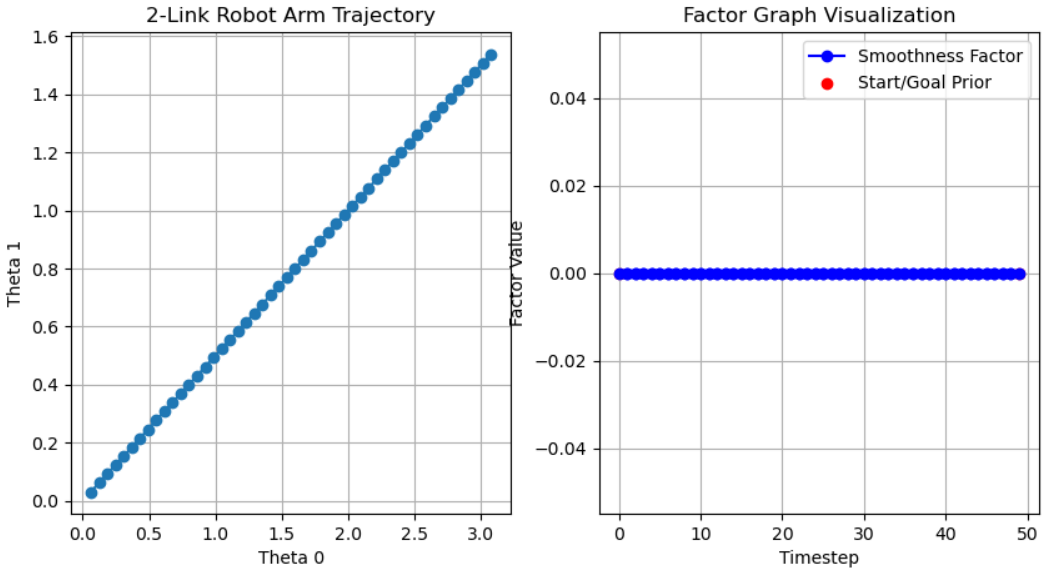


Trajectory with constraints:



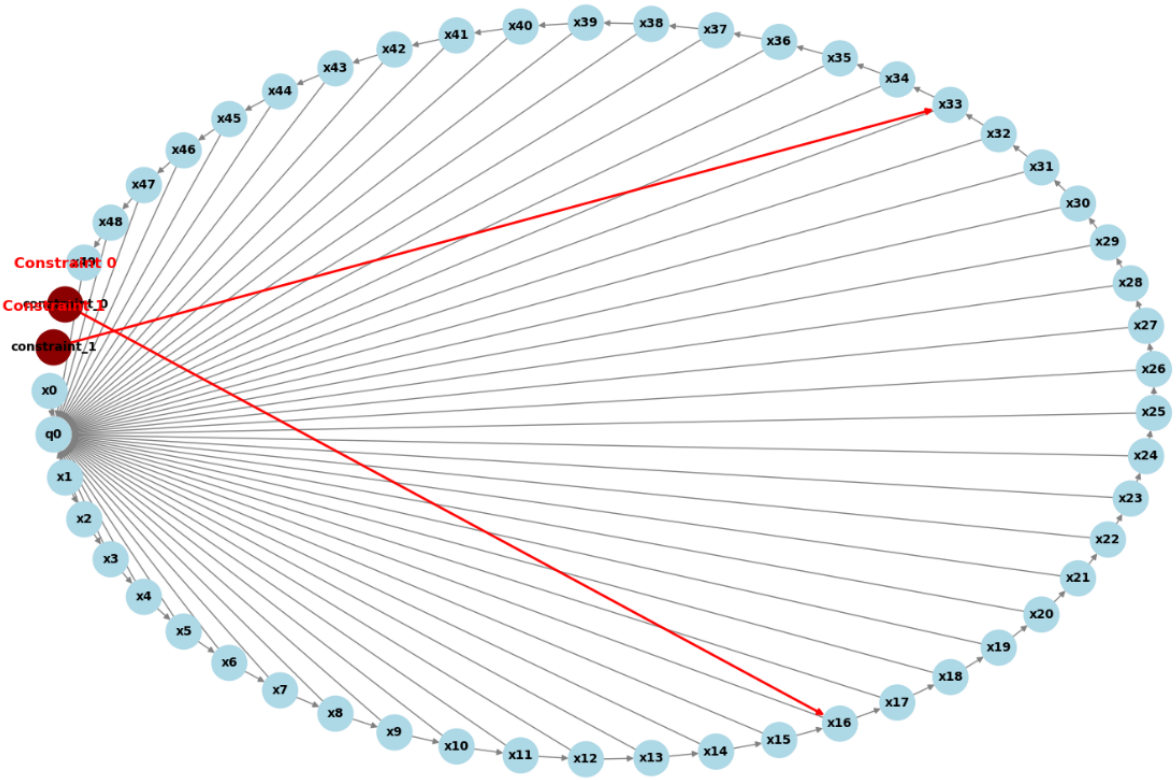
2-Link Robot Arm:

This script performs trajectory optimization for a 2-link robotic arm using GTSAM. It defines a trajectory from a specified start configuration to a goal configuration over a number of timesteps (T). A Nonlinear Factor Graph is constructed, where prior factors constrain the start and goal states, and smoothness constraints are enforced between consecutive timesteps to ensure a smooth trajectory. The initial trajectory estimate is linearly interpolated between the start and goal configurations. The visualization is saved as an image, making it useful for understanding robotic arm motion planning and factor graph-based optimization.

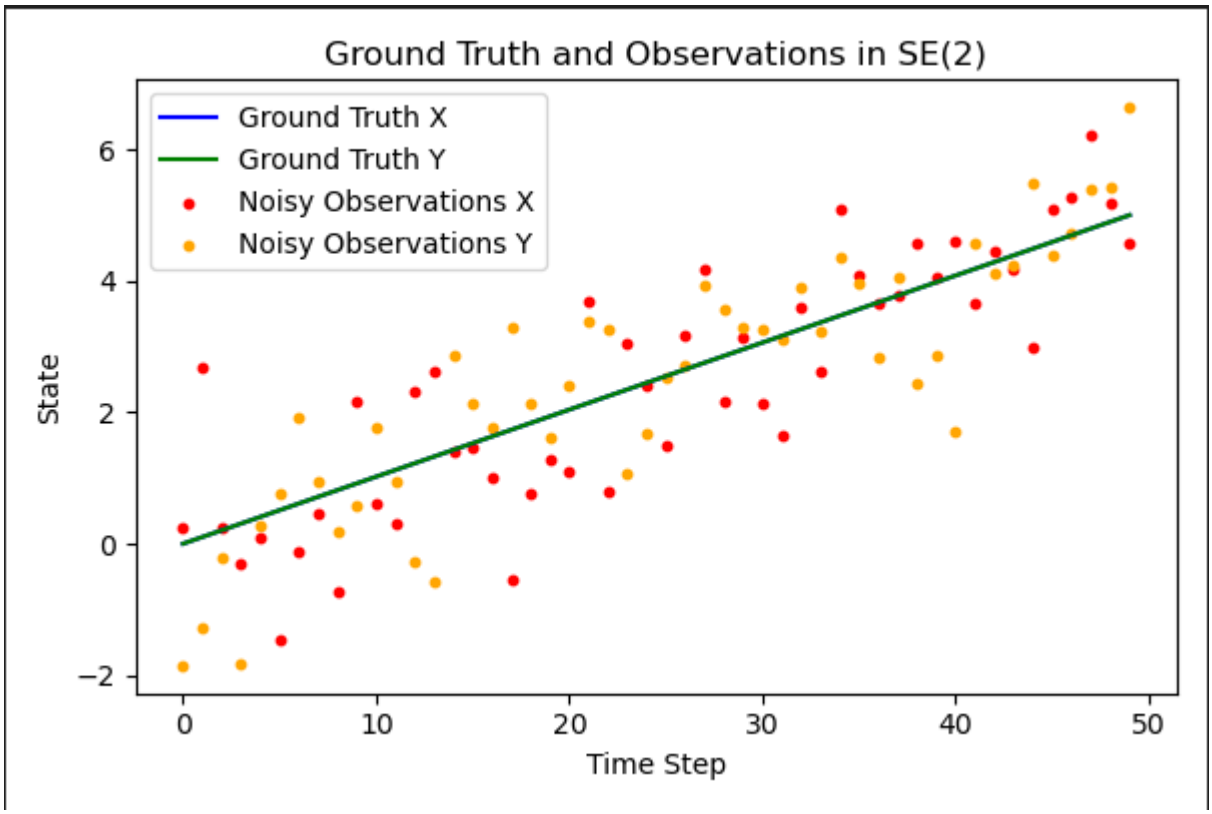


Extra Credit (Trajectory Optimization in SE(2)):

Factor Graph with constraints:



Trajectory With Constraints SE(2):


Ground Truth and Observations in SE(2)
Ground Truth X
Ground Truth Y
Noisy Observations X
Noisy Observations Y
State
Time Step
6
4
2
0
−2
0
10
20
30
40
50