Digital Twin Capstone Project

Report

Kareem Wael Abosamra

24th September, 2025

Introduction

1. **Introduction**
2. **System Architecture and Phase 1**
3. **Phase 2**
4. **Experiments**

Introduction

The project aims to make use of the VSI Fabric to simulate a line follower robot.

Where it is split into two phases:

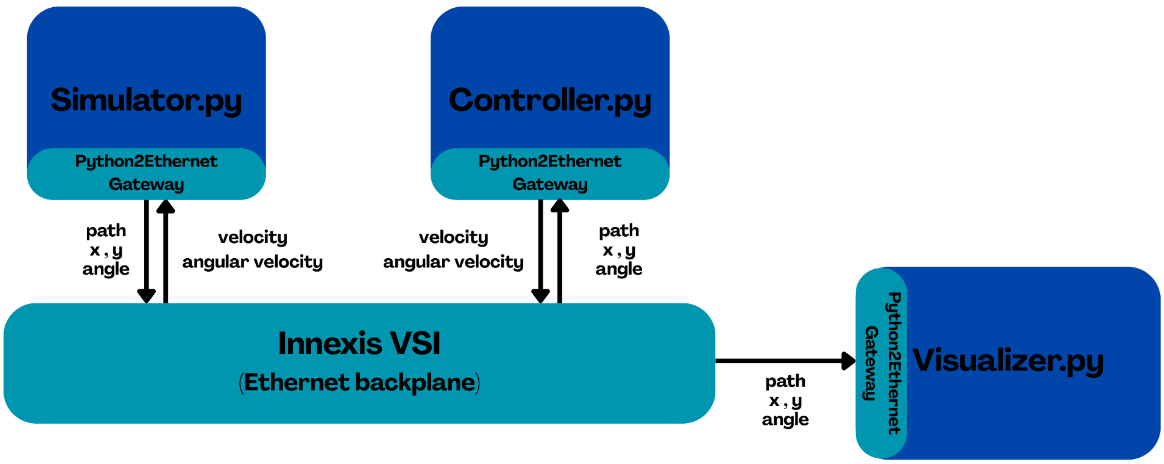
Phase 1: making a working prototype using Scokets (zmq)

Phase 2: Building a Workspace using Innexis VSI then Implementing the prototype into it

Tools used: Python - NumPy - Matplotlib for plotting - zmq for communication in Phase 1

System Architecture and Phase 1

System Architecture:



In Phase 1 I made the 3 clients with python (Simulator.py -Controller.py -Vistualizer.py) and ran them simultaneously as a standalone process.

Where:

1. Simulator

- **Purpose:** Models the robot's motion in the environment (a unicycle kinematic model).
- **Receives:**
  - Linear velocity command v (m/s)
  - Angular velocity command ω (rad/s)
- **Sends:**
  - Robot pose: (x, y, θ)
  - Path points (pathX, pathY) and path length pathLen
- **Mathematical model (unicycle kinematics):**

$$\dot{x} = v * \cos(\theta), \quad \dot{y} = v * \sin(\theta), \quad \dot{\theta} = \omega$$

2. Controller

- **Purpose**: Computes control actions to keep the robot on the path.
- **Receives**:
    1. Robot pose (x, y, θ) from Simulator
    2. Path points (pathX, pathY, pathLen)
- **Sends**:
    1. **Control commands:** linear velocity v and angular velocity ω
- **Math**:

    1. **Error computation**:
        - Find nearest path point (x_p, y_p) to robot (x, y).
        - Path tangent angle:

          $$\theta p = arctan2(yp + 1 - yp - 1, xp + 1 - xp - 1)$$

        - **Lateral error**:
          $el = (x - xp)(- sin\theta p) + (y - yp)(cos\theta p)$
        - **Heading error**:
          $eh = \theta - \theta p \ \ (wrapped\ to\ [- \pi, \pi])$
    2. **Control law (PID on combined error)**:

        - Error signal:
          $e = el + kh\, eh$
        - PID output for angular velocity:
          $\omega =- (Kpe + Ki\int edt + Kd\,\frac{de}{dt})$
        - Linear velocity modulated by path progress:
          $v = v_{nominal} \ \ (1 - min(\frac{|el|}{e_{max}}, 1))$

3. Visualizer

- **Purpose**: Observes the robot and evaluates performance.
- **Receives**:
  - Robot pose (x, y, θ) and path (pathX, pathY) from Simulator
- **Sends**:
  - Nothing (only logs and saves data/plots)

- **Math / KPIs**:
  - **Lateral error**:
    Distance from robot (x,y) to nearest path point
  - **Overshoot**:
    Maximum deviation beyond steady-state error
  - **Settling time**:
    First time the error stays within a tolerance band (e.g. ±0.05 m)
  - **Steady-state error**:
    Mean of error in the last window of simulation
  - **RMSE**:

    $$RMSE = \sqrt{\frac{1}{N}\sum_{i=1}^{N} e_i^2}$$

  - **IAE (Integral of Absolute Error)**:
    $IAE = \int |e(t)|dt \approx \sum|ei|\Delta t$

When i validated that it is the correct behavior i proceeded to Phase 2

Phase 2

First I implemented the System architecture by analyzing what I needed to define in the vsiBuildCommands from:

- Components
- Ports
- Signals
- Connections
- Simulation Configurations

Then i wrote the vsiBuildCommands and Built the system

Then i implemented my code with the following constrains:

- Due to the slowness of the VPC i made the simulation lasts only 10 seconds with the simulation step = 0.02 second (each run was about 5-7 minutes with offline plotting)
- Made the random spawning bounded so the robot doesn’t spawn to far from the path taking more time (the simulation could end before it reaches the path)

Experiments

Experiment 1:

I used different Gain sets, i ran each one **3 times on a straight line with random spawning and zero noise** and got these results:

# Set	Kp Ki Kd	run	Overshoot (m)	Settling Time (s)	Steady-state Error (m)	RMSE (m)	Max Deviation (m)	IAE (m·s)
		1	0.033	0.71	0.025	0.031	0.058	0.251
		2	0.238	2.31	0.031	0.076	0.268	0.444
		3	0.238	2.31	0.031	0.076	0.268	0.444
1	11.2 0.8 3.2	avg	0.1696666667	1.776666667	0.029	0.061	0.198	0.3796666667
		1	0.134	1.61	0.025	0.049	0.16	0.335
		2	0.039	0.51	0.024	0.03	0.063	0.251
		3	0.15	1.65	0.024	0.052	0.174	0.348
2	6.0 0.05 1	avg	0.1076666667	1.256666667	0.02433333333	0.04366666667	0.1323333333	0.3113333333
		1	0.038	0.39	0.024	0.03	0.062	0.25
		2	0.38	3.19	0.025	0.125	0.405	0.652
		3	0.308	2.75	0.025	0.095	0.333	0.527
3	8.0 0.1 0.8	avg	0.242	2.11	0.02466666667	0.08333333333	0.2666666667	0.4763333333
		1	0.058	0.49	0.025	0.035	0.084	0.293
		2	0.18	2.81	0.025	0.07	0.205	0.476
		3	0.094	1.49	0.025	0.044	0.119	0.338
4	4.0 0.2 0.3	avg	0.1106666667	1.596666667	0.025	0.04966666667	0.136	0.369
		1	0.172	2.09	0.026	0.057	0.198	0.35
		2	0.138	1.91	0.026	0.049	0.164	0.321
		3	0.213	2.29	0.027	0.068	0.24	0.387
5	8.0 0.2 2.4	avg	0.1743333333	2.096666667	0.02633333333	0.058	0.2006666667	0.3526666667

Noticing that Sets 1,2 and 5 got the best results I continued with them into the next experiment.

***You can check the output graphs in their folders respectively**

Experiment 2:

I used the best 3 sets on **Curved path with zero noise at the starting position (1,0.7)** and got the following Results:

# Set	Kp Ki Kd	Starting Point	Overshoot (m)	Settling Time (s)	Steady-state Error (m)	RMSE (m)	Max Deviation (m)	IAE (m·s)
1	11.2 0.8 3.2	(1,0.7)	0.081	0.65	0.031	0.069	0.112	0.575
2	6.0 0.05 1	(1,0.7)	0.065	0.65	0.024	0.052	0.089	0.089
5	8.0 0.2 2.4	(1,0.7)	0.094	0.71	0.045	0.084	0.139	0.694

Noticing that:

- Set 2 had the best results
- The overshoots after the first overshoot was more than that of the straight line
- The errors were less (this might be because of the starting position) compared to the straight line

***You can check the output graphs in their folders respectively**

Experiment 3:

I Tried **3 different levels of noise on the position and angle using set 2** and got the following results:

# Set	Noise Amount (pos, angle)	Starting Point	Overshoot (m)	Settling Time (s)	Steady-state Error (m)	RMSE (m)	Max Deviation (m)	IAE (m·s)
2	0.01, 0.005	(1,0.7)	0.103	4.95	0.092	0.112	0.195	0.938
2	0.05 , 0.01	(1,0.7)	0.366	N/A	0.242	0.308	0.608	2.568
2	0.1 , 0.05	(1,0.7)	0.639	N/A	0.734	0.685	1.373	5.671

Noticing that:

- The settling time was way worse than that of the denoise experiments with the 2nd and 3rd run not settling at all
- With increasing the value of the noise the amount of the covered path decreased

***You can check the output graphs in their folders respectively**

Experiment 4:

I Compared the PID vs PD controllers on a curved path with noise and got these results:

Controller	Noise Amount (pos, angle)	Starting Point	Overshoot (m)	Settling Time (s)	Steady-state Error (m)	RMSE (m)	Max Deviation (m)	IAE (m·s)
PID	0.01, 0.005	(1,0.7)	0.103	4.95	0.092	0.112	0.195	0.938
PD	0.01, 0.005	(1,0.7)	0.116	4.83	0.076	0.109	0.193	0.914

Noticing that weirdly the PD controller had better result in some metrics.

***You can check the output graphs in their folders respectively and you can check the results of all the experiments in the file "results"**

Thank You