(blank row)

OPTIMOTION

City University of Seattle

School of Technology & Computing

GROUP 5

Verónica Elze

[ElzeVeronica@CityUniversity.edu](mailto:ElzeVeronica@CityUniversity.edu)

Master of Artificial Intelligence

Sumit Chahar

[ChaharSumit@CityUniversity.edu](mailto:ChaharSumit@CityUniversity.edu)

Master of Data Science

1. **Abstract**

Optimotion, explores the mathematical modeling and optimization of robotic arm movement applying foundational concepts from linear algebra, vector calculus, and numerical methods. We simulate a simplified two-joint planar robotic arm with the goal of minimizing a custom-defined cost function that reflects distance to a target and optionally penalizes extreme joint angles. Using only NumPy and Matplotlib, we manually implement gradient descent to iteratively update joint angles and optimize motion. Forward kinematics determine end-effector positions, and numerical approximation is used to compute gradients that guide each step toward convergence. Visualizations illustrate cost reduction, motion improvement, and final pose accuracy. This project directly applies course concepts including matrix operations, partial derivatives, and multivariable function minimization, demonstrating how mathematical tools can drive interpretable, simulation-based robotic control. Stretch goals such as using supervised machine learning to predict joint angles are reserved for TP03. Deliverables include source code, simulation outputs, visual plots, and a documented analysis of performance.

**Keywords:** Gradient Descent, Robotic Arm Kinematics, Cost Function Minimization, Numerical Optimization, Forward Kinematics, Simulation-Based Visualization.

A hexagon with a white robot arm and text

AI-generated content may be incorrect.

**Figure 1** Team Optimotion logo representing the core focus of our project: applying mathematical and statistical concepts to optimize robotic arm motion. The articulated robotic arm and hexagonal badge symbolize the integration of linear algebra, vector calculus, and gradient-based optimization—key learning objectives of DS623. This visual identity reflects our commitment to simulation-driven modeling, analytical reasoning, and real-world data science applications in control systems.

1. **INTRODUCTION**

**Project Description**

This project implements a simulation of a two-joint, planar robotic arm optimized using gradient descent. The goal is to minimize a cost function that calculates the Euclidean distance between the end-effector and a target point, with an optional penalty term for joint angle magnitude. Forward kinematics is used to compute end-effector positions from joint angles, and numerical gradients guide the optimization. The Proof of Concept (PoC) includes complete function implementation, simulation, and visualization. Figures in this report show the robot arm’s pose before and after optimization, as well as a cost convergence plot over 50 iterations. Inspired by Mithi (n.d.), future work could include an animated stretch goal using Matplotlib’s FuncAnimation.

**Usefulness**

Optimizing robotic arm movements has significant applications in fields such as industrial automation, surgical robotics, and assistive devices. This simulation reinforces key machine learning principles, including cost function minimization, numerical differentiation, and iterative model refinement. The visual outputs (cost convergence and pose comparison) offer intuitive insights into how the optimization behaves over time. This project not only builds a strong foundation in applied optimization but also connects academic concepts with real-world robotics challenges (Carabin & Scalera, 2020).

**Data Set**

Synthetic data was generated internally by initializing joint-angle configurations and setting fixed target points. This allowed full control over the experimental setup and reproducibility of results. While integration with external datasets was initially explored—such as those from OpenAI Gym (Brockman et al., 2016), DeepMind Control Suite (DeepMind, 2020), and D4RL (Fu et al., 2020)—the final PoC relies solely on internally generated data. These external sources remain promising avenues for future extension and validation.

1. **LITERATURE REVIEW**

**Foundations of Robotic Arm Kinematics**

A comprehensive understanding of robotic arm mechanics begins with the basic architecture and principles governing their movement. As outlined by Standard Bots (2025), robotic arms comprise interconnected joints and segments powered by actuators that mimic muscular functions. Their flexibility and precision depend on the degrees of freedom and the arrangement of these joints, typically forming configurations akin to human limbs. These systems are designed to operate within a constrained "work envelope" and employ task-specific end-effectors such as grippers or tools. To mathematically represent the arm's motion, Craig (2005) introduces homogeneous transformation matrices that combine rotation and translation operations. His detailed explanation of the Denavit-Hartenberg (D-H) convention provides a structured approach to assigning coordinate frames to each link, facilitating consistent and scalable modeling of kinematic chains. This framework is essential for calculating forward and inverse kinematics in both planar and spatial robotic configurations.

Further elaborating on these principles, Lafifi (2009) differentiates between forward kinematics—calculating the end-effector’s position based on known joint parameters—and inverse kinematics, where joint values are derived from a desired position. His work reinforces the importance of transformation matrices and coordinating assignments in building accurate models for robotic motion. Similarly, Salman and Roman (2022) focus on practical implementations of the D-H method in real-world robotic systems. Their applied perspective helps translate theoretical modeling into effective simulations. Expanding on this, Qiu (2024) presents a targeted analysis of a 3-DOF RRR (revolute-revolute-revolute) manipulator. By applying D-H parameters to model its structure, Qiu demonstrates how motion and precision can be achieved through kinematic optimization for specific arm designs.

**Optimization in Robotics**

Optimization is central to efficient robotic control. Siciliano et al. (2009) highlight how cost functions—typically designed to minimize energy, time, or motion smoothness—are integrated into control systems using a variety of mathematical strategies. These include gradient descent for differentiable objectives, sampling-based planners for complex trajectories, and convex optimization for constrained problems. Optimal control techniques like Linear Quadratic Regulators (LQR) further refine system performance by minimizing specific indices. In a modern reinforcement learning context, Hassan and Sanaullah (2024) introduce a gradient-based inverse reinforcement learning (IRL) system. Their framework enables robotic arms to learn manipulation tasks directly from visual demonstrations. By extracting visual keypoints and training a predictive dynamics model, the system derives cost functions from behavior patterns and refines them using temporal-difference model predictive control (TD-MPC). This method advances unsupervised learning of control strategies in environments where explicit action data is unavailable.

Building on human-influenced behavior, Lee et al. (2012) propose a bimanual robotic system where each arm is guided by a cost function inspired by human task division. Using task-compatibility indices, they design a model where coarse and fine motor roles are strategically assigned to mimic natural coordination—enhancing overall efficiency in dual-arm systems. Energy efficiency, a growing concern in robotics, is addressed in Carabin and Scalera’s (2020) work. They develop a methodology for minimum-energy trajectory planning using dynamic and electromechanical modeling. Validated on linear Cartesian manipulators and servo-based test benches, their approach emphasizes sustainability in industrial systems by optimizing speed profiles for energy savings.

**Simulation and Control Benchmarks**

To evaluate and compare optimization strategies in robotic control, simulation frameworks play a critical role. Brockman et al. (2016) developed OpenAI Gym, a benchmark suite that standardizes reinforcement learning environments, including robotic simulations. Its consistent API enables reproducibility and comparative testing of diverse algorithms under unified settings. Similarly, the DeepMind Control Suite (2020) provides structured continuous control tasks built on the MuJoCo physics engine. These tasks, ranging from pendulums to manipulators, offer consistent reward schemes and dynamics models—making them ideal for developing and testing learning agents in a simulation context. The suite’s clarity and uniformity are especially beneficial for generating synthetic datasets. In the offline learning domain, the D4RL dataset (Fu et al., 2020) supports reinforcement learning by offering standardized datasets from locomotion and manipulation environments. This enables researchers to test algorithmic performance under limited interaction conditions, which is crucial for real-world applications like robotics and healthcare.

For simpler, code-driven simulations, Mithi (n.d.) provides a hands-on 2D robotic arm model implemented in NumPy. This project manually calculates forward kinematics and performs gradient descent to optimize joint configurations. Beyond its algorithmic clarity, it offers intuitive visualization tools using Matplotlib, making it an effective baseline for understanding robotic motion control without advanced libraries. The project’s structure directly influenced the simulation logic and visual outputs of our Optimotion model.

1. **METHODOLOGY**

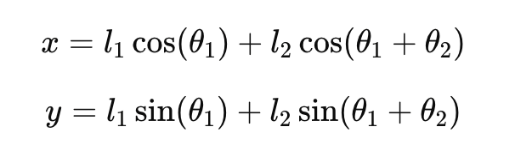
The Optimotion model simulates a two-joint planar robotic arm and employs gradient descent to optimize its movement toward a predefined target position. This methodology integrates mathematical modeling, numerical approximation, and visualization to iteratively refine the robot's motion path.

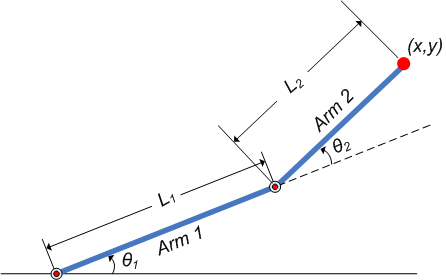
Using forward kinematics, the end-effector's position is computed based on joint angles. The cost function, defined as the Euclidean distance between the end-effector and the target, serves as the metric for optimization. To minimize this cost, gradients are approximated numerically, and the joint angles are adjusted iteratively. This process allows the robot arm to converge toward a more accurate configuration using purely mathematical formulations, without reliance on external machine learning libraries or pretrained models.

**Robotic Arm Configuration**

The simulated robotic arm is designed as a two-joint planar structure—commonly used in robotics for two-dimensional motion control. It consists of two rigid links connected by rotational joints. The first joint anchors the arm to a base, while joint angles θ₁ and θ₂ define the orientation of each link.

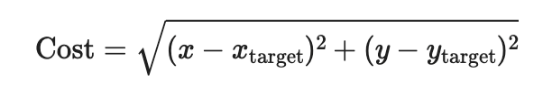
Using trigonometric functions and forward kinematics (Craig, 2005), the model calculates the position (x, y) of the end-effector from known link lengths (L1, L2) and joint angles. This transformation from joint space to Cartesian space enables precise tracking of the arm’s movement and target-reaching capabilities.



*Figure 1 This image demonstrates* ***forward kinematics*** *for a 2-link robotic arm.*

**Cost Function Design**

The cost function is a core component, calculating the Euclidean distance between the end-effector and the target. A lower cost indicates a better configuration, guiding the optimization process toward the desired position.



*Figure 2 The cost function calculates the Euclidean distance between the end-effector and the target position.*

The cost function outputs a single value representing the distance to the target. A lower cost indicates closer proximity to the desired position, which is crucial during the optimization process.

To promote energy-efficient movements, an optional penalty term can be added—drawing from Carabin & Scalera (2020). This term discourages large angular deviations, ensuring smoother transitions and longer hardware lifespan. In delicate applications such as surgery or precision assembly, this refinement promotes both functional efficiency and physical reliability.

**III. Gradient Descent Implementation**

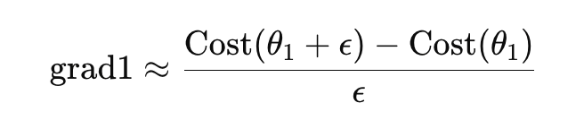
To optimize the movement of a two-joint planar robotic arm, we employ gradient descent, a technique for minimizing a defined cost function that measures the distance of the arm's end-effector from a target point. This section outlines the process of computing gradients and the iterative optimization strategy.

The first step in the optimization process involves computing the gradients of the cost function with respect to the joint angles of the arm. We utilize a numerical approach known as the finite difference method. This involves slightly perturbing each joint angle (theta1 and theta2) and evaluating the change in cost:

**1. Current Cost Calculation**

The current cost (distance to the target) is computed with the existing joint angles as shown in the section above.

**2. Perturbation of Angles**



*Figure 3 The first joint angle (theta1) is increased by a small value (epsilon) to see how the cost function is affected. The difference in cost gives an estimate of the gradient with respect to theta1.*

The same process is repeated for the second joint angle (theta2), allowing us to compute its gradient. The resulting gradients indicate how much to adjust each joint angle to decrease the cost effectively.

Once the gradients are determined, the next step is to update the joint angles iteratively:

**1. Initialization**

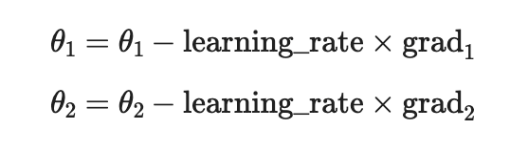
Begin with predefined initial angles for theta1 and theta2, as well as a set learning rate and number of iterations for the optimization process.

**2. Iterative Updates**

- For each iteration, calculate the current cost based on the current joint angles.

- Use the computed gradients to update each angle: the angles are adjusted in the direction opposite to the gradient (to minimize the cost function).

- Record the updated angles and the associated cost at each step for analysis of the optimization progress.

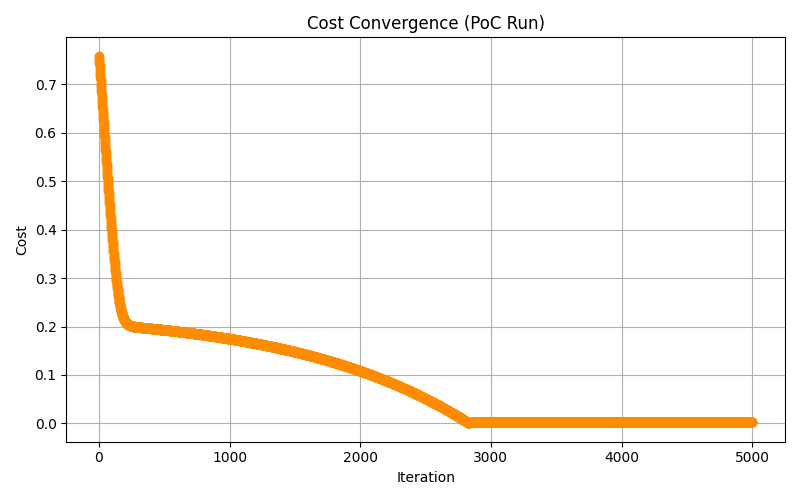


*Figure 4 The update rule for each joint angle*

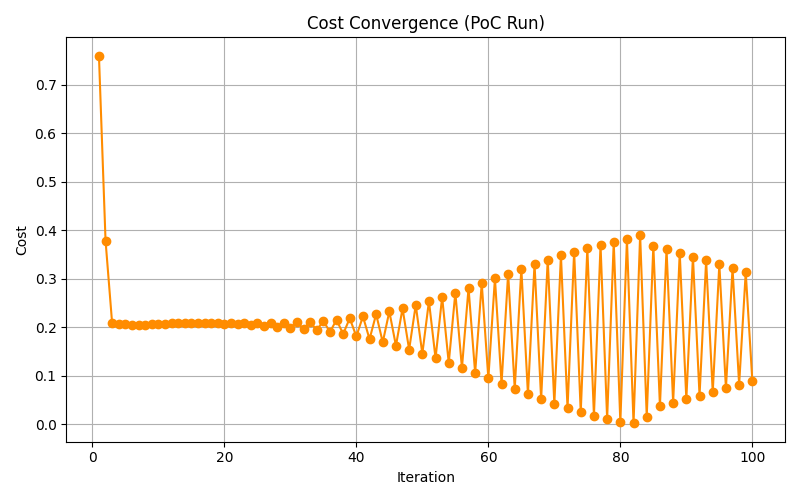
**3. Tracking Progress**

Maintain a history of angles and costs over iterations to assess the convergence and behavior of the optimization.

By systematically applying these steps, the optimization process converges towards the joint angles that minimize the distance to the target, thereby improving the performance and efficiency of the robotic arm's movements.

*Figure 5 Cost convergence (reduction) graph for 0.001 learning rate and 5000 iterations*

The cost reduction graph might oscillate and fail to converge if the learning rate used in the gradient descent optimization is too high. This is caused due to the optimizer overshooting the minimum, resulting in repeated fluctuations between values rather than smooth convergence. To resolve this, the learning rate needs to be reduced. The smaller step size allows the optimizer to approach the minimum more gradually, promoting stable and smooth convergence of the cost function.

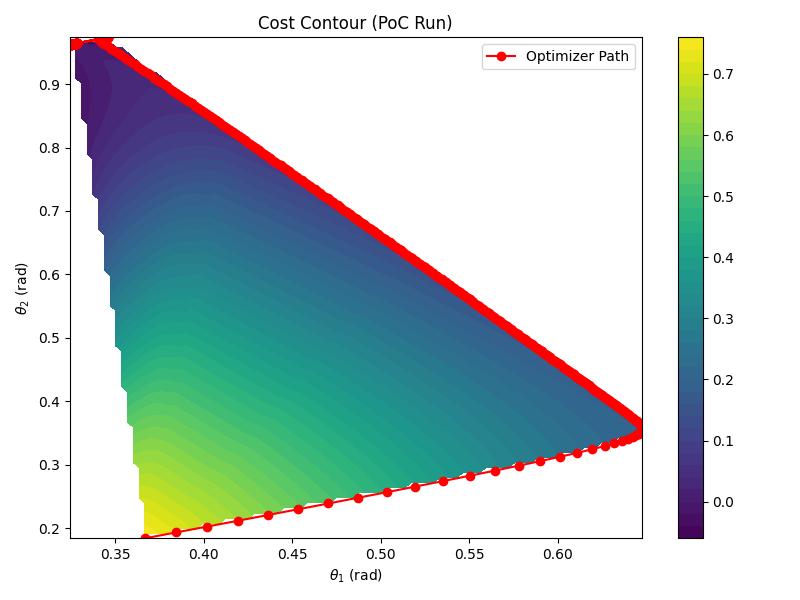
*Figure 6 Cost convergence (reduction) graph with high (0.1) learning rate. Notice that the optimizer overshot the minima due to that.*

**IV. Simulation Tools and Dataset Strategy**

Our implementation relies on:

* **NumPy** for numerical computations.
* **Matplotlib** for static visualizations and planned animation.
* **Synthetic dataset generation**: Random angle initialization and fixed target points.

While the TP02 implementation focuses on internally generated data, external frameworks such as OpenAI Gym, DeepMind Control Suite, and D4RL offer promising benchmarks for future validation and scaling.

*Figure 7 The 2D contour graph generated through matplotlib showing theta1 and theta2 changes along with minimizing cost. First, theta1 changes faster than theta2 but later theta1 gets smaller whereas theta2 increases to get the optimized cost.*

To verify the correctness of the core mathematical functions, unit tests were implemented for forward kinematics and cost calculation. These tests ensure that the end-effector reaches the expected position when joint angles are known, and that the cost function returns zero when the arm reaches the target. The tests are saved in /tests/test\_robotic\_arm.py.

1. **RESULTS**

Once the core simulation functionality was implemented, we generated dataset with 1000 instances and used various optimization techniques such as:

**1. Learning Rate Decay**

Learning rate decay is a technique that modifies the learning rate over time, usually by decreasing it gradually. This method is especially effective when high learning rates are desirable early in training to speed up convergence, while smaller learning rates are needed later to fine-tune the parameters around the optimal solution. In our implementation, we used an inverse decay formula: lr = initial\_lr / (1 + decay\_rate \* iteration). This results in a progressively smaller step size as iterations increase. The primary benefit of learning rate decay is that it balances speed and precision — it allows for rapid movement toward minima early on and prevents overshooting as the optimizer homes in on the best solution. In robotic arm optimization, this contributes to more stable and precise control as the joint angles approach their final values.

**2. Energy Penalty (L2 Regularization)**

Inspired by L2 regularization in machine learning, the energy penalty term penalizes large values of joint angles to simulate the concept of minimizing energy or effort. The term is added to the cost function as: λ \* (theta1² + theta2²). The idea is to discourage configurations where the arm extends to extreme positions, which may be energetically inefficient or physically stressful for motors and joints. Incorporating this penalty helps produce more compact and realistic arm movements, especially important in real-world settings where minimizing energy consumption or avoiding mechanical wear is a concern.

**4. Angle Change Penalty (Smoothness Regularization)**

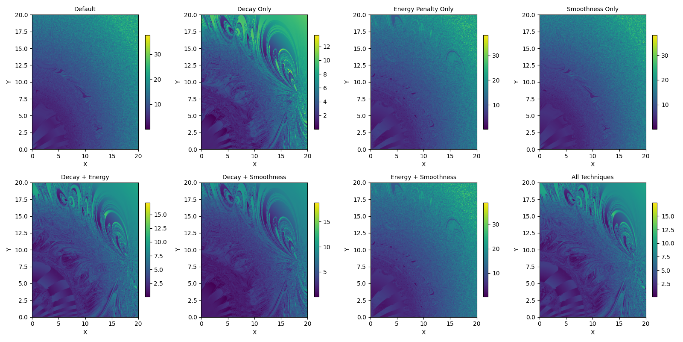
Smoothness regularization penalizes large changes in joint angles between consecutive iterations. This is implemented by comparing the current angle values with those from the previous iteration and adding a penalty term: λ \* ((theta1 - prev\_theta1)² + (theta2 - prev\_theta2)²). The purpose of this term is to promote gradual transitions and discourage abrupt angle updates. In physical systems, such abrupt changes could lead to mechanical instability, increased wear, or unnatural motion. By including this smoothness constraint, the optimizer learns to produce fluid, continuous trajectories, which is essential for robotics, animation, and human-interaction tasks where motion aesthetics and stability matter.

**5. Convergence Threshold (Early Stopping)**

Convergence thresholding is a stopping criterion based on the rate of improvement in the cost function. Rather than running a fixed number of iterations, the optimizer stops when the absolute difference in cost between iterations falls below a small threshold ε. This prevents unnecessary computations once the solution has stabilized. In our simulation, it allows the system to identify when further updates are no longer meaningful, effectively reducing computation time and avoiding overfitting noise or local fluctuations.

**6. Combination of Techniques**

Each technique described above addresses a specific limitation of vanilla gradient descent. When combined, they offer synergistic improvements. For instance, combining learning rate decay with energy penalty results in an optimizer that not only converges efficiently but also prefers more efficient arm configurations. Adding smoothness further improves the realism of the motion. Our heatmap visualizations show that combinations of all techniques together produce the best overall performance in terms of convergence quality of joint movement.

*Fig 7. Heatmap showing the cost impact of various optimization techniques*

The combined heatmap visualization clearly illustrates the impact of each optimization technique on the final cost of the robotic arm's movement. The plain gradient descent without any enhancements, exhibits higher final costs and smoother but less precise convergence patterns, particularly in the upper-right regions. When Learning Rate Decay is applied, the optimizer demonstrates significantly sharper convergence, as seen in the low-cost contours; however, this also introduces more complex behavior in some regions, likely due to abrupt changes in step sizes as the learning rate reduces.

The Energy Penalty alone helps regularize joint angle magnitudes, leading to more efficient movements, but doesn't significantly alter the overall convergence pattern compared to the baseline. In contrast, Smoothness Penalty promotes gradual transitions in joint angles, producing similar heatmap results to the default, with subtle improvements in continuity and reduced sharp fluctuations.

When techniques are combined, the results become more pronounced. Decay + Energy and Decay + Smoothness both lead to more defined cost basins, especially around central and upper regions, suggesting that learning rate decay synergizes well with regularization. The Energy + Smoothness pairing maintains a balance between compactness and continuity, although it lacks the aggressive convergence seen with decay.

Finally, the All-Techniques configuration yields the most refined optimization landscape — it produces the lowest cost zones with high consistency across a wide range of target positions. This indicates that combining decay, energy efficiency, and smoothness leads to optimal and physically realistic convergence.

Overall, these results validate the hypothesis that combining multiple optimization techniques enhances the behavior of gradient-based robotic arm control, both in terms of minimizing cost and producing smoother, more stable solutions.

1. **CONCLUSION**

This report has documented the design, implementation, and evaluation of a gradient-based optimization model for a two-joint planar robotic arm. A complete Proof of Concept (PoC) was developed to simulate motion, calculate cost, and iteratively update joint angles using manually implemented forward kinematics and numerical gradient descent. Visual outputs demonstrate the success of optimization in both cost convergence and pose improvement.

**1. Stretch goal**

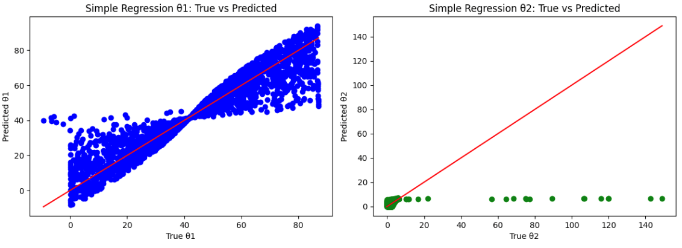
As a follow-up and stretch goal, we applied linear regression techniques to predict the joint angles theta1 and theta2 instead of calculating cost by gradient descent every time. These techniques were tested through two different approaches: simple linear regression (separate models for theta1 and theta2) and multivariate linear regression (a single model predicting both angles simultaneously). The dataset for these models was generated using the above gradient descent approach where target end-effector co-ordinates x and y were input and the final optimized angles were labels/outputs. The generated dataset was then broken down into train and test dataset using 80:20 split.

**2. Simple Linear Regression**  
In the first approach, we trained two separate linear regression models—one for predicting theta1 and another for theta2. Each model was trained independently using the input features. These models were fitted on the training dataset, and predictions for theta1 and theta2 were made separately for each target co-ordinate of test dataset. The performance of each model was evaluated using the **R² score**, which indicated how well each model captured the variance in the angles based on the input features.

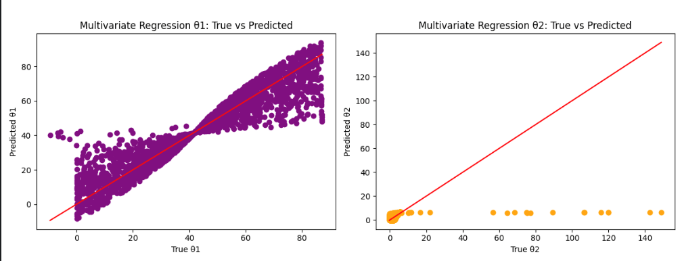
**3. Multivariate Linear Regression**  
In the second approach, we combined both theta1 and theta2 into a multivariate regression model. This approach allows the model to predict both angles simultaneously from the input features, treating the task as a multi-output prediction problem. This method enabled the model to learn a joint relationship between both angles and the input coordinates, which could potentially improve the predictive performance if there was any interdependence between theta1 and theta2.

**4. Interaction Term for Multivariate Regression**  
 To improve the performance of the multivariate regression model; we added an **interaction term** between *x* and *y* as an additional feature. The interaction term x\*y was included to help the model capture more complex relationships between the input features and the target angles. This transformation allowed the model to consider the combined effect of both *x* and *y* on the prediction of theta1 and theta2, thus accounting for any non-linearities in the data that a simple linear regression might miss.

**5. Evaluation and Comparison**:  
After training both the simple and multivariate models, we evaluated their performance using the R² score. The R² score for each model indicates the proportion of variance explained by the model.



*Fig. 8 R2 scores graph of theta1 and theta2 predication using simple linear regression*



*Fig. 9 R2 scores graph of theta1 and theta2 predication using multivariate linear regression*

The Simple Linear Regression for theta1 showed an R² score of 0.8668, indicating that the model explains approximately 87% of the variance in theta1, suggesting a strong linear relationship between the target position (x, y) and the first joint angle. However, the R² score for theta2 was only 0.0663, showing that the simple model struggles to predict theta2, capturing only 6.63% of its variance. This highlights that the relationship between (x, y) and theta2 is more complex. The Multivariate Linear Regression model, which predicted both angles simultaneously, achieved an R² score of 0.4666, meaning it explained 46.66% of the variance for both theta1 and theta2 combined. While this is a moderate fit, the model still underperforms, especially for theta2, when compared to the simple regression for theta1.

**6. Future Work**

To improve the performance of the model, particularly for theta2, a non-linear model would be a promising direction. Since the current linear models fail to capture the complexity of the relationship between the input coordinates (x, y) and theta2, non-linear techniques like Decision Trees, Random Forests, or Neural Networks could be explored. These models are better suited to capturing non-linear relationships and interactions between features, which may be critical for accurately predicting theta2. Furthermore, exploring feature engineering techniques such as polynomial features, kernel methods, or deep learning models could significantly enhance model accuracy. A non-linear approach would likely offer more flexibility in modeling the complex dependencies between the input variables and the joint angles, leading to better performance across both angles.

1. **REFERENCES**

Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J., & Zaremba, W. (2016). *OpenAI Gym* [Preprint]. *arXiv*. <https://arxiv.org/abs/1606.01540>

Carabin, G., & Scalera, L. (2020). On the trajectory planning for energy efficiency in industrial robotic systems. *Robotics*, *9*(4), 89. <https://doi.org/10.3390/robotics9040089>

Craig, J. J. (2005). *Introduction to robotics: Mechanics and control* (3rd ed.). Pearson Education.

DeepMind. (2020). *DeepMind Control Suite*. <https://github.com/deepmind/dm_control>

Fu, J., Kumar, A., Nachum, O., Tucker, G., & Levine, S. (2020). *D4RL: Datasets for deep data-driven reinforcement learning* [Preprint]. *arXiv*. <https://arxiv.org/abs/2004.07219>

Hassan, M. S., & Sanaullah, S. M. (2024). *Robotic arm manipulation with inverse reinforcement learning & TD-MPC* [Preprint]. *arXiv*. <https://arxiv.org/abs/2407.12941>

Lafifi, M. M. (2009). *Kinematic analysis for robot arm*. ResearchGate. [<https://www.researchgate.net/profile/Mohamed-Mourad-Lafifi/post/What_is_the_best_platform_for_implementing_inverse_kinematics_of_a_6-axis_robotic_arm/attachment/5fd8c1de3b21a20001622f80/AS%3A969008031477765%401608040926850/download/KINEMATIC%2BANALYSIS%2BFOR%2BROBOT%2BARM.pdf>](https://www.researchgate.net/profile/Mohamed-Mourad-Lafifi/post/What_is_the_best_platform_for_implementing_inverse_kinematics_of_a_6-axis_robotic_arm/attachment/5fd8c1de3b21a20001622f80/AS%3A969008031477765%401608040926850/download/KINEMATIC%2BANALYSIS%2BFOR%2BROBOT%2BARM.pdf)

Mithi. (n.d.). *2D robotic arm simulation using NumPy* [Jupyter Notebook]. GitHub.

Qiu, X. (2024). Forward kinematic and inverse kinematic analysis of a 3-DOF RRR manipulator. In Y. Yue (Ed.), *Proceedings of the 2024 International Conference on Mechanics, Electronics Engineering and Automation (ICMEEA 2024)* (pp. 532–539). Atlantis Press. [<https://doi.org/10.2991/978-94-6463-518-8_50>](https://doi.org/10.2991/978-94-6463-518-8_50)

Salman, A. E., & Roman, M. (2022). *Robot kinematics*. ResearchGate. [<https://www.researchgate.net/publication/365193182_ROBOT_KINEMATICS>](https://www.researchgate.net/publication/365193182_ROBOT_KINEMATICS)

Siciliano, B., Sciavicco, L., Villani, L., & Oriolo, G. (2009). *Robotics: Modelling, planning and control*. Springer.

Standard Bots. (2025, April 23). *How do robotic arms work? A comprehensive 2025 guide*. [<https://standardbots.com/blog/how-do-robotic-arms-work-a-comprehensive-guide>](https://standardbots.com/blog/how-do-robotic-arms-work-a-comprehensive-guide)

# Appendix A: GitHub Repository

To promote transparency, collaboration, and reproducibility, all source code, figures, simulation outputs, and draft documents associated with the Optimotion project are version-controlled and publicly available (or available to the instructor) through our team's GitHub repository. All PoC development (code, visual outputs, and notebook simulation) was completed by Verónica Elze. GitHub repo contributions and visualizations cited in this report reflect this work.

**Repository Information**

Repository Name: Optimotion-DS623

URL: https://github.com/YourGitHubUsername/Optimotion-DS623

**Repository File Structure**

/config

    READMEcfg.md           # Placeholder for future configuration files

/docs

    REFERENCES.md           # APA-formatted references for TP01 & TP02

/notebooks

    run\_poc\_sim.ipynb       # Complete PoC notebook for TP02 Appendix B

    run\_poc\_sim.html/pdf   # Exports of the notebook for TP02 submission

run\_multi\_target\_sim.ipynb # Generate and calculate costs for multiple target co-ordinates

run\_optimized\_simulation.ipynb # Optimized final simulation

[linear\_regression\_predictor.ipynb](https://github.com/MissVz/DS623-Optimotion/blob/main/notebooks/linear_regression_predictor.ipynb) # Linear regression predictor for TP03 submission

    READMEnb.md             # Notebook folder overview

/outputs

/multi\_target

Heatmap\_final\_costs.png # Heatmap of final costs while calculating for multiple targets

/optimized

Optimzed\_costs\_heatmap.png # Heatmap of costs after applying multiple optimzation techniques

/regression

simple\_linear\_regression\_performance.png # Simple linear regression accuracy plot

multivariate\_linear\_regression\_performance.png # Multivariate linear regression accuracy plot

    poc\_cost\_convergence.png # Line plot of cost function values across optimization iterations

    poc\_pose\_comparison.png # Side-by-side plot of initial and optimized robotic arm configurations

    run\_poc\_sim.html/pdf   # Output copy for portability

    READMEouts.md           # Description of plots and simulation results

/src

    /data

        READMEdata.md       # Notes: no persistent datasets used in TP02

    /visualizations

        cost\_convergence\_plot.py # Plots cost vs. iteration to show optimization progress

        fwd\_kinematics\_plot.py # Visualizes a single arm pose from joint angles

        pose\_comparison\_plot.py # Compares initial vs. optimized arm configurations

        READMEviz.md       # Overview of plot script responsibilities

    optimizer.py           # Cost, gradients, and optimization loop

    robotic\_arm.py         # Forward kinematics calculation

/tests

    test\_robotic\_arm.py     # Unit tests for forward kinematics and cost function

/.gitignore # Files and folders to be excluded from version control

requirements.txt # Python packages required to run the Optimotion simulation & outputs

README.md # Overview of project, structure, technologies, and instructions

**Technologies Used**

Python 3.11+

NumPy

Pandas

Matplotlib

JupyterLab

GitHub for version control

**Usage Notes**

Code execution instructions and environment setup are detailed in the repository's README.md.

A requirements.txt file is included for easy environment replication.

# Appendix B: Visualizations and Simulation Outputs

This appendix provides an overview of the anticipated and final visualization components developed for the Optimotion project. These outputs support the TP03 Methodology and demonstrate the behavior and performance of the optimized robotic arm as calculated through forward kinematics and gradient descent.

**Summary of Visual Outputs**

|  |  |  |
| --- | --- | --- |
| Visualization | Anticipated for TP03 | Completed in TP03 |
| Multi target simulation heatmap | ✅ | ✅ |
| Simple linear regression prediction accuracy | ✅ | ✅ |
| Multivariate linear regression prediction accuracy | ✅ | ✅ |
| Multiple optimizations impact on cost heatmap |  | ✅ |

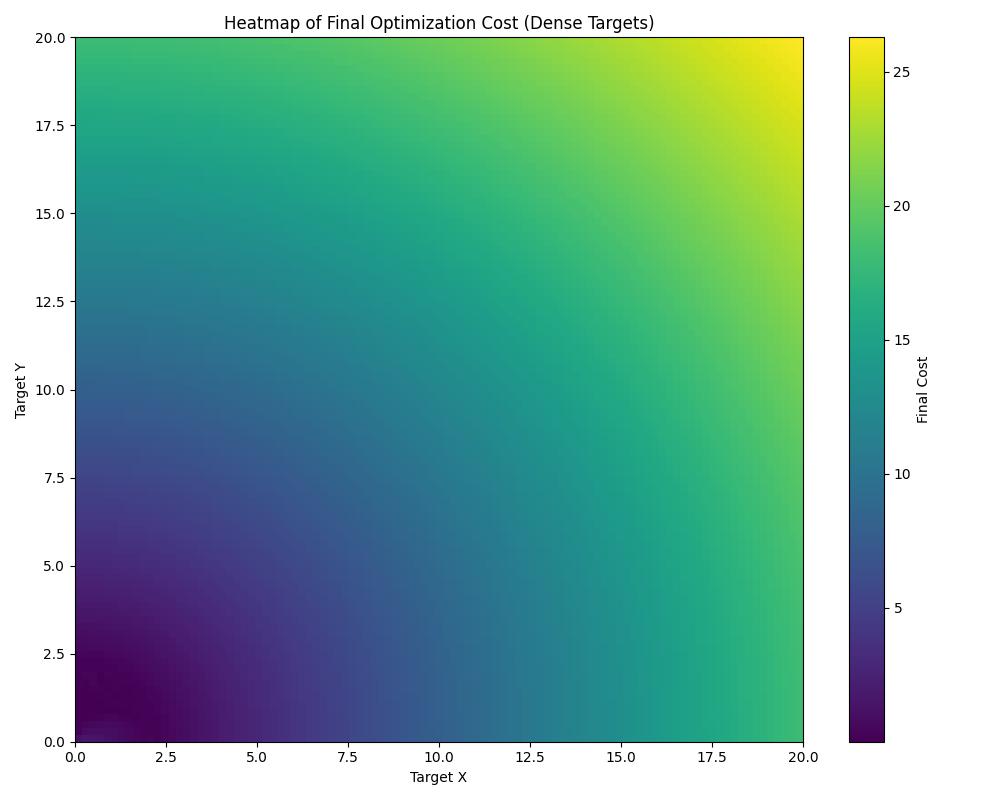


Figure 1e

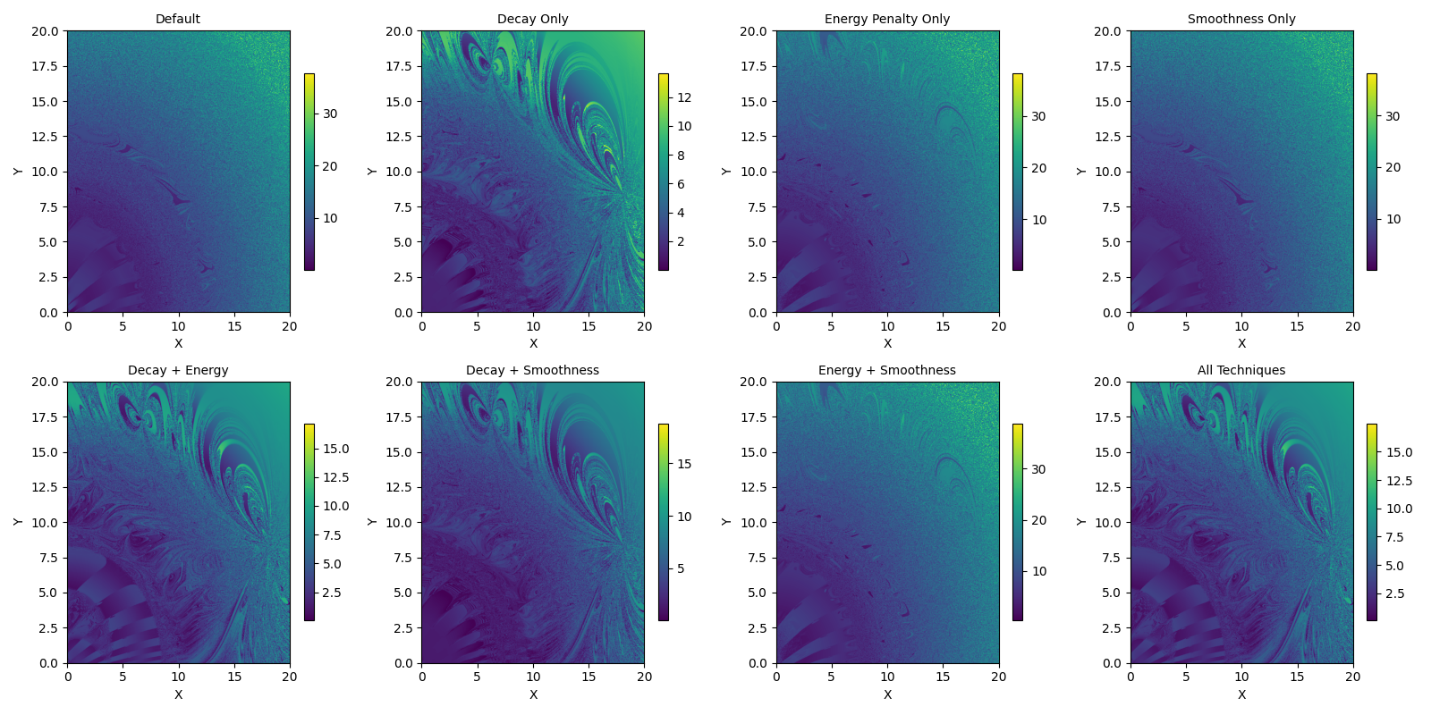


Figure 2 Cost heatmap after using combination of multiple optimization techniques for gradient descent

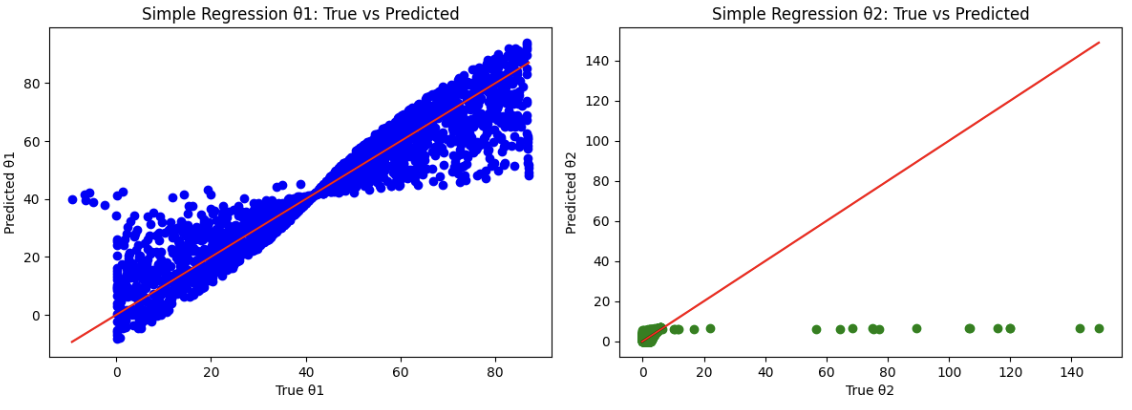


Figure 3 Simple regression predicts θ₁ well but fails to capture the variance in θ₂, leading to low prediction accuracy

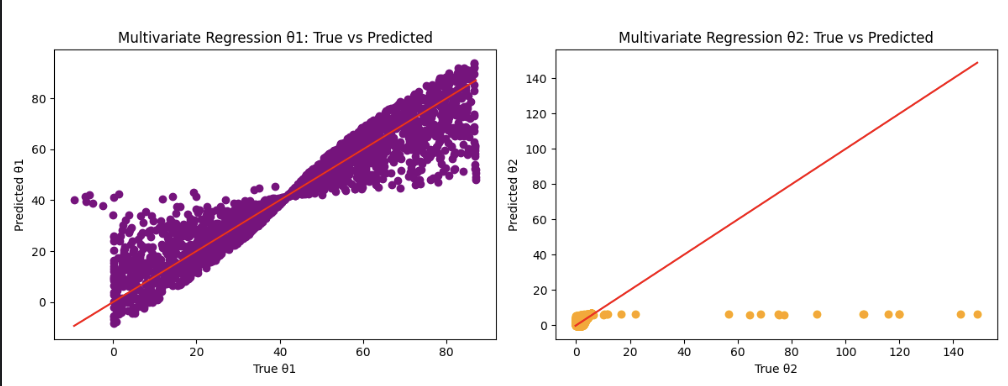


Figure 4 Multivariate regression predicts θ₁ well but fails to capture the variance in θ₂, leading to low prediction accuracy

**Output File Details**

All visualization files are saved in the /outputs/ folder of the GitHub repository.

|  |  |
| --- | --- |
| Filename | Description |
| multi\_target/heatmap\_final\_costs.png | Cost heatmap after running vanilla gradient descent on multiple targets |
| optimized/heatmap\_optimized\_costs.png | Cost heatmap showing reduced cost after applying various optimization techniques for gradient descent |
| regression/ simple\_linear\_regression\_performance.png | (Optional) Accuracy graph for predicting theta1 and theta2 using linear regression |
| regression/ multivariate\_linear\_regression\_performance.png | (Optional) Accuracy graph for predicting theta1 and theta2 using multivariate linear regression |

**Technical Summary**

Visualizations are generated using:

* matplotlib.pyplot for all static visualizations
* numpy for all coordinate and cost computations
* Plot functions stored in below /src/notebooks/ (at the end of each notebook):
  + run\_multi\_target\_sim.ipynb
  + run\_optimized\_simulation.ipynb
  + linear\_regression\_predictor.ipynb

# APPENDIX C: WORKLOAD ASSIGNMENT TABLES

The workload assignments below reflect our alignment with the DS623 Team Project Final Report and Final Presentation instructions. Per the course guidelines, the TP03 report requires original analytical writing, a literature review (four sources per student), and a detailed explanation of modeling methodology and results. Sumit serves as the Primary Lead for TP03, responsible for the full development of the written report, findings, and MVP improvements, while Verónica serves as Reviewer and Tester, ensuring accuracy and completeness based on the completed PoC. For TP04, Verónica leads the 15-minute presentation development, coordinating visualizations and content delivery, with Sumit contributing as Reviewer and Narrator. This structure ensures that all report and presentation deliverables meet course expectations for professional collaboration, writing quality, and data-driven analysis.

**TP03 FINAL REPORT**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Section / Task | | Primary / Lead | Reviewer / Tester | |
| Multi-target Simulation | | Sumit | | | Verónica | |
| Simple Linear Regression Predication Model | | Sumit | | | Verónica | |
| Multivariate Linear Regression Prediction Model | | Sumit | | | Verónica | |
| Learning Rate Decay Optimization | | Sumit | | | Verónica | |
| Penalty Term Exploration (λ) Optimization | | Sumit | | | Verónica | |
| Convergence Threshold Optimization | | Sumit | | | Verónica | |
| Angle Change Penalty Optimization | | Sumit | | | Verónica | |
| Final Abstract | | Sumit | | | Verónica | |
| Final Introduction | | Sumit | | | Verónica | |
| Final Literature Review | | Sumit | | | Verónica | |
| Final Methodology | | Sumit | | | Verónica | |
| Findings / Results | | Sumit | | | Verónica | |
| Future Work | | Sumit | | | Verónica | |
| Conclusion | | Sumit | | | Verónica | |
| Final References | | Sumit | | | Verónica | |
| Final Development/Modeling (MVP) | | Sumit | | | Verónica | |
| Final Cloud Configuration (if applicable) | | Sumit | | | Verónica | |
|  | | **17** | | |  | |

**TP04 PRESENTATION**

|  |  |  |
| --- | --- | --- |
| Slide / Task | Primary / Lead | Reviewer / Narrator |
| Goal & Purpose | Verónica | Sumit |
| Background | Verónica | Sumit |
| Related Work / Research | Verónica | Sumit |
| Approach / Process | Verónica | Sumit |
| Descriptions | Verónica | Sumit |
| Visualizations / Metrics | Verónica | Sumit |
| Demonstration 1 (MVP) | Verónica | Sumit |
| Demonstration 2 (Cloud Sim or Stretch Goal) | Verónica | Sumit |
| Conclusion | Verónica | Sumit |
| Future Work | Verónica | Sumit |
| Key References | Verónica | Sumit |
|  | **11** |  |