

Optimizing PID controller using Genetic Algorithm

Introduction

PID controllers are one of the most used control architectures in the automation industry. This is due to the relative simplicity of the model, while achieving high efficiency and flexibility to perform a wide range of tasks. However, the primary disadvantage of using a PID controller is the lack of analytical methods to derive the desired coefficients i.e., K_p , K_I and K_D , corresponding to each relevant error signal. In most application scenarios the coefficients corresponding to the use-case need to be intuitively determined through trial and error. A better approach would be the usage of numerical methods to find the optimal combination of coefficients. To redesign the problem as an optimization of design variables, an objective function needs to be designed which incentivizes the desired behavior above any other. We also need to consider constraints corresponding to the feasibility of the design.

In most applications the objective evaluation of any design cannot be expressed as a unimodal, purely convex function. This leads to a scenario with multiple local and a singular optimum. In case of mathematical methods such as gradient descent, the unimodality of the function is often a prerequisite. Classic optimization algorithms such as gradient descent often converge to the nearest minima without the guarantee of it being the global best. In non-traditional nature inspired optimization algorithms, this drawback is circumvented. One such popular algorithm is the evolutionary genetic algorithm. For such an algorithm there is no guarantee of convergence to the global minima, but often manages to do so given enough time and resources due to its varied and efficient search process.

In our case we apply this problem to an open world navigation setup where a differential drive two-wheeled ground vehicle needs to reach a pre-designated target in a non-contested obstacle map.

Problem Formulation

➤ Simulation Environment

The environment corresponding to the control and navigation of the robot was prepared from scratch in Pygame and other Python simulation environments. The PID controller is designed such that it generates the desired linear and angular velocities, which is incorporated in the model using Euler's method. The x, y coordinates define the positions and the angle θ define the orientation of the robot in the global coordinate system. The vector v determines the linear velocity, while ω determines the angular velocity of the robot. The kinematics of the mobile robot TURTLEBOT 2 is described by the

following equation in the form in which the control signals are the linear and angular velocities of the robot:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} v \\ \omega \end{bmatrix}$$

The robot's velocities are denoted as: v_x —forward velocity, v_y —lateral velocity, ω —angular velocity. The driving forces of the robot are denoted as F_{rrx}, F_{rry} —the longitudinal and lateral force of the right wheel and F_{rlx}, F_{rly} —the longitudinal and lateral force of the left wheel. The dynamics model of the robot was formulated by performing a law of conservation of linear and angular momentum, in the coordinate system associated with the robot, following:

$$\begin{aligned} \sum F_x &= F_{rlx} + F_{rrx} = m(\dot{v}_x - v_y\omega) \\ \sum F_y &= F_{rly} + F_{rry} = m(\dot{v}_y - v_x\omega) \\ \sum M_z &= I_z\omega = \frac{d}{2}(F_{rrx} - F_{rlx}) \end{aligned}$$

In our simulations the robot was fitted with a LIDAR sensor, which would detect the distance to the nearest obstacle within a finite limit. This LIDAR system is used as the collision detection system. When the distance to the closest obstacle becomes equal to the robot's size, a collision has occurred.

➤ Controller Design

In the current experiment the movement of the robot is controlled by a non-adaptive PID controller. Therefore, the bot needs to navigate based on a fixed set of coefficients corresponding to the PID controller. This unique set of coefficients is considered to be the design variable for the controller. The controller needs to provide control commands corresponding to linear and angular velocities to the plant or the robot. In order to encapsulate maximal information about the environment, a vector of error signals is considered. The error vector consists of the following:

1. **Positional error:** Corresponds to the distance from target
2. **Directional error:** Corresponds to the deviation in travel direction from the straight-line path to target
3. **LIDAR signal:** Corresponds to the obstacle distance array as per the LIDAR

For collision detection LIDAR rays uniformly spread across 360° are used. For the control decision only the LIDAR in the forward-facing 180° azimuth angle is considered. As a design choice, the controller coefficients for determining angular velocity corresponding to complementary LIDAR distances are said to be negatives of each other. This ensures rotational balance in a symmetrically perceived environment. The coefficients or design variables corresponding to such a controller are then determined using the genetic algorithm.

➤ Genetic Algorithm

Genetic algorithm is one of the most popular non-traditional optimization algorithms which is inspired by the natural process of evolution. It simultaneously considers multiple randomly selected candidate solutions which form a population. For each member in the population a binary string called the chromosome is specified which uniquely translates to a set of design variables. In our problem, each design variable is represented by 8 binary digits which means that the range of search for each design variable is divided into 2^8 with this resolution the optimal choice of design variables is determined by the algorithm. Assuming convergence to the global optima, the maximum error corresponding to the optimal choice of the design variables is given by the following equation,

$$\Delta x = \frac{x^{(u)} - x^{(l)}}{2^q - 1}$$

In this case, given the range of search $[X_l, X_u]$, the value of design variable can be decoded from the corresponding binary block as follows:

$$x = x^{(l)} + \frac{x^{(u)} - x^{(l)}}{2^q - 1} \sum_{k=0}^q 2^k b_k$$

Where X_l and X_u are the lower and upper limits of the search range respectively.

The algorithm searches for the optimal set of design variables, using three genetic operators: Reproduction, Crossover and Mutation

▪ **Reproduction**

The process of reproduction emulates that of evolution and natural selection whereby each chromosome is assigned a fitness value based on its performance in a simulation environment. Probabilities of retention of each chromosome are determined by normalizing the fitness values or the performance scores. Therefore, a better performing chromosome will have a higher chance of retention while the worse chromosome has a higher chance of extinction based on this discrete probability mass function a cumulative probability distribution is determined which demarcates boundaries dividing the finite range of $[0, 1]$ into segments whose size is proportional to each member's fitness and number is equal to the population size. For the determination of each member in the offspring generation, a random number is selected between 0 to 1 and the member corresponding to the bucket in which the random number lands is selected for the following generation. This method of selection or sampling of the offspring is called the Roulette wheel method. In this experiment the simulations were elitist, where the highest performing member is guaranteed to be carried over. This promotes the selection of the better performing genes and elimination of the worst performing ones over multiple generations.

▪ **Crossover**

In this experiment crossover is implemented as swapping of blocks corresponding to design variables between 2 parents before passing the chromosome onto the offspring. A standard single point crossover induces too many perturbations for the system to converge in this problem. The blocks corresponding to each variable are crossed over between parents based on a crossover probability ($p_c = 0.001$). The elite chromosome is unaffected by crossover, which thereby preserves the best solution from degradation while promoting the spreading of favorable characteristics throughout the population.

▪ **Mutation**

Before finally passing on the chromosomes to the offspring generation, the final genetic operation of mutation is performed. In this process, random genes are mutated, corresponding to flipping of bits in the chromosome based on a mutation probability ($p_m = 0.0001$). This operation provides a way for the algorithm to search through nearby regions in the search space that helps in exploration of the design space. It is by this method, that randomness is incorporated, and better genes are found. Once again, the elite chromosome is immune from any mutation to prevent degradation of the best solution. Given enough time and resources, the algorithm will converge to the globally optimal solution.

➤ Design of the objective function

The objective function needs to be defined such that irrespective of any local maxima, the global maxima should correspond to the design solution which allows the robot to successfully navigate to the target while avoiding obstacles. The objective function is designed as a rewards function to be maximized, as follows:

1. **Completion:** Large bonus reward for reaching the target.
2. **Collision:** Large negative penalty for obstacle collision
3. **Heuristic:** Penalty proportional to distance from target at end of episode
4. **Deviation:** Penalty proportional to average distance from target in simulation
5. **Pace:** Penalty proportional to number of time steps bounded by a finite episode length, which is applied only if the episode is not terminated by collision

Simulation Results

➤ Experimental Setup

The model was used with 8 LIDAR rays for collision detection, 5 of which are for control. Therefore, the error vector length was 5 (3 for LIDAR distances and 2 for positional and directional error). Therefore, there were 15 design variables (K_p , K_I and K_D corresponding to each error signal). The range of values was uniformly set to $[-0.1, 0.1]$ each design variable was represented by 8 bits in the binary chromosome ($q=8$). Thus, the chromosome was a binary string of length $n*q = 15*8 = 120$. These simulations considered a population of size 50 with all of them simulated and evaluated in parallel using Python multiprocessing. The population was updated through iterations of reproduction, crossover (crossover probability=0.001) and mutation (mutation probability=0.0001) as a generation. The algorithm was used for 20 generations for learning and evolution.

Using the above settings the simulations yielded the following results.

1. **Optimality:** Given the above settings the robot can find nearly optimal solutions for navigation in complex environments with use of extremely limited computational resources, the algorithm converges to the optimal design very quickly. The optimal path followed in a complex environment and the corresponding optimality are shown in the following figure,

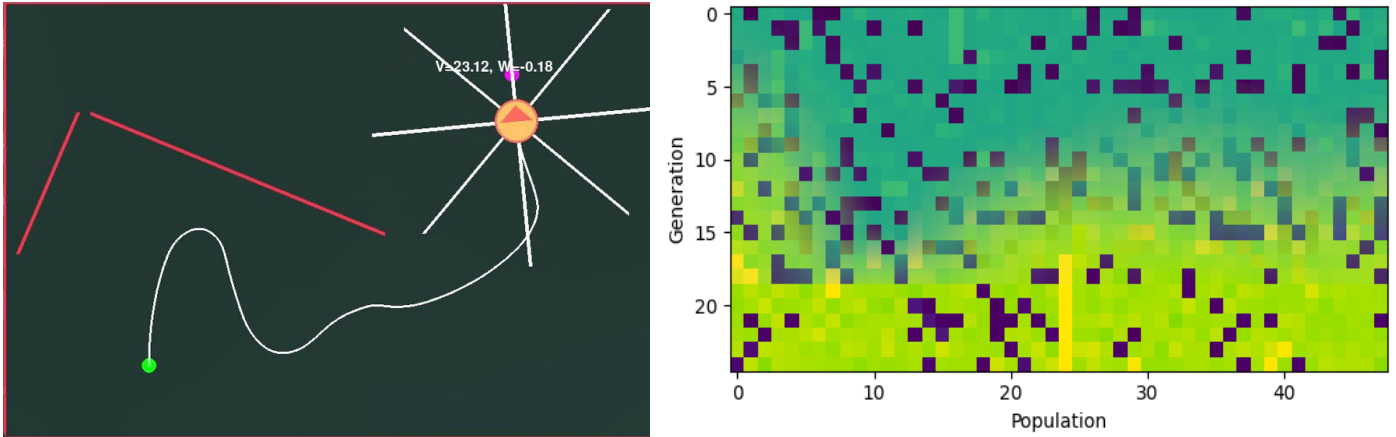


Fig: Bot's training results (warmer color means higher fitness)

- II. **Transferability:** The algorithm can find optimal controller design, not only for the current environment, but on a more universal level. The adaptive nature of the controller forces the algorithm to learn the principles of navigation as opposed to specific directions corresponding to a specific map. As a result, the bot becomes capable of navigating even in completely new unobserved maps which are much more complex than the training map itself.

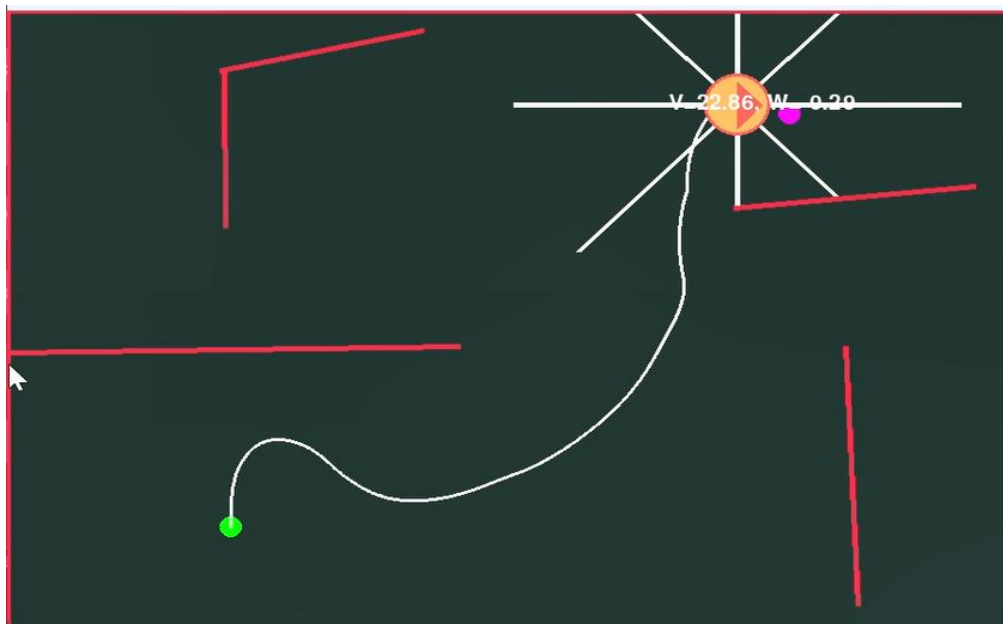


Fig: Transferability of the learned coefficients

- III. **Robustness:** Even in such small scaled simulations with just 50 members in a population which evolve for only 20 generations, the algorithm is robust enough to converge to the optimal solution out of the 2^{120} feasible design solutions on multiple runs.

➤ Disadvantages of traditional optimization algorithm

Traditional optimization algorithms can also be used for potential navigation controllers but they present their own drawback. There are many efficient uni-variate optimization algorithms such as Golden Ratio Search, Bisection Method, Secant Method, Interval Halving Method as shown (for a quadratic single variable objective function):

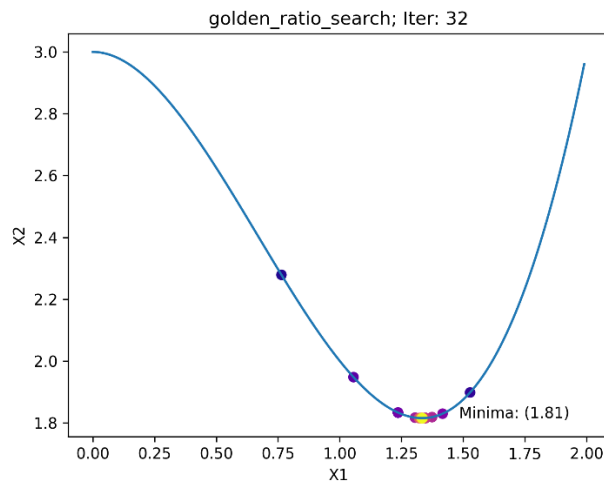


Fig: Golden Ratio Search with 32 iterations

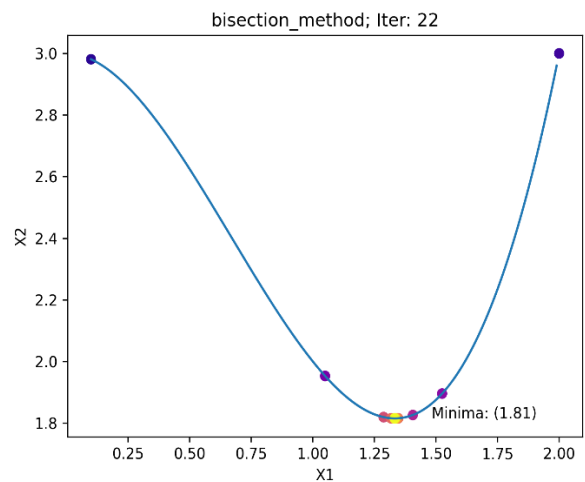


Fig: Bisection Method with 32 iterations

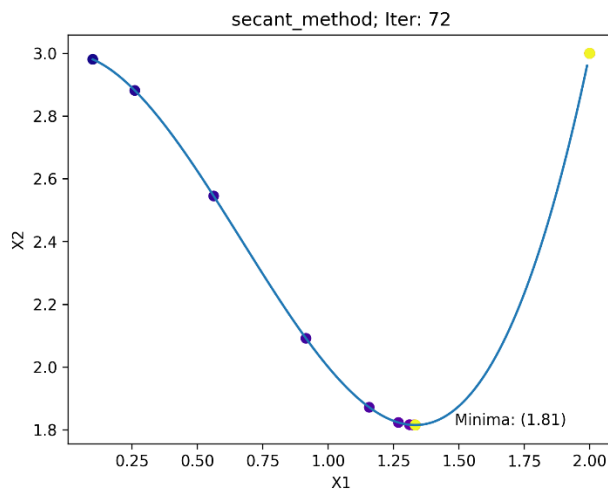


Fig: Secant Method with 32 iterations

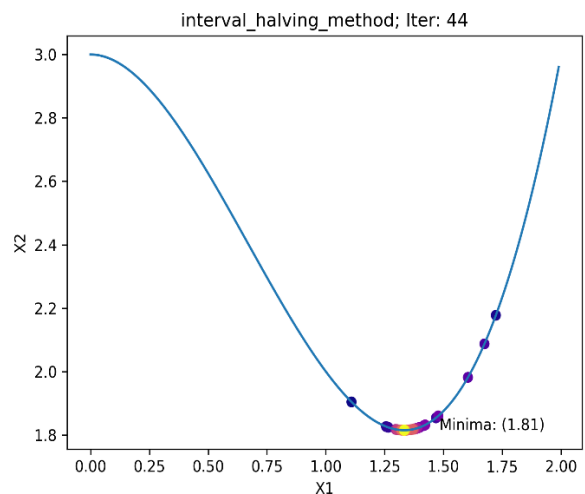


Fig: Interval Halving Method with 32 iterations

These univariable optimization algorithms can then be used for multi-variable optimization algorithms by using additional techniques for narrowing down the direction of search. Optimization of a 2 variable objective function using Box's Evolutionary Algorithm, Cauchy's Steepest Descent Algorithm and Marquardt's Algorithm, as shown below,

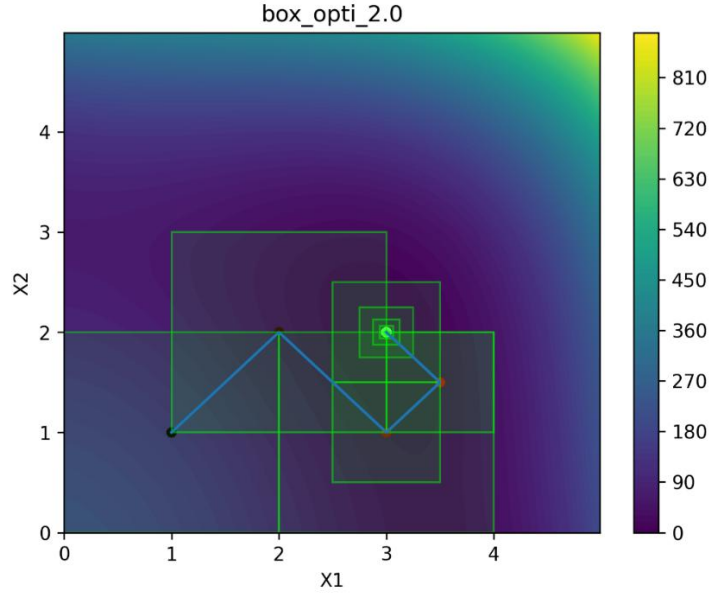


Fig: Box's Evolutionary Algorithm

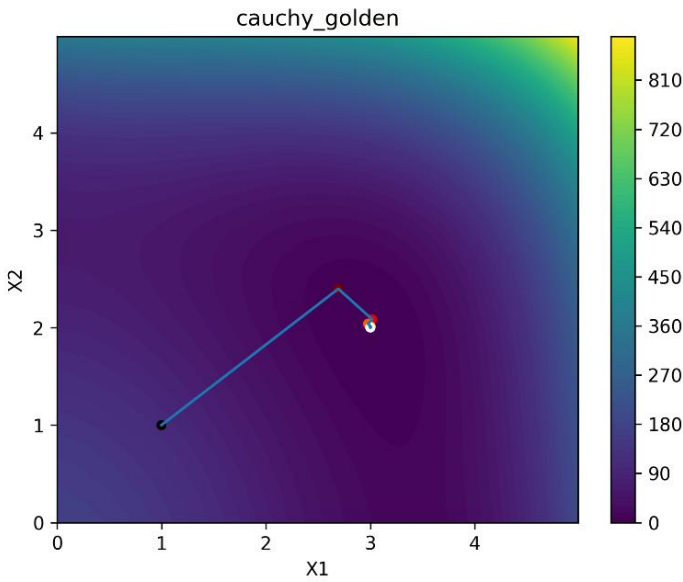


Fig: Cauchy's Steepest Descent Method

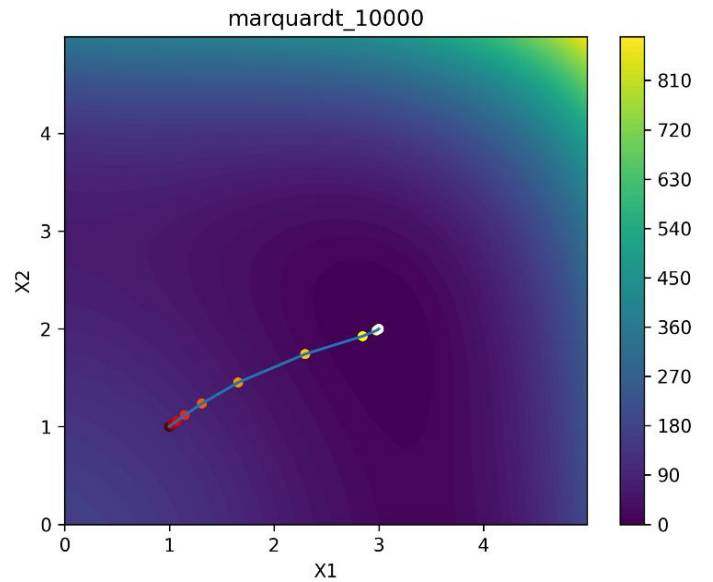


Fig: Marquardt's Method

Therefore, with the knowledge of the map, the navigation problem can be framed as an optimization one. A pseudo-objective function is designed with a parabolic structure centered at the target. The circular obstacles are modeled as Gaussian hills with set standard deviation. The 2D optimization algorithms are used to find the optima which we know corresponds to the target and the updated estimated points on each iteration act as the navigation pointer for the robot.

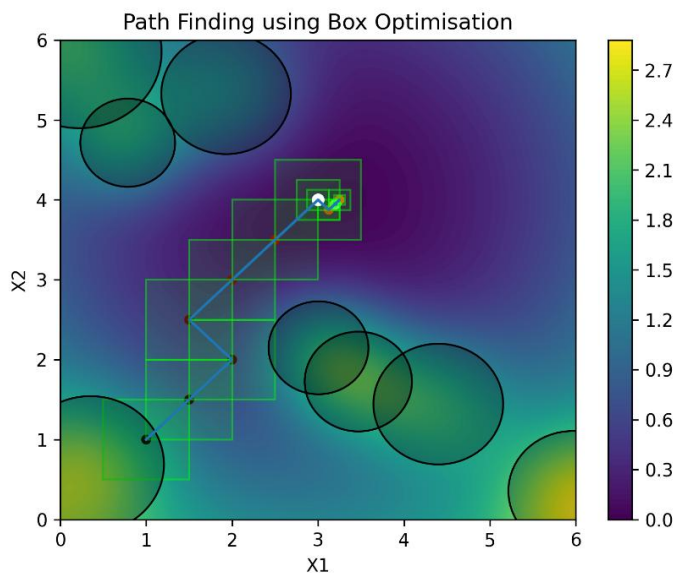


Fig: Path Finding using Box's Algorithm

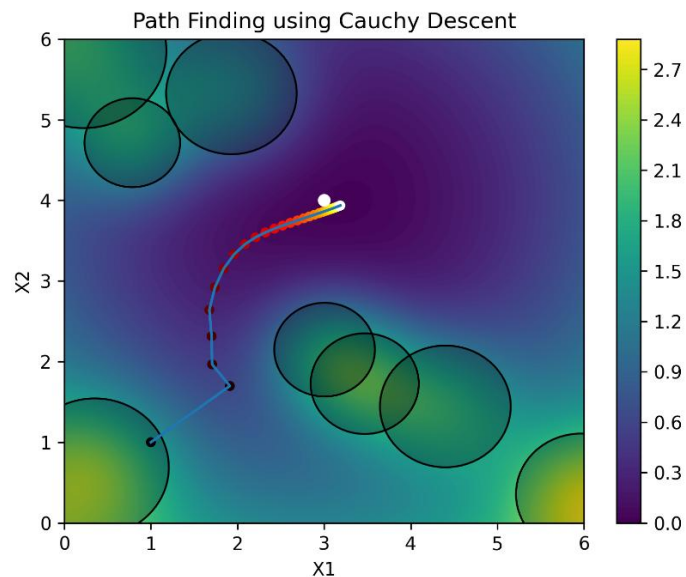


Fig: Path Finding using Cauchy's Algorithm

This can often work as a simplistic solution for navigation as shown below,

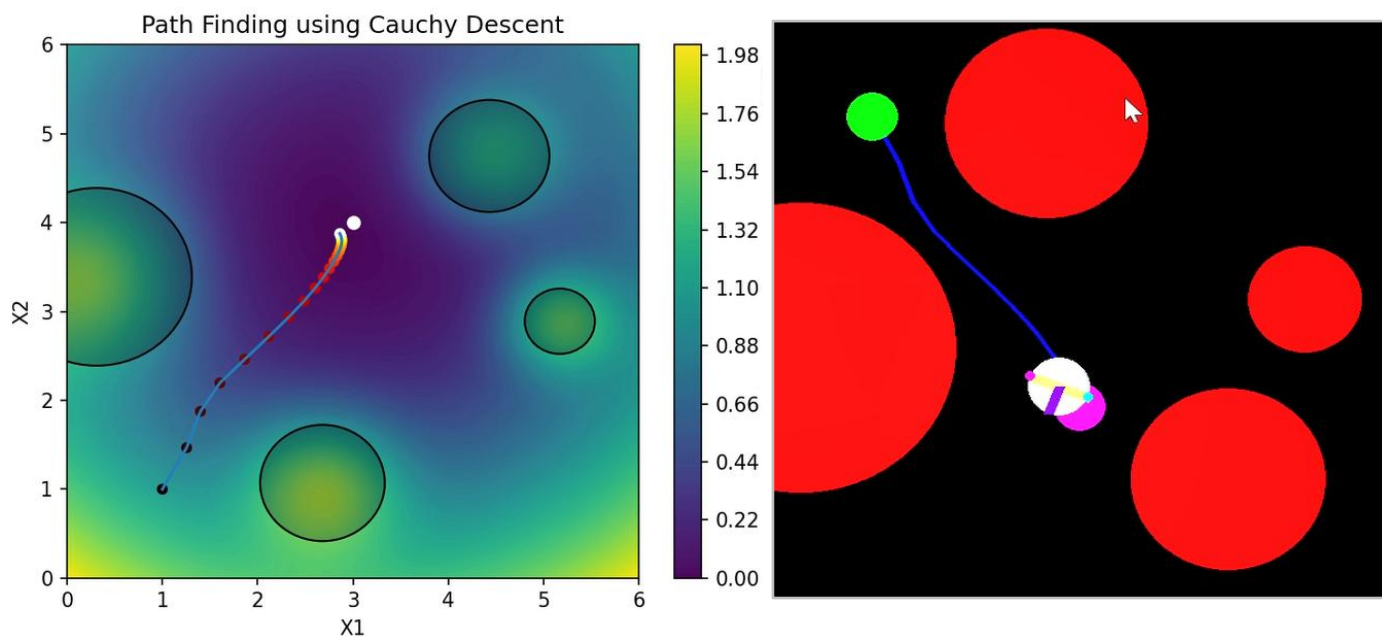


Fig: Path Finding successful using Gradient Descent and Line Following Bot

However, this breaks down as the map gets slightly complex with multiple Gaussian-like obstacles as the objective function may no longer be unimodal. The following is an example where a direct optimization leads to convergence to a local minimum which corresponds to the robot getting stuck behind the obstacles even though it has plenty of space to circumnavigate them.

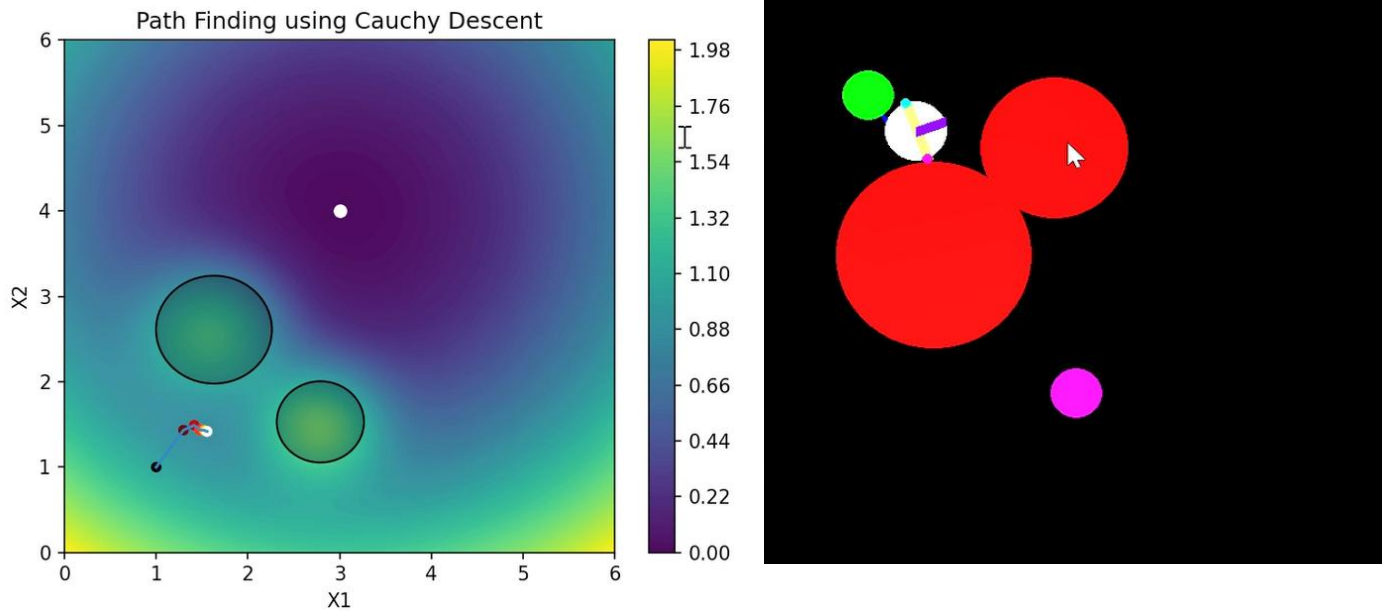


Fig: Path Finding failed using Gradient Descent and Line Following Bot

Conclusion

1. **Versatility:** The independence of the learning mechanism from modality and convexity of the objective function allows for diverse applications in highly complex design problems where the desired behavior can almost never be truly represented using a strictly convex function.
2. **Parallelism:** The algorithmic structuring where multiple simulations in the population are independent of each other allows for the prospect of massive parallelization and optimized computation.
3. **Robustness:** The exploration of multiple probable solutions parallelly avoids convergence to local minima, and will almost always converge to the global minima due to expansive and efficient exploration of the design space.
4. **Interpretability:** The derivation of such an algorithm from a natural process allows for logical interpretation of the way in which the algorithm learns.