Department of Computer Science

Bachelor Project Report

Counterexample-Guided Optimisation Applied to Mobile Robot Path Planning

Author: Mengze Li

Supervisor: Dr. Lucas Cordeiro

April 2021

Abstract

The purpose of this project is to present description, implementation and evaluation of the Counterexample Guided Inductive Optimisation (CEGIO) algorithm based off-line path planning for mobile robots, and to discuss Bounded Model Checking and software model checkers as verification techniques which assist the algorithm to generate paths.

In addition, the original contribution of this project is that it further describes the design, implementation and evaluation of Assisted CEGIO-based path planning (ACEGIO-based path planning) which combines Gradient Descent to significantly improve the efficiency. Besides, a possible design of mobile robot and an implementation of trajectory planning are also provided in this project.

In particular, this report presents that paths which are extremely close to the global optimal path, consisted of a sequence of points in a predetermined environment, have been successfully achieved by CEGIO and ACEGIO based path planning. Besides, smooth trajectories based on the obtained paths have been achieved by trajectory planning and applied to the mobile robot.

Acknowledgements

First and foremost, I would like to express my heartfelt gratitude to Dr Lucas Cordeiro, my supervisor, for his constant encouragement, patience and guidance on my project. He has provided me with helpful materials which pushed me to find a novel and challenging idea.

Second, I would also like to thank my family and friends for their support and great confidence in my abilities.

Impact of lockdown

I will briefly discuss the impact of lockdown on my project. Firstly, I planned to be back to Manchester from China at the beginning of the first semester, but because of lockdown most flights were cancelled including my flights. Thus, my travel was delayed for several times, and this led me wasting some time on repacking luggage and rebooking flight, which affected my original project plan. Additionally, since I live alone in Manchester, I hardly see my friends and rarely speak to other students in person for nearly half a year, along with the stress of uncertain future postgraduate program, I had slight mental problem, which made me feel negative and really anxious. This distracted me from doing my project for several weeks. Overall, the lockdown did have impact on me, and I spent some time to handle the unexpected situations. But the project has been done, following the timetable of new project plan.

Contents

List of Figures

List of Abbreviations

1. CEGIO: Counterexample-Guided Inductive Optimisation
2. ACEGIO: Assisted Counterexample-Guided Inductive Optimisation
3. SAT: Boolean Satisfiability
4. SMT: Satisfiability Modulo Theories
5. BMC: Bounded Model Checking
6. ESBMC: Efficient SMT-based Bounded Model Checker
7. CEGIO-F: Fast version of CEGIO
8. GD: Gradient Descent
9. ACEGIO-GD: Assisted CEGIO by Gradient Descent
10. ACEGIO-F-GD: Assisted CEGIO-F by Gradient Descent
11. Algorithm 1: CEGIO-based path planning algorithm
12. Algorithm 2: ACEGIO-based path planning algorithm

1 Introduction

This chapter will begin with discussing the reason of choosing this project and explaining the shortcoming of other algorithms for solving optimisation problem. Motivation section will also briefly introduce and define CEGIO and ACEGIO with presenting their pros and cons. Next, it will list the objectives of my project and criteria for identifying their success. Follow this will be the structure of my report.

1.1 Motivation

The last several decades have seen a growing trend towards automated societies. Mobile robots are replacing humans in repeated and monotonous work such as transforming parcels in Amazon's warehouses and more resources are provided for mobile robot development [1]. This is the reason why, autonomous navigation, especially path planning has a pivotal role in development of mobile robots. Path planning algorithm is a computational problem to find a valid path from initial location to desired destination that avoids all possible obstacles in the motion space [2][3].

A considerable amount of literature has provided series of algorithms and various methods on solutions to path planning. Previous research by Araujo et al. [4] has considered path planning problem as optimisation problem and established the description, which is "a decision variable represents a given path, i.e., the sequence of points (or movements) by which the robot must move; the cost function is certain criteria or metric whose value is optimized (e.g., distance, energy consumption, and execution time)" [5]. To solve this optimisation problem, algorithms, such as A* algorithm [6], genetic algorithm (GA) [7], particle swarm optimisation (PSO), ant colony optimisation, gravitational search algorithm (GSA) [8], can be applied to on-line path planning due to their optimal efficiency. However, these methods have an uncertain possibility that global optimality cannot be achieved.

Study of Counterexample Guided Inductive Optimisation Algorithm (CEGIO) by Araujo et al. [9] shows the ability of CEGIO that can ensure global optimality, thus CEGIO can be applied to path planning. In term of Araujo et al.'s description about CEGIO [10], it bases on Boolean Satisfiability (SAT) and Satisfiability Modulo Theories (SMT). Besides, the basic process and working principle of CEGIO is that it is executed iteratively by SAT and SMT solvers to generate counterexamples, which are employed to update both decision variables and the cost function and guide the optimisation towards global optimality. However, because of high time consumption on CEGIO-based path planning application, CEGIO is only applied to off-line path planning in this report.

Assisted Counterexample Guided Inductive Optimisation Algorithm (ACEGIO) is described by Chitoraga [11] which combines CEGIO with an auxiliary algorithm, in order to improve the efficiency of CEGIO. Previous study [12] shows CEGIO assisted by gradient descent can ensure global optimality in all the cases that CEGIO can achieve. Therefore, ACEGIO algorithm with gradient descent as the auxiliary algorithm is applied to off-line path planning for mobile robots.

1.2 Project objectives and criteria

Objectives of my project are summarized briefly and clearly. The objectives have two main categories, the two path planning algorithms and trajectory planning. The following is a comprehensive list of the objectives.

- Implement and evaluate CEGIO-based path planning algorithm
- Propose, implement and evaluate ACEGIO-based path planning algorithm
- Design a mobile robot, and build a physical robot
- Implement trajectory planning
- Apply trajectory planning to the virtual mobile robot and physical robot, respectively

CEGIO-based path planning algorithm and ACEGIO-based path planning algorithm are implemented and evaluated in a pre-defined environment with static obstacles. The criteria for identifying success of the objectives are as follows.

- Whether these two algorithms (CEGIO-based path planning and ACEGIO-based path planning) are capable of generating the global optimal path or paths close to the global optimality
- Whether ACEGIO-based path planning algorithm improves the efficiency of finding the global optimal path or paths close to the global optimality
- Whether smooth trajectories based on the obtained 2D paths are achieved by trajectory planning
- Whether the virtual robot and the physical robot follow the trajectories smoothly

1.3 Report structure and outline

Firstly, this report will begin with a technical background about optimisation problem, path planning for mobile robots, bounded model checking, CEGIO, ACEGIO and trajectory planning. Secondly, it will present description of modelling path planning problem as optimisation problem and implementation of CEGIO algorithm for solving the path planning problem. Then it will present the methodology steps for proposing ACEGIO-based path planning algorithm, and it will offer description and implementation of the algorithm. Next, the Trajectory Planning chapter will describe the implementation of the trajectory planning. In this chapter, it will also present the virtual robot simulation and designs of physical robot. Following this will be the evaluation and experimental results of these two path planning algorithms and trajectory planning. At last, conclusions and future work will be presented.

2 Background

This chapter will cover technical background, which is necessary for understanding my project achievements. Firstly, it will define optimisation problem and discuss difficulty of finding global optimal solution for optimisation problems in different categories. Then, it will explain the path planning problem that I will focus on in my project. Next, it will introduce the Model Checking with its steps and Bound Model checking with its basic principle. Additionally, it will discuss how CEGIO works with assistance of Bounding Model Checking. It will also discuss how ACEGIO works with assistance of Gradient Descent. Finally, it will cover trajectory planning preliminaries.

2.1 Optimisation problem

Optimisation problems and global optimal solution can be defined as Follows:

Definition 1. Optimisation problems in the field of Computer Science is finding the global optimal solution from all feasible solutions.

In general, optimisation problems are to minimize the cost function. If there are optimisation problems which need to maximize the cost function, then minimize the negation of the cost function. Therefore, general optimisation problems can be written as [13]

$$\begin{aligned} \underset{x}{\text{minimize}} \quad & f(x) \\ \text{subject to} \quad & g_i(x) \leq 0, i = 0, \dots, m \;, \\ & h_j(x) = 0, j = 0, \dots, p \end{aligned} \tag{1}$$

where

- $f(x)$ is the objective function as known as cost function, which needs to be minimized over a vector x composed n variables as known as decision variables.
- $g_i(x) \leq 0$ and $h_j(x) = 0$ are inequality constraints and equality constraints, respectively.
- $m \geq 0$ and $p \geq 0$
- If m and p are both 0, then there is no constraint, and the problem is an unconstrained optimisation problem.

Definition 2. A vector x^* such that $g_i(x^*) \leq 0$ or $h_j(x^*) = 0$ is a global optimal solution of $f(x)$ iff $f(x^*) \leq f(x)$, for $\forall x$ such that $g_i(x) \leq 0$ or $h_j(x) = 0$.

There are different categories of optimisation problems, which depend on the number of objective functions, objective function nature, constraints and whether the decision variables are continuous or discrete.

Different nature of optimisation problems has impacts on ways and difficulty of finding the global optimal solution. For instance, discrete optimisation problems tend to be harder to find the best solution than continuous optimisation problems. Problems with multiple objectives functions are often converted to single objective function problem, in order to make them easier

[14]. Furthermore, optimisation problems with non-convex cost function are the most difficult to achieve global optimality because of the cost function nature. Some optimisation techniques fail to find the global optimal solution within the given time and computational memory, and some optimisation algorithms trap in local optimal solutions while finding the global optimal solution [15]. In contrast, convex optimisation problems are easier, and more optimisation techniques are available because of the following properties

- Every local optimality is global optimality.
- There is at most one global optimality for the optimisation problems whose cost function is strictly convex.

2.2 Path planning for mobile robots

Path planning problem can be described as finding a safe and available path from a starting position to a target position in the environment. As one step in mobile robot navigation, efficient path planning algorithms have considerable contribution on safe and effective mobile robot navigation. Path planning can be classified as static or dynamic, global or local, exact or heuristic, respectively depending on environment nature, robot's knowledge about the environment and completeness [16]. In the present work, I focus on static, global, exact path planning algorithm in bi-dimensional environment with following properties

- The environment is fixed, where the starting position, the target position, obstacles and map information are unvarying.
- The mobile robot has priori knowledge about the environment.
- The algorithm finds an optimal path if one exists or proves no feasible solution exists.

Definition 3. An optimal path is composed by a sequence of points, which are consecutively and sequentially connected by straight segments. The optimal path is valid for mobile robots to move from a starting position to a target position, and it minimizes the cost function related to that path such as energy consumption, distance. [17]

According to the definition 3 and the above properties, proposed path planning algorithm in this report is to find an optimal path, consisting of a set of straight segments that formed by points, from the source to the destination, which meets the path specification (obstacle avoidance) and minimizes the distance as the principal objective in the predefined environment.

2.3 Model checking

In this section, the definition and steps of Model Checking will be presented. Model checking is an automated verification technique. Given a finite-state model of a system and a formal property, model checking systematically explores and checks all reachable systems states in a brute-force manner [18]. Model checking is a successful approach for verifying requirements with a wide range of applications. Model checking process can be divide into three steps: modelling, specification, and verification [19].

Modelling converts the system to a formal characterization of a finite set of states and a set of transition to be checked. Then specification prescribes the system behaviour and properties that all relevant systems states should satisfy. In this process, it has no requirement on complete specification which allows focus on essential properties to be checked instead of all properties. Therefore, it makes model checking support partial verification in next step. The final step verification examines the model and checks whether the states satisfy the given property [20]. Verification succeeds if the model satisfies the specification. Otherwise, counterexample, which describes the execution path from the initial system state to property violation state, is generated if verification fails [21]. The counterexample also contains diagnostic information that plays a vital role in debugging.

2.4 Bounded Model Checking

This section will cover definition and basic idea of Bounded Model Checking. Additionally, it will discuss a Bounded Model Checker.

Model Checking checks if all given states in model satisfy a property, while Bounded Model Checking (BMC) which is based on SAT or SMT solvers as a verification technique checks if a subset of states within the bound satisfies a property. Bounded Model Checking can be described as follows:

Definition 4. "Given a transition system M, a property ϕ, and a bound k; BMC unrolls the system k times and translates it into a verification condition (VC) ψ, which is satisfiable iff ϕ has a counterexample of depth less than or equal to k" (Araujo et al., 2017) [22].

Study of Bounded Model Checking by Araujo et al. (2017) [23] shows that the basic idea of BMC is to check the negation of a given property and the logical formula of associated problem in BMC is as following:

$$\psi_k = I(S_0) \wedge \bigvee_{i=0}^{k} \bigwedge_{j=0}^{i-1} \left(\gamma(s_j, s_{j+1}) \wedge \neg\phi(s_i)\right), \tag{2}$$

where ϕ is a property and S_0 is a set of initial states of M, and $\gamma(s_j, s_{j+1})$ is the transition relation of M between time steps j and j + 1. The verification condition ψ_k is satisfiable iff for some $i \leq k$ there exists a reachable state at time step i in which the property ϕ is violated. In this case, the last step of model checking (verification) fails, and the SAT or SMT solver provides a satisfying assignment of the above logic formula which is counterexample. Counterexample can be described as follows:

Definition 5. "A counterexample for a property ϕ is a sequence of states $S_0, S_1, \ldots, S_k$ with $s_0 \in S_0, s_k \in S_k$, and $\gamma(s_i, s_{i+1})$ for $0 \leq i \leq k$ that makes Eq. (2) satisfiable. If it is unsatisfiable (i.e., returns false), then we can conclude that there is no error state in k steps or less" (Araujo et al., 2017)) [24].

ESBMC is an efficient SMT-based Bounded Model Checker for both single- and multi-threaded C / C++ programs which is used as software verification tool in this study. ESBMC is able to verify program properties violations such as memory safety, overflows, array bounds and also additional properties which are stated by users. In C / C++ programming language, ASSUME and ASSERT are two directives which can be used in modelling and specification in the process of model checking. In particular, ASSUME directive can set constraints by employing variables in non-deterministic representation and ASSERT directive can be used to specify the given property. The corresponding intrinsic functions in ESBMC is __ESBMC_assume and __ESBMC_assert.

2.5 CEGIO

The working process of CEGIO with the help of Bounded Model checking is presented as follows. CEGIO algorithm is an optimisation algorithm which requires iterative executions to achieve global optimisation by requesting counterexamples from SAT and SMT solvers. According to definition 1,4 and formula (1)(2), optimisation problems can be solved by Bounded Model Checking and be modelled as system states and formal property via cost function, decision variables and its constraints. In each iteration of CEGIO, decision variables and its constraints are defined by ASSUME directive as constraints, and optimal condition is specified by ASSERT directive as property. Then bounded model checker, in particular ESBMC is applied with its intrinsic functions in this study, generates counterexamples, which are employed to update both domain boundaries and the optimal candidates, and guide the optimisation towards global optimality.

Additionally, the previous study by Araujo et al. demonstrates that CEGIO algorithm has strong ability of finding the global minima of optimisation problems [25]. The study also presents three variants of CEGIO, which are the Generalized CEGIO (CEGIO-G), the Simplified CEGIO (CEGIO-S) and the Fast CEGIO(CEGIO-F) [26]. CEGIO-G is suitable for any optimisation problems in relatively slow speed. CEGIO-S is faster than CEGIO-G and is suitable for optimisation problems whose knowledge about local minima is provided in advance. CEGIO-F is the fastest among these three types of algorithm, but it can only be applied to convex functions.

2.6 ACEGIO

This section will cover technical background information of ACEGIO algorithm. Firstly, it will begin with its definition and how it works. Assisted CEGIO (ACEGIO) proposed by Dumitru [27] combines CEGIO with an auxiliary algorithm, which significantly improves the optimisation speed without restricting the range of problems CEGIO can be applied to. The process of achieving optimisation by ACEGIO is as following. In each iteration of generating a more optimized position by ACEGIO, instead of only extracting from counterexamples, the position is calculated by the auxiliary algorithm based on the counterexamples from CEGIO till it reaches local optimum or global optimum. If it reaches local optimum, a new position is generated from counterexamples, and repeat the above process till global optimum is found. Since requesting counterexamples from SMT solvers is the most time-consuming step in CEGIO, and ACEGIO greatly reduces the times of requesting counterexamples. For this specific reason, ACEGIO significantly improves the execution speed of finding global optimum.

Additionally, it will present a particular auxiliary algorithm as follows. Gradient Descent, an efficient optimisation technique for optimizing convex functions, is applied as auxiliary algorithm to assist CEGIO in this report. Given a differentiable cost function, Gradient Descent algorithm has the capability to find the local minimum. The basic idea of general Gradient Descent algorithm is to iteratively move towards the opposite direction of the gradient of the function at current position, which is the direction of steepest descent at current point. To be more specific, Gradient Descent algorithm can be pictured as a hiker who would like to climb down a mountain (cost function) into valley (local minimum), and each step is determined by direction of steepest descent at current position and the leg length of the hiker. In each iteration of Gradient Descent Optimisation, compute the gradient of the cost function and move-in the opposite direction with a step. The hiker iteratively takes steps till reaching the valley (local minimum).

The following property enables Gradient Descent to be the auxiliary algorithm in ACEGIO:

$$\forall y, \text{ if } \exists x \text{ such that } g(f(y)) > g(x) \text{ then } g(f(y)) > g(f(x)), \tag{3}$$

where

- x is decision variable
- g is cost function, f is the function of Gradient Descent
- $f(x)$ is the result of applying Gradient Descent on x

Finally, it will discuss how ACEGIO-GD (CEGIO assisted by Gradient Descent) algorithm can ensure global optimality. If the cost function is convex, $f(x)$ can always point to the global optimality. However, if the cost function is non-convex, Gradient Descent has some possibility of being trapped in local optimum. In this case, CEGIO would generate counterexamples which contains more optimized decision variable position than previous local optimum. Then $f(x)$ is applied, either to global optimum or a better local optimum. This process will be repeated till global optimum is found. Therefore, global optimum can always be found by ACEGIO-GD.

2.7 Trajectory planning preliminaries

After achieving the optimal path from path planning, smooth movements of physical robots are achieved by trajectory planning. Trajectory planning can be described as motion planning, which plans how to move based on velocity, time, etc. As for implementation, VEXcode VR is a browser-based platform which can realize robot simulation with programming languages like Python. Besides, Lego Mindstorms EV3 Home is a robotics kit, which contains a programmable brick, motors and various building components. The kit allows users to design and build their own robots with different functionalities and import their own programs into the programmable brick to control the motion of robot. As for coding the controlling programs, different tools are available based on the user's preference to a specific programming language. For instance, ROBOTC is a C programming language-based IDE with built-in variables and functions that control the robot's hardware like motors. In this report, VEXcode VR, Lego Mindstorms EV3 kit and ROBOTC are respectively applied to simulate robot motion online, build a physical robot and control the physical robot motion by programs.

3 CEGIO-based Path Planning Algorithm

In this project, the main objective of path planning problem is to find the global optimal path composed by a set of points which can guide the mobile robots from a starting position to a target position with avoiding obstacles in an environment. A great deal of previous research has explored the impacts of obstacles number and obstacles type on complexity of path planning problem. With the increase in number of obstacles and types of obstacles, the complexity substantially increases which requires more processing time to find the optimal path. As a result, there is high demand of developing novel methods that can achieve optimal path, considering time and system consumption.

In this report, CEGIO based path planning is applied to generate points on path in a pre-defined environment with static obstacles. This method consists of two steps: (1) formulate the path planning problem as an optimisation problem (i.e., model the environment, and static obstacles as constraints, set the cost function) (2) apply CEGIO to find the optimal path that satisfies the constraints.

Following the above steps, this chapter will firstly discuss how path planning problem is formulated as an optimisation problem. Next, it will present description and implementation of CEGIO-F algorithm, which is employed to solve the optimisation problem. Finally, it will present description and implementation of applying CEGIO-F algorithm to solve path planning problem.

3.1 Optimisation problem formulation

In order to solve the path planning problem by CEGIO algorithm, it is necessary to formulate the path planning problem as an optimisation problem. Thus, its cost function and constraints need to be defined. Araujo et al. [28] proposed the following definitions (Definition 6, 7) about cost function and constraints for path planning problem.

Definition 6. Cost function: Define the starting position (S) and target position (T) as $\mathrm{S} = \mathrm{P}_1$ and $\mathrm{T} = \mathrm{P}_\mathrm{n}$, respectively. The objective is to find a decision variable matrix, $\mathbf{L} = [\mathrm{P}_1, \mathrm{P}_2, \dots, \mathrm{P}_{\mathrm{n}-1}, \mathrm{P}_\mathrm{n}]$, which minimizes the cost function $\mathrm{J(L)}$. $\mathrm{J(L)}$ is the distance function to calculate the total distance of the path, also $\mathrm{J(L)}$ is the cost function of the path planning problem in this project. The cost function is defined as:

$$J(L) = \sum_{i=1}^{n-1} \|P_{i+1} - P_i\|_2, \tag{4}$$

where n is the number of points (including the starting position and the target position) that form the path. A a smooth trajectory will be achieved if n is infinite [29].

From Definition 3 and 6, the path consists of n points on the path which is connected by $n-1$ straight segments, such that the i-th straight segment is from position P_i to position P_{i+1}.

Definition 7. Constraints: Each straight segment on the path must not intercept any obstacle and must be within the pre-defined environment.

According to Definition 1 and Eq.(4), the optimisation problem for path planning can be written as:

$$\begin{aligned} \min_{L} \quad & J(L), \\ \text{s.t.} \quad & p_{i\lambda}(L) \notin \mathbb{O} \\ & p_{i\lambda}(L) \in \mathbb{E} \\ & i = 1, \dots, n-1, \end{aligned} \quad , \tag{5}$$

where "$\mathbb{O}$ is the set of points defined by obstacles; $\mathbb{E}$ is the set of points defined by environment limits; n is the number of points that consist the path; and $p_{i\lambda}(L)$ is all points belonging to the i-th straight segment of the path defined by vector L, each $p_{i\lambda}(L)$ point is defined as" (Araujo et al., 2017) [30]:

$$p_{i\lambda}(L) = (1-\lambda)P_i + \lambda P_{i+1}, \forall \lambda \in [0,1]. \tag{6}$$

After defining the cost function and constraints, movement environment ($\mathbb{E}$) and static obstacles ($\mathbb{O}$) need to be encoded. In this report, the environment of movement is modelled in a two-dimensional Cartesian system as rectangle with lower and upper boundaries. Each point on path must be within this rectangle. As for obstacles ($\mathbb{O}$), one circle is used to model one obstacle, such that a centre of circle is the geometric centre of a physical obstacle, radius of circle is the largest distance from the centre to edge of the physical obstacle. This ensures all the points formed physical obstacles are surrounded by the corresponding circles. Therefore, for each obstacle, constraints $p_{i\lambda}(L) \notin \mathbb{O}$ such that $i = 1, \dots, n-1$ can be written as:

$$(x_{i\lambda} - x_0)^2 + (y_{i\lambda} - y_0)^2 \geq (r + \sigma)^2,$$

where $p_{i\lambda} = (x_{i\lambda}, y_{i\lambda})$, and σ is a safety margin, (x_0, y_0) is the center of an obstacle, r is the radius of the obstacle.

3.2 CEGIO-F

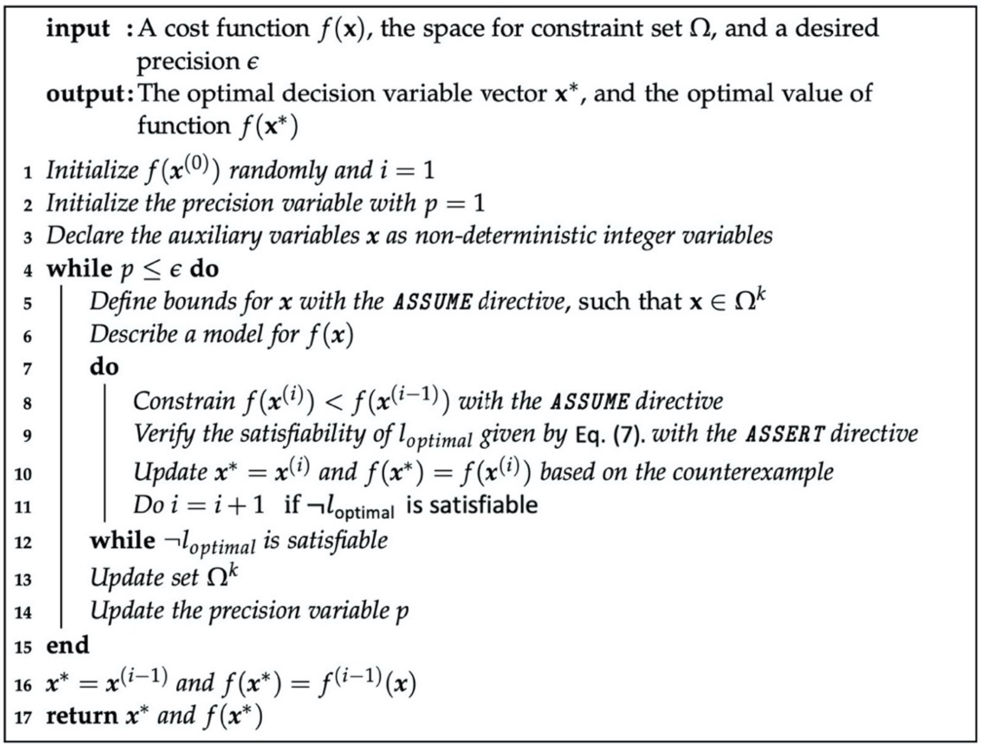
Previous research has demonstrated that the cost function (Eq.(4)) is convex, thus CEGIO-F which is aforementioned is applied to solve path planning due to its efficiency. The pseudocode of CEGIO-F algorithm proposed by Araujo et al. [31] is in figure 1. The explanation of the pseudocode is as follows.

CEGIO-F takes the cost function $f(\mathbf{x})$, the space for constraint set Ω, and a desired precision ϵ as input. The output includes the optimal decision variable vector $\mathbf{x}^*$, and optimal cost function value $f(\mathbf{x}^*)$. Firstly, from line 1 to line 3 of the pseudocode, it begins with initialization and declaration of value $f(x^{(0)})$, precision p and non-deterministic decision variable $\mathbf{x}$. The space for constraint set in line 5 is defined as Ω^k, such that $k = \log p$, where k is the number of decimal places of the points coordinate values (i.e., if p is 1, then k is 0 and coordinates are considered as integers). Precision p is initialized with value 1 and is updated in line 14 by multiplying p by 10. Finally, CEGIO-F ends if precision p reaches the desired precision ϵ.

The core in CEGIO-F algorithm of finding optimal solution is as follows. Aforementioned ASSUME and ASSERT directives in line 5,8 and 9 are used to define constraints and check specific constraints. Line 5 defines constraints by setting the bounds for decision variable $\mathbf{x}$ within the space Ω^k with ASSUME directive. Besides, line 8 sets constraint $f(\mathbf{x}^{(i)}) < f(\mathbf{x}^{(i-1)})$, which defines that objective function $f(\mathbf{x}^{(i)})$ must be smaller than the value obtained as optimal candidate $f(\mathbf{x}^{(i-1)})$. Then check the literal $l_{optimal}$ with ASSERT directive，$l_{optimal}$ is described as:

$$l_{optimal} \Leftrightarrow f(\mathbf{x}^{(i)}) \geq f(\mathbf{x}^{(i-1)}). \tag{7}$$

If $\neg l_{optimal} \Leftrightarrow (f(\mathbf{x}^{(i)}) < f(\mathbf{x}^{(i-1)}))$ is satisfiable，a counterexample is generated which contains the new decision variables vector $\mathbf{x}^*(\mathbf{x}^{(i)})$ and new optimal candidate $f(\mathbf{x}^*)$ ($f(\mathbf{x}^{(i)})$). The new optimal candidate is smaller than previous optimal candidate $f(\mathbf{x}^{(i-1)})$ and closer to global optimality. Otherwise, if $l_{optimal}$ is satisfiable, $\mathbf{x}^*$ and $f(\mathbf{x}^*)$ would not be updated and would remain $\mathbf{x}^{(i-1)}$ and $f(\mathbf{x}^{(i-1)})$, respectively. Besides, variable i stores the index of new optimal candidate, which means i is only updated by adding 1 in each iteration when a counterexample is generated in this iteration. As a result, in each iteration, either a new optimal candidate which is closer to global optimality is found, or the state-space is updated. After substantial times of iterations, the optimal value and optimal solution is generated.



```
input : A cost function $f(\mathbf{x})$, the space for constraint set $\Omega$, and a desired
        precision $\epsilon$
output: The optimal decision variable vector $\mathbf{x}^*$, and the optimal value of
        function $f(\mathbf{x}^*)$
1  Initialize $f(\mathbf{x}^{(0)})$ randomly and $i = 1$
2  Initialize the precision variable with $p = 1$
3  Declare the auxiliary variables $\mathbf{x}$ as non-deterministic integer variables
4  while $p \leq \epsilon$ do
5      Define bounds for $\mathbf{x}$ with the ASSUME directive, such that $\mathbf{x} \in \Omega^k$
6      Describe a model for $f(\mathbf{x})$
7      do
8          Constrain $f(\mathbf{x}^{(i)}) < f(\mathbf{x}^{(i-1)})$ with the ASSUME directive
9          Verify the satisfiability of $l_{optimal}$ given by Eq. (7). with the ASSERT directive
10         Update $\mathbf{x}^* = \mathbf{x}^{(i)}$ and $f(\mathbf{x}^*) = f(\mathbf{x}^{(i)})$ based on the counterexample
11         Do $i = i + 1$  if $\neg l_{optimal}$ is satisfiable
12     while $\neg l_{optimal}$ is satisfiable
13     Update set $\Omega^k$
14     Update the precision variable $p$
15 end
16 $\mathbf{x}^* = \mathbf{x}^{(i-1)}$ and $f(\mathbf{x}^*) = f^{(i-1)}(\mathbf{x})$
17 return $\mathbf{x}^*$ and $f(\mathbf{x}^*)$
```

Figure 1. Pseudocode of CEGIO-F

3.3 CEGIO-based Path Planning Algorithm

This section will cover the development and explanation of CEGIO-based path planning algorithm. Following this will be the brief analysis of its efficiency. Finally, it will present how the algorithm is applied to path planning problem. In this section, it will firstly begin with the development and explanation of CEGIO-based path planning algorithm as follows.

Previous study of Araujo et al. [32] provides the algorithm (in Figure 2.) that CEGIO-F is applied to solve path planning problem based on its optimisation formula. The algorithm takes cost function $\mathrm{J}(\mathbf{L})$, constraint set Ω defined by a set of obstacles constraints $\mathbb{O}$ and a set of environment constraints $\mathbb{E}$, and a desired precision η as input. And the output is the optimal path $\mathbf{L}^*$, and the optimal cost function value $\mathrm{J}(\mathbf{L}^*)$. It starts with initialization of $\mathrm{J}\left(\mathbf{L}^{(0)}\right)$, which is in high value, in order to make $\neg \mathrm{J}_{\mathrm{optimal}}$ is satisfiable and generate a new optimal candidate in first iteration. $\mathrm{J}_{\mathrm{optimal}}$ is described as:

$$\mathrm{J}_{\mathrm{optimal}} \Leftrightarrow \mathrm{J}\left(\mathbf{L}^{(\mathrm{i})}\right) \geq \mathrm{J}\left(\mathbf{L}^{(\mathrm{i}-1)}\right). \tag{8}$$

Besides, the precision p is initialized with value 1 (coordinates are considered as integers), and number of points n composed the path is initialized with value 1, which means there is only one point excluding the starting point and the target point on the path. Decision variables vector $\mathbf{L}^{\mathrm{i}}$ is declared over non-deterministic variables which contains all the path points information.

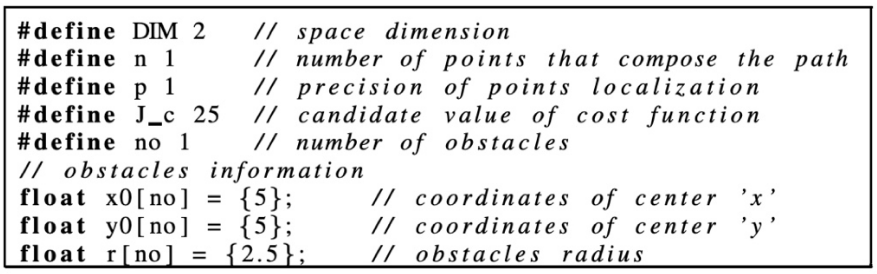
```
input : Cost function $J(\mathbf{L})$, is a set of obstacles constraints $\mathbb{O}$ and a set of
        environment constraints $\mathbb{E}$, which define $\Omega$ and a desired precision $\eta$
output: The optimal path $\mathbf{L}^*$ and the optimal cost function value $J(\mathbf{L}^*)$
1  Initialize $J(\mathbf{L}^{(0)})$ randomly;
2  Initialize precision variable with $p = 1$, $k = 0$ e $i = 1$;
3  Initialize number of points, $n = 1$;
4  Declare decision variables vector $\mathbf{L}^i$ as non-deterministic integer variables;
5  while $k \leq \eta$ do
6      Define upper and lower limits of $\mathbf{L}$ with directive ASSUME, such as $L \in \Omega^k$;
7      Describe the objective function model $J(\mathbf{L})$;
8      do
9          do
10             Define the constraint $J(\mathbf{L}^{(i)}) < J(\mathbf{L}^{(i-1)})$ with directive ASSUME;
11             Verify the satisfiability of $J_{optimal}$ given by Eq. (7);
12             Update $\mathbf{L}^* = \mathbf{L}^{(i)}$ e $J(\mathbf{L}^*) = J(\mathbf{L}^{(i)})$ based on the counterexample;
13             Do $i = i + 1$;
14         while $\neg J_{optimal}$ is satisfiable;
15         if $\neg J_{optimal}$ is not consecutively satisfiable then
16             break
17         end
18         else
19             Update the number of points, $n$;
20         end
21     while TRUE;
22     Do $k = k + 1$;
23     Update the set $\Omega^k$;
24     Update the precision variable, $p$;
25 end
26 $\mathbf{L}^* = \mathbf{L}^{(i)}$ e $J(\mathbf{L}^*) = J(\mathbf{L}^{(i)})$;
27 return $\mathbf{L}^*$ e $J(\mathbf{L}^*)$;
```

Figure 2. Pseudocode of CEGIO-F applied to path planning problem

Similar to CEGIO-F, ASSUME directive is used to set the constraints, and ASSERT directive is used to verify the specific property. According to line 6, 10, 11, and 12 in Figure 2, the algorithm defines all boundary constraints with ASSUME directive and set the constraint $\neg \mathrm{J}_{\mathrm{optimal}} \Leftrightarrow \mathbf{J}(\mathbf{L}^{(\mathrm{i})}) < \mathrm{J}(\mathbf{L}^{(\mathrm{i}-1)})$ is satisfiable for a given n and p. Then use ASSERT directive to check the specific property $\mathrm{J}_{\mathrm{optimal}}$, if verification fails, which means $\neg \mathrm{J}_{\mathrm{optimal}}$ is satisfiable. In this case, a counterexample, which contains decision variables vectors $\mathbf{L}^{*} = \mathbf{L}^{(\mathrm{i})}$ is generated. The counterexample contains current optimal path which is composed by the set of n points, and it also current optimal cost function value which is the total distance of current path. Then the optimal candidate is updated as $\mathbf{J}(\mathbf{L}^{(\mathrm{i})})$. Otherwise, if the verification is successful, the current optimal candidate is still $\mathrm{J}(\mathbf{L}^{(\mathrm{i}-1)})$. Therefore, new optimal candidates who is closer to global optimality is generated in this way. For a given n and p, if it is not possible to find a new optimal candidate, number of points n is updated by adding 1 to n. If $\neg \mathrm{J}_{\mathrm{optimal}}$ is not consecutively satisfiable, then update precision p by multiplying p by 10. And the number of decimal places of the points coordinate values k is updated by adding 1 to k, since $\mathrm{k} = \log \mathrm{p}$. Therefore, the precision is increased by adding one decimal place in the coordinate values.

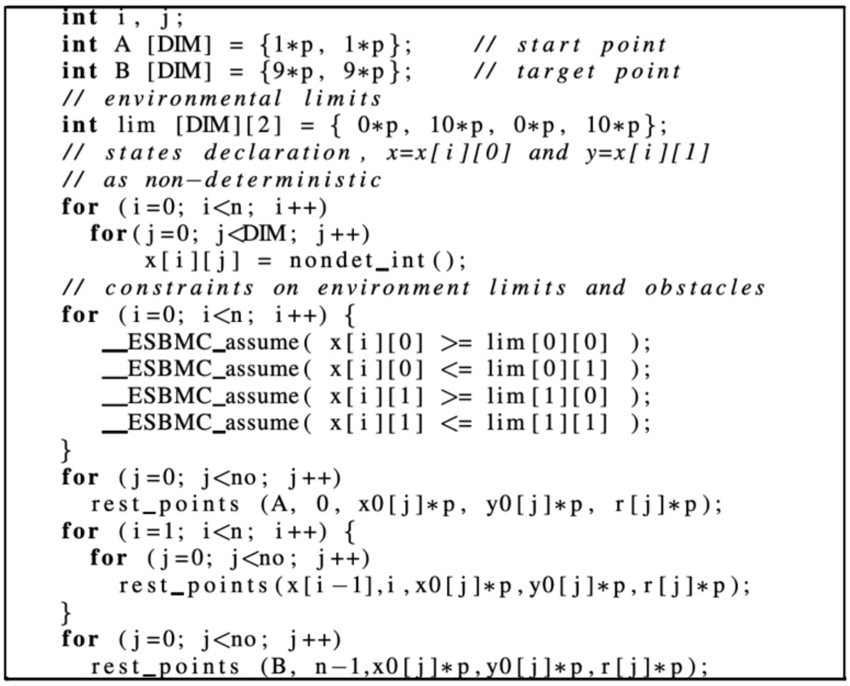
The analysis of efficiency is as follows. The number of points on the path depends the efficiency of the algorithm and number of obstacles have impacts on the algorithm performance. With the number of points increases, the time complexity and space complexity are substantially increasing, resulting in a large execution time.

In this project, I focused on applying the CEGIO-based path planning algorithm to static movement environment and employing ESBMC verification tool as Bounded Model Checking tool. Constraints of path planning problem are encoded in C programming language as follows. Figure 3 shows the definition and declaration of space dimension, initialized precision, candidate value, and number of points. In this example, only one obstacle is defined in this environment. These information are necessary for defining constraints.



```
#define DIM 2    // space dimension
#define n 1      // number of points that compose the path
#define p 1      // precision of points localization
#define J_c 25   // candidate value of cost function
#define no 1     // number of obstacles
// obstacles information
float x0[no] = {5};      // coordinates of center 'x'
float y0[no] = {5};      // coordinates of center 'y'
float r[no] = {2.5};     // obstacles radius
```

Figure 3. Example of related variables and macro constants [33]



```
int i, j;
int A [DIM] = {1*p, 1*p};      // start point
int B [DIM] = {9*p, 9*p};      // target point
// environmental limits
int lim [DIM][2] = { 0*p, 10*p, 0*p, 10*p};
// states declaration, x=x[i][0] and y=x[i][1]
// as non-deterministic
for (i=0; i<n; i++)
  for(j=0; j<DIM; j++)
      x[i][j] = nondet_int();
// constraints on environment limits and obstacles
for (i=0; i<n; i++) {
   __ESBMC_assume( x[i][0] >= lim[0][0] );
   __ESBMC_assume( x[i][0] <= lim[0][1] );
   __ESBMC_assume( x[i][1] >= lim[1][0] );
   __ESBMC_assume( x[i][1] <= lim[1][1] );
}
for (j=0; j<no; j++)
  rest_points (A, 0, x0[j]*p, y0[j]*p, r[j]*p);
for (i=1; i<n; i++) {
  for (j=0; j<no; j++)
    rest_points(x[i-1],i,x0[j]*p,y0[j]*p,r[j]*p);
}
for (j=0; j<no; j++)
  rest_points (B, n-1,x0[j]*p,y0[j]*p,r[j]*p);
```

Figure 4. Code of setting constraints [34]

Figure 4 shows the environment constraints. Firstly, it initializes points with non-deterministic variables and set every point on the path within the environment limits. Then set the constraints on obstacles by using function rest_points showed in Figure 5, which make sure aim (1) every point on the path is outside the obstacles, and aim (2) every straight segment composed the path would neither be within the range of any obstacle nor intersect with the any obstacle. The function takes a point and its next point that form a straight segment on the path as input. First six line ensure that the points on the path are outside the obstacles. In order to achieve aim (2), consider the intersection between the support line of the straight segment from (x[i-1], y[i-1]) and (x[i], y[i]) and its perpendicular passing through the obstacle centre (x[0], y[0]). Py is the y coordinate of that intersection point. Then if condition (line 18) is true if and only if the intersection point belongs to the straight segment. In this case the distance d between the obstacle centre and the straight segment should be greater than the obstacle radius to ensure the obstacle avoidance. Such an assumption is not necessary if the intersection point does not belong to the straight segment.

```
void rest_points(int P1 [DIM], int i, float x0, float y0,
                 float r) {
    float sigma = 0.5;    // safety margin
    // constraint given by Eq. 7
    __ESBMC_assume( (x[i][0]-x0)*(x[i][0]-x0) +
    (x[i][1]-y0)*(x[i][1]-y0) > (r+sigma)*(r+sigma) );
    float a, b, c;
    if (P1[0]-x[i][0]==0){
       a = 1;
       b = 0;
       c = -P1[0];
    }
    else{
       a = (float) (P1[1]-x[i][1])/(P1[0]-x[i][0]);
       b = -1;
       c = (float) -a*P1[0]+P1[1];
    }
    float Py = (a*a*y0-a*b*x0-b*c)/(a*a+b*b);
    if (((Py-x[i][1])/(P1[1]-x[i][1])>=0) &&
       ((Py-x[i][1])/(P1[1]-x[i][1])<=1))) {
       float d=(float) abs2(a*x0+b*y0+c)/sqrt2(a*a+b*b);
       __ESBMC_assume( d > r );
    }
}
```

Figure 5. Code of the rest_points function [35]

After defining the environment and obstacles constraints, cost function $J(\mathbf{L})$ is calculated according to Eq. 4 and constraint $J(\mathbf{L}^{(i)}) < J(\mathbf{L}^{(i-1)})$ is defined by __ESBMC_ASSUME directive. To obtain a new optimal candidate, $J_{optimal}$ (Eq. 8) is verified with __ESBMC_ASSERT directive at the end of each iteration. In this report, the above C program is iteratively passed to ESBMC to generate counterexample contains new optimal candidate based on the current optimal candidate, resulting in a smooth global optimal path if the execution time is infinite. A script is implemented and used to control the iteration and the update of precision p and number of points n.

4 ACEGIO-based path planning Algorithm

This chapter will cover development of ACEGIO-F-GD algorithm, and it will also cover description and development of ACEGIO-GD based path planning algorithm.

4.1 ACEGIO-F-GD

CEGIO-F based path planning is available for generating the optimal path. However, the execution process of the algorithm is extremely time-consuming, due to high time consumption on requesting counterexamples from SMT solvers, especially when the number of points on the path is relatively large. For this specific reason, to improve the efficiency of CEGIO based path planning algorithm, times of requesting counterexamples must be reduced. Therefore, CEGIO assisted by Gradient Descent (ACEGIO-GD) is applied. According to the capability of

ACEGIO-GD which is discussed in Background chapter, it can be applied to optimisation problems, no matter the cost function is convex or not. Thus ACEGIO-GD can generate the optimal path for the path planning problem.

Since the cost function of path planning is convex, CEGIO-F assisted by Gradient Descent is capable of finding the global optimality. Thus, ACEGIO-F-GD (CEGIO-F assisted by Gradient Descent) is applied which is proposed specifically for solving optimisation problems whose cost function is convex. ACEGIO-F-GD employs Gradient Descent algorithm to assist CEGIO-F algorithm, which is relatively faster than general ACEGIO-GD algorithm. Based on Eq. 3 and CEGIO-F in figure 1, the pseudocode of ACEGIO-F-GD is described in figure 6.

```
input : A cost function $f(\mathbf{x})$, the space for constraint set $\Omega$, and a desired
        precision $\epsilon$ , a Gradient Descent function $G$
output: The optimal decision variable vector $\mathbf{x}^*$, and the optimal value of
        function $f(\mathbf{x}^*)$
1  Initialize $f(\boldsymbol{x}^{(0)})$ randomly and $i = 1$
2  Initialize the precision variable with $p = 1$
3  Declare the auxiliary variables $\boldsymbol{x}$ as non-deterministic integer variables
4  while $p \leq \epsilon$ do
5      Define bounds for $\boldsymbol{x}$ with the ASSUME directive, such that $\mathbf{x} \in \Omega^k$
6      Describe a model for $f(\boldsymbol{x})$
7      do
8          Constrain $f(\boldsymbol{x}^{(i)}) < f(\boldsymbol{x}^{(i-1)})$ with the ASSUME directive
9          Verify the satisfiability of $l_{optimal}$ given by Eq. (7). with the ASSERT directive
10         Update $\boldsymbol{x}^* = \boldsymbol{x}^{(i)}$ and $f(\boldsymbol{x}^*) = f(\boldsymbol{x}^{(i)})$ based on the counterexample
11         If $\neg l_{\text{optimal}}$ is SAT, Do $i = i + 1$, then $x^i = G(x^{(i-1)})$ and $f(x^i) = f(G(x^{(i-1)}))$
12     while $\neg l_{optimal}$ is satisfiable
13     Update set $\Omega^k$
14     Update the precision variable p
15 end
16 $\boldsymbol{x}^* = \boldsymbol{x}^{(i-1)}$ and $f(\boldsymbol{x}^*) = f^{(i-1)}(\boldsymbol{x})$
17 return $\boldsymbol{x}^*$ and $f(\boldsymbol{x}^*)$
```

Figure 6. Pseudocode of ACEGIO-F-GD

The different between CEGIO-F and ACEGIO-F-GD is that ACEGIO-F-GD employs Gradient Descent to iteratively calculate optimal candidates in line 11 instead of iteratively generating optimal candidates by requesting counterexamples. To be more specific, for a given precision in ACEGIO-F-GD, requesting counterexamples only happens once when it reaches the inner loop, resulting in an optimal candidate. Then Gradient Descent is iteratively used to achieve a new optimal candidate who is closer to the global optimal solution for the given precision. Contrary to CEGIO-F, ACEGIO-F-GD greatly reduces the time consumption.

An example in Figure 7 shows how ACEGIO-GD can achieve global minimum and how ACEGIO-GD improves the efficiency with a simple non-convex cost function. The example presents possible optimisation processes for CEGIO and ACEGIO-GD optimising the same cost function f(x), who has one variable x as decision variable. The optimising processes of CEGIO and ACEGIO-GD are presented on the left side and right side of Figure 7, respectively. The red dot represents global minimum or local minimum. The orange dot is current optimal candidate position updated by requesting counterexamples from SMT solvers over non-deterministic decision variable. The green dot is current optimal candidate position updated by Gradient Descent. The orange arrow is the optimising path from last optimal candidate to current optimal candidate, generated by requesting counterexamples. The green arrow is the optimising path generated by Gradient Descent.

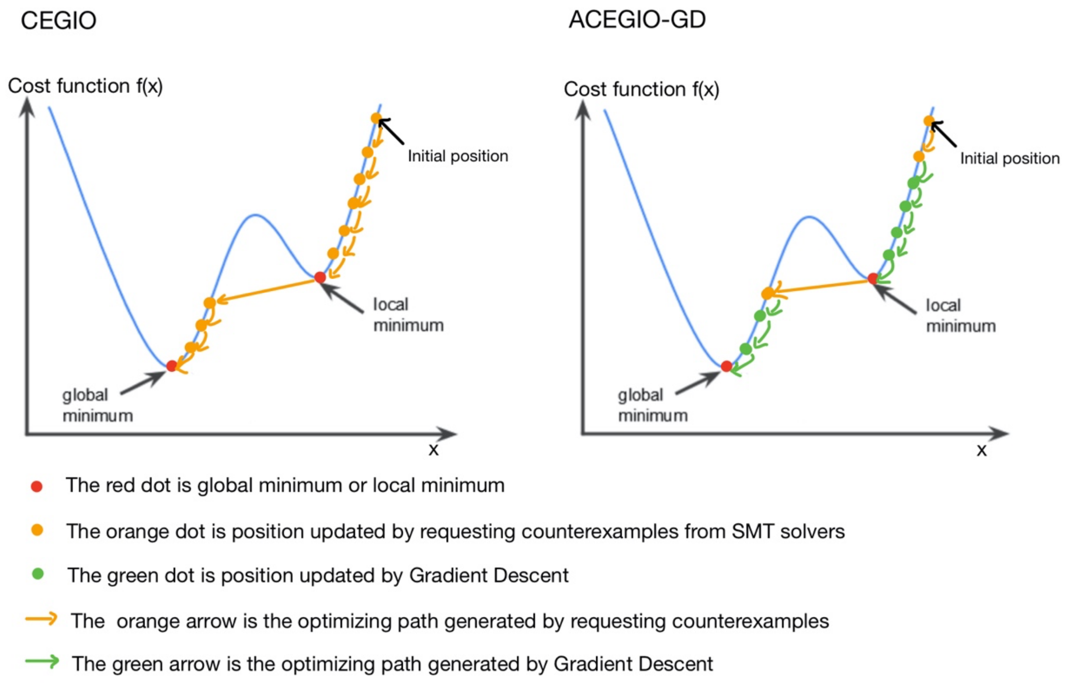

CEGIO
Cost function f(x)
Initial position
local minimum
global minimum
x
ACEGIO-GD
Cost function f(x)
Initial position
local minimum
global minimum
x
The red dot is global minimum or local minimum
The orange dot is position updated by requesting counterexamples from SMT solvers
The green dot is position updated by Gradient Descent
The orange arrow is the optimizing path generated by requesting counterexamples
The green arrow is the optimizing path generated by Gradient Descent

Figure 7: Example explains how ACEGIO achieves global optimality and improves efficiency

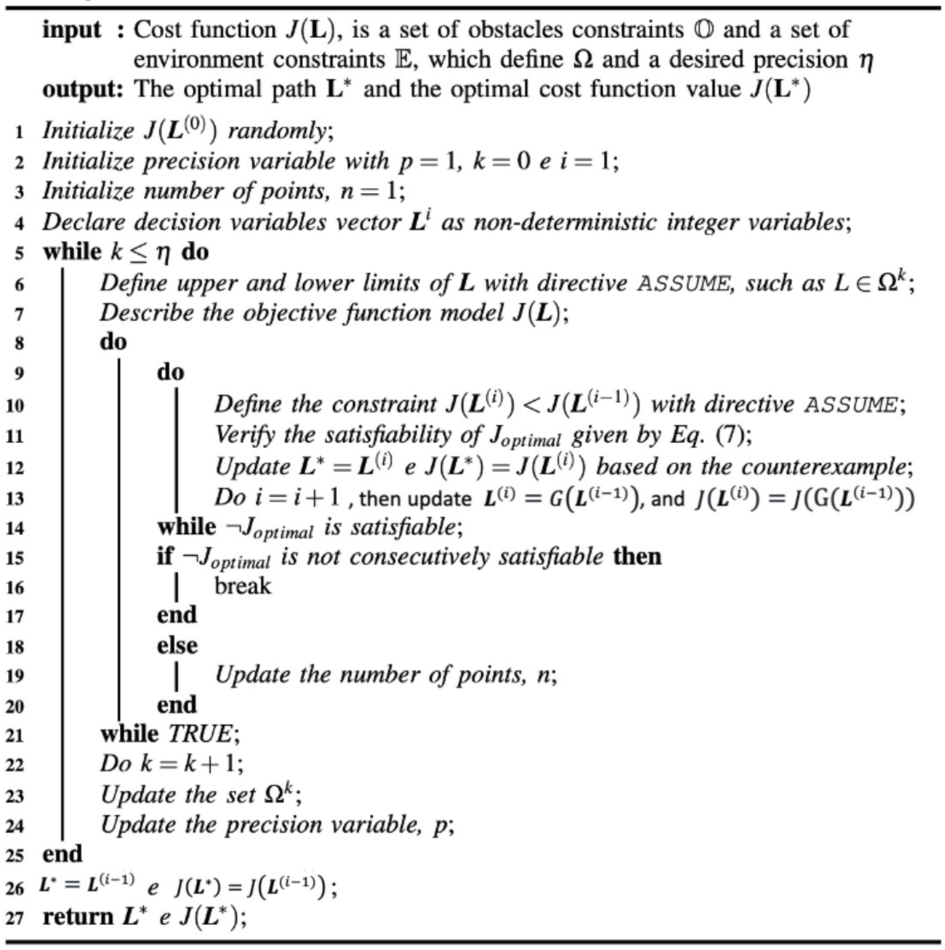
As for how ACEGIO-GD achieves global optimality, in the above example, from the initial position, ACEGIO-GD generated a new optimal candidate position by requesting counterexamples. Then iteratively employ Gradient Descent to achieve new optimal candidate till it traps in the local minimum, where general Gradient Descent algorithm cannot get it out. Thus, it needs to request counterexamples from SMT solvers and generates a more optimised candidate. Then Gradient Descent is iteratively applied till it reaches the global minimum. This example demonstrates how ACEGIO-GD achieves global optimum, and the working principle is similar for other cost functions.

As for how ACEGIO-GD improves the efficiency comparing with CEGIO, in the above example, a possible path of optimising the cost function by CEGIO is presented on the left side of Figure, and the optimising process of ACEGIO-GD is on the right. Comparing to the time consumption of requesting counterexamples (orange arrow), the time consumption of Gradient Descent (green arrow) can be ignored. Comparing the paths of optimising processes for CEGIO and ACEGIO-GD, much fewer orange arrows for ACEGIO-GD, which means times of requesting counterexamples are greatly reduced. Therefore, the time efficiency of ACEGIO-

GD is significantly improved. The working principle can be expanded to all cost function optimisation problem.

4.2 ACEGIO-GD-based Path Planning Algorithm

In this report, ACEGIO-F-GD is applied to solve path planning problem. Figure 8 shows the pseudocode of ACEGIO-F-GD based Path Planning. The only difference between this ACEGIO-GD (ACEGIO-F-GD is simply written as ACEGIO-GD) based path planning (Figure 8) and CEGIO-based path planning (Figure 2) is that Gradient Descent is applied in line 13 to update $\mathbf{L}^{(i)} = G(\mathbf{L}^{(i-1)})$ and $J(\mathbf{L}^{(i)}) = J(G(\mathbf{L}^{(i-1)}))$, if $\neg J_{\text{optimal}}$ in line 11 is satisfiable. Besides, the input additionally includes a Gradient Descent function G.



```
input : Cost function $J(\mathbf{L})$, is a set of obstacles constraints $\mathbb{O}$ and a set of
        environment constraints $\mathbb{E}$, which define $\mathbf{\Omega}$ and a desired precision $\eta$
output: The optimal path $\mathbf{L}^*$ and the optimal cost function value $J(\mathbf{L}^*)$
1  Initialize $J(\boldsymbol{L}^{(0)})$ randomly;
2  Initialize precision variable with $p = 1$, $k = 0$ e $i = 1$;
3  Initialize number of points, $n = 1$;
4  Declare decision variables vector $\boldsymbol{L}^i$ as non-deterministic integer variables;
5  while $k \leq \eta$ do
6      Define upper and lower limits of $\boldsymbol{L}$ with directive ASSUME, such as $L \in \Omega^k$;
7      Describe the objective function model $J(\boldsymbol{L})$;
8      do
9          do
10             Define the constraint $J(\boldsymbol{L}^{(i)}) < J(\boldsymbol{L}^{(i-1)})$ with directive ASSUME;
11             Verify the satisfiability of $J_{optimal}$ given by Eq. (7);
12             Update $\boldsymbol{L}^* = \boldsymbol{L}^{(i)}$ e $J(\boldsymbol{L}^*) = J(\boldsymbol{L}^{(i)})$ based on the counterexample;
13             Do $i = i + 1$ , then update $\boldsymbol{L}^{(i)} = G(\boldsymbol{L}^{(i-1)})$, and $J(\boldsymbol{L}^{(i)}) = J(G(\boldsymbol{L}^{(i-1)}))$
14         while $\neg J_{optimal}$ is satisfiable;
15         if $\neg J_{optimal}$ is not consecutively satisfiable then
16             break
17         end
18         else
19             Update the number of points, n;
20         end
21     while TRUE;
22     Do $k = k + 1$;
23     Update the set $\Omega^k$;
24     Update the precision variable, p;
25 end
26 $\boldsymbol{L}^* = \boldsymbol{L}^{(i-1)}$ e $J(\boldsymbol{L}^*) = J(\boldsymbol{L}^{(i-1)})$;
27 return $\boldsymbol{L}^*$ e $J(\boldsymbol{L}^*)$;
```

Figure 8. Pseudocode of ACEGIO-F-GD based Path Planning algorithm

Gradient Descent algorithm is an optimisation algorithm which is capable to find local optimality of optimisation problems whose cost function is differentiable. The basic idea has been discussed in Background chapter, which is to iteratively move towards the direction of steepest descent. Gradient descent is defined as follows.

Definition 8. For a multi-variable function $F(x)$ which is differentiable in its whole domain. For every point $\mathbf{x}_n$ of the function, $F(\mathbf{x}_n)$ decreases fastest if one moves from $\mathbf{x}_n$ against the gradient of F at $\mathbf{x}_n$, $-\nabla F(\mathbf{x}_n)$. It can be described as:

For every point $\mathbf{x}_n$ of the function, if $\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), n \geq 0$,

$$\text{then } F(\mathbf{x}_n) \geq F(\mathbf{x}_{n+1}), \tag{9}$$

where γ_n is the value of step size, which is allowed to change at every iteration, $-\nabla F(\mathbf{x}_n)$ is the steepest direction of function $F(x)$ at current point $\mathbf{x}_n$.

According to Eq. 4, cost function can be written as:

$$J(x_1, y_1, x_2, y_2, \dots, x_n, y_n) = \sum_{i=1}^{n-1} \sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}, \tag{10}$$

where i is the index of points, (x_i, y_i) is point on path, n is the number of points on the path.

Assume every point on the path $(x_i, y_i) \in \mathbb{R}^2$, thus the cost function of the path planning problem can be considered as a differentiable function with $2n$ variables. Combining Definition 8 and equation $\mathbf{L}^{(j+1)} = G(\mathbf{L}^{(j)})$, $\mathbf{L}^{(j+1)}$ can be written as:

$$\mathbf{L}^{(j+1)} = G(\mathbf{L}^{(j)}) = \mathbf{L}^{(j)} - \gamma_j \nabla J(\mathbf{L}^{(j)}), \tag{11}$$

where $\mathbf{L} = [x_1, y_1, x_2, y_2, \dots, x_n, y_n]$, j is the index of iteration in Gradient Descent, $\mathbf{L}^{(j+1)}$ is the new decision variable vector generated by Gradient Descent.

For each variable x_i or y_i, the partial derivate can be written as:

$$\nabla J(x_i) = \frac{x_i - x_{i-1}}{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}} + \frac{x_i - x_{i+1}}{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}} \tag{12}$$

$$\nabla J(y_i) = \frac{y_i - y_{i-1}}{\sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2}} + \frac{y_i - y_{i+1}}{\sqrt{(x_i - x_{i+1})^2 + (y_i - y_{i+1})^2}} \tag{13}$$

Gradient descent G of the cost function J is defined as above equations. A new decision variable vector $\mathbf{L}^{(j+1)}$ can be achieved by following Eq. 11, Eq. 12, Eq. 13. Since every point on the path (x_i, y_i) is assumed in the range of $\mathbb{R}^2$, it is necessary to make the new calculated x_i, y_i keep k decimal places for a given precision p. Then a new optimal candidate based on the new decision variables can be generated by Gradient Descent.

5 Trajectory planning

A desired path consisting of a set of points is achieved by previous path planning algorithms, then trajectory planning is applied to ensure smooth movements of the mobile robot. In this project, trajectory planning is firstly implemented on a virtual robot, in order to test the trajectory planning algorithm and simulate robot motion in a virtual environment before applying in real world. Robot simulation also decreases workload than directly testing the algorithm on a physical robot. After obtaining the appropriate trajectory planning algorithm, it is applied to a physical robot in a real pre-defined environment.

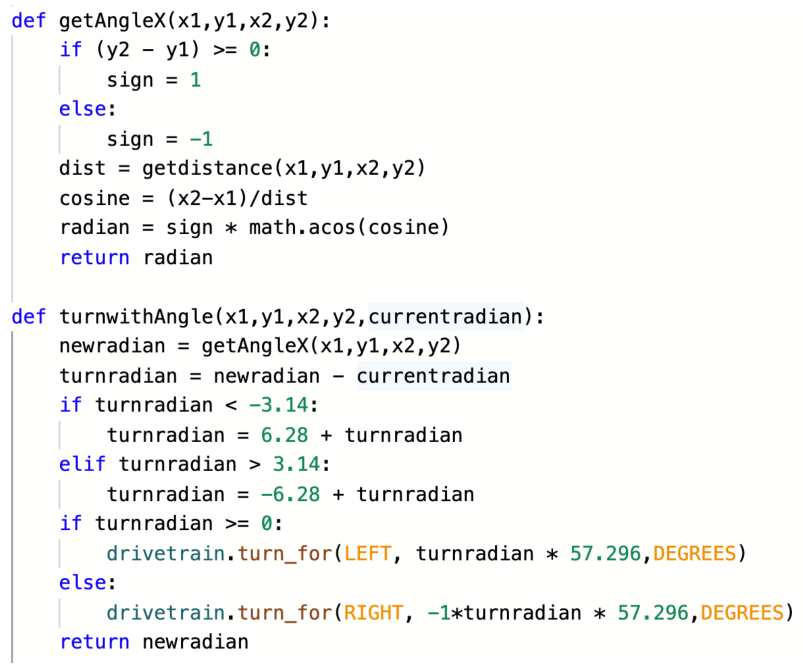
In this chapter, it will firstly discuss robot simulation tool and present steps of smooth trajectory planning. Following this will be physical robot design and implementation.

5.1 Robot simulation

VEXcode VR is a platform that allows you to code a virtual robot using a custom developed Text-based Python interface. The virtual robot is pre-built which use drivetrains to navigate and use pen features to code a creative drawing and record the path. Different virtual three dimensional playgrounds are available, including a grid world, an art canvas, and a walled maze. Besides, point of view can be controlled by users as the robot runs the code.

As the path is consisted of a sequence of straight segments, which is determined by a set of points on the path, trajectory planning in this case takes the set of points as input and make sure the virtual robot consecutively reaches every point on the path following the corresponding straight segment smoothly. By repeating the following two steps, smooth robot motion can be achieved. Step 1: rotate the robot with a certain angle in current position to make it point towards next position. Step 2: move forward to next position.

To achieve step 1, the certain rotation angles need to be calculated. The rectangle environment is defined in the first quadrant in Cartesian coordinate system with its left bottom at origin of coordinate. Figure 9 shows the code of calculating the rotation angle.



```python
def getAngleX(x1,y1,x2,y2):
    if (y2 - y1) >= 0:
        sign = 1
    else:
        sign = -1
    dist = getdistance(x1,y1,x2,y2)
    cosine = (x2-x1)/dist
    radian = sign * math.acos(cosine)
    return radian

def turnwithAngle(x1,y1,x2,y2,currentradian):
    newradian = getAngleX(x1,y1,x2,y2)
    turnradian = newradian - currentradian
    if turnradian < -3.14:
        turnradian = 6.28 + turnradian
    elif turnradian > 3.14:
        turnradian = -6.28 + turnradian
    if turnradian >= 0:
        drivetrain.turn_for(LEFT, turnradian * 57.296,DEGREES)
    else:
        drivetrain.turn_for(RIGHT, -1*turnradian * 57.296,DEGREES)
    return newradian
```

Figure 9. Code of calculating the rotation angle

The angles and robot orientations are all represented relatively to X-axis in the range from -180 degree to 180 degree. The positive direction of X-axis is 0 degree, and the degrees in first or second quadrant are positive. The certain rotation angle depends on current orientation and the angle of intersection between next segment (current point to next point) and the positive direction of X-axis. The function getAngleX returns a radian value of the angle between the certain segment and positive direction of X-axis with the help of cosine and arccosine. Since the range of arccosine output is from 0 to 180, the quadrant which the angle is in can be generated with the help of sign value (1 or -1) in function getAngleX.

The rotation angle is calculated, and robot rotation is achieved by function turnwithAngle. It computes the angle of the next segment on the path and get the rotation angle by subtracting current orientation from the new angle. Then rescale the angle in the range of semicircle to get the smallest rotation angle.

To achieve Step 2, the distance of the straight segment is calculated based on Euclidian distance formula. Then use the drivetrains to move forward with certain distance.

5.2 Physical robot

As aforementioned in background chapter, the physical robot in this report is built with Lego Mindstorms EV3 robotics kit, which contains two motors, a programmable and intelligent brick to use programs to control the motors, and various building components. ROBOTC is a cross-robotics-platform programming language, and it is used to write the controlling program in this report.

The final version of the robot is showed in Figure 10. It uses two caterpillar tracks, and each track is consisted of a rear-wheel, a front-wheel and a rubber track, to move and rotate. And only the two rear-wheels are connected by the two large motors. The design of the robot is based on military tank, which has the capability to rotate in place. As the virtual robot can rotate in place in previous robot simulation, the physical robot needs to have the same ability to ensure smooth trajectory with applying previous trajectory planning method. It rotates by making the rear-wheels spin with the same speed in opposite directions (one clockwise, another counter clockwise).

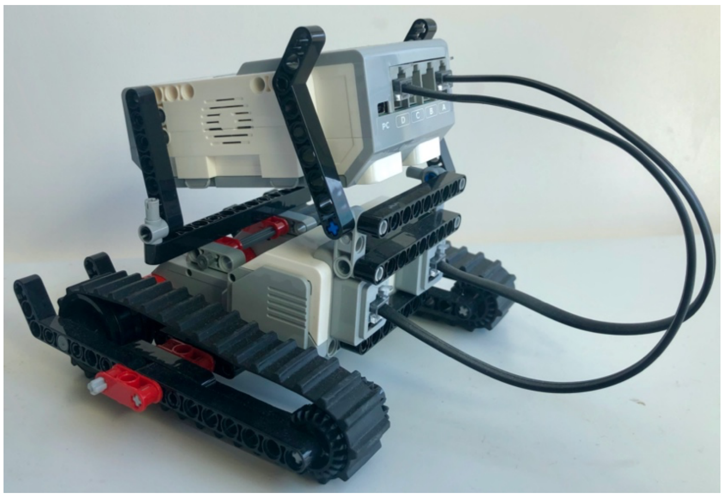


Figure 10. Final version of physical robot

The previous version of robot was based on the same skeleton as rear-wheel-drive car. This design was aborted because it did not fit well with the previous trajectory planning method, since it cannot accurately rotate without changing its current position. Besides, wheel-drive car has more requirements on the surface than caterpillar.

The program of trajectory planning for this physical robot is similar to the previous one for the virtual robot. It also achieves smooth motion by repeating the two steps which are aforementioned.

6 Experimental evaluation

In this chapter, it will cover evaluation of CEGIO-based path planning algorithm, ACEGIO-based path planning algorithm, and trajectory planning. It will also discuss achievements of the project aims. Firstly, it will begin with the experimental objectives and description of the path planning algorithms and trajectory planning. Next, it will introduce the experimental setup. Finally, it will discuss the experimental results and present results analysis.

6.1 Experimental Objectives and Description

In this section, it will firstly discuss the experimental objectives and description of both path planning algorithms as follows. To be brief, CEGIO-based path planning algorithm is labelled with Algorithm 1, and ACEGIO-based path planning algorithm is labelled with Algorithm 2.

CEGIO-based path planning algorithm (Algorithm 1) and ACEGIO-GD-based path planning algorithm (Algorithm 2) are evaluated and compared via following experiments. The goal of all the experiments in different environment settings is to generate a set of points which composed the global optimal path or a path close to the global optimal path from starting position to target position.

Two environment settings are designed with meter as the measurement unit. The shape of both settings is square, whose side length is 10. Starting points and target points are the same in both settings, which is (1, 1) and (9, 9) respectively. The only difference is the obstacles. In first setting (setting 1 in Figure 11), one obstacle is applied whose centre is in (5,5) and radius is 2.5. In Setting 2, two obstacles are applied which are centred in (4,3) and (8,7) with 1.5 and 1 as their radius, respectively. The safety margin in both settings is 0.5. Figure 11 shows the obstacles with blue line and safety margin with dotted red line.

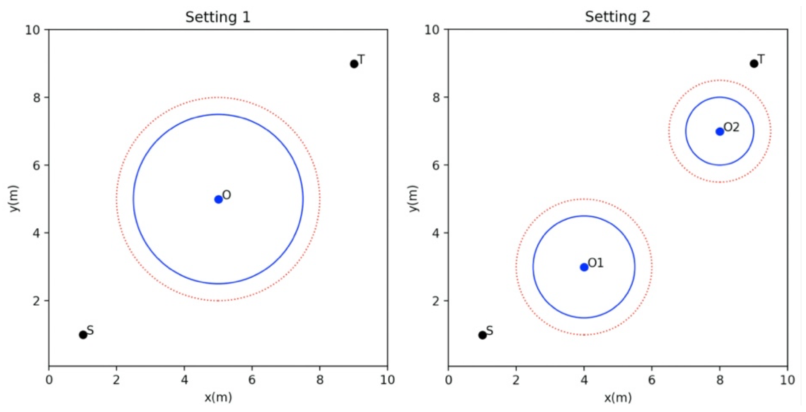
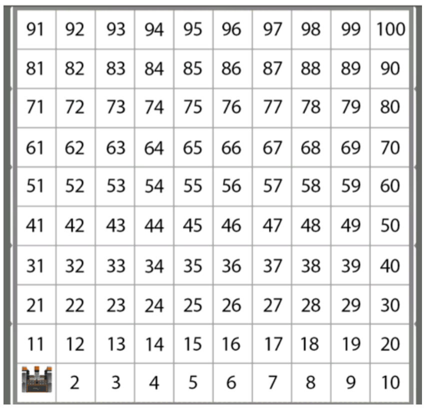


Figure 11. Environment settings

In total, four experiments were conducted. For each path planning algorithm which needs to be evaluated in this chapter, it is applied to both settings, in order to compare its performance in different environments. For each setting, Algorithm 1 and Algorithm 2 is respectively applied to achieve the goal, in order to compare their accuracy, efficiency, and performance in the same environment.

Additionally, this section will discuss experimental objectives and description of trajectory planning as follows. Trajectory planning is evaluated by applying to both virtual robot and physical robot. Firstly, trajectory planning applied to virtual robot is evaluated. Then the physical robot will be evaluated if the robot simulation successes. The goal of experiments for trajectory planning is to achieve smooth robot motion (including rotation and moving forward) based on the trajectory.

Virtual movement environment is designed as in Figure 12. The 10x10 square environment is consisted of 100 small squares with their indices. The virtual robot in Figure 12 is at the starting point (1, 1) and it is in small square 1. The experiment starts with (1, 1) as starting point and makes the robot move towards a sequence of points in order, which is point (2, 6), point (4, 8) and destination point (9, 9). The corresponding square indices of these points are 1, 52, 74 and 89, respectively.



91	92	93	94	95	96	97	98	99	100
81	82	83	84	85	86	87	88	89	90
71	72	73	74	75	76	77	78	79	80
61	62	63	64	65	66	67	68	69	70
51	52	53	54	55	56	57	58	59	60
41	42	43	44	45	46	47	48	49	50
31	32	33	34	35	36	37	38	39	40
21	22	23	24	25	26	27	28	29	30
11	12	13	14	15	16	17	18	19	20
	2	3	4	5	6	7	8	9	10

Figure 12. Virtual environment

As for experiment of trajectory planning applied to the physical robot, it takes four (2x2) square ground tiles as experimental environment. A coordinate system is built, and the left bottom point of the left bottom tile is set as original point, whose coordinate position is set as (1, 1). The side length of each tile is 40 centimetres, and each side length in the coordinate system takes 4 units. Therefore, each unit in the coordinate system is 10 centimetres. Besides, coordinate position of physical robot depends on its geometric centre position. In Figure 13, the position of physical robot is at (1, 1). The trajectory planning on physical robot experiment starts with point (1, 1), and follows the sequence of points, which is point (2, 6), point (4, 8) and destination point (9, 9). The criterion is to check whether it can achieve smooth motion based on the desired trajectory, which is presented as the blue line in Figure 13.

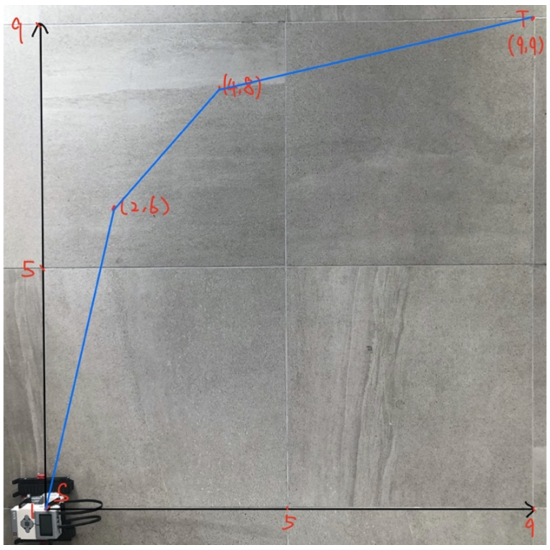


Figure 13. Physical robot experimental environment

6.2 Experimental Setup

All the path planning experiments were conducted on 2.3 GHz OCTA Intel Core i9 processor with 16GB of RAM, running macOS Catalina 10.15.6 64-bits. For each experiment, the execution time, which is the average of five executions, is measured in seconds based on CPU time. The maximum execution time for setting 1 and setting 2 is two days and three days respectively. However, memory consumption is not restricted. As for software, ESBMC 6.4.0 is selected as verification tool and boolector is chosen as SMT solver.

As for trajectory planning, virtual robot is simulated via platform VEXcode VR. Besides, physical robot is built with Lego EV3 kit and is controlled by ROBOTC based programs.

6.3 Experimental Results

In this section, it will firstly discuss experimental results of the path planning algorithms and present the analysis. Following this will be the results of trajectory planning.

The paths obtained by CEGIO-based path planning algorithm and ACEGIO-based path planning algorithm are presented in Figure 14.

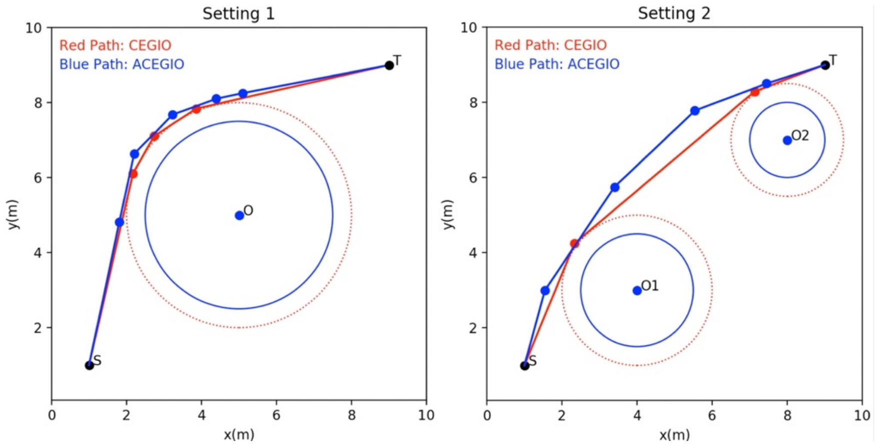


Figure 14. Paths generated by the two algorithms in different settings

For both settings, red paths in Figure 14 were obtained by Algorithm 1, and the blue paths were obtained by Algorithm 2. For Setting 1, a path composed by five points (including starting point and target point) was generated by Algorithm 1, and a path with seven points was generated by Algorithm 2. As for Setting 2, a path with four points and a path with six points was obtained by Algorithm 1 and Algorithm 2, respectively. Both algorithms in both settings suffered the pre-set timeout.

Figure 15 shows the value of cost function in path planning problem and the values were obtained by these two algorithms in different iteration. The iteration in abscissa represents the times of requesting counterexamples over non-deterministic variables, which is the most time-consuming step in both algorithms. The red line represents the tendency for cost function values obtained by Algorithm 1, and blue line is the tendency for cost function values generated by Algorithm 2. The cost function value continuously decreases at each iteration and converges to the global optimality. Note that all the solutions in both scenarios are not the global optimal solution because of the pre-set timeout, but they are very close to the global optimality. Therefore, the aim of path planning has been achieved, since both path planning algorithms have successfully generated paths close to global optimum within the timeout.

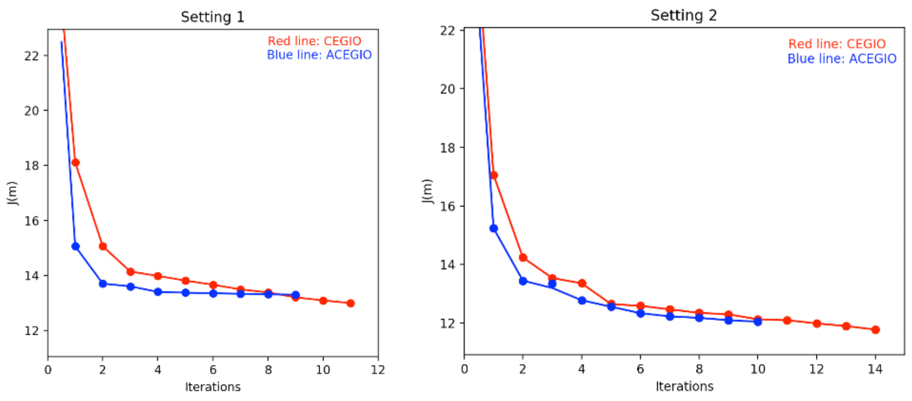


Figure 15. Value of cost function

In Figure 15, ACEGIO-based path planning algorithm can get a solution which is closer to global optimal solution with fewer times of iteration. Therefore, ACEGIO-based path planning algorithm does improve the efficiency of finding a more optimized solution within the first several iterations, comparing with CEGIO-based path planning. Hence, the extended section goal of improving CEGIO-based path planning algorithm's efficiency has been achieved with assistance of Gradient Descent.

However, setting the same timeout (like three days) with CEGIO-based path planning algorithm, the solution generated by Algorithm 2 is slightly farther from the global optimal solution. Thus, if the timeout is the same and long enough, CEGIO-based path planning algorithm has higher accuracy to get the solution which is closer to global optimal path. The following two points can explain. Firstly, an appropriate step in Gradient Descent algorithm, which is determined by the gradient and also leg length, is hard to be selected due to changes of precision. For example, after applying Gradient Descent, the new generated decision variables, which are the points on the path, have some possibilities to be in the range of obstacles, especially like points at the edge of safety margin on the red path from Setting 1 in Figure 14. In this case, it is difficult to choose a suitable step and keep the new decision variables outside the obstacles in its precision. Secondly, there were more points on the path obtained by Algorithm 2, which takes much longer execution time for each iteration. Although Algorithm 2 has generated a path which is greatly close to the global optimum. However, comparing with Algorithm 1 within the same and long execution time, Algorithm 2 has fewer iterations, resulting in solutions which are relative farther from the global optimal solution.

With longer execution time, CEGIO-based path planning algorithm and ACEGIO-based path planning algorithm can find a solution which is closer to global optimal solution. Note that after several iterations, the cost function value only slightly decreases. However, the execution time of each iteration significantly increases. Thus, there is trade-off between execution time and how optimal the path solution is.

Additionally, the result of trajectory planning on virtual robot is in Figure 16. The black line is the trajectory which tracks the robot motion. Following the experiments, both virtual robot and physical robot can achieve smooth motion, including the process of rotating and moving towards the desired position. Besides, both robots have high motion accuracy, which means the robots will make accurate rotation and move forwards for accurate distances in order to reach each point on the path. The results of trajectory planning are the same as the expected experimental results. Therefore, the aims of trajectory planning have been achieved.

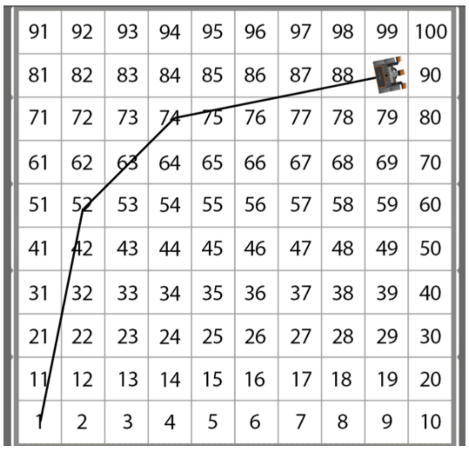


Figure 16. Result of trajectory planning on virtual robot

7 Conclusion

In this chapter, it will firstly discuss my reflection in my project, and it will present and highlight achievements of my project. Next, it will discuss possible future work.

7.1 Reflection

The main reason of choosing this project was originally from the path planning algorithms learnt from Course Unit “Algorithms and Imperative Programming” in second year, including depth first search, Dijkstra, and A* algorithm. However, all these algorithms can only solve node-based path planning problems based on graph data structure. Therefore, CEGIO-based path planning algorithm is a novel and refreshing approach without limitation of encoding movement environment by a specific data structure. Besides, I was eager to dive into the interesting field that I was completely unfamiliar with. These attracted me to choose this project.

My original project “Counterexample-Guided Optimisation Applied to Mobile Robot Path Planning” is to develop and evaluate off-line path planning based on CEGIO algorithm. This led me to begin my project with investigating technical background such as propositional logic, SMT solver, Bounded Model Checking, and optimisation problem. Since I was not familiar with these areas which were hard to comprehend, it took several weeks to choose and understand related background materials. After comprehending the recommended materials, I implemented CEGIO-based path planning algorithm and evaluated it with experiments in different environment settings. The experiments generated the desired paths took several days, which achieved the project aim but was a significantly time-consuming process. Therefore, I further proposed Assisted CEGIO-based path planning with Gradient Descent as project extension, which can greatly improve the efficiency theoretically. Since ACEGIO algorithm was relatively new and had never been applied to solve path planning problem, it took several weeks on choosing the Gradient Descent as auxiliary optimisation algorithm and applying it to path planning problem whose cost function is discrete. Then evaluation of ACEGIO-based path planning algorithm on the same environment setting was completed, resulting in great improvement on the efficiency within specific execution time. At this point in my project, I had completed the original project and expanded my project with proposing ACEGIO-based path planning algorithm. Besides, development and evaluation of the proposed algorithm was also completed as extension section in my project. Additionally, after generating the desired path, smooth trajectory planning based on the path could be achieved. Therefore, I implemented trajectory planning and tested it with virtual robot which realized smooth trajectory planning. Then, I built a physical robot with Lego components and evaluated the trajectory planning in real world. In my project, smooth trajectory planning on both virtual and physical robot has been achieved and also been added as extension. Overall, all the aims of my project including original sections and extension sections had been achieved.

7.2 Future work

There are two main categories for possible future work, regarding path planning environments and ACEGIO-based path planning algorithm. Firstly, in this project, obstacles were modelled as circles, and movement environment was modelled as square. Therefore, future work for encoding different environment settings would be presented, which could increase the application range of CEGIO-based path planning algorithm and ACEGIO-based path planning algorithm. Additionally, this project proposed Gradient Descent algorithm as its auxiliary algorithm for CEGIO algorithm. There are other algorithms like Genetic Algorithm which can also improve the efficiency of CEGIO, and different auxiliary algorithm improves the CEGIO to different degrees. Therefore, another future work would focus on the algorithms which could be applied to ACEGIO.

References

[1] Edwards, D., 2021. Amazon now has 200,000 robots working in its warehouses. Robotics & Automation News. Available from: https://roboticsandautomationnews.com/2020/01/21/amazon-now-has-200000-robots-working-in-its-warehouses/28840 [Accessed 25 February 2021].

[2] En.wikipedia.org. 2021. Motion planning. Available from: https://en.wikipedia.org/wiki/Motion_planning [Accessed 25 February 2021].

[3] Liu, H., 2020. Robot systems for rail transit applications. San Diego: Elsevier.

[4] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 1. Available from: https://arxiv.org/pdf/1708.04028.pdf

[5] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 1. Available from: https://arxiv.org/pdf/1708.04028.pdf

[6] Wikipedia Contributors (2019). A. Wikipedia. Available from: https://en.wikipedia.org/wiki/A.

[7] BAJPAI, P. and M. Kumar (2010). (PDF) Genetic Algorithm - an Approach to Solve Global Optimization Problems. ResearchGate. Available from: https://www.researchgate.net/publication/283361244_Genetic_Algorithm_-_an_Approach_to_Solve_Global_Optimization_Problems.

[8] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 1. Available from: https://arxiv.org/pdf/1708.04028.pdf

[9] R. Araujo, I. Bessa, L. C. Cordeiro, and J. E. C. Filho, "SMT-based verification applied to non-convex optimization problems," in 2016 VI Brazilian Symposium on Computing Systems Engineering (SBESC), Nov 2016, pp. 1–8.

[10] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 1-2 Available from: https://arxiv.org/pdf/1708.04028.pdf

[11] Dumitru CHITORAGA .(2019). Assisted Counterexample Guided Optimisation. The final year project. The University of Manchester. pp. 3

[12] Dumitru CHITORAGA .(2019). Assisted Counterexample Guided Optimisation. The final year project. The University of Manchester

[13] Wikipedia contributors. (2021a, January 29). Optimization problem. Wikipedia. Available from: https://en.wikipedia.org/wiki/Optimization_problem

[14]Types of Optimization Problems | NEOS. (n.d.). Neos-Guide. Available from: https://neos-guide.org/optimization-tree

[15] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 1-2. Available from: https://arxiv.org/pdf/1708.04028.pdf

[16] Koubaa, Anis & Bennaceur, Hachemi & Chaari, Imen & Trigui, Sahar & Ammar, Adel & Sriti, Mohamed-Foued & Alajlan, Maram & Cheikhrouhou, Omar & Javed, Yasir. (2018). Introduction to Mobile Robot Path Planning.

[17] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 1-2. Available from: https://arxiv.org/pdf/1708.04028.pdf

[18] C. Baier and J. Katoen, Principles of model checking. MIT Press, 2008.

[19] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 2-3. Available from: https://arxiv.org/pdf/1708.04028.pdf

[20] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 2-3. Available from: https://arxiv.org/pdf/1708.04028.pdf

[21] C. Baier and J. Katoen, Principles of model checking. MIT Press, 2008.

[22] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 2. Available from: https://arxiv.org/pdf/1708.04028.pdf

[23] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 2. Available from: https://arxiv.org/pdf/1708.04028.pdf

[24] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 2-3. Available from: https://arxiv.org/pdf/1708.04028.pdf

[25] R. Ara´ujo, I. Bessa, L. Cordeiro, and J. E. C. Filho, "Counterexample guided inductive optimization," in arXiv:1704.03738 [cs.AI], April 2017, pp. 1–32. Available from: http://arxiv.org/abs/1704.03738

[26] R. Ara´ujo, I. Bessa, L. Cordeiro, and J. E. C. Filho, "Counterexample guided inductive optimization," in arXiv:1704.03738 [cs.AI], April 2017, pp. 18–24. Available from: http://arxiv.org/abs/1704.03738

[27] Dumitru CHITORAGA .(2019). Assisted Counterexample Guided Optimisation. The final year project. The University of Manchester. pp. 3

[28] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 3-4. Available from: https://arxiv.org/pdf/1708.04028.pdf

[29] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 3. Available from: https://arxiv.org/pdf/1708.04028.pdf

[30] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 3. Available from: https://arxiv.org/pdf/1708.04028.pdf

[31] R. Ara´ujo, I. Bessa, L. Cordeiro, and J. E. C. Filho, "Counterexample guided inductive optimization," in arXiv:1704.03738 [cs.AI], April 2017, pp. 18–19. Available: http://arxiv.org/abs/1704.03738

[32] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, pp. 3-5. Available from: https://arxiv.org/pdf/1708.04028.pdf

[33] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 4. Available from: https://arxiv.org/pdf/1708.04028.pdf

[34] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 4. Available from: https://arxiv.org/pdf/1708.04028.pdf

[35] R. F. Araujo, A. Ribeiro, I. V. Bessa, L. C. Cordeiro, J. E. C. Filho. (2017). Counter- example Guided Inductive Optimisation Applied to Mobile Robots Path Planning, p. 5. Available from: https://arxiv.org/pdf/1708.04028.pdf