On the Use of Optimal Search Algorithms with Artificial Potential Field for Robot Soccer Navigation

DONG, Chen

Chief Supervisor

Reyes, Napoleon H., Ph.D

Co-Supervisor

Barczak, Andre L., Ph.D

Computer Science

Master of Science

January 23, 2018

Contents

List of Figures

List of Tables

Abstract

The artificial potential field (APF) is a popular method of choice for robot navigation, as it offers an intuitive model clearly defining all attractive and repulsive forces acting on the robot [3] [25] [29] [43] [50]. However, there are drawbacks that limit the usage of this method. For instance, the local minima problem that gets a robot trapped, and the Goal-Non-Reachable-with-Obstacle-Nearby (GNRON) problem, as reported in [51] [5] [23] [2] and [3]. In order to avoid these limitations, this research focuses on devising a methodology of combining the artificial potential field with a selection of optimal search algorithms. This work investigates the performance of the method when using different optimal search algorithms such as the A* algorithm and the any-angle path-planning Theta* Search, in combination with different types of artificial potential field generators. We also present a novel integration technique, whereby the Potential Field approach is utilized as an internal component of an optimal search algorithm, considering the safeness of the calculated paths. Furthermore, this study also explores the optimization of several auxiliary algorithms used in conjunction with the APF-Optimal search integration: There are three different methods proposed for implementing the line-of-sight (LOS) component of the Theta* search, namely the simple line-of-sight checking algorithm, the modified Bresenham's line algorithm and the modified Cohen-Sutherland algorithm. Contrary to the studies presented in [5], [42], [48] and [40] where the APF and the optimal search algorithms were used separately, in this research, an integrative methodology involving the APF inside the optimal search with a newly proposed Safety Factor (SF) is explored. Experiment results indicate that the APF-A* Search with the SF can reduce the number of state expansions and therefore also the running time up to 19.61%, while maintaining the safeness of the path, as compared to APF-A* when not using the SF. Furthermore, this research also explores how the proposed hybrid algorithms can be used in developing multi-objective behaviours of single robot. In this regard, a robot soccer simulation platform with a physics engine is developed as well to support the exploration. Lastly, the performance of the proposed algorithms is examined under varying environment conditions. Evidences are provided showing that the method can be used in constructing the intelligence for a robot goal keeper and a robot attacker (ball shooter). A multitude of AI robot behaviours using the proposed methods are integrated via a finite state machine including: defensive positioning/parking, ball kicking/shooting, and target pursuing behaviours.

Keywords : Artificial Potential Field, Optimal Searches, Robot Navigation, Multi-objective Behaviours.

Chapter 1

Introduction

In the robot navigation problem domain, the most fundamental research issue is how to select the appropriate motion-planning algorithm for specific working environments. From the MIT IAP robotics course [27], various types of useful navigation algorithms can be combined together. Among these algorithms, we have the optimal search algorithms and the artificial potential field; they are widely used in different robotic platforms such as Unmanned Aerial Vehicle (UAV) control [3] [23], indoor robot control [47] and the robot soccer game [32] [35] [36] [39] [41] [43].

There are two major components in the robot navigation problem: target pursuing and obstacle avoidance. Target pursuing is the most fundamental one, which makes the robot move towards the target position. Then, the obstacle avoidance is involved in making the robot avoid the obstacle while pursuing the target. In order to deal with these requirements, the robot navigation problem is divided into two major aspects: path-planning and motion control. Path planning is used to find out the shortest path from the starting position to the goal with the shortest running time that avoids all the obstacles, then the motion control is engaged to control the robot moving along the path and respond to any events happening before the robot arrives at the goal.

In this research, the methodology of using both the Artificial Potential Field (APF) and the optimal search for the robot navigation problem are investigated. Due to the limitation of using individual algorithms separately, the integration of using both the artificial potential field and the optimal search to resolve the limitation of each other is developed and examined. To customize the results of optimal search, a concept from the artificial potential field which is called safety factor (SF) is combined into the optimal search that makes the search algorithm find out not only the shortest path but the best path which has a balance of path length and robot safety as well. Also, alternative generator functions of the artificial potential field are studied to improve the performance of motion planning. Moreover, with the utilization of the optimal search

algorithms, the flaws of the artificial potential field are addressed.

The researches like [54] and [28] have stated that a necessary distance from the obstacle is essential for the path planning problem because in the real practice, the robot is not a solid point but has a volume, which means a margin to the neareast obstacle is compulsory. The drawback of using constant margin will be illustrated in the section 4.2.2, and the methodology of evaluate the dynamic margin using SF will be discovered.

For the purpose of examining the efficacy of the navigation algorithms in real practice, the robot soccer game is preferred which is a suitable research platform requiring the navigation algorithms. Then, due to the complexity of the robot soccer game, multiple intelligent behaviours for the soccer players were developed and examined. Moreover, three different line-of-sight algorithms are developed and examined, as an auxiliary component of the Theta* Search algorithm, since the literature does not provide specific details on how to actually implement it.

For better game simulation, the implementation includes physics mechanics as well as collision detection. A dynamic cell size technique for the grid-world representation is also developed, for the purpose of using optimal search in unknown terrains.

Last but not least, the performance of the navigation algorithms are compared against the experiment results from the other researches. The strengths and the limitations of the proposed methods are also summarized, with a discussion of areas for future work.

1.1 The Problem Domain: Robot Soccer Game

The robotics soccer game is to simulate the real soccer game, which purpose is controlling the robots to pursue and kick the ball into the gate that belongs to the opponent team, and defending the attacking from the opponent team [22]. The behaviors of the robots are controlled by the artificial intelligence automatically. The researches about the robot soccer game like [32], [36],[39] and [41] have indicated that there are multiple abilities to play the game, such as the target pursuing, obstacle avoidance, decision making and cooperating. Moreover, each robot player requires multi-model behaviours for various tasks. For instance, the attacker should have the ability to find out a proper position to kick the ball toward the gate, which requires both the path planning [36] and the obstacle avoidance when moving to the attacking position [41]. Also, the keeper should have the ability of making decision under different circumstance to behave various purposes.

In the robot soccer game, one of the most important issues is the robotics navigation: first, the purpose of the path planning is to find out a proper path from the current position of the robot to the target position which could be either the ball or the position for further attacking or defensing. Then, the algorithm of motion control should have the ability to make the robot moving along the given path and avoiding the obstacles at the same time.

There are also other problem domains of the robot soccer game like the vision processing and the decision making [22]. These domains are combined with the navigation algorithm to manipulate the robot together. Nevertheless, this research will focus on the navigation problem only: All the other components of the robot soccer game are considered as idealized.

1.2 Research Objectives

The major purpose of this research is to investigate the efficacy of using optimal search algorithms with the artificial potential field for the robot navigation problem. Optimal search algorithms are widely used in the area of navigation, but have drawbacks in terms of running time [8][17][42]. On the other hand, the artificial potential field is a high speed algorithm for real-time navigation but has some weaknesses.

The local minima problem is one of the the major concerns for using the artificial potential field [5] [24]. The fundamental mechanism of the artificial potential field is to use the gradient to indicate the moving direction of the agent. However, the local minima may problem appear if there are more than one point having a zero gradient, which causes the agent has the probability of being trapped other than the goal position.

Another problem is called the Goal-Non-Reachable-with-Obstacle-Nearby (GNRON) problem [2]. The GNRON problem will happen if the goal position is too close to the obstacle so that the gradient at the goal position is not 0, or all the gradients of each point around the goal are all pointing away from the goal. The effect is that it prevents the agent to move towards the goal.

This research explores a proper combination of optimal search algorithms and artificial potential field to address the flaws of artificial potential field in robot navigation. Since both of the local minima and the GNRON problems could arise if there is no unique goal on the field, the optimal search algorithms are selected to indicate the only distinct goal even there are multiple or no position having zero gradient. Moreover, the methodology of keeping the advantages of the artificial potential field for the robot navigation will be explored, where the alternated potential field generator functions will be discussed

and examined.

In order to implement the Theta* Search Algorithm, the study of implementing the line-of-sight component of the Theta* Search should be investigated. Due to the various types of the environment, different algorithms are developed to adapt corresponding cases. Furthermore, the utilization and modification of existing algorithms such as the Bresenham's Line Algorithm [12] which is used for mapping the vector graph into pixel and the Cohen-Sutherland Algorithm [14] which is to determine the necessary of rendering in the area of computer graphics will be explored to apply them for line-of-sight check.

In this research, the robot soccer game is used as platform to evaluate the performance of the study result. As it is mentioned in the section 1.1, an idealized simulation platform of the robot socces game will be established to restrict the variation of the research. That is, the vision processing system of the robot soccer game will not be considered.

1.3 Significance of the Research

Although there have been many researches about the navigation problem in the robot soccer game, such as [32], [35] [39], [41], [43], [36], and also a lot of researches about the navigation problem using various algorithms under different environment, for instance, the flight path planning [1], the UAV control [3], the virtual motion [45] and so on, there is few study about using the combination of multiple algorithm in the robot soccer navigation.

This research introduces a new technique of combining the A* and Theta* algorithms with the Potential Field algorithm, in order to add a safety factor to the calculated paths. The novel integration technique makes use of the Potential Field algorithm as an internal component of the A* algorithm, whereby it is used to calculate the g-cost values, taking into account the presence of obstacles around each visited waypoint.

By implementing the corresponding simulation platforms, the comparison between the performances of different algorithms are explored and exposed as well. The implementation includes the mechanics engine for aspects of rigid body motion and collision detective.

In order to implement the Theta* Search, the detailed implementation of the line-of-sight will be addressed and resolved. Moreover, the strategies of the robot soccer players are also discussed and explored.

1.4 Structure of the Thesis

In the chapter 2, the theoretical framework will be introduced, plus the review of the related literature when explaining the corresponding algorithms. The section 2.1 will introduce the A* Search Algorithm and the Theta* Search Algorithm which are based on the Dijkstra Algorithm. Then, the artificial potential field will be introduced in the section 2.2.

The chapter 3 will explore the performance of the related algorithms. The majority of the preliminary exploration is the optimal search algorithms.

In the chapter 4, the algorithm of how to combine the optimal search algorithms with the artificial potential field will be expressed and proved. The selection of the artificial potential field functions will be explored. Also, the discussion of using alternative heuristic in the optimal search are exposed as well. Furthermore, the algorithms of implementing the concept of line-of-sight will be introduced as well, including the modified Bresenham's Line Algorithm and the modified Cohen-Sutherland Algorithm.

After that, the strategies of the robot soccer behaviours will be expressed in the chapter 5, including the generic strategies, the attacker strategies and the keeper strategies. The algorithms of implementing the strategies will be introduced at the same time.

In order to explore the performance of the algorithms, a platform should be constructed as well. In the chapter 6, the implementation of experiment platform will be introduced. The section 6.1 will expose the algorithm of mechanics simulation will is focused on the collision detective and applying the motion control of the artificial potential field. Then the implementation details of the search algorithms will be discussed in the section 6.2. Then, the architecture of the platform will be exposed in the section 6.3.

The experiment result of corresponding algorithms are presented in the chapter 7. First, the performances of using different optimal algorithms with the artificial potential field under different environment have been illustrated. Also, the performance of the safety factor is explored at the same time. Then, the comparison with the related studies are demonstrated. Last but not least, the experiment of the performances of various robot behaviours are exposed.

Chapter 2

Theoretical Framework and Review of Related Literature

In general, the robot navigation problem contains two major parts: path planning and intelligent control. For path planning, the algorithms should have the ability of finding the best path from the start position two the goal. On the other hand, for the intelligent control, the algorithms should have the ability to control the motion of the robot to move along the found path and avoid the obstacle.

These two parts are not totally individual: when a path is planned, the way-points should have a proper tolerant that allow the robot to move through, which is a partial obstacle avoidance as well. Furthermore, the appearance of the path is generally a set of way points $P = \{s_1, s_2, ...s_n\}$, which means the control algorithm should also plan the path between two connected way-points.

That is, the robot navigation could not be implemented by any single algorithm, but requires a combination of multiple algorithms to cooperate.

2.1 Optimal Search Algorithms

The optimal search algorithm is a not a single algorithm, but a set of algorithms based on the Dijkstra Algorithm [9]. The purpose of the optimal search is to find out the shortest path from the start to goal as fast as possible.

2.1.1 Dijkstra Algorithm

The Dijkstra algorithm was originally addressed and proved by Dijkstra, E. W in 1956 [6][10]. As a basis of the optimal search algorithms, this algorithm generating the search tree from a directed map. A sample directed map is shown in figure 2.1, the value of the edges is the cost of moving from one node to the connected node with given direction.

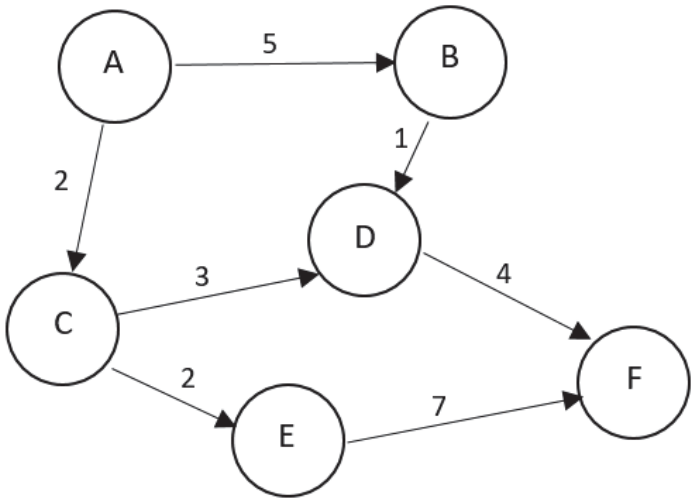


Figure 2.1: A directed map for dijkstra algorithm

In Dijkstra algorithm, the top two essential aspects are the **search queue** S which is a priority queue, and the **visited list** V which is a set that records the node has been searched. During the search processing, the total cost from the start node to the current visited node will be used as the key value of the priority queue, and updated the smallest cost to the visited list.

The structure of a node is

```
1 Node:
2         Node parent := null
3         UNSIGNED cost := +INF
```

And the pseudo code of the Dijkstra algorithm is

```
Initialize:
        {S}:=null
        {V}:=null
        create Node n := {null,0}
        {S}.insert(n)

FindPath:
  WHILE NOT {S}.empty()
        u := {S}.top()
        {S}.pop()

        IF u is Goal THEN
          return u
    ELSE
          IF u in {V} THEN
                NEXTLOOP
          ENDIF

          {V}.insert(u,u.cost)

          FOREACH v as neighbour of u
                IF v in {S} THEN
                  IF  cost(u,v) + u.cost < v.cost THEN
                        {S}.remove(v)
                        {S}.parent = u
                        v.cost = cost(u,v) + u.cost
                        S.insert(v,v.cost)
                  ENDIF
                 ELSE NOT v in {V} THEN
                  v.cost = cost(u,v) + u.cost
                  v.parent = u
                  {S}.insert(v)
                 ENDIF
          ENDLOOP
        ENDIF
  ENDLOOP
  return null
```

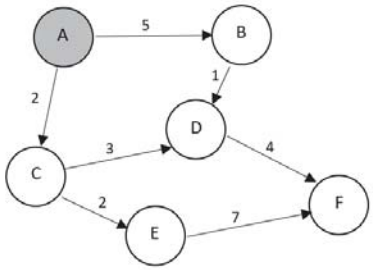
In the initializing phase, both the search queue and the visited list will be empty, then the first node which is also the start node will be inserted into the search queue with the initial cost of 0. Then, during the search process, at each iteration, the node with the

smallest cost will be popped from the search list and examined. First, if the current node u is the goal node, the search will be terminated and return u as the result. Otherwise, u will be added to the visited list, and all the neighbour nodes v will be examined.

If v has been visited, but cost of the path of moving from start through u to v is smaller than the recorded cost, then the parent of v will be replaced by u, and the cost of v will be updated as well. On the other hand, if v is a new node which has not been visited, it will be added to the search queue, and also the cost from s through u to v as the cost of v.

If any goal node has been found, by the traversal of the parent node from goal node to start node, the shortest path will be found as well. However, if there is no result found before the search queue is empty, a result of null will be given out to indicate that there is no valid path from the start node to the goal.

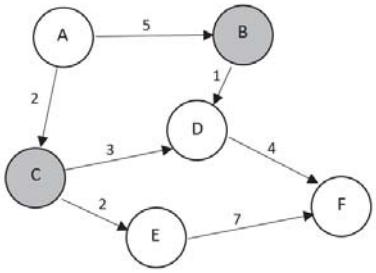
In the sample of figure 2.1, assuming the requirement of the search is to find out a path from the node A and the node F. The following figures demonstrate the major steps of the search: the gray nodes are in the search list, and the cost of the nodes are listed as well.



Search	Visited	Cost
A		0

Figure 2.2: Step 1 - Insert node A into S as initial

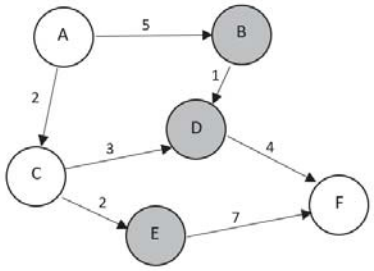
First, as it is shown in the figure 2.2, the node A is inserted into the search list during the initializing phase. The rest of the nodes are still in unknown status.



Search	Visited	Cost
	A	0
B		5
C		2

Figure 2.3: Step 2 - Pop A and insert B, C into S

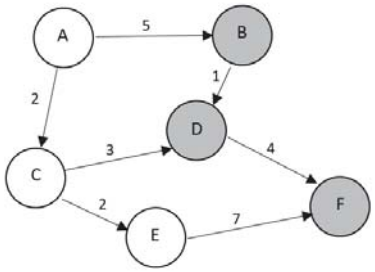
Then the major process will start. The node which has the smallest cost will be popped and examined if it is the goal. Currently, there is only one node A in the search list, so that A is popped and the neighbours of A, which have not been searched or visited are added to the search list, with the cost separately.



Search	Visited	Cost
	A	0
B		5
	C	2
D		5
E		4

Figure 2.4: Step 3 - Pop C which cost is 2, insert D, E

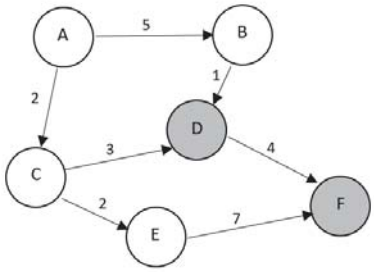
Moving to the next iteration, since C has the smallest cost in the search list, it will be popped like A in previous round. The neighbours of C are inserted into the search list as well.



Search	Visited	Cost
	A	0
B		5
	C	2
D		5
	E	4
F		11

Figure 2.5: Step 4 - Pop E, insert F, but not end

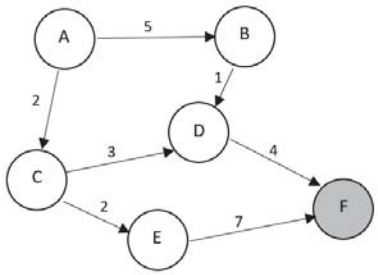
In the next iteration, after the node E is popped and examined, the node F which is the goal node will be added into the search list, with a cost of 11 and parent of E. However, since the goal examining will only be occurred when the node is popped from the search list but not inserted, the search will not be terminated immediately but keep going.



Search	Visited	Cost
	A	0
	B	5
	C	2
D		5
	E	4
F		11

Figure 2.6: Step 5 - Pop B, D is in S but no update

Although F has been in the search list already, there two nodes has smaller cost, and also their cost are exactly the same. Depend on the implementation of the priority queue, both of the could be popped. In this sample, B is popped first and examined, and the node D is left in the search list. Because the cost from A t D through B is 6, which is greater than the current cost of D, D will not be updated, and also no new node will be inserted into the search list.



Search	Visited	Cost
	A	0
	B	5
	C	2
	D	5
	E	4
F		(9)11

Figure 2.7: Step 6 - Pop D, F is in S which has new lower cost

While the node D is popped as well, as the neighbour of D, the cost of the node F will be updated since the cost through D to F is smaller than that through E to F. At the same time, the parent of F will be changed to D as well, so that the shortest path from A to F is not A-C-E-F anymore, but A-C-D-F.

Then the final step, since there is only one node in the search list, F will be popped and returned as the final result of the entire search process.

This sample demonstrates the procedure of the Dijkstra algorithm, and also the significance of recording the cost of each node which ensuring the shortest path will be found.

Although the Dijkstra algorithm has the ability to find out the shortest path, the efficiency is not quite satisfied since all the nodes in the search list will be visited, which reduces the running speed. The minimal time complexity of the Dijkstra algorithm is

O(—E— + —V—log—V—) when using fipponacci heap as the implementation of the priority queue, where V is the count of node and E is the count of edges [11].

2.1.2 A* Search Algorithm

The A* Search Algorithms is a classical search algorithm for finding out the shortest path on a known field. It is widely used in the area of path planning for known terrain [8] [13] [16] [23].

Different from the Dijkstra algorithm, the A* Search introduced another crucial aspect: heuristic. In the A* Search, an evaluated cost from a node n to the possible target node t is established by the heuristic function $\hat{h}(n) \leq h(n,t)$, which had been proved by Hart, Nilsson and Raphael in 1968 [9], where $h(n,t)$ is the actual shortest cost from n to t.

This condition is also known as the consistency of the heuristic [9]. In the original algorithm, the heuristic is evaluated by the euclidean distance d(x,y) from the current node to the target node t.

Another aspect used in the A* algorithm is the expanded list $\{E\}$, which is also known as close set. Different from the visited list in the Dijkstra algorithm, a node will be expanded if and only if the shortest path from the start node to this node has been found. The condition of using expand list is the consistency heuristic, which had been proved by Hart, Nilsson and Raphael as well [9].

In order to simplify the computation work in practice, the search node is introduced. The search node is defined as a series of connected nodes which present that path from the back one to the front one. For example in the figure 2.1 on page 8, a search node $s = \{DCA\}$ indicates that there is a path from the node A through C to the node D. Furthermore, the evaluated cost of this search node $\hat{f}(s) = g(s) + \hat{h}(s)$ is also used as the key value in the search list, as result, a complete search node will be marked like $s = \{8DCA\}$, where the $g(D) = 5$ and $\hat{h}(D) = 3$.

The search list $\{S\}$ of the A* algorithm is the same as that in the Dijkstra algorithm, which is a priority queue as well. Furthermore, the expanded list in the A* algorithm is a set which ensures no duplicate expand may happen.

The pseudo code of implementing the A* algorithm is:

```
SearchNode{
  UNSIGNED gcost := +INF
  UNSIGNED hcost := +INF
  {N} as Node := null
}
GetKey SearchNode N:
  return N.gcost + N.hcost

Initialize:
  {S} := null
  {E} := null
  Create Search Node N
  N.gcost = 0
  N.Add(start)
  N.hcost = distance(start,goal)
  S.insert(GetKey(N),N)
```

```
FindPath:
  WHILE NOT {S}.empty()
          M = {S}.front()
         {S}.pop()
         Node u = M.last()
         IF u is goal THEN
           return M
         ENDIF

         {E}.insert(u)
         FOREACH v is successor of u
             IF v in {E} THEN
                  NEXTLOOP
                ENDIF

                Crate Search Node N:= M
                N.Add(v)
                N.gcost += distance(u,v)
                N.hcost := distance(v,goal)
                {S}.insert(GetKey(N),N)
         ENDLOOP
  ENDLOOP
  return null
```

During the initialization phase of the A* algorithm, both the search list $\{S\}$ and the expand list $\{E\}$ are cleared, and then the first search node N is created. The start node will be added to the search node, and the gcost of N is set to 0 since there is no actual cost. After that, the heuristic function is applied, which is the distance between the start node and the goal node. At last, the initial search node N will be inserted into $\{S\}$, with its fcost as the key value.

When processing the search, at each iteration, the search node M with a key value is the smallest in the search list will be popped and examined. The last added node in M, u, will be compared with the goal node to determine if M is the shortest path found or not. If the node u is not the goal node, it will be expanded immediately, and then all the successors of u will be examined.

The existence in the expanded list of each successor will be checked at this step, and new a search node N is created only for the successor that has not been expanded. The initial value of N is a copy of the search node M, then the successor v will be added to the end of N, and the actual cost from node u to v will be added the gcost of N. The heuristic of N is also need to be replaced by the hcost of the node v but not u anymore. Then the final step is to add the new search node N back to the search list $\{S\}$, and moving to the next iteration.

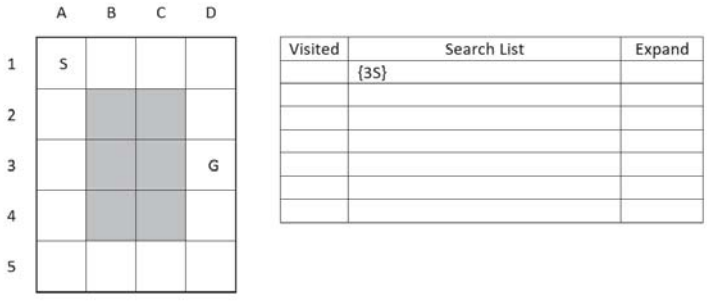
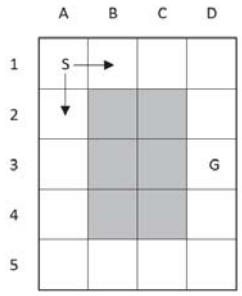


Figure 2.8: The initial status of A* Search

Figure 2.8 demonstrates an initial state of the A* search, where A1 (marked as S) is the start node and D3 (marked as G) is the goal node. The search is restricted to 4-connected, that is only the nodes have shared edge are connected together. The distance between connected nodes is set as 1 as well. In order to simplify the computation, the manhattan distance is also used to replace the euclidean distance, which is defined as the equation 2.1

$$d(P, Q) = max(|P.x - Q.x|, |P.y - Q.y|). \tag{2.1}$$



Visited	Search List	Expand
{3S}	~~{3S}~~	S
{3S-B1}	~~{3S-B1}~~,{4S-A2}	S,B1

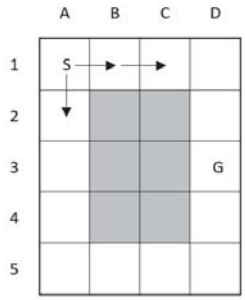
Figure 2.9: Two successors of S

In the main procedure of the search, the search node with the lowest cost will be popped from the list. As it is shown in the figure 2.9, the search node M = 2S is popped, then the last element in the this search node u = S is examined that if it is the goal node or not. Since the goal node is D3 but not S, S is inserted into the expand list, and then the successors of S, B1 and A2, will be selected and checked as well.

Because both B1 and A2 have not been expanded yet, which means the shortest path from S to these two nodes are not found, two new search nodes are generated by extending the path to them separately. For the first new search node S-B1, the gcost is 0+1, where 0 is the gcost of the previous search node S, and 1 is the cost from S to B1. Then the hcost is the manhattan distance from the node B1 to G which is 2, as a result, the fcost of S-B1 is 3.

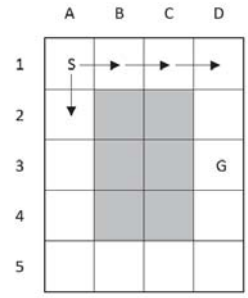
The same processing is also applied to the other new search node S-A2 which fcost is 4 = 1+3. After the successor is processed, the new search node is inserted back to the search list.

After all the successors have been processed, the procedure is moved to the next iteration and the first step is repeated. Since the original search node 3S has been removed already, currently, the search node with the lowest cost is 3S-B1. As a result, this search node is popped, and the last element, B1, is examined at inserted into the expand list as well.



Visited	Search List	Expand
{3S}	~~{3S}~~	S
{3S-B1}	~~{3S-B1}~~,{4S-A2}	S,B1
{3S-B1-C1}	~~{3S-B1-C1}~~,{4S-A2}	S,B1,C1

Figure 2.10: Only one new node, the other are blocked



Visited	Search List	Expand
{3S}	~~{3S}~~	S
{3S-B1}	~~{3S-B1}~~,{4S-A2}	S,B1
{3S-B1-C1}	~~{3S-B1-C1}~~,{4S-A2}	S,B1,C1
{3S-B1-C1-D1}	~~{3S-B1-C1-D1}~~,{4S-A2}	S,B1,C1,D1

Figure 2.11: Further step

The following processing are illustrated in the figure 2.10 and the figure 2.11. For the search node 3S-B1, since only one successor is accessible, the new search node is generated as 3S-B1-C1, with a cost is still smaller than 4S-A2 generated in the previous iteration. As a result, 3S-B1-C1 will be popped before 4S-A2, and the further process is based on the node C1 but not A2.

2.1.3 Object Representation in the 2D Gridworld

In general, the A* search is mainly focussed on the path finding in euclidean space, that $v = r\,\theta$

The 2D search field is defined as a 2D-plane, which divided by numbers of lines along x and y axis. The neighbour grids are connected. The cell is defined as the smallest rectangles on the field crossing by the lines, and the vertex, or the grid is the intersection point of the lines.

There will also be obstacles on the search field. In order to simplify the computation, despite the shape of the obstacles, the blocked cells are used instead of the obstacles.

The following figures 2.12, 2.13 and 2.14 illustrate a 2D search field that which has been divided by the lines with an obstacle on it.

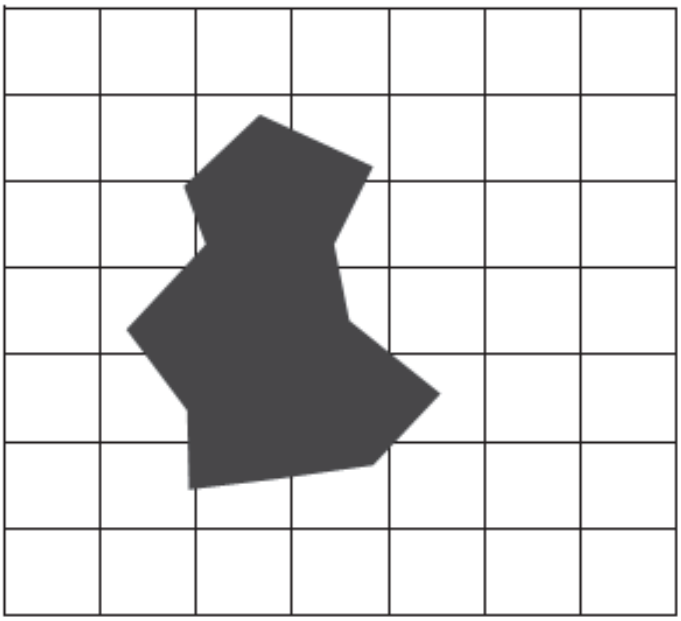


Figure 2.12: A 2D-Plain with obstacle

The cells which are partly or totally covered by the obstacle are marked as gray, which means these node are blocked, and there will not be any path through the blocked cell.

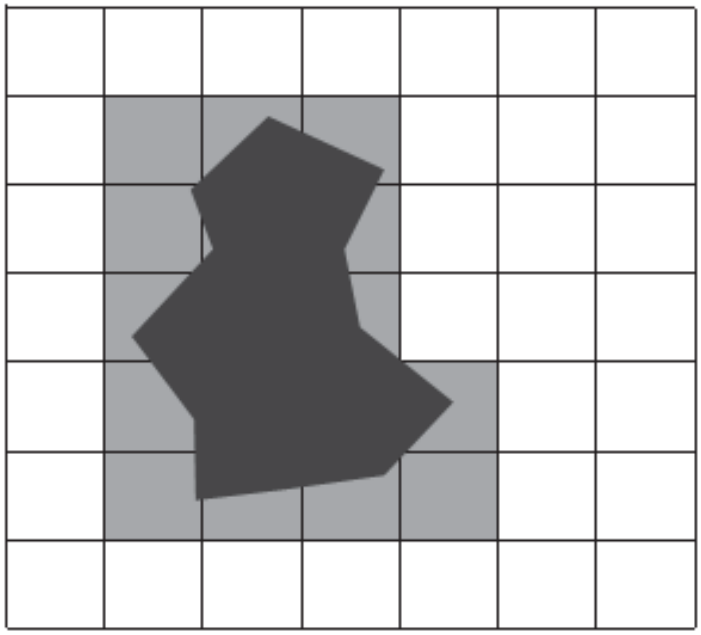


Figure 2.13: The cell partly or totally covered by the obstacle are marked

After removing the obstacle, the blocked cells have replaced, where any path that may go through them are not allowed.

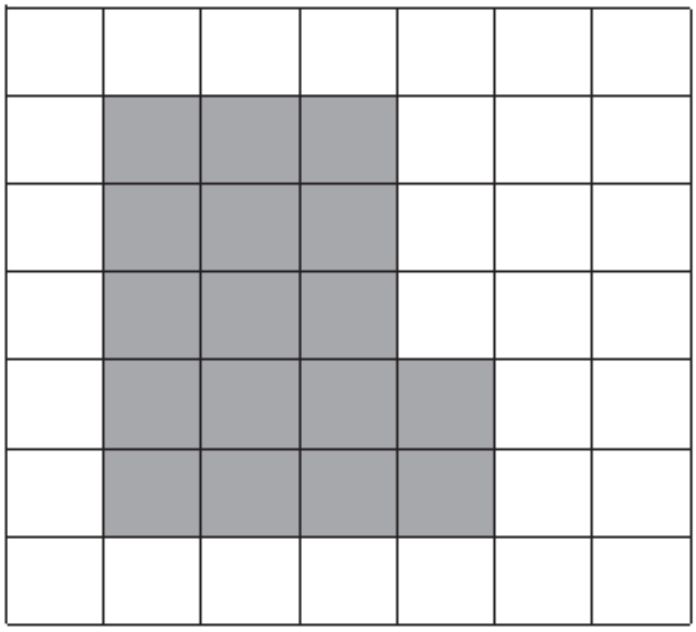


Figure 2.14: The marked cells are blocked after removing the obstacle

2.1.4 Search in Vertex-Based and Cell-Based Worlds

When searching path on the search field, in general, there are two methods to mark the node. Considering the sample illustrated in the figure 2.15, assuming the requirement of the search algorithm is to find out the path from the top-left corner to the bottom-right corner.

Figure 2.15: The original field that has not been divided by the grid

After dividing the search field, the next option is to decide whether the search is based on the cells or based on the vertices. For instance, in the figure 2.16, the coordinate C-3 represents a vertex, but in the figure 2.17, the coordinate C-3 represents a cell.

The cell of the search field is defined as the minimum rectangle area, which is crossed by four edges. The cell based search will find out a path which is a series of connected cell. The vertex of the search field is defined as a intersection point of the search field. The vertex based search will find out a series of connected vertices as the path.

The major difference between them is the cell represents an area, that there may be some information missed or ignored during the search. For instance, consider the cell B2 in the figure 2.17, since it is blocked due to the obstacle, it will not belong to any of the path. However, the vertices and edges of this blocked cell are still accessible if the connected cell is not blocked. That is, the path that passes through these vertices or edges could be exist. The figure 2.18 demonstrates a path from A4 to C1, where B3

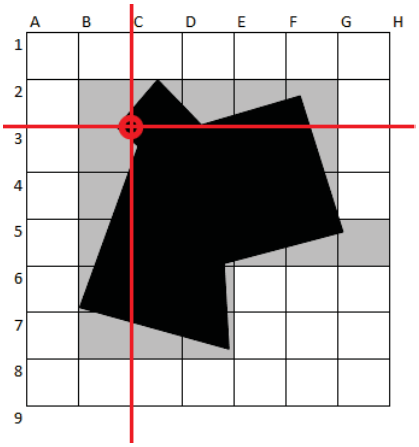


Figure 2.16: Search based on the vertices

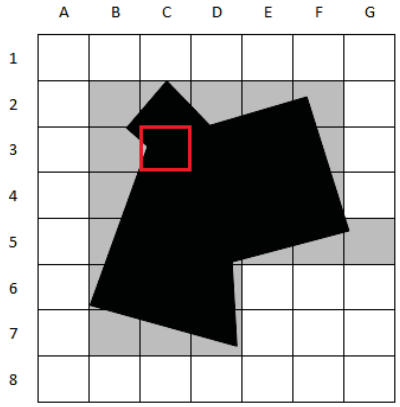


Figure 2.17: Search based on the cells

and B2 are two vertices that belong to the blocked cell.

Since most of the searches on the 2D-field are to find out the shortest path between two point, the vertex based search is more suitable because all the search nodes are considered as a point but not a rectangle area. In the following chapters, all the search will be based on the vertex if there is no specification.

2.1.5 Any-Angle Search and the Line-of-sight

In the A* search, the angle relationships between neighbour nodes on the path are concentrated on vertical, horizontal or diagonal, because the successors are physically connect to the source node. Consider the sample illustrated in the figure 2.19, the shortest path found by A* from A1 to H5 is A1-B2-C2-D3-E3-F4-G4-H5.

It is obvious that there is no obstacle within this area, that is, the actual shortest path is not that found by A*, but from A1-H5 directly, as it is shown in the figure 2.20.

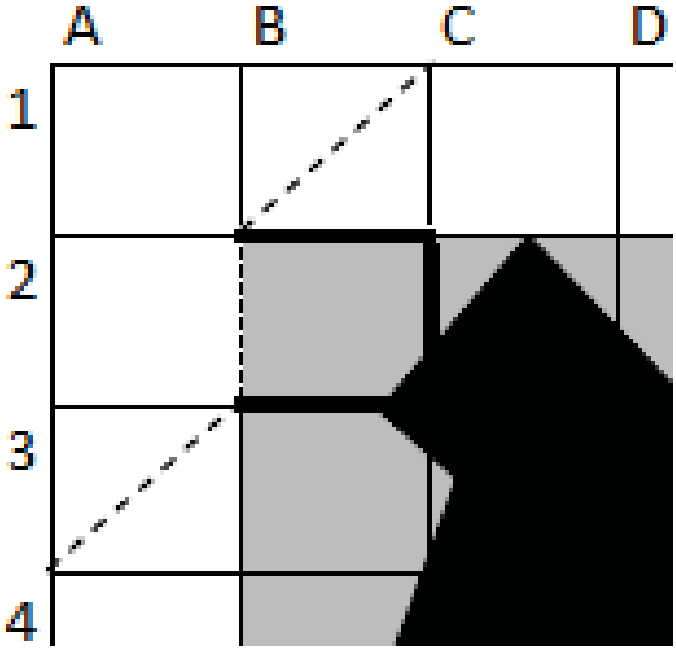


Figure 2.18: A path between A4 and D1 is valid

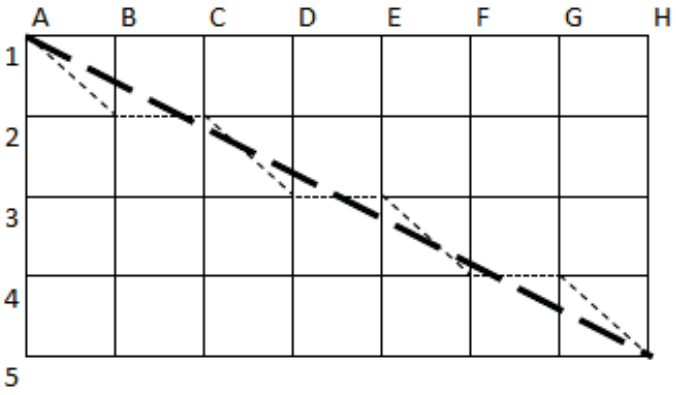


Figure 2.20: The same path of the figure 2.19 could be smoothed

The concept of line-of-sight is introduced by Yap, 2002 [17], with the purpose of removing the unnecessary intermediate node on the path. The algorithm of examining line-of-sight depends on how the search field is divided, or how the obstacle is defined.

There are multiple algorithms that can implement the line-of-sight check in different scenarios. The details of the algorithms will be discussed in the other chapter.

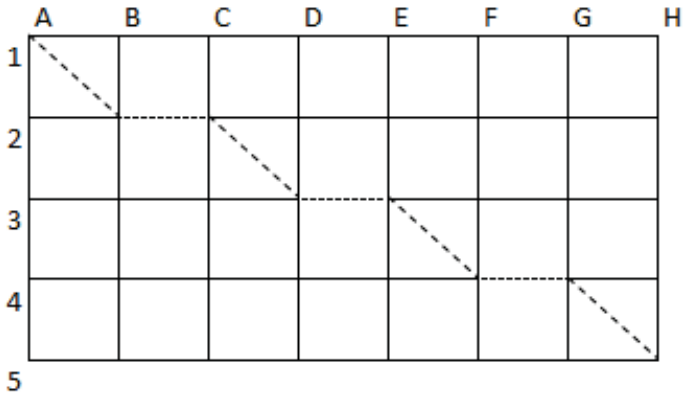


Figure 2.19: The path found by A* is wiggled

2.1.6 Post-smoothing of the A* Search Algorithm

The purpose of Post-smoothing is to remove the unnecessary way point on the path found by the A* search [20]. As soon as a valid path has been found by the A* search, the Post-smoothing is executed. Assuming the way points of the path is connected as a linked-list, the post-smoothing is to remove some of the way points:

Exist $Path := \{S_0, S_1, ...S_n\}$
FOREACH $s \in (Path \setminus \{S_0, S_n\})$
    $s_p := s.Prev$
    $s_n := s.Next$
    IF $LineOfSight(s_p, s_n)$ THEN
        $RemoveFromPath(s)$
    End IF
ENDLOOP

The Post-smoothing reduces the count of the way points on the path. However, it also increases the workload of the search since the line of sight checking is only applied after the path is found.

2.1.7 Theta* Search Algorithm

Theta* search is an alternative version of A* search, which was introduced by Daniel, K, Nash, A, Koenig, S and Felner, A in 2010. The basic idea of Theta* search is to remove a constraint of 8-edges A* search that all the path nodes must be the neighbour of their parent nodes and their child nodes.

In A* search, as soon as a search node S has been popped from the search list and added to the expanded list, all the neighbour path nodes s' of the latest path node

s from the search node S will be examined, then the successors which has not been expanded will be joined into the search node S to generate new search nodes S', and then S' will be added into the self-sorted search list. In this procedure, when a new search nodes is generated, the latest path node will always be the neighbour of previous path node, that is, the direction between two connected node will be concentrated to vertical, horizontal or 45-degrees.

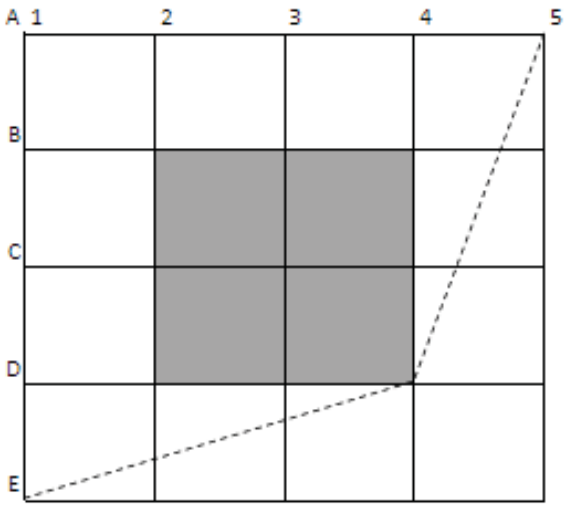


Figure 2.21: A possible path found by Theta* Search

Different from the A* algorithm, Theta* algorithm involves an extra examining during this procedure: the line-of-sight checking[15] between the parent node of current, and the successor.

The figure 2.21 illustrates a possible result path found by the Theta* Search Algorithm from E1 to A5. By applying the feature of line-of-sight checking, the path is not concentrated to certain angles anymore.

Considering the case shown in the figure 2.22: find out the path from E1 to A5. In A* Search with 8-connected node, an possible result path would be E1-D2-D3-D4-C4-B4-A5. Same as the A* Search, the Theta* search will begin from checking the successors of the start node E1. Assuming the euclidean distance ($d(s, s') = EuclideanDistance(s, s')$) is applied in the entire procedure, the fcost of D1 and E2 are both 6 (1+5), while the fcost of D2 is $4\sqrt{2}$ ($\sqrt{2} + 3\sqrt{2}$), so that D2 will be at the top in the search list and expanded in the next iteration.

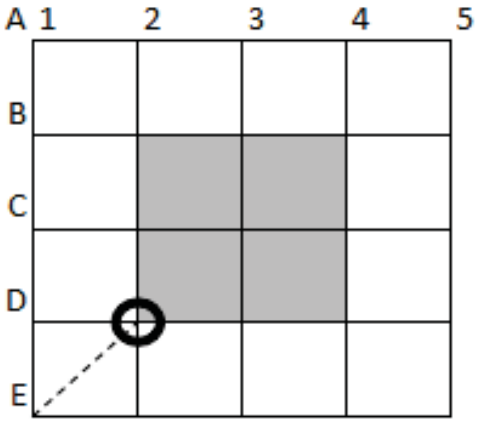


Figure 2.22: Examine the vertex D2

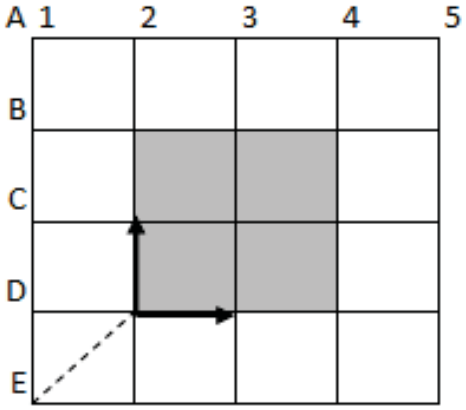


Figure 2.23: Visit the successor of D2: C2 and D3

Then the successors of D2: both C2 and D3 have the smallest heuristic value, so that new search nodes C2-D2-E1 and D3-D2-E1 might be added to the search list with the same fcost.

Before adding the new nodes to the search list, the Theta* search will process the line-of-sight checking from the parent of current node D2 to the successors C2 and D3. As it is shown in the figure 2.23, the line-of-sight checking is applied between (E1, C2) and (E1, D3).

Since both of the results of line-of-sight checking are true, the current node D2 will be removed from the search nodes, and the new search nodes will be update to E1-C2 and E1-D3, also the gcost will be updated to gcost(E1) + d(E1,C2) and gcost(E1) + d(E1, D3) separately. Then, the new search nodes will be inserted into the search list.

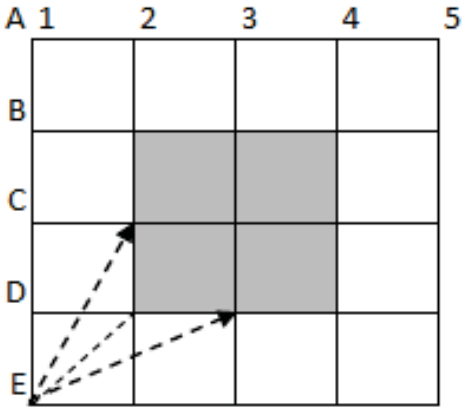


Figure 2.24: Examining the line-of-sight from E1 to the successors

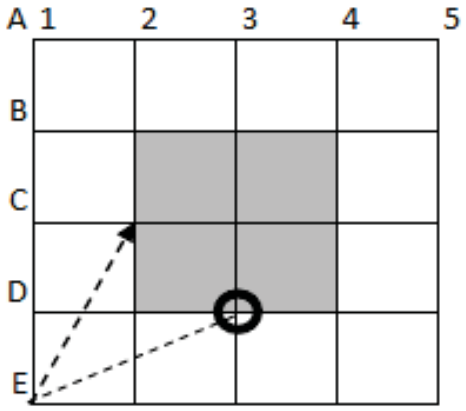


Figure 2.25: Remove D2 on the path since there is a line-of-sight

The next iteration will repeat the same procedure. Since the search node E1-C2 and E1-D3 have the same fcost, one of the will be popped from the search list depend on the implementation. Assuming E1-D3 will be popped first, then the successor of D3 will be checked and the search will advance.

Figure 2.26 illustrates that the search has advanced to where D4 has been expanded, and successors of D4 are going to have the line-of-sight checking. Although there is only one successor C3 is blocked, to simplify the demonstration, only C4, C5 and D5 are considered due to the low heuristics of them.

Checking the successors C4, C5 and D5, as it is illustrated in the figure 2.27: the line-of-sight checking from E1 to them are not all success. There are not sights from E1 to C4 and C5, so that D4 will not be removed from the new search nodes E1-D4-C4 and E1-D4-C5, but will be removed from the node E1-D4-D5. The table 2.1 lists the cost

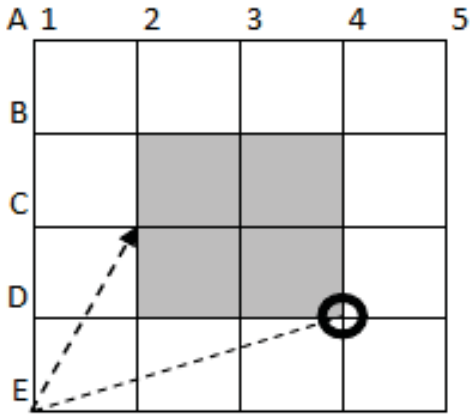


Figure 2.26: Search has be advanced to D4

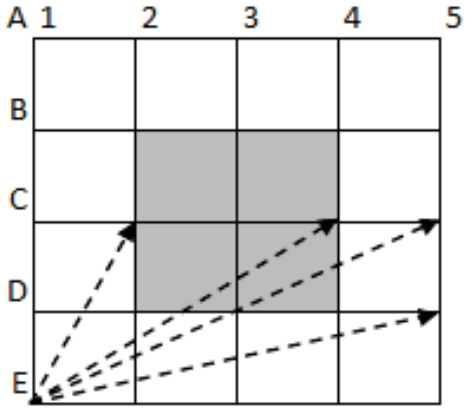


Figure 2.27: Checking the line-of-sight for the successors of D4

of these search nodes.

Table 2.1: gcost and heuristic of new search nodes

	heuristic	gcost
E1-D4-C4	$\sqrt{5}$	$\sqrt{10}+1$
E1-D4-C5	2	$\sqrt{10}+\sqrt{2}$
E1-D5	3	$\sqrt{17}$

When C4 or C5 is expanded, like the sample in the figure 2.28, the line-of-sight checking will be from the parent of C4 or C5, which is D4 but not E1, to the successors.

Comparing with original A* search, Theta* search results a much smoother path, however, due to the extra examining procedure during the search, the run time is not satisfactory under the cases where obstacles covering big area. It only performs quicker than A* search if there are only a few obstacles on the plain [17][7].

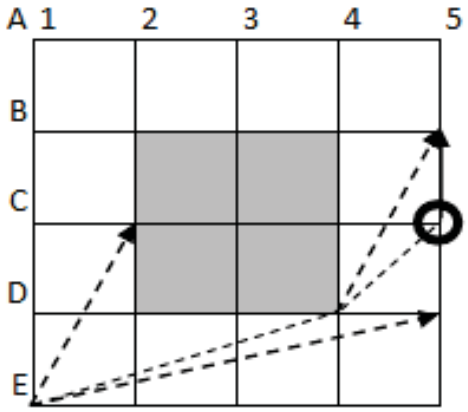


Figure 2.28: The line-of-sight examining for the successors of C5 is from D4 but not E1

2.1.8 Limitations of the Optimal Search Algorithms

The search algorithms have a limitation of cannot cope with dynamic environment. While the agent is moving along the path found by the search algorithm before starting, the environment may not be changed, otherwise the previous path could be overlapped by the obstacle. Van Toll & Geraerts exposed an improvement of the search algorithm for dynamic environment in 2015[4], which is to detect the change of the environment at each iteration and do re-planning by reusing part of the previous path. However, there is no guarantee of the performance if there are multiple changes happening in same iteration.

There is another research by Wang, Zhou, Zheng & Liang in 2014 use Hierarchical A* Search for navigation for complex environment [8], and also a similar research by Yap, Burch, Holte, Schaeffer in 2011 called Block A* Search [17]. Both of them divided the search field in several blocks to reduced the complexity of planning and re-planning by reusing the known information of the field. However, if most of the obstacles are moving, or most of the blocks are changed frequently, re-initializing is neccesary, without any guarantee about the speed as well.

The research by Pochmara, Grygiel, Koppa & Kaminski in 2013 expressed the performance of the A* Search in real-time navigation. With different distance functions, the performance of the A* Search varies, but overall the performance is good. This research also discusses the involving of extra cost in the algorithm other than the distance.

2.2 The Artificial Potential Field Algorithm

The artificial potential field was introduced by Oussama Khatib, 1985. The original idea of artificial potential field is to generate a virtual field in the space, all the target objects which need to be manipulated will be affected by this virtual field. The artificial potential field is a simulating of the natural potential fields like gravity field, which apply a virtual force to the target objects, so that the motion of objects will be affected by the virtual force.

The types of artificial potential field could vary. The most fundamental type is a point that generates a scalar field where the magnitude is inversely proportional to the square distance. Also, the magnitude is not constrained to that, but could be self-defined depend on the circumstances. The other types could be considered as the combination of multiple fundamental potential fields. The magnitude of an artificial potential field at point k could be presented as the equation 2.2:

$$N(k) = \iint P(x, y)dxdy. \tag{2.2}$$

In equation 2.2, P(x,y) is the magnitude which is generated by the point (x,y) to target point k, and P is the pre-defined generator function. Furthermore, for discretization, the artificial potential field could be considered as a sum magnitude of multiple objects that generate separate artificial potential field affecting the point k at same time, so that the magnitude at point s is:

$$m(s) = \sum_{i=1}^{n} P_i(s). \tag{2.3}$$

2.2.1 Artificial Potential Field for Navigation

Considering the equation 2.3: the scalar field also generate the vector field of the gradient, which is how the artificial potential field affects the moving objects. For each point s on the artificial potential field, the gradient at s determines the force affected on the agent x placed at s, both the direction and the magnitude:

$$\overrightarrow{a(x)} = \frac{\overrightarrow{F(x)}}{m_x} = \frac{\bigtriangledown P(ks}{m_x}. \tag{2.4}$$

In the equation 2.4, m_x is the virtual mass of the agent as a constant. According to

the newtons law of motion, there will be :

$$\begin{aligned} v_x(t) &= \int_{t_0}^{t_1} a(x)dt + v_x(0), \\ S_x(t) &= \int_{t0}^{t_1} v(x)dt + S_x(0). \end{aligned} \tag{2.5}$$

According to the equation 2.5, for the case of static field where the obstacle are not movable, the path S could be determined as soon as the start and the goal position are set. That is, the artificial potential field have the ability of path-planning for static field.

However, if the field is dynamic and the motion status of the obstacles are unknown, the path-planning of the artificial potential field does not work. That is, there is an assuming of the artificial potential field that during $t_0 \rightarrow t_1$, the field should be static.

2.2.2 Simplification of the Artificial Potential Field

In most of the cases, the obstacles are not points but have a shape. A set of the points where the distance from the point to the obstacle are the same are supposed to have the same magnitude on the potential field. If the magnitude is proportional to the distance, then the gradient of the potential field generated by this obstacle will have an attractive force [19]. Oppositely, as the magnitude is inversely proportional to the distance, the obstacle will have repulsive force [19].

This simplification is because the major purpose of the potential field is to add an extra virtual force that makes the object to have a tendency to move towards or away from the obstacle. Since for any curve the distance from a point outside to the curve is the straight line orthogonal to the curve passing through the point, the computation of the magnitude of the potential field could be reduced to only two parts: the distance from the target agent to the obstacles, and the direction from the edge of the obstacles to the agent.

In the case of multiple obstacles, this simplification also have the effect that, the magnitude at point k is proportional or inversely proportional to the minimum distance from the agent to all the obstacles. In the other words, only the nearest obstacle of the agent need to be considered.

2.2.3 Limitations of the Artificial Potential Field

Although the artificial potential field could be used for navigation, it has some limitations that cannot deal with the navigation problem in complex environment.

Potential Well

The concept of potential well is the main reason of this limitation: since the motion of agent on potential field is based on the gradient, and gradient of the artificial potential field is a vector field, there could be 0, 1 or multiple points on the artificial potential field where the magnitude is 0.

As it is shown in figure 2.29 on page 31: The gray scale demonstrates the magnitude of the artificial potential field, where the lighter cells have smaller magnitudes. The white areas are the potential wells, where the direction of the gradient around the white cells are pointing to it. When the agent is staying in the potential well without enough speed, it cannot escape from the potential well since the force will always push it back.

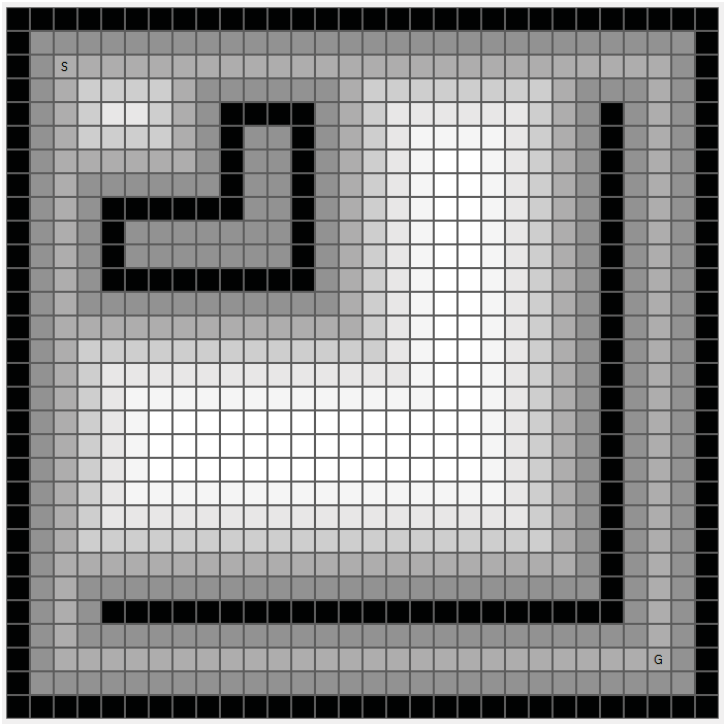


Figure 2.29: Magnitude of the APF

Because of the existence of the potential well, the artificial potential field cannot be used individually in complex scenarios. That is, there should be some other algorithm

combined with the artificial potential field to avoid this trap.

Goal-Non-Reachable-with-Obstacle-Nearby (GNRON) Problem

The Goal-Non-Reachable-with-Obstacle-Nearby problem is another drawback of the artificial potential field [2] [3]: when the target is near an obstacle, the distribution of the gradient around the target has a possibility that it does not point to target but away from it since the magnitude distribution is affected by multiple objects.

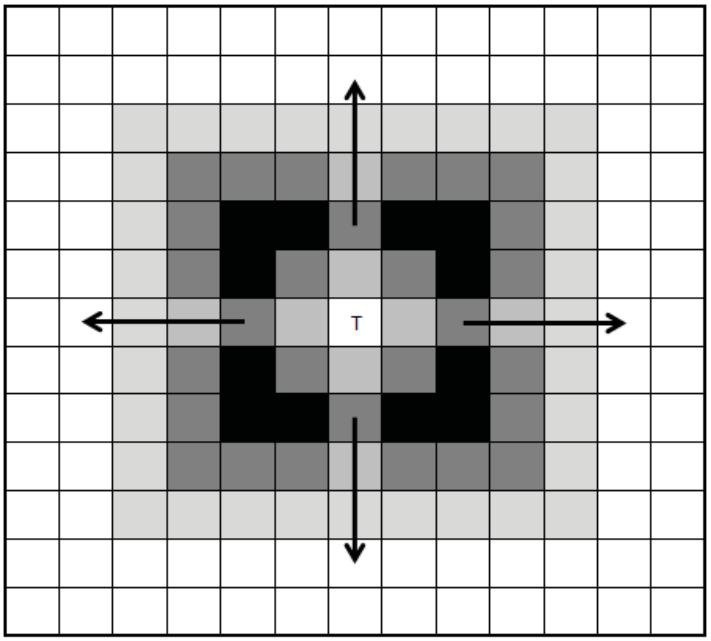


Figure 2.30: Target is not reachable due to the obstacles around

The figure 2.30 illustrates a sample of the GNRON problem: although the target in the centre has a smaller magnitude, the cells around it affected by the obstacle have higher magnitude that causes the direction of the gradient around it are point to the outer sides. As the result, when the agent is moving towards this area, it cannot reach the target.

2.2.4 Related Works based on the Artificial Potential Field

Although the artificial potential field is an effective method of robotics navigation, the limitations of the artificial potential field can pose problems. Yang X, Yang W and Zhang H, introduced a modified solution of the artificial potential field to solve the limitation due to the potential well in 2016 [2]. However, it consumes extra detecting

time to identify the existence of the potential well, as it cannot figure out all the probable wells on the way. Also, Sun & Han explored the usage of the artificial potential field for 3D path planning in 2016 [1] by designing a better model. Another research by Liu & Zhao in 2016 [3] uses virtual way-points that contained the concept of search algorithms.

The research by Ge & Cui in 2000 [51] explored an improvement of resolving the GNRON problem of the artificial potential field. By adding an extra factor of torque, the agent will always take a tangential force near the obstacle, where the centre of the torque is inside the obstacle. This factor allows the agent to move around the obstacle towards the goal. However, the solution didn't exposed the effects of moving obstacles, as the torques of different obstacle may cancel each other if the distance between the obstacles change. Also, for unknown terrain, the direction of the torque could not be decided.

The research by Wang & Chirikjian in 2000 [24] illustrates an alternative methodology of using the artificial potential field: by extending the space and involving the z-axis limited to $[0, 2\pi)$, which represents the orientation of the agent and the obstacles. With this extension, the artificial potential field has the ability to resolve the navigation of an agent as long as it is not in the shape of sphere.

In 2012, Li, Yamashita, Asama, & Tamura demonstrated a solution of using regression search to improve the artificial potential field [5]. In this research, the regression search algorithm is used to keep finding the way-points around the obstacle and uses a methods like line-of-sight to remove the intermediate way-points when the agent is close to the obstacle.

Another research from Zhejiang University and the University of Tokyo in 2015 explored an improved method of the artificial potential field called SIFORS [25], which is partly involved with search algorithms. Similarly, this algorithm optimizes the path after the path planning by the artificial potential field, by removing the way-points close to the obstacles with the concept of line-of-sight and keep checking the direction of the current way-point.

The research by Manalu in 2014 [36] illustrates the method of double targets potential field (DTPF). Different from the traditional potential field that has only one target, the DTPF generates the attractive point by using two targets at same time. However, there is still only one attractive target in the potential field, and the purpose of DTPF is to figure out a better position for the future behaviour of the robot.

2.3 Introduction of the Robot Soccer Game

The robot soccer game uses the robots controlled by the artificial intelligence playing soccer. The general rules of the robot soccer are similar to the normal soccer game. The purpose is to kick the ball on the field into the goal of the opponent, while defensing the attacks from the opponent.

Eslami, Asadi, Soleymani & Azimirad discussed the efficiency of using various algorithms for robot path planning in 2014. [52], and exposed that the path-planning and motion control could be manipulated in the grid-based field.

There are two types of the robot in the game: the attacker and the keeper.

During the game, the attacker should have the abilities to pursue the target and avoid the obstacles which could be the opponent robot or the walls. Also, it should be able to kick the ball towards the gate of the opponent.

Different from the attacker, the keeper should have the ability to protect the goal by various behaviors. Furthermore, when the ball is far away from the goal, it should not keep moving but should park at a certain position waiting for the attacks. It should not be too far away from the goal area.

2.3.1 Dimensions of the Playing Field and the Agents

According to the manual of the robot soccer game [22] introduced by Yujin Robotics, Co., Ltd. Korea, 2003, the playing field is rectangle ground with a size of 220×180, and the length of the goals are 60 in the middle of the left and right side. At each corner of the playing field, there is a bevel which angle is 45° where the length of short edges are 7. The figure 2.31 illustrates the overall size of the playing field.

The robot is a square where the length of edge is 7.

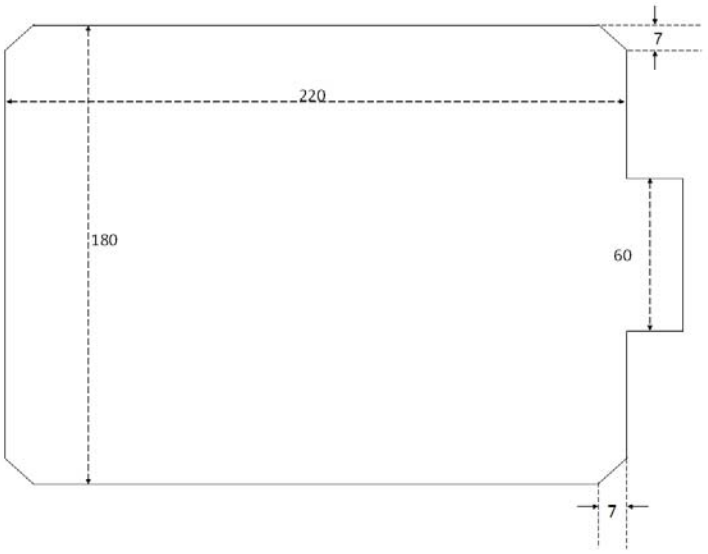


Figure 2.31: Size of the field

Chapter 3

Preliminary Experiments

The preliminary exploration focuses on the the optimal search algorithms to explore the usability and compare their performances. It also explores the limitation of the artificial potential field by designing the potential well.

3.1 The Platform

The platform of the preliminary exploration is developed by Visual C# with framework of .NET Framework 4.5.2. All the experiments of the optimal search algorithms are cell-based. The items which are explored include: the average runtime of the search, the number of the visited nodes, the number of the expanded nodes, the number of the way-points and the length of the resulting path.

3.1.1 Terminologies and Statistical Measurements

Average Runtime

The average time is computed by the time difference between the start and the time when the result is found. In order to represent the actual runtime, it didn't record the time of initialization phase. That is, the preparation of the collections and the insertion of the start node were not counted.

Number of Visited Nodes and Expanded Nodes

As soon as a cell which is the successor of the current examined cell is visited, and it is still not expanded yet, the number of visited nodes will be increased by 1, even though

the same cell has been visited multiple times before the expansion. Also, since the start cell will be expanded immediately when the search begins, it will not be counted as a visited.

The number of expanded nodes is the size of the expand list (close set) when the search ends.

Number of the Way-points and the Length of the Path

The number of the way-points is the size of the result cell list found by the search algorithms, which includes the start and the goal cell.

The length of the path is the accumulated euclidean distance $d(s, s')$ between the closed way-points. Assuming the set of the way-points is $P = \{p_1, p_2, ...p_n\}$, then the length of the path L is:

$$L = \sum_{i=2}^{n} d(p_{i-1}, p_i). \tag{3.1}$$

Ratios

The experiment also cares about the ratios between the A* Search and the Theta* Search in the area of the time consumption, visited count and the expanded count.

3.1.2 Search Field

The search field contains $x \cdot y$ cells. The cell marked as "S" is the start cell of the search, while the one marked as "G" is the goal cell. As the illustration of the figure 3.1, the black cells are "blocked" (cannot be visited), and the white cells are accessible.

Figure 3.2 shows a sample result after searching. The red cells represent the way-points of the search result, from start to goal. The orange cells are the expanded ones, which are inserted into the expand list (close set), and the yellow cells the ones visited during the search.

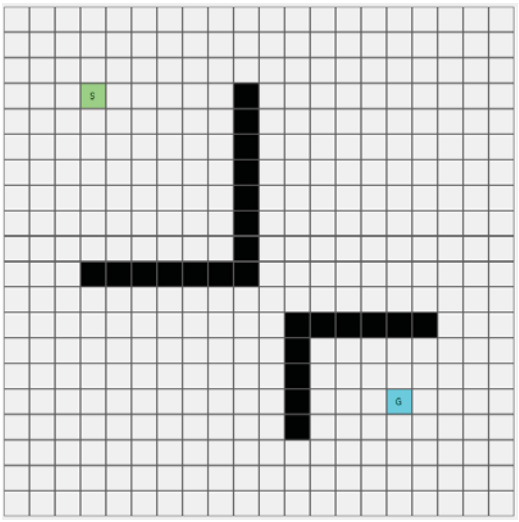


Figure 3.1: Black cells are blocked, light green is the start cell and cyan is the goal cell

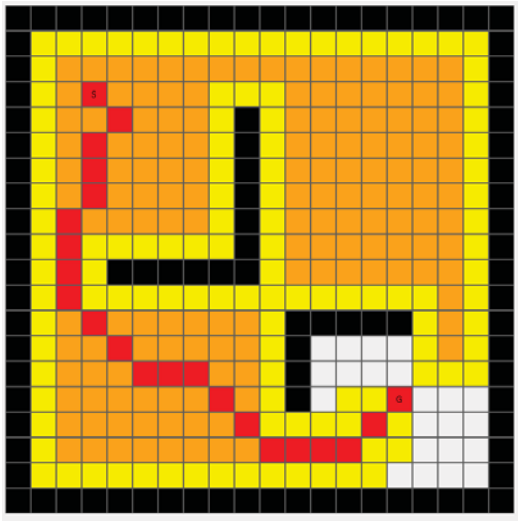


Figure 3.2: A sample path found by a search algorithm

3.2 Case Studies Using the Optimal Search Algorithms

In this section, optimal search algorithms, such as A* and Theta* were tested on path-planning on multiple 8-connected grid worlds. We measured their performance based on runtime, states expanded, path length and cells added into the queue Q. 8 grid worlds were used in the experiments, with varying degree of complexity of obstacle formations. For each test case, 100 trials were performed to guarantee that the results are reliable. However, due to the garbage collection feature of the programming language used (i.e. C#, for this chapter), we have observed that there are some discrepancies in the time measurements. Therefore, we have eliminated those outliers, before calculating the average standard the deviation of the measurements.

3.2.1 One Big Obstacle in the Centre, Size 20x20

The first experiment is the performance on the map that has a big obstacle in the middle, as it is shown in the figure 3.3. The illustration of the performance of the A* Search and the Theta* Search are listed in figures 3.4 and 3.5, separately.

Table 3.1: Statistics in the case of Big Obstacles

Type	Speed(ms)	Visited	Expanded	Path Length
A*	1.55917	566	153	32.246
Theta*	1.38774	508	137	31.539
Ratio	89.00%	89.75%	89.54%	-

It is clear that in this case, the length of the paths found by the A* Search and the Theta* Search are very close. The time consumption of the Theta* Search is also better than that of the A* Search.

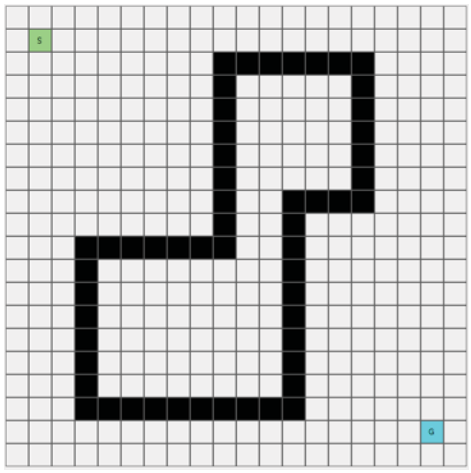


Figure 3.3: Test map of a big obstacle in the centre

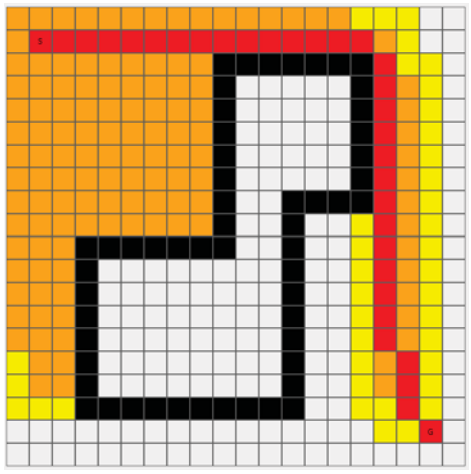


Figure 3.4: The path found by A* Search

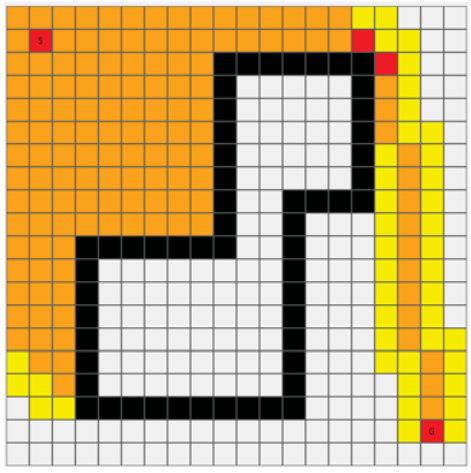


Figure 3.5: The path found by Theta* Search

3.2.2 Four Medium Obstacles, Size 20x20

Similar to the previous one, this case is also on the field with size 20x20. However, the wall is added to the field, and the big obstacle is split into four medium obstacles and replaced, as it is shown in the figure 3.6. The results are shown in figures 3.7 and 3.8:

Table 3.2: Statistics in the case of Four Medium Obstacles

Type	Speed(ms)	Visited	Expanded	Path Length
A*	1.01157	340	97	26.971
Theta*	0.81868	268	68	25.724
Ratio	81.92%	78.82%	70.10%	-

The performance of the Theta* Search in this case is better than the A* Search. After excluding a data from the A* Search where the error is too big (over 10ms), the average runtime of the A* Search is still longer than the Theta* Search. The lower numbers of the visited and expanded nodes reduced the computing workload.

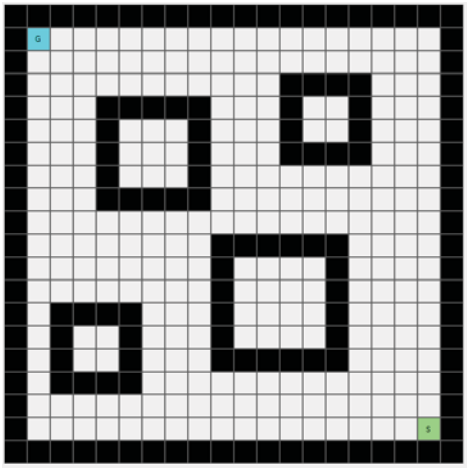


Figure 3.6: The map of 4 medium obstacles

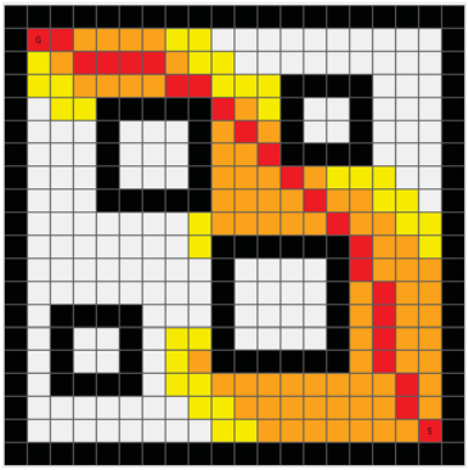


Figure 3.7: The path found by A* Search

Figure 3.8: The path found by Theta* Search

3.2.3 Five Small Obstacles, Size 20x20

The next experiment reduces the size of the obstacle again: five small obstacles are placed on the 20x20 field, and the other conditions are same as the previous one. The figures 3.9, 3.10 and 3.11 show the case and the search results by the A* Search and Theta* Search separately.

Table 3.3: Statistics in the case of Five Small Obstalces

Type	Speed(ms)	Visited	Expanded	Path Length
A*	1.07786	355	88	25.799
Theta*	0.94593	307	68	24.723
Ratio	87.76%	86.48%	77.27%	-

The Theta* Search works faster than A* in this case.

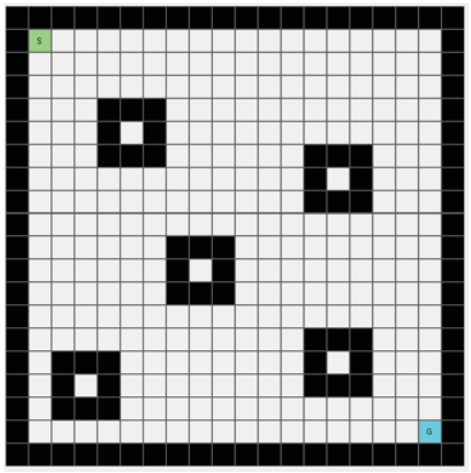


Figure 3.9: The map of 5 small obstacles

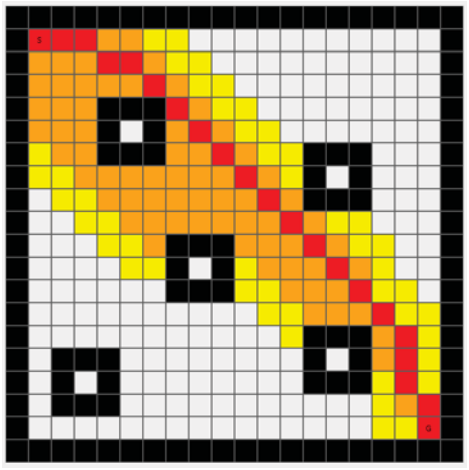


Figure 3.10: The path found by A* Search

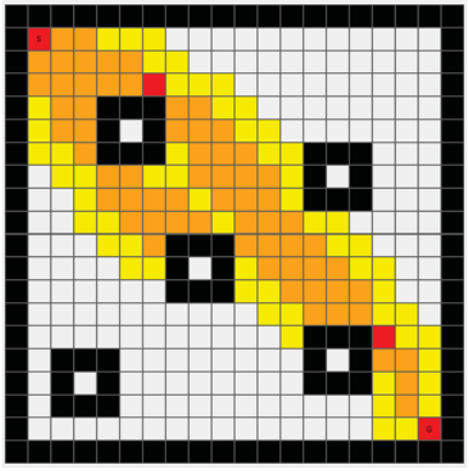


Figure 3.11: The path found by Theta* Search

3.2.4 Random Dots, Size 30x30

The next experiment reduced the size of the obstacles to 1x1 and placed them randomly, as it is shown in the figures 3.12, 3.13 and 3.14:

Table 3.4: Statistics in the case of Random Dots

Type	Speed(ms)	Visited	Expanded	Path Length
A*	2.51323	758	187	37.113
Theta*	1.73988	535	117	36.135
Ratio	69.23%	70.58%	62.57%	-

The advantage of the Theta* Search in this case is clear it compared with the previous experiment: it visited and expanded less nodes than the A* search.

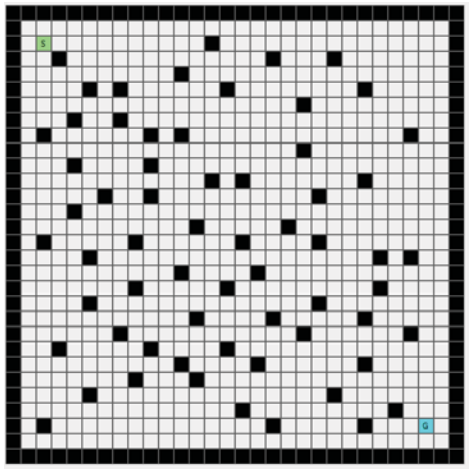


Figure 3.12: The ma of a set of dots

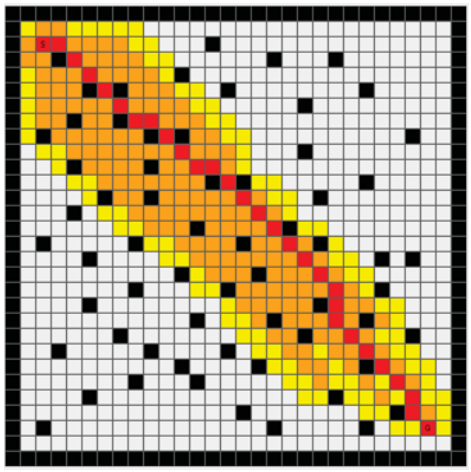


Figure 3.13: The path found by A* Search

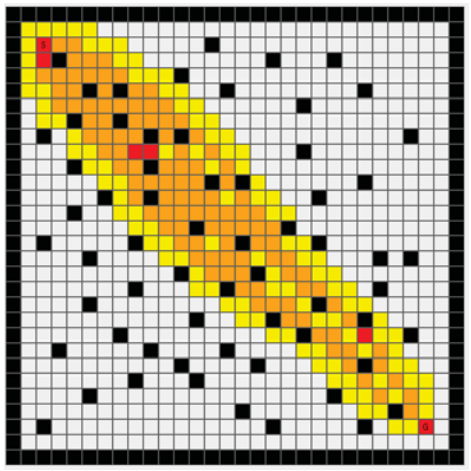


Figure 3.14: The path found by Theta* Search

3.2.5 Four Walls with a Wiggled Lane, Size 20x20

Here we show an example of a complex terrain. The figure 3.15 demonstrates a field with four walls, where the path would be wiggled. The results are shown in the figures 3.16 and 3.17:

Table 3.5: Statistics in the case of Four Walls with a Wiggled Lane

Type	Speed(ms)	Visited	Expanded	Path Length
A*	2.77335	725	234	73.213
Theta*	2.56903	725	234	71.129
Ratio	92.63%	100.00%	100.00%	-

There is an interesting result here: even though the Theta* Search has the same visited and expanded number of nodes as the A* Search, the runtime is still shorter.

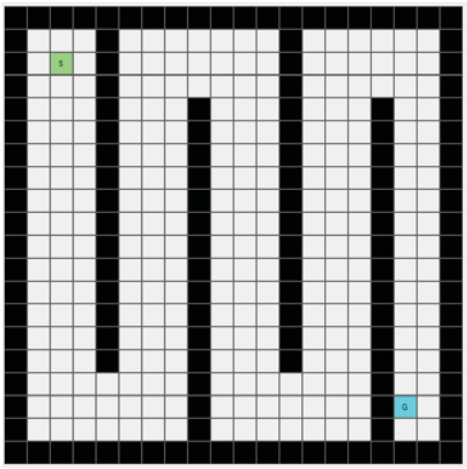


Figure 3.15: The map of 4 walls with a wiggled lane

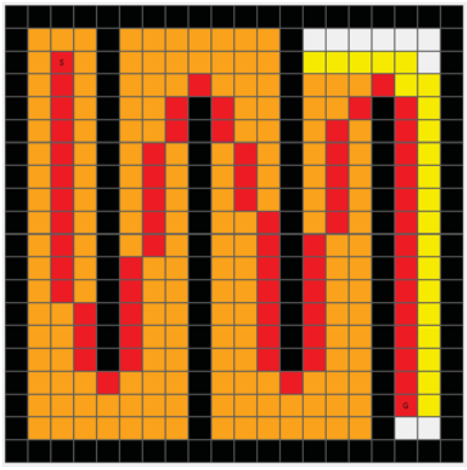


Figure 3.16: The path found by A* Search

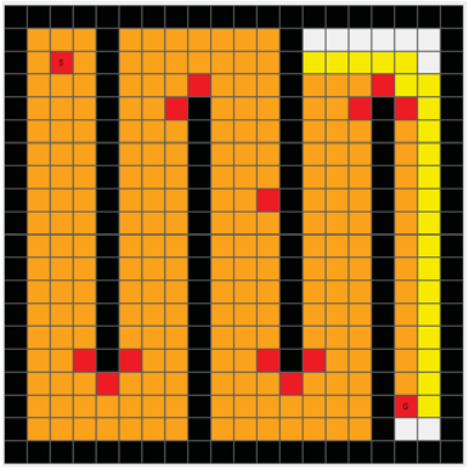


Figure 3.17: The path found by Theta* Search

3.2.6 A Series of Walls with a Lane in the middle, Size 30x30

This experiment is similar to the previous one, which has many walls. However, the possible path result is not wiggled.

Table 3.6: Statistics in the case of Walls with a Lane in the Middle

Type	Speed(ms)	Visited	Expanded	Path Length
A*	4.57833	1187	349	45.899
Theta*	3.87714	1099	320	44.475
Ratio	84.68%	92.59%	91.69%	-

The performance of the Theta* Search here is still better than the A* Search.

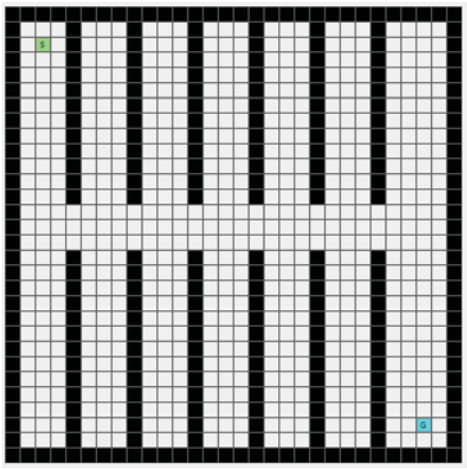


Figure 3.18: The map of multiple walls with a lane in the middle

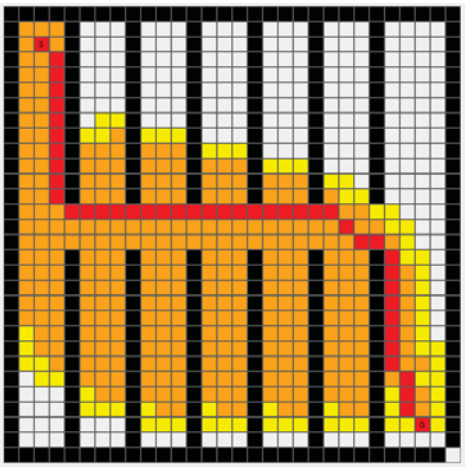


Figure 3.19: The path found by A* Search

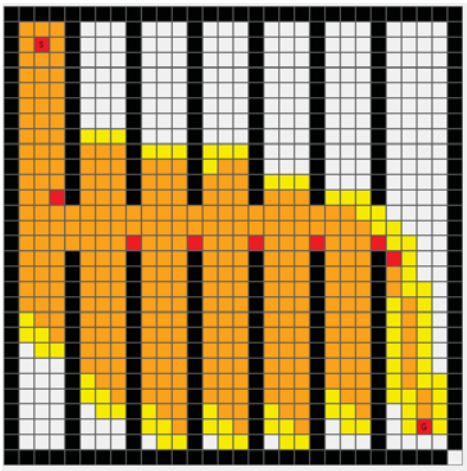


Figure 3.20: The path found by Theta* Search

3.2.7 Maze, Size 30x30

The case of a maze has a much more complex terrain. The figures 3.21, 3.22 and 3.23 illustrate the field and corresponding results of the A* Search and the Theta* Search.

Table 3.7: Statistics in the case of Maze

Type	Speed(ms)	Visited	Expanded	Path Length
A*	17.37805	1722	565	126.326
Theta*	13.44539	1703	558	122.432
Ratio	77.37%	98.90%	98.76%	-

The number of visited nodes and the expanded nodes in this case are very close. However, the Theta* Search still holds the advantage of the runtime.

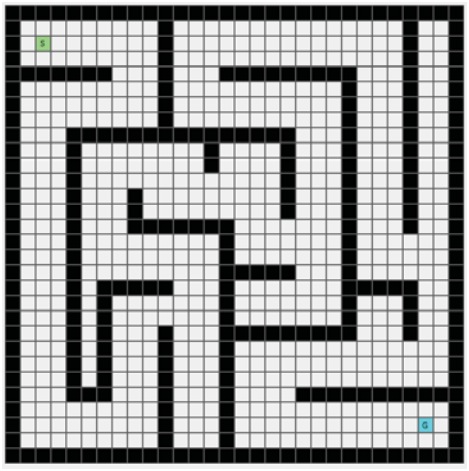


Figure 3.21: The map of maze

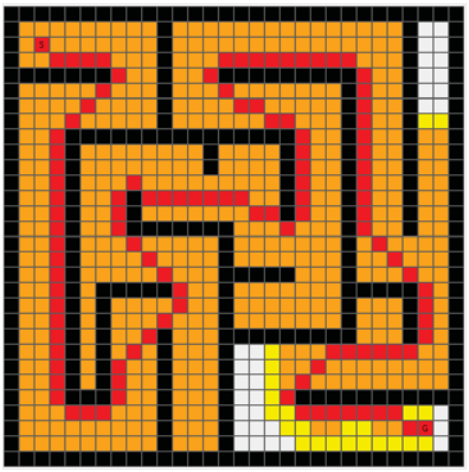


Figure 3.22: The path found by A* Search

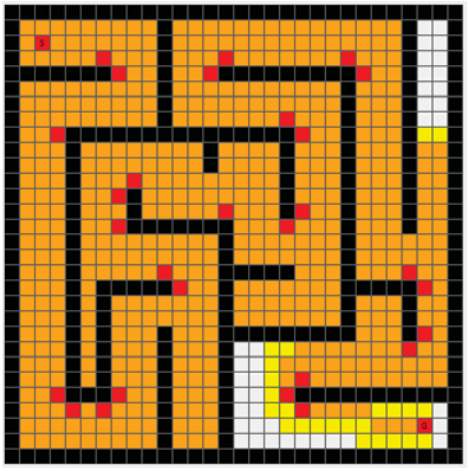


Figure 3.23: The path found by Theta* Search

3.2.8 Potential Well, Size 30x30

The last test is to explore the performance in the case of a potential well. Both algorithms should have the ability of finding a valid path, so that the agent can escape away from the potential well.

Table 3.8: Statistics in the case of Potential Well

Type	Speed(ms)	Visited	Expanded	Path Length
A*	8.45702	2252	604	48.243
Theta*	7.23375	2102	560	47.458
Ratio	85.53%	93.34%	92.72%	-

There is no surprise about the performance. Both algorithms found the shortest way. Although both algorithms expanded most of the nodes, compared with the previous maze, the runtime was shorter, even though the number of visited nodes and the expanded nodes are greater than the previous maze.

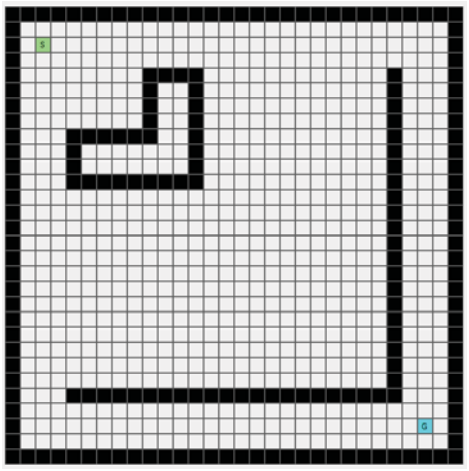


Figure 3.24: The map have a potential well

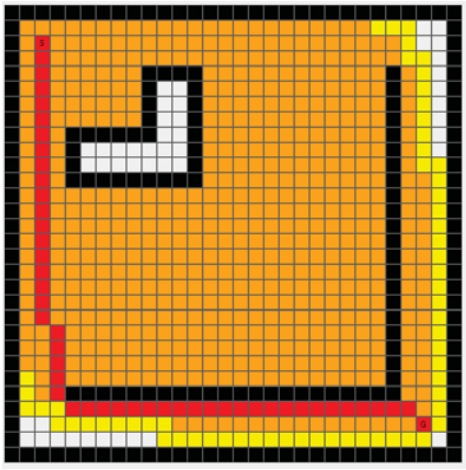


Figure 3.25: The path found by A* Search

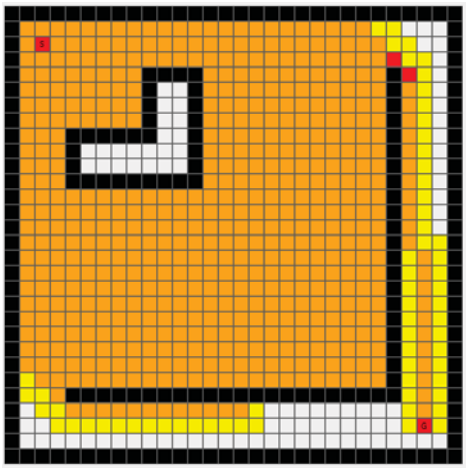


Figure 3.26: The path found by Theta* Search

3.3 Analysis of the Relationship between Speed and the Visited/Expanded States

The following table list the ratios from the preliminary experiments, sorted by the speed ratio.

Table 3.9: Statistics of the Performance Ratios between the Optimal Searches

Speed Ratio	Visited Ratio	Expanded Ratio
69.23	70.58	62.57
77.37	98.90	98.76
81.92	78.80	71.10
84.68	92.59	91.69
85.53	93.34	92.72
87.76	86.48	77.27
89.00	89.75	89.54
92.63	100.00	100.00

Overall, the speed ratio is directly proportional to the number of visited nodes more than the expanded count. The main reason is the computation of the visited nodes: when a node is visited, both the gcost and the hcost should be computed, and for the Theta* search the line-of-sight should be checked as well, that is the major computation. The number of expanded nodes indicates the depth of the search tree, but the process of the expansion will not take extra computation.

The data at the row 2 in the table 3.9 which has the Speed Ratio of 77.37% but the Visited Ratio of 98.90% and the Expanded Ratio of 98.76% is an abnormal case, the data of the case of Maze.

3.4 Conclusion of the Preliminary Exploration

The conclusion is focused on two significant factors: the runtime and the length of the path.

The runtime reflects the speed of the search algorithm, for real-time navigation, a faster algorithm can find out the result path in shorter time, which will be the biggest advantage that will ensure the decision will be made up in time.

The length of the path reflects how accurate the algorithm is. The advantage of the Theta* Search in this case is clear.

In all the cases, the Theta* Search has better runtime than the A* Search. Although the computation of the line-of-sight check consumed extra time, the reduced number of visited nodes and expanded node brings the advantage.

When the results from both the algorithms have the same key way-points, where the set of way-points found by the Theta* Search is a subset of the way-points found by the A* Search, the computation resulting a shorter path for the Theta* Search.

Chapter 4

Algorithms Using Optimal Search with the Artificial Potential Field

4.1 Overview

The artificial potential field can be used for real-time navigation, and is also a method which has a smooth path to avoid the obstacles. However, because of the limitations of the artificial potential well, it sometimes cannot ensure the navigation result. In order to resolve that problem, the search algorithms are involved, so they can ensure that the correct path will always be found if it exists.

The research proposed by Tan, Zhao, Wang, Zhang & Li in 2016 [23] introduced a basic method of using the the A* Search with the artificial potential field for navigation, which is using the search algorithm to find out the path first, and then driving the agent by the artificial potential field. The two methods are applied separately but not actually combined together. In the following sections, the limitations of using them separately will be expressed, which may cause the problem of no-path-found even though there is a path.

4.2 Potential Field with Optimal Search

While searching on square cells, the $c(s, s') = EuclideanDistance(s, s')$ are generally constrained to 1 or $\sqrt{2}$ for 8-connected cells, but the search algorithm accepts various cost between nodes. That is, if a constant value P_x is assigned to a certain node x, the

cost from node s to s' will be changed to $c(s, s') = EuclideanDistance(s, s') + P_{s'}$.

Because the artificial potential field generates a scalar field, each node x on the field affected by that will have a constant P value at a certain time frame, according to equation 2.3 on page 29. That is, the fundamental aspect of the combination of the artificial potential field and the optimal search algorithms is to involve the magnitude of the artificial potential field as an extra dimension upon the search field.

When searching on a 2D-field, both the gcost and the hcost are the the euclidean distance between two points. Then, the magnitude of the artificial potential field at a specified point p represents the the third dimension, that is, the euclidean distance d(s,p) between two point s, p on this field, which magnitudes are $m(s)$ and $m(p)$, is

$$\begin{cases} r(s,p) & = \sqrt{(x_s - x_p)^2 + (y_s - y_p)^2}, \\ d(s,p) & = \sqrt{r^2 + [m(s) - m(p)]^2}. \end{cases} \tag{4.1}$$

That is, searching a possible path on artificial potential field, the costs between the node s and the successor s' of s is:

$$\begin{cases} fcost = gcost + hcost, \\ gcost = \sum_{i=start}^{s} [d(i, i')], \\ hcost = d(s', goal). \end{cases} \tag{4.2}$$

In the equation 4.2, one of the most important factors is the range of the potential field magnitude at node x: if it is too big, the gcost at all the nodes will have a tendency to the same, and the factor of hcost would be ignored, so that the search will be degenerated to best-first-search. On the other hand, if it is too small, the search will be turned back to a normal optimized search, and only the shortest path could be found, but not the safest path to avoid the obstacles.

Figure 4.1 demonstrates the difference between Theta* search with and without artificial potential field. When robot is navigating from A0 to I8, if it follows the dashed line, it may touch the corner of the obstacle at H2. On the other hand, the path found with artificial potential field (straight line) has a tolerant so that the robot will not hit the corner when turning.

4.2.1 Safety Factor

There is an obvious drawback in the equations 4.1 and 4.2 as well: the gradient. According to the equation 2.4 on page 29, the artificial potential field has a direction that

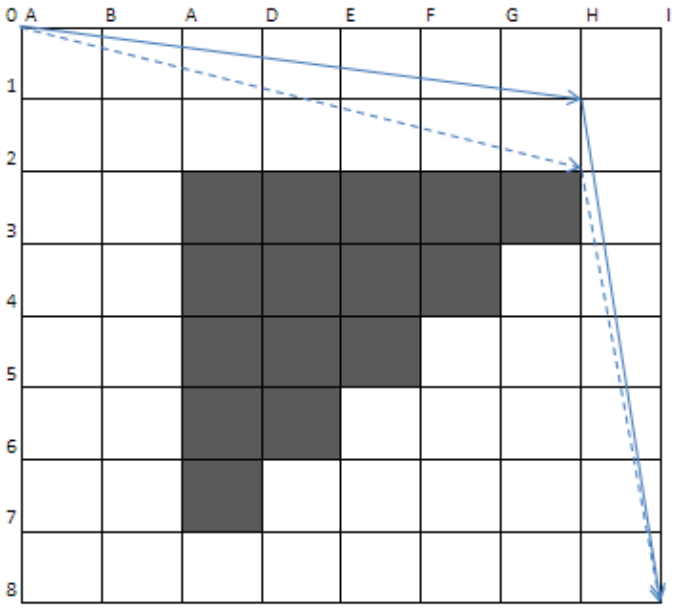


Figure 4.1: The shortest path vs. The safest path

indicates where the magnitude reduced fastest at the point s. However, the equation 4.1 ignore the direction due to the square root. That is, the successors where the magnitude are closer to the source may have a higher priority.

Assuming there are three nodes s, p, q & q' where p is the successor of s, and q, q' are the successors of p, and having $m(s) < m(p)$, $m(q) < m(q') < m(p)$. Also, the distance from p to q and q' are identical ($d(p, q) = d(p, q')$).

If the agent is moving from s to p, with the magnitude of the artificial potential field, it will perform like a climbing. On the other hand, if the agent moves from p to q or q', the performance is like a downhill. Since $m(q) < m(q')$, the direction should point from p to q, but not q'.

However, since $r(p, q) = r(p, q')$ and $|m(q') - m(p)| < |m(q) - m(p)|$, then there is $c(p, q') < c(p, q)$, which causes $g(q') < g(q)$ that is not the expected result in the search. As a result, it cannot guarantee the path it optimal.

In order to resolve this issue, the safety factor (SF) is introduced to the algorithm. Let $\delta(s, p) = m(p) - m(s)$ Consider the equation 4.1:

$$c(s, p) = r(s, p) + \delta(s, p). \tag{4.3}$$

Other than the 3D euclidean distance, the magnitude is used as a safety factor (SF) calculated from the artificial potential field. For any vertex s_n belonging to a partial

path with n vertices, where the start vertex is s_0, the new g-cost of s_n can be expressed as:

$$\begin{aligned} g(s_n) &= \sum_{i=0}^{n-1}[r(s_i, s_{i+1}) + \delta(s_i, s_{i+1})] \\ &= \quad r(s_0, s_1) + m(s_1) - m(s_0) \\ &\quad + r(s_1, s_2) + m(s_2) - m(s_1) \\ &\quad + r(s_2, s_3) + m(s_3) - m(s_2) \\ &\quad \quad ... \\ &\quad + r(s_{n-1}, s_n) + m(s_n) - m(s_{n-1}) \\ &= \delta(s_0, s_n) + \sum_{i=0}^{n-1}[r(s_i, s_{i+1})]. \end{aligned} \tag{4.4}$$

From equation 4.4, it can be observed that the new g-cost contains two components: the summation part corresponds to the original A* g-cost, while the δ part can be considered as a heurstic that depends on the start and goal vertices. Therefore, the equation obeys the consistency requirements of optimal search algorithms [9].

In equation 4.4, the safety factor δ is the difference between the protential field magnitude at the start vertex and the last vertex, but will not be affect by any intermediate node. That is, it only indicates the direction where the magnitude has the largest decrease from start node to the current node. Furthermore, when $s- > start$, there is

$$\lim_{s \to start} \vec{\delta}(start, s) = \bigtriangledown P(start)$$

.

4.2.2 The Basic Artificial Potential Field Generator

According to equation 2.3 on page 29, the simplification of implementing the scalar potential field is to computing the magnitude based on the distance between current point to the nearest obstacles. That is, any decreasing function which domain and range are both $(0, \infty)$ could be applied.

Assuming for any point s on the field, the minimum distance from s to the nearest obstacle is $dmin(s)$, and the magnitude of the artificial potential field at the s is m(s).

The figure 4.2 illustrates the distances from s to the obstacles, where

$$dmin(s) = min[d(s, o_1), d(s, o_2), ...d(s, o_n)]$$

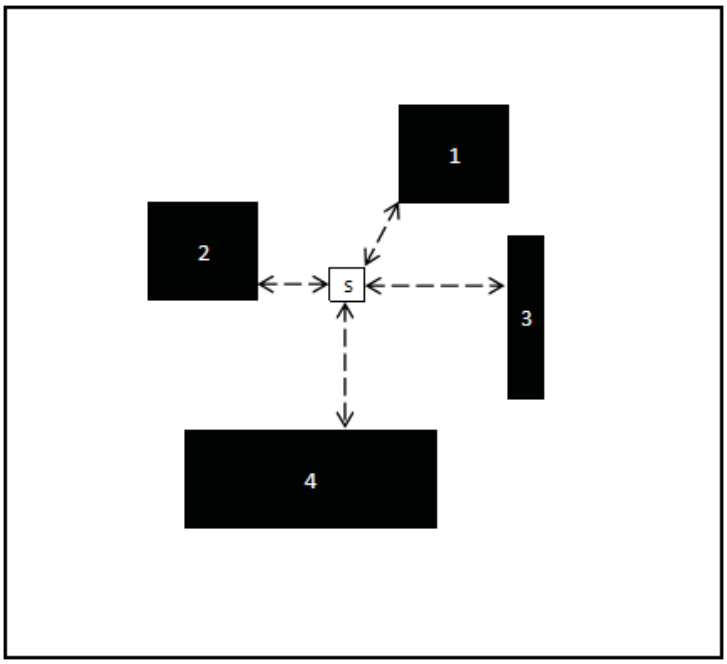


Figure 4.2: The distance from s to the obstacles

The most simple one is margin function:

$$m(s) = \begin{cases} 0 & if \quad dmin(s) > margin, \\ \infty & otherwise. \end{cases} \tag{4.5}$$

Figure 4.4 is a sample of the use of the margin function. In this case, all the gray nodes are not accessible, that is, the margin areas are blocked so that no path may go through these nodes. The margin function ensures a safe path could be found because it blocks extra nodes on the field, but the drawback is also obvious: it sometime may not find the correct way even if it existed due to the margin areas that overlap the path.

The diagram of the margin function is expressed in the figure 4.3, where the $m(s) = 0$ if $dmin(s) > margin$.

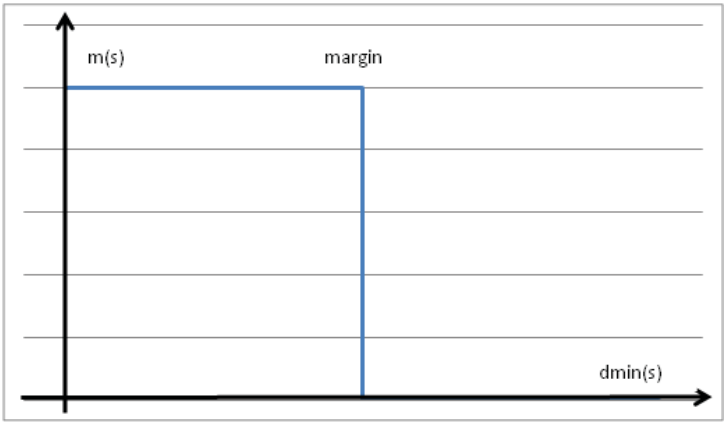


Figure 4.3: The diagram of the margin function

Figure 4.4 illustrates that the margin area (light gray) generated by the artificial potential field depends on the obstacle (dark gray). Figure 4.5 demonstrates a sample where even though a valid path exists (straight arrow), the usage of margin function in the artifical potential field may cause a result that blocks all the nodes on the path so that the search algorithm cannot find the path.

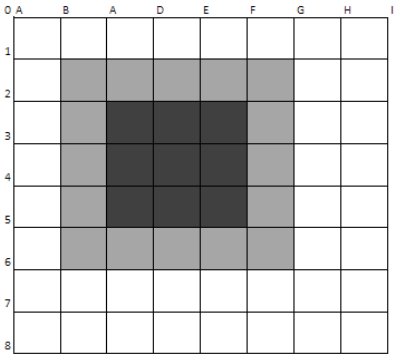


Figure 4.4: Search with Margin function

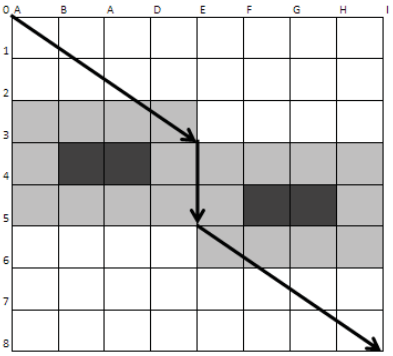


Figure 4.5: The drawback of margin function

4.2.3 Linear Functions

Other than margin function, linear and hyperbola are both decreasing functions that may satisfy the requirement of potential field as well. The advantage of these functions is that they may not always block the cells around the obstacle, so that the nodes next to the obstacle may have a proper $m(s)$. The search algorithm will not ignore the node totally, but only reduce the priorities. The figure 4.6 is a sample of a possible path found with the artificial potential field, where the gray scale illustrates the magnitude of the cells (darker cells have higher values).

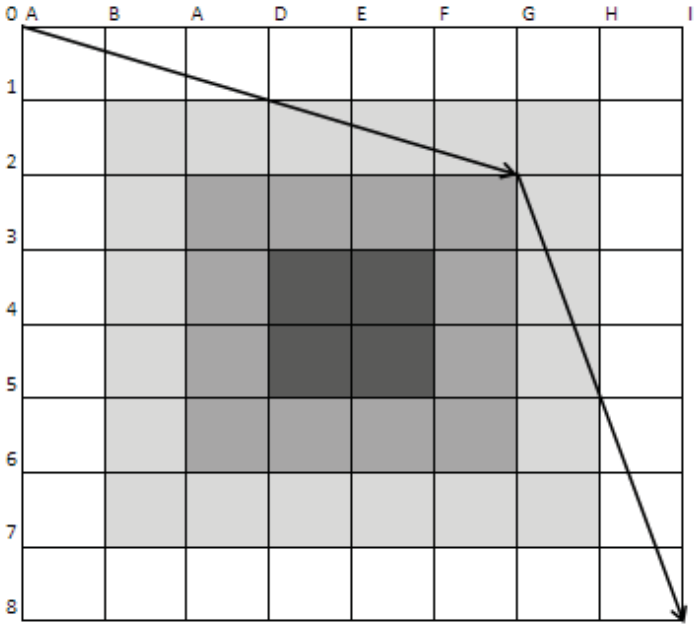


Figure 4.6: The gray scale illustrates the magnitude of the artificial potential field

For linear functions, if the domain is not $(0, \infty)$ but $(0, k]$, then the range could be $(0, t]$ where both $k, t \in \mathbb{R}^+$:

$$m(s) = max[-k \cdot dmin(s) + t, 0]. \tag{4.6}$$

The diagram of the linear function is illustrated in the figure, where $k = -1$, $t = 10$.

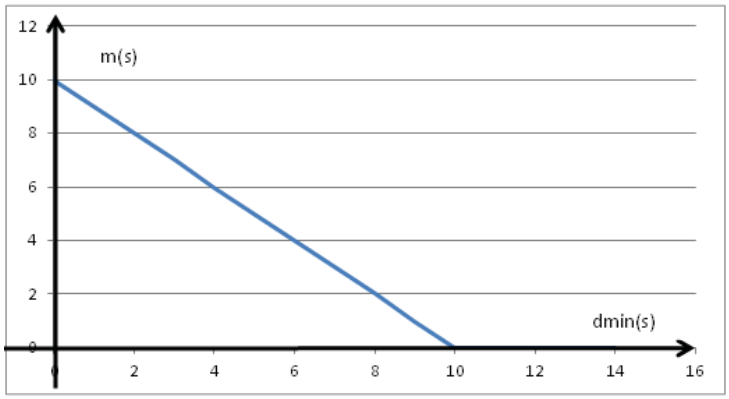


Figure 4.7: The diagram of the linear function

The appropriate range of t in equation 4.6 is around half of the side length of the searching area. For instance, a suitable value of t in figure 4.6 should be 16 (8 + 8).

Also, despite the fact that there's no restrict requirement of the value of k, a value of around 1 would be a reasonable choice.

The figure 4.8 is the manhattan distance from the bottom-right corner to the top-left corner, which is also the gcosts of the cells. Then the figure 4.9 illustrates when a 3x3 obstacle is place on the field, with an artificial potential field of $m(s) = 7 - dmin(s)$, the change of the gcosts. Assuming the gcost of each node is equal to the gcost of the cell at its bottom-right side, consider the search process at the node G6: current search node is I8-H7-G6, with the artificial potential field, the next search nodes with minimum fcost will be I8-H7-G6-F6 and I8-H7-G6-G5, but not I8-H7-G6-F5. This is because despite F6, F5 and G5 have same heuristic, the gcost is not the same, so that search algorithm will select the node which are not too close to the obstacle. So the final result might be I8-G1-A0, but this will depend on how the algorithm is implemented.

0 A	B	A	D	E	F	G	H	I
1	7	7	7	7	7	7	7	7
2	7	6	6	6	6	6	6	6
3	7	6	5	5	5	5	5	5
4	7	6	5	4	4	4	4	4
5	7	6	5	4	3	3	3	3
6	7	6	5	4	3	2	2	2
7	7	6	5	4	3	2	1	1
8	7	6	5	4	3	2	1	0

Figure 4.8: Manhattan Distance

0 A	B	A	D	E	F	G	H	I
1	12	12	12	12	12	12	12	11
2	12	12	12	12	12	12	11	10
3	12	12				11	10	9
4	12	12				10	9	8
5	12	12				9	8	7
6	12	12	11	10	9	8	7	6
7	12	11	10	9	8	7	6	5
8	11	10	9	8	7	6	5	4

Figure 4.9: Manhattan Distance plus artificial potential field magnitude

A major disadvantage of using linear function is if the number of obstacles is small, the path selected by linear function are too conservative and it is not short enough. Since the major purpose of using search algorithm is to find the shortest path, and the use of artificial potential field is to find the safest path, a better balance is always needed.

Another problem of using linear functions is that sometimes the artificial potential field may not be efficient. Considering the figure 4.9, the nodes around top-left side havethea same gcost of 12, that is, the only difference when the search algorithm is examining the nodes around this part is its heuristic. The gcost used in grid based search algorithm is linear as well, and also proportional to the distance from the start node to current node if there is no obstacle. The gcost could be written as $gcost = kr_2$, so if the magnitude is added, the result is $gcost = kr_2 - k'dmin(s) + t$, and it has a possibility that $kr_s - k'dmin(s) = CONSTANT$, usually along a straight line. Furthermore, with multiple obstacles, there will be more nodes that have same gcost, and only hcost is

effective. The search algorithm would only work depending on the heurstic, finding the shortest path again. The main purpose of using it to find a safe path would be nullified.

4.2.4 Hyperbola function

Comparing with linear function, the hyperbola function has an big advantage that, with the increasing of distance from the nearest object, the magnitude of the potential field will be decreased much quicker:

$$m(s) = k \cdot dmin(s)^{-t} \qquad where\ k, t \in \mathbb{R}^{+}. \tag{4.7}$$

The diagram of the hyperbola function is illustrated in the figure, where $k = 10$, $t = -1$.

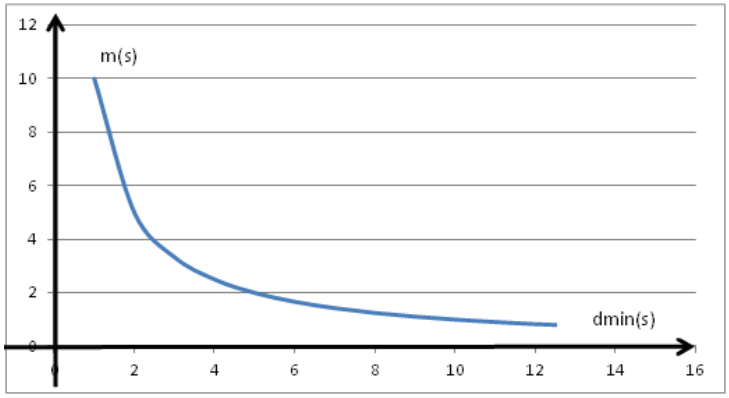


Figure 4.10: The diagram of the hyperbola function

The hyperbola function has the benefit that the gcost of different nodes have a low chance of being the samet. However, opposite to the linear function, the hyperbola function causes the nodes close to the obstacle to also have a low magnitude of the artificial potential field, so that it cannot make sure that the path is safe enough.

4.2.5 Sigmoid Function

The sigmoid function combines the advantages from both the linear and the hyperbola function: There is garantee that the gcost of the nodes which close to the obstacles will be large enough, and also the avoidance that the gcost around a obstacle might be same. A typical sigmoid function is the logistic function:

$$m(s) = \frac{k}{1 + e^{dmin(s)-t}} \qquad where\ k, t \in \mathbb{R}^+. \tag{4.8}$$

The diagram of the sigmoid function is illustrated in the figure, where $k = 10$, $t = 5$.

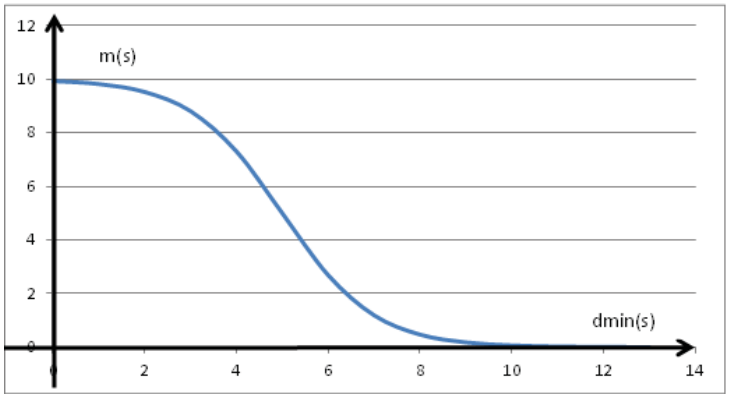


Figure 4.11: The diagram of the sigmoid function

Similar to the linear function, the range of k should be proportional to the size of the search area, and the value of t determines the where is the largest slope of the sigmoid function. For instance, if the size of the searching area is 60x60, and the expected radius of the robot is 5, then k = 3600 and t = 5 could be an appropriate range.

There are also some other sigmoid functions could be applied, for example, some algebraic functions and the arccot function:

$$\begin{cases} m(s) = \frac{-k(dmin(s)-t)}{\sqrt{(x-t)^2+1}} + k, \\ m(s) = k\, arccot(dmin(s) - t). \end{cases} \tag{4.9}$$

All of these sigmoid functions have a similar effect within the search algorithm: while the distance between the node and the obstacle reduced, the gcost of the node will be increased dramatically. As a result, the node close to the obstacle have a high gcost that will not be cancelled by the original distance cost. On the other hand, when the distance increases, the gcost will be turned back to the original distance cost so that the shorter path can have a higher priority.

Although the sigmoid function avoids the disadvantages from both linear function and hyperbola function, it still have the drawback: the node very close to the obstacle still have chance to be selected as the path but not ignored. Since the robot is not a point but has a certain size, the gap problem may happen as it is illustrated in the figure 4.12, where light gray is the robot with size 2x2, and dark gray represents the obstacles.

The gap between two obstacle has a valid path that the search algorithm found from the robot to the goal node, but in fact the path is too narrow and the robot cannot pass.

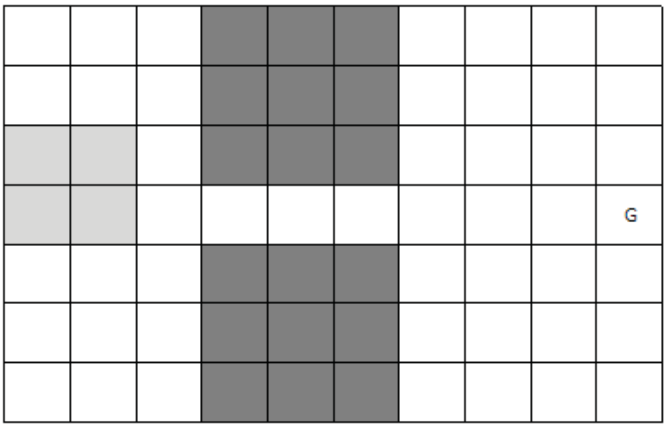


Figure 4.12: The gap where the robot cannot pass

Furthermore, all the sigmoid function have same limits that $\lim_{dmin(s)\to\infty} f(d(min)) = 0$ and $\lim_{dmin(s)\to\infty} f'(d(min)) = 0$, so that value of P of the nodes which are far away from the obstacle are meaningless more or less. That is, there is no necessary to computing the actual magnitude of the potential field at these nodes.

4.2.6 Double Thresholds

Inheriting the concept of margin function, two threshold values could be involved to resolve the flaws of the sigmoid function. Equation 4.10 demonstrates that if the distance is smaller than the first threshold, the node will be blocked like an obstacle, so there will be no way-point inside this area. Contrarily, if the distance is greater than the second threshold, there is no need to calculate the P value but set it to 0 directly.

$$m(s) = \left\{ \begin{array}{ll} \infty & dmin(s) < t_1, \\ \frac{k}{1+e^{dmin(s)-t_1}} & dmin(s) \in [t_1, t_2], \\ 0 & dmin(s) > t_2. \end{array} \right\} where\ k, t1, t2 \in \mathbb{R}^+,\ t1 < t2. \quad (4.10)$$

The figure 4.13 is the logistic sigmoid with two thresholds, both blocking the node that too close to the obstacle, and ignoring the magnitude of the artificial potential field at the nodes far away from the obstacles.

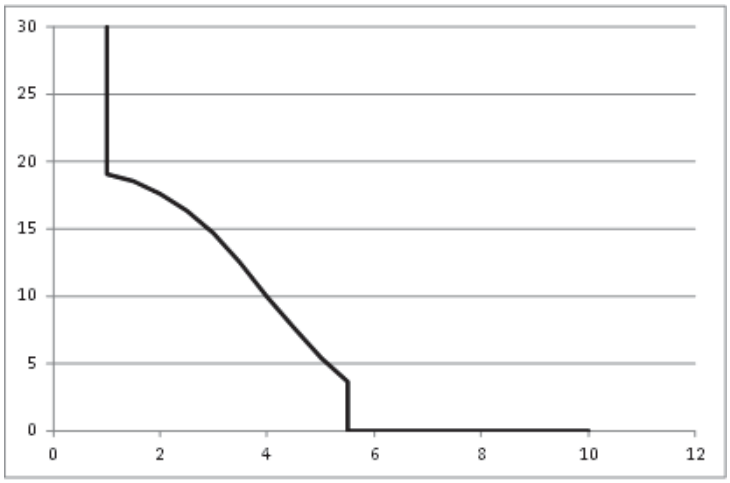


Figure 4.13: Logistic sigmoid with two thresholds

4.2.7 Comparison with Related Study

The study by Ge and Cui in 2000 plotted the local minima problem and the GNRON problem by assuming the agent, obstacle and the goal are placed on the same line [51], as shown in the figure 4.14:

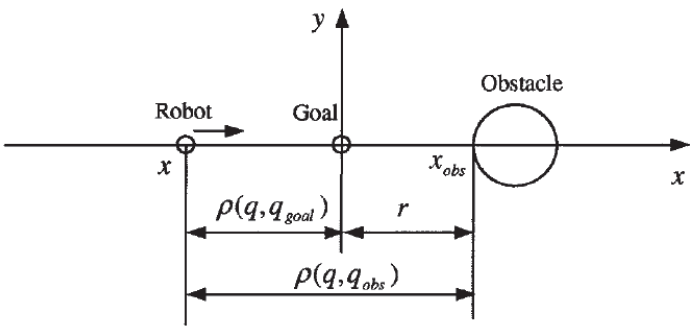


Figure 4.14: According to the study in [51], the position relationship between the robot, goal and obstacle in 1-D scenario.

With this method, a brief magnitude distribution of the artificial potential field could be plotted to indicate the local minima and the GNRON problem in the most simple case. For instance, the figure 4.15 illustrated the magnitude distribution of the figure 4.14 in the study by Ge and Cui [51]:

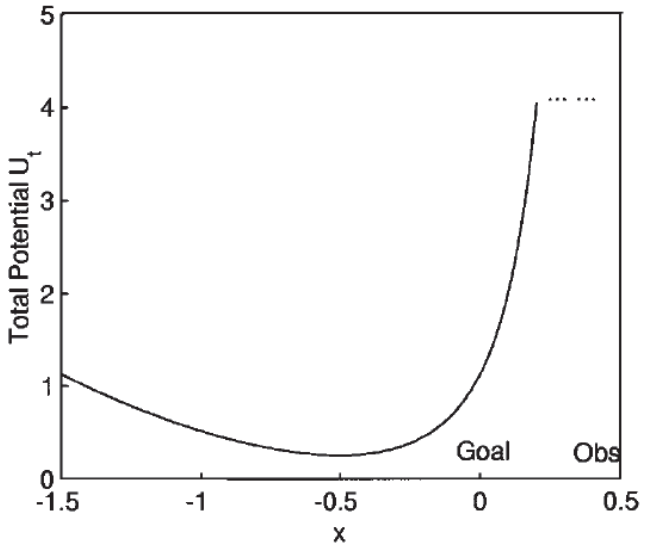


Figure 4.15: According to the study in [51], the magnitude distribution of the APF in 1-D scenario.

With various generator functions, the magnitude distribution will be different as well. For the case in figure 4.14, the resulting magnitude by different generator functions are listed in figures 4.16, 4.17 and 4.18:

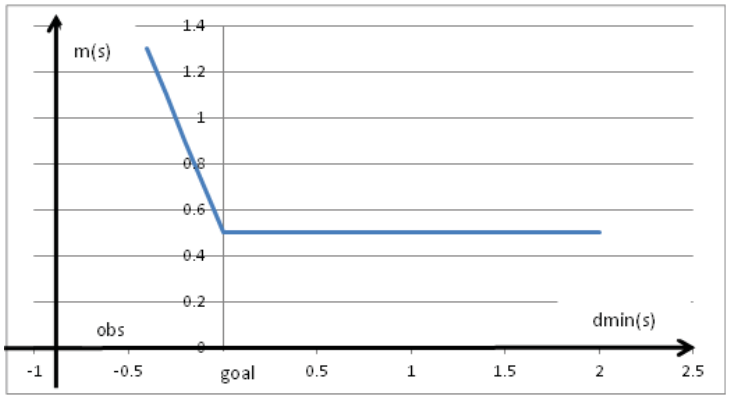


Figure 4.16: The magnitude distribution by linear function

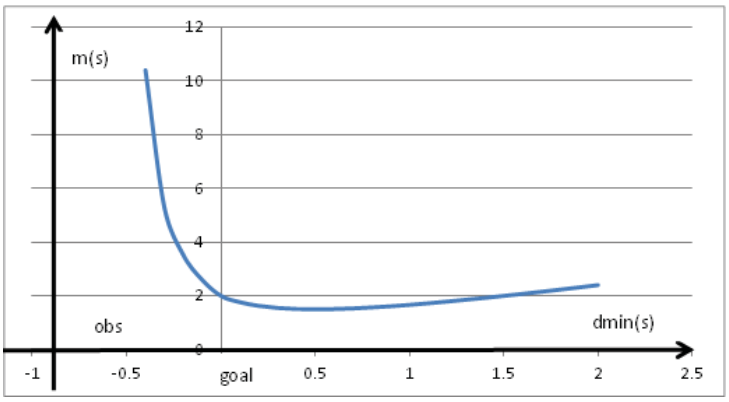


Figure 4.17: The magnitude distribution by hyperbola function

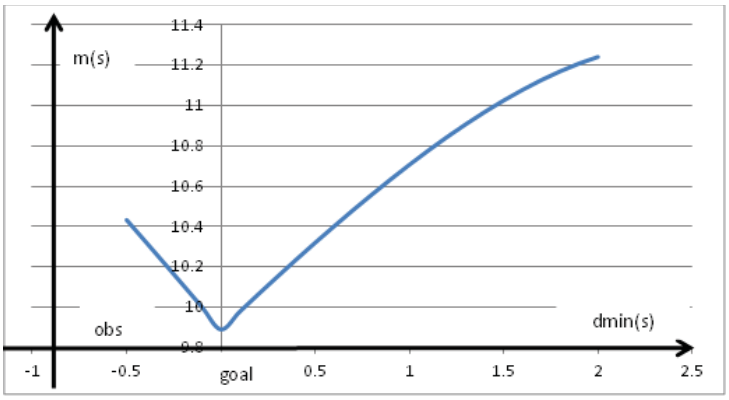


Figure 4.18: The magnitude distribution by Sigmoid function

4.3 Optimal Search Algorithms with Alternative Heuristics

4.3.1 Sharp Bend Problem

Another problem of the search algorithms is not intuitive: on a complex terrain which has a lot of scattered obstacles, the fcost of the nodes could be messed even though a heuristic function exits. That results from the fact that the path may have some "shape bend", where the robot has to make a big turn driving along the path. Figure 4.19 illustrates a sample where the robot will have a U-turn when moving towards the goal, which may cause unexpected accidents in real practice.

The research exposed by Fernandes, Costa, Lima & Veiga in 2015 [16] uses the method of Orientation Enhanced A* Search to resolve the sharp-bend problem. However, the

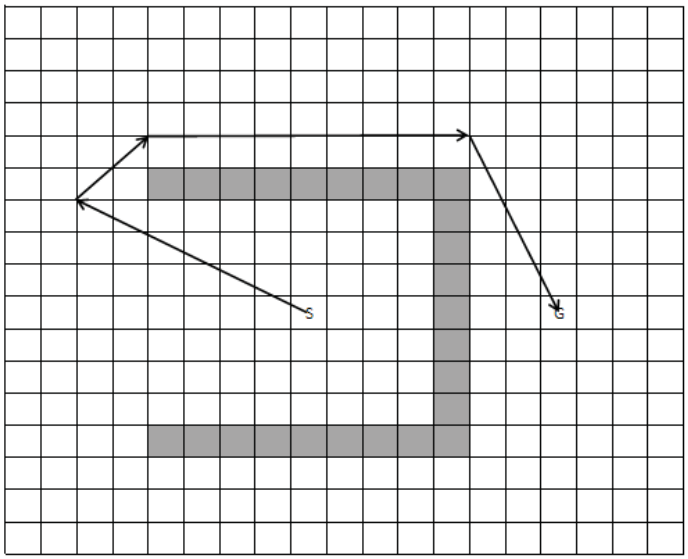


Figure 4.19: The sharp bend in the path

result didn't illustrate the efficacy of flexibility of angle selection nor the guarantee of the search result in a complex environment.

4.3.2 Dynamic Attractive Point of the Artificial Potential Field

The simplest solution to avoid the sharp bend problem is to use artificial potential field again: Since the search algorithm has found a path to the goal point already, the set of the way points excluding the start point could be used as the attractive point of the artificial potential field. As soon as the path is found, the robot begins to move to the first way point, applies the force from the artificial potential field only, and then when the first way point is reached, the attractive point should be switched to the second one in the path. Keep iterating this procedure until the goal node is reached.

This solution is based on the fact that the artificial potential field executes a smooth motion of the agent. However, there is another factor that may not be ignored: mass. Similar to the virtual force, all the moving agents affected by the potential field should have a virtual mass, that determines the acceleration of the agent with the force. That is, there is inertia. Consider the scenario illustrated in the figure 4.19 again: marking the path from start node to goal node as $\{s_0, s_1, ...s_n\}$, the acute angle is $\angle s_0 s_1 s_2$. While the agent is moving along the straight line $s0s1$ and passes the point $s1$, due to the inertia, it may not turn right immediately but keep moving forward for a short while then turning back. At this point, inertia ensure that the actual motion is smooth enough. However, when moving towards s_2, because of the inertia in the previous

motion, there may be the possibility that the moving is not from bottom-left to top-right, but from top-left to bottom right. The potential field will not fix that motion because of the attractive point s_2. If the attractive force is bigger than the repulsive force since the distance to the attractive point is shorter than that to the obstacles, obviously, the agent can reach the way point s2, then have a high possibility to hit the obstacle. Otherwise, due to the bigger repulsive force from the obstacle, the agent may not reach the way point s_2, and instead keep patrolling around s_2.

4.3.3 The Angular Factor in Optimal Search

An alternative method is to reduce the acute angles in the search result: this involves the angular factor in the search algorithms. Consider the scenario demonstrated in the figure 4.20:

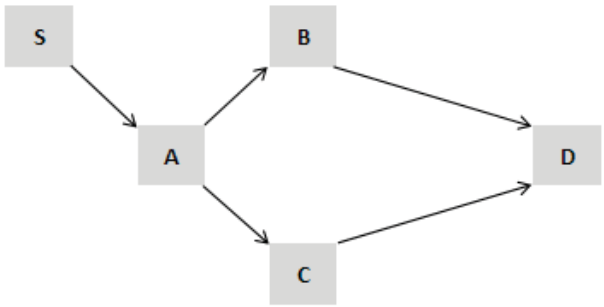


Figure 4.20: Two paths have same fcost but different angular cost

The path P_1: [S-A-B-D] and P_2: [S-A-C-D] have the same fcost. However, comparing with P_1, there are less bends in P_2, and the only angle in P_2 is an obtuse angle. That is, the path of P_2 is much smoother than P_1.

The angular factor α is defined as: the angle of the agent that has to turn while it is moving through the way point N. Assuming that there are two generalized vectors: $\overrightarrow{u}$ is the incoming direction, and $\overrightarrow{v}$ is the outgoing direction. The angle factor between these two vectors is:

$$\alpha = abs[arccos(\frac{\overrightarrow{u} \cdot \overrightarrow{v}}{||\overrightarrow{u}||\,||\overrightarrow{v}||})] = abs[arccos(\overrightarrow{u} \cdot \overrightarrow{v})]. \tag{4.11}$$

4.3.4 Involving the Angular Factor in the Search Algorithm

4.3.5 The Second Key Value

In A* Search, the key value is the most significant factor that determines the order of the search nodes. The fcost is the only key in the A* search and the alternative versions. While the angular factor is involved, it could be used as the second key:

```
struct NodeKey{
        double fcost;
        double angularFactor;

        friend bool operator (
                const NodeKey& k1,
                const NodeKey& k2){
                if(k1.fcost < k2.fcost){
                  return true;
                }else if(k1.fcost > k2.fcost){
                  return false;
                }else{
                  return
                  k1.angularFactor < k2.angularFactor;
                }
        }
};
```

The second key indicates the search procedure towards the direction with smaller bend angle, but only if the fcost of two search nodes are exactly the same. However, in the Theta* search with euclidean distance and the artificial potential field magnitude, it is almost impossible to find two search node that would have same fcost. As a result, a better way of using the angular factor is required.

4.3.6 Unified Key with Separate Coefficients

The unified key combines the two keys together like a convolution. For example, the equation 4.12 demonstrates one of the most simple way of the combination is using add them together with separate coefficients:

$$key = s \cdot fcost + t \cdot angularFactor \qquad where\ s, t \in \mathbb{R}^{+}. \tag{4.12}$$

The drawback of this combination is that the key value cannot distinguish the angular factor from the entire key, so that it is impossible to figure out the difference between the two search nodes. That is, there might be such case: there are two search nodes which have very similar key value. For one of them the path is sharper but farther away from the obstacles. The other is smoother but closer to the obstacles. It is hard to figure out which is safer, and the actual value in this case cannot be calibrated.

4.3.7 Alternative Heuristic Function

The heuristic is another significant part of the optimized search algorithms. The requirement of heuristic is admissible but not must be consistent [21]. For the angular factor, the major issue of involving it in the heuristic is to maintain the admissible.

Considering the Node x and all the parent nodes $p \in \{P\}$, there will be a direction that moving from p to x, marked as $\overrightarrow{u}$. Then for all the child nodes x', there will also be a direction moving from x to x', marked as $\overrightarrow{v}$. In order to maintain the admissible of the heuristic, the expand list should be modified to not store a single node, but a node pair with the smallest fcost.

The simplest heuristic is to add the two values together:

$$\begin{cases} \alpha & = abs[arccos(\overrightarrow{u} \cdot \overrightarrow{v})], \\ h(x) & = ManhattanDistance(x, g) + \alpha. \end{cases} \tag{4.13}$$

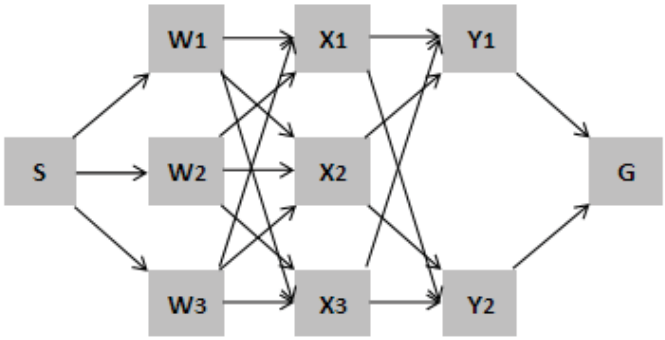


Figure 4.21: Same node may have different angular heuristic from different parent

Consider the scenario illustrated in the figure 4.21. Assuming that the manhattan distance is applied here other than euclidean distance, when the start node is added to the first search node and expanded, there are three search nodes generated:$S - W_1$,

$S - W_2$ and $S - W_3$. Now, the gcosts of these three are 1, and since it is the first step, the angular factor α is 0 as well. As a result, the heuristic is 3 for all.

Table 4.1: Search nodes at first step

	fcost	gcost	heuristic
$S - W_1$	4	1	3
$S - W_2$	4	1	3
$S - W_3$	4	1	3

Then, the search node will be popped from the search list and W_1 is going to expand. As a result, the first two nodes for $S - W_1$ are expanded as a single node, and then moving to the next step, examining the successors of W_1.

The angular factors of the three successors are:

Table 4.2: Angular Factors

	α
$W_1 - X_1$	0.785
$W_1 - X_2$	1.571
$W_1 - X_3$	2.678

Further steps follows. The heuristic of $S - W_1 - X_1$ is 2 + 0.785 = 2.785, and the gcost is 1+1, so that the fcost is 4.785. After all the three successors are examined, the heuristic and cost all the W-X nodes are :

Table 4.3: gcost and heuristic of all the nodes

	heuristic	gcost
S-W1-X1	2.785	2
S-W1-X2	3.571	2
S-W1-X3	4.678	2
S-W2-X1	2.785	2
S-W2-X2	2	2
S-W2-X3	2.785	2
S-W3-X1	4.678	2
S-W3-X2	3.571	2
S-W3-X3	2.785	2

The admissible in A* search is for node x, $h(x) \leq c(x, g)$. The node stored in the expand list is not a single node but the node pair, that is admissible requirement has been split

into two parts: the first part is same as the normal one that only comparing the distance relationship, and the second is comparing the direction relationship. Assuming there are two connect node pairs (x,y) as vector $\overrightarrow{u}$ and (y,z) as vector marked as $\overrightarrow{v}$, the heuristic should satisfied the condition of $h(\overrightarrow{u}) \leq c(\overrightarrow{u}, \overrightarrow{v})$.

That is, if the parent node x, current node y and the child node z are put together, an inverse vector could be use that $\overrightarrow{v} = z- > y$ and $\overrightarrow{u} = y- > x$, then it is easy to figure out that $c(\overrightarrow{u}, \overrightarrow{v})$ is the angular factor α, and let $h(\overrightarrow{u}) = \alpha$, then the admissible is satisfied. That is, when the vector y-¿z is expanded, the heuristic of y-¿z is the angular factor from x-¿y to y-¿z plus the distance cost from y to z, since both of the side satisfied the admissible condition, the result will be satisfied as well.

Then the consistency of the heuristic: the requirement of consistent is $h(\overrightarrow{u}) \leq c(\overrightarrow{u}, \overrightarrow{v})+ h(\overrightarrow{v})$. Since $h(\overrightarrow{u})$ is the angle between $\overrightarrow{u}, \overrightarrow{v}$ already, there is $h(\overrightarrow{u}) = c(\overrightarrow{u}, \overrightarrow{v})$. the equation 4.11 has demonstrated that the range of the angular factor is $\alpha \in [0, 2\pi)$ (will not be π because the parent node will not be searched and the constraint of the line-of-sight), that is, for all the possible child of the node z in vector $\overrightarrow{v}$, the angular factor will not be smaller than 0. That is, $h(\overrightarrow{u}) = c(\overrightarrow{u}, \overrightarrow{v})\, and\, h(\overrightarrow{v}) \geq 0 \Rightarrow h(\overrightarrow{u}) \leq c(\overrightarrow{u}, \overrightarrow{v}) + h(\overrightarrow{v})$.

This heuristic indicates that when the searching processes a certain node, which direction have the smallest bend angle. This angle will not be accumulated in the gcost so that search algorithm can find the result has the most smooth path but not the least bend count.

4.4 Algorithms Implementing the Line-of-sight Detection

The line-of-sight (LOS) check is an important component of the Theta* Search, which is to indicate if there is any visual between two given positions or not. As which has been introduced in the section 2.1.4, the search field could be divided based on the vertex or the cell, that is, the LOS algorithms should be implementd based on the vertex or the cell separately.

The inputs of the LOS algorithms are two different coordinates, and the output is a boolean value which is true if there is a visual between the input coordinates, otherwise false.

4.4.1 Simple Line-of-Sight Checking for Vertex-based Search

In the case of vertex-based search, the sample of line-of-sight check is illustrated in the figure 4.22, where the input coordinates are A1 and H5:

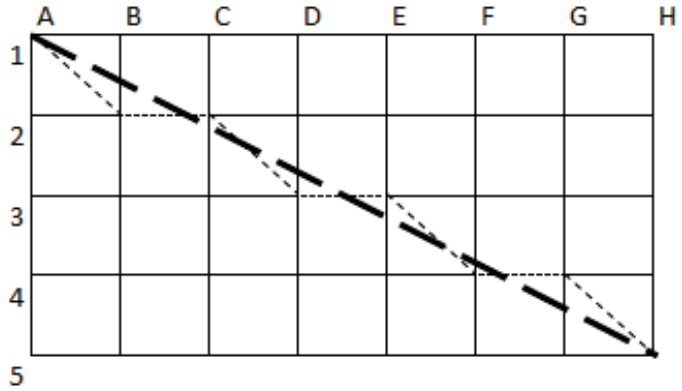


Figure 4.22: The same path of the figure 2.19 could be smoothed

The edges are marked like $< C2, D2 >, < D2, D3 >$, etc., depending on the cells. The LOS check is to find out possible intersections between the segment of part of the path and the edges of the cells. It could be found like this: first, comparing the x distance and y distance between the two given end point of the segment, and then selecting the minimum one as the minstep, and the maximum one as the maxstep. Then, traversing and examining the accessibility of all the possible vertices by iteration if xstep is greater than ystep, otherwise swap x and y in the loop blocks (see listing).

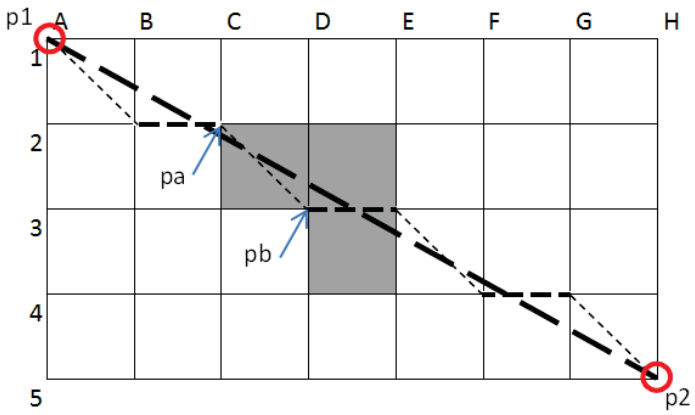


Figure 4.23: An intermediate step when processing LOS checking

```
//INPUTS: p1, p2 are the coordinates of the two way-points
//OUTPUT: Specify if there is a LOS between p1 and p2
int dx = p2.X - p1.X;
int dy = p2.Y - p1.Y;
int xop = dx / abs(dx); //step direction of x
int yop = dy / abs(dy); //step direction of y

for(i=0;i<minstep;i++){
        int p = i*maxstep / minstep;
        int pn = (i+1)*maxstep/minstep;
        for(j = p;j<pn;j++){

                Point pa,pb;
                pa.x = p1.x - xop*(j+1);
                pa.y = p1.y - yop*(i);
                pb.x = p1.x - xop*(j+1);
                pb.y = p1.y - yop*(i+1);

                //check the cell between pa & pb is
                   accessible or not
                bool b = checkAccessible(pa,pb);
                if(!b){
                        return false;
                }
        }
}
return true;
```

Since the x and y directions have been checked previously, there is no need to check both possible edges of a single cell.

4.4.2 Modified Bresenham's Algorithm for Line-of-Sight Detection

The algorithm of modified Bresenham's line algorithm is discussed next, where the original algorithm is to mapping a line to pixels [12]. Figure 4.24 illustrates a sample of the Bresenham' line algorithm: while checking the LOS from the cell A1 to the cell F3, the light gray cells should be examined to check if they are blocked or not and determine if there is a LOS.

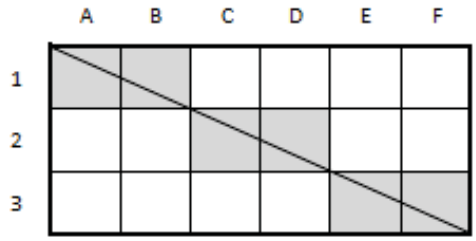


Figure 4.24: The light gray cells are where should be examined

The pseudo code for examining the accessibility of the cells in the grid-world is:

```
IsCellAccessible:
        INPUTS: x, y as the coordinates of the cell
        OUTPUT: cell blockage status, true if accessible,
           otherwise false
```

The pseudo code of use Bresenham's line algorithm to examine the line-of-sight is:

```
Bresenham Line for Line-of-sight check:
        INPUTS: p1, p2 are the coordinates of the two way-
           points
        OUTPUT: true if there is a LOS between p1 and p2;
           otherwise false

        Point p1,p2
        int x0 := p1.x
        int y0 := p1.y
        int x1 := p2.x
        int y1 := p2.y
        bool needSwap := abs(y1-y0) > abs(x1-x0)
        IF needSwap THEN
                swap(x0,y0)
                swap(x1,y1)
        ENDIF
        IF x0 > x1 THEN
                swap(x0,x1)
                swap(y0,y1)
        ENDIF
        int dx := x1 - x0
        int dy := abs(y1 - y0)
        int err := dx / 2
        int i := y0
```

```
23          int ystep := y0 < y1 ? 1 : -1
24          FOR j := x0 to x1 Increment by 1
25                  BOOL accessible
26                  IF needSwap THEN
27                          accessible := IsCellAccessible(i,j)
28                  ELSE
29                          accessible := IsCellAccessible(j,i)
30                  ENDIF
31
32                  IF NOT accessible THEN
33                          RETURN FALSE
34                  ENDIF
35                  err := err - dy
36                  IF err < 0 THEN
37                          i := i + ystep
38                          err := err + dx
39                  ENDIF
40          ENDLOOP
41          RETURN TRUE
```

However, the original Bresenham's line algorithm cannot check all the cells where the segment go through. Considering the sample illustrated in the figure 4.25, the cells B2 and D3 are cannot be found by the original algorithm, but stay on the path. A modification of the algorithm should be applied.

Back to the original algorithm, the case of missing examined cells is happened when the line is moving from the previous step to the next step (line 37). There should be an extra examination when err is 0, before err := err -dy (line 35), that indicates the examination is moving to the next step.

```
1           IF err is 0 THEN
2                   IF needSwap THEN
3                           accessible := IsCellAccessible(i+
                               ystep,j)
4                   ELSE
5                           accessible := IsCellAccessible(j,i+
                               ystep)
6                   ENDIF
7           ENDIF
8
9           err := err - dy
```

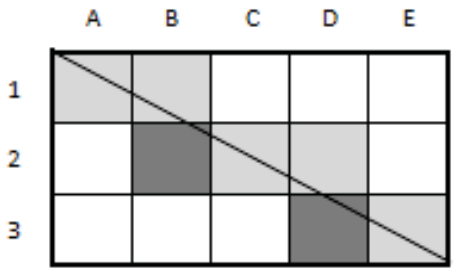


Figure 4.25: The dark gray cells will not be found by the original algorithm but should be examined

Nevertheless, this additional examination also causes an unexpected problem. That the segment is going through a vertex of the cells. Since this case only happened when the width could be divided by the height, it could be avoid by checking that:

```
1         int dx := x1 - x0
2         int dy := abs(y1 - y0)
3         int err := dx / 2
4
5         bool remainderCheck := (dx + 1) % (dy + 1) != 0
6
7         /* ...  */
8
9         IF err is 0 AND remainderCheck  THEN
10                IF needSwap THEN
11                        accessible := IsCellAccessible(i+
                             ystep,j)
12                ELSE
13                        accessible := IsCellAccessible(j,i+
                             ystep)
14                ENDIF
15        ENDIF
```

4.4.3 Obstacle as Polygon

In some of the scenarios, it is better to examine the line-of-sight depending on the original obstacles before they are simplified. Assuming that the original obstacles are define as a polygon by a series of connected vertices $V = \{v1, v2, ...vn\}$, the line-of-sight could be checked by finding any intersection between the segment and any of the edges of the obstacle.

Considering the scenario illustrated in the figure 4.26: the segment is plotted by two endpoints $S = \{P_a, P_b\}$, and the polygon is a plotted by six points $P = \{P_1, P_2, P_3, P_4, P_5, P_6\}$. Then, the LOS checking is processed by examining the intersection between the segment S and each of the edge of the polygon P.

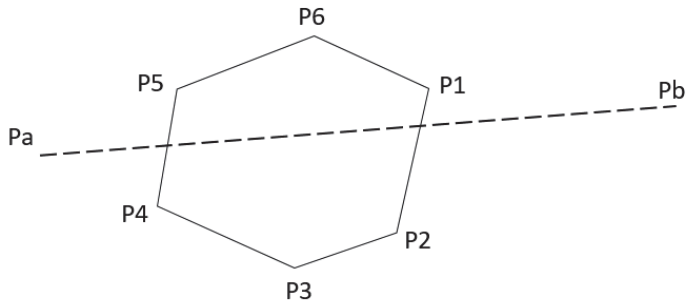


Figure 4.26: The intersection between a segment and a polygon

The following pseudo code shows the procedure of checking if there is any intersection between two segments:

```
Input: p1, p2 are the endpoints of segment 1
Input: p3, p4 are the endpoints of segment 2
Output: TRUE if there is an intersection, otherwise FALSE

HasIntersection: Point p1, p2, p3, p4
        a1 := p2.y - p1.y;
        b1 := p1.x - p2.x;
        c1 := p2.x * p1.y - p1.x * p2.y;
        r3 := a1 * p3.x + b1 * p3.y + c1;
        r4 := a1 * p4.x + b1 * p4.y + c1;

        IF abs(r3) > 0 && abs(r4) > 0 && r3*r4 > 0 THEN
                RETURN FALSE
        ENDIF

        a2 := p4.y - p3.y;
        b2 := p3.x - p4.x;
        c2 := p4.x * p3.y - p3.x * p4.y;
        r1 := a2 * p1.x + b2 * p1.y + c2;
        r2 := a2 * p2.x + b2 * p2.y + c2;

        IF abs(r1)>0 && abs(r2) > 0 && r1*r2 > 0 THEN
                RETURN FALSE
```

```
24          ENDIF
25          RETURN TRUE
```

Using the procedure HasIntersection on the segment and all the edge of the input polygon, the result of the LOS checking will be FALSE if there is any intersection between the segment and the edge of the polygon, otherwise it is TRUE.

4.4.4 Modified Cohen-Sutherland Algorithm

Alternatively, sometimes the obstacle could be simplified as a rotated rectangle, so the line-of-sight check is to detect the intersection between a segment and a rectangle. The Cohen-Sutherland algorithm is the most efficient way to do that[14], where the only modification that should be applied is to rotated the rectangle and the segment back to horizontal.

The Cohen-Sutherland Algorithm was introduced by Cohan, D. and Sutherland, I. in 1967[14]. It is an efficient algorithm to determine if there is an intersection between a segment and a rectangle. This algorithm divides the rectangle into 9 sections, marked by a 4-digits binary flag, as it is illustrated in the figure 4.27:

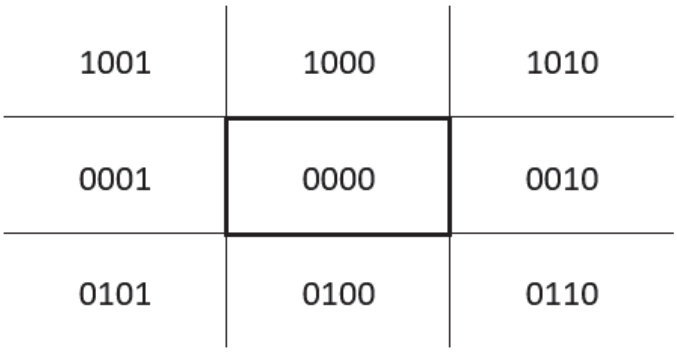


Figure 4.27: The flags of the Cohen-Sutherland Algorithm

The first digit is marked as 1 if the section is at the top (1000_2); The second digit is marked as 1 if the section is at the bottom (0100_2); The third digit is marked as 1 if the section is to the right (0010_2); And the fourth digit is marked as if the section is to the left (0001_2).

Then, the algorithm detects the possible intersection by examining the end points of a segment: if the higher two or lower two are the same and non-zero, then there is no intersection, otherwise move to further examination.

if the higher two digits or the lower two digits are both zero, check the other two. If they are the same and non-zero, there is no intersection, otherwise there is.

If both the higher two or the lower two are different, the segment is not horizontal or vertical. By comparing the value between the end points and the vertices of the rectangle, the algorithm examines the possibilities case by case:

Marking the end point p_1 of the segment which has a lower x value, the vertices whose x value is smaller than p_1 could be ignored. Then, comparing the rest of the vertices to figure out if there is any vertex in which the x value is greater than the x component of the other end point p_2. If it is, the cases are converted to find out the intersection by using the concept of similar triangles.

Assuming the centre point pivot, rotating angle θ, width w and height h of a rectangle has been given, with the endpoints p1 and p2 of the segment, the first step is to rotate them back:

$$\begin{cases} R & = \begin{bmatrix} cos(-\theta) & sin(-\theta) \\ -sin(-\theta, & cos(-\theta) \end{bmatrix}, \\ p1' & = R(p1 - pivot) + pivot, \\ p2' & = R(p2 - pivot) + pivot. \end{cases} \tag{4.14}$$

Also, the two docking vertices of the rectangle are

$$\begin{cases} rpmin = p0 & = pivot - \{w/2, h/2\}, \\ rpmax = p1 & = pivot + \{w/2, h/2\}. \end{cases} \tag{4.15}$$

with these new points, the Cohen-Sutherland algorithm could be applied. The pseudo code is:

```

//Enums used in the Cohen-Sutherland Algorithms
//That indicates the position of a point
enum CSCode{
        CS_INSIDE = 0x0,
        CS_LEFT = 0x1,
        CS_RIGHT = 0x2,
        CS_BOTTOM = 0x4,
        CS_TOP = 0x8
};
```

```
//pt is one of the coordinate of the segment's endpoint
//rpmin and rpmax are the coordinates of the rectangle's diagonal
int outcode( pt, rpmin, rpmax){
        int code = CS_INSIDE;

        if (pt.x < rpmin.x){
                code |= CS_LEFT;
        }else if (pt.x > rpmax.x){
                code |= CS_RIGHT;
        }if (pt.y < rpmin.y){
                code |= CS_BOTTOM;
        }else if (pt.y > rpmax.y){
                code |= CS_TOP;
        }
        return code;
}
```

```
//INPUTS: p1 and p2 are the coordinates of the segment's endpoints corresponding to the two adjacent way-points
//               rpmin and rpmax are the coordinates of the rectangle's diagonal, corresponding to the obstacle being examined
//OUTPUT: TRUE if there is a line-of-sight between p1 and p2, otherwise FALSE
bool Cohen_Sutherland(p1, p2,rpmin,rpmax){
        int outcode0 = outcode(p1,rpmin,rpmax);
        int outcode1 = outcode(p2,rpmin,rpmax);
        bool accept = false;
        while (true) {
                if (!(outcode0 | outcode1)) {
                        accept = true;
                        break;
                } else if (outcode0 & outcode1) {
                        break;
                } else {
                float x, y;
                int outcodeOut =
                    outcode0 ? outcode0 : outcode1;
                if (outcodeOut & CS_TOP) {
                  x = p1.x + (p2.x - p1.x) *
```

```
20                    (rpmax.y - p1.y) / (p2.y - p1.y);
21                    y = rpmax.y;
22                  } else if (outcodeOut & CS_BOTTOM) {
23                    x = p1.x + (p2.x - p1.x) *
24                    (rpmin.y - p1.y) / (p2.y - p1.y);
25                    y = rpmin.y;
26                  } else if (outcodeOut & CS_RIGHT) {
27                  y = p1.y + (p2.y - p1.y) *
28                    (rpmax.x - p1.x) / (p2.x - p1.x);
29                    x = rpmax.x;
30                  } else if (outcodeOut & CS_LEFT) {
31                    y = p1.y + (p2.y - p1.y) *
32                    (rpmin.x - p1.x) / (p2.x - p1.x);
33                    x = rpmin.x;
34                  }
35
36                  if (outcodeOut == outcode0) {
37                          p1.x = x;
38                          p1.y = y;
39                          outcode0 =
40                            outcode(p1,rpmin,rpmax);
41                  } else {
42                          p2.x = x;
43                          p2.y = y;
44                          outcode1 =
45                            outcode(p2,rpmin,rpmax);
46                  }
47          }
48    }
49    return accept;
50  }
```

Chapter 5

Strategies for Building Robot Behaviours

In the robotics soccer game, both the attacker and the keeper should have the strategies to indicate how to select the algorithms. In general, both the attacker and the keeper must have the abilities of moving towards a target point, and avoid the obstacle on the way. Then, the unique strategies are applied separately to perform the different behaviors due to their unique characteristics.

5.1 Generic Strategies

The generic strategies are applied to both the attacker and the keeper, including the target pursuing and the object avoidance. The fundamental of these generic strategies is using the optimal search algorithms with the artificial potential field to do the path planning and re-planning, then driving the robot along the way points of the path and avoiding the obstacles by the artificial potential field.

5.1.1 Target Pursuing

The target pursuing is consists in path planning by search algorithms and driving by the artificial potential field. The figure 5.1 demonstrates the procedure of the path planning and re-planning when pursing the target. First, the the goal g is assigned as the target's position, and the start s is assigned as the current position of the agent. Then, the search algorithm will plan a path by given start and goal position. As soon as a valid path is found, the agent will move toward the first way point s on the path and avoid the possible obstacles by the artificial potential field. When the agent has

arrived at s, it will check if the target is still around the goal or not. The next way point will be extracted from the path and pursued (if the target is still there), otherwise a re-planning will be processed depend on the current path. In the process of re-planning, the way point s_i which is the nearest to the target will be set as the new start position, and the way points after s_i are deleted. Then, a new path from s_i to $g = target$ is computed and appended to the end of the previous path.

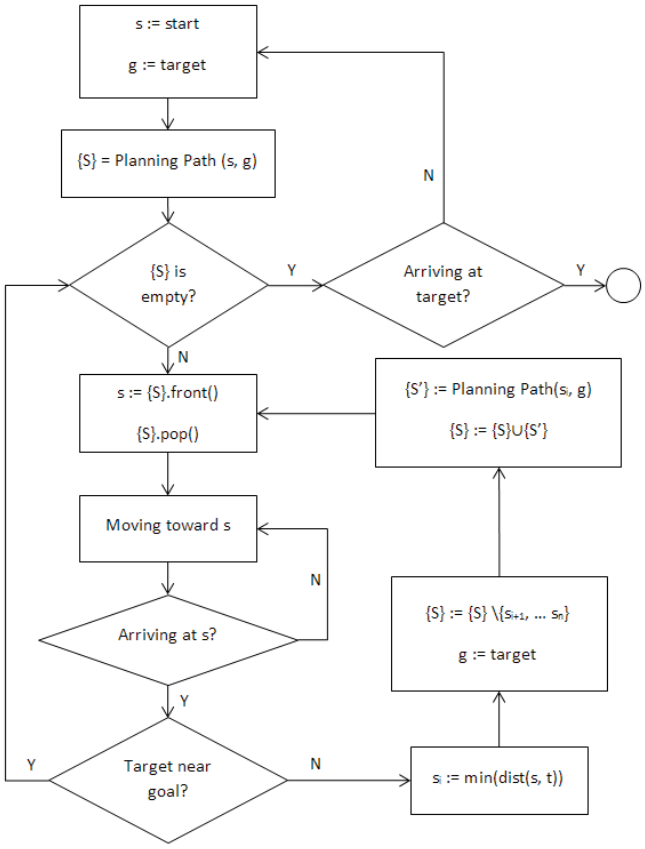


Figure 5.1: The flow chat of path planning and re-planning

5.1.2 Obstacle Avoidance

The obstacle avoidance is processed during different phases: as soon as the search algorithm has planned a path, the way-points are plotted forming a path around the current positions of the obstacles. However, the characteristic of the search algorithms determines only that the way-points can keep a proper distance away from the obstacles, but the segment plotted by neighbouring way-points may have a smaller distance to the obstacle.

Figure 5.2 shows an example: for the case of the segment from D3 to E4, both the way-points D3 and E4 have a proper distance away from the obstacle, but the segment is too close. While the agent is moving from D3 to E4 along a straight line, it may hit the obstacle which should be avoided.

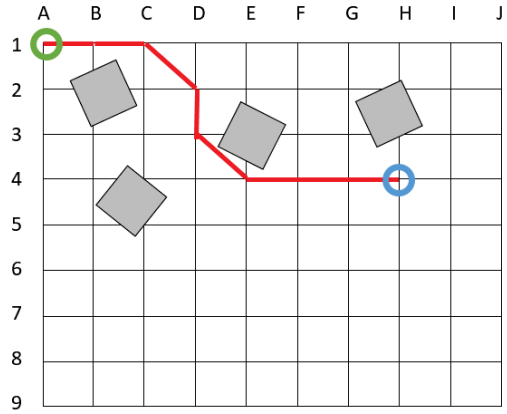


Figure 5.2: A possible path found by the A* Search

Considering the artificial potential field around the obstacle 2, as it is shown in figure 5.3: the obstacle 2 is generating the repulsive force which direction is to push the agent away from it, and the E4 is the next way-point which has an attractive force that direction is point to E4. As a result, the direction of the net force will neither point at E4 nor away from the obstacle, but a curve that goes around the obstacles.

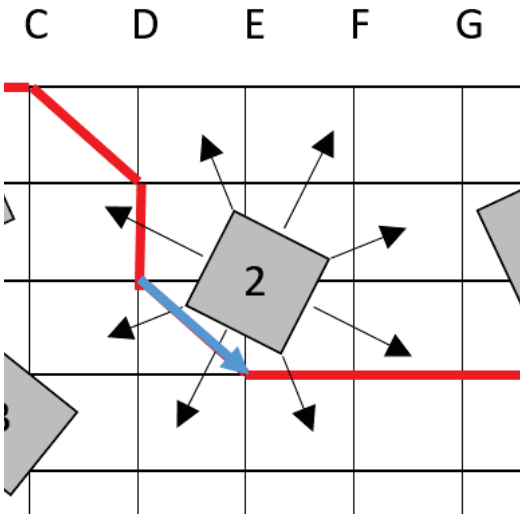


Figure 5.3: The distribution of the artificial potential field around the obstacle 2

Figure 5.4 illustrates the expected trail of the agent moving around the obstacle 2 driven by the artificial potential field. The drawback of the artificial potential field due to the potential well still exists, but in a small area. Due to the thresholds, the artificial

potential field will be affected by only one obstacle and the attractive point, where the potential well does not exist.

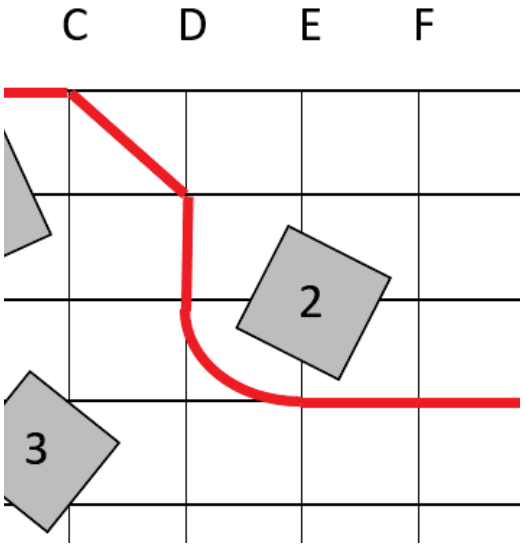


Figure 5.4: The expected actual trail of the agent

5.2 Attacker Strategies

Other than pursuing the target object, the purpose of the attacker is to push the ball into the goal and not simply pursuing the ball. So that an alternative strategy of finding an appropriate position to kick the ball is essential to the attacker behavior.

5.2.1 Dynamic Attacking Position

The strategy of dynamic attacking position is to navigate the attacker to a proper place for attacking. Considering the example shown in figure 5.5:

Point A is the position of the ball, X is the position of the attacker 1, B is the centre of the goal and C is the expected attacking position. The angle $\alpha + \beta$ is the kicking area where the attacker can process the attack. The angle $\theta = \angle CAX$ is to determine if the attacker is in the kicking area or not. Assuming the length of the segment AC is d_1, and the length of the segment AB is d_2, then the position of C could be found by using a similar triangle:

$$\begin{cases} \frac{x_3 - x_1}{d_1} = \frac{x_1 - x_2}{d_2}, \\ \frac{y_3 - y_1}{d_1} = \frac{y_1 - y_2}{d_2}. \end{cases} \tag{5.1}$$

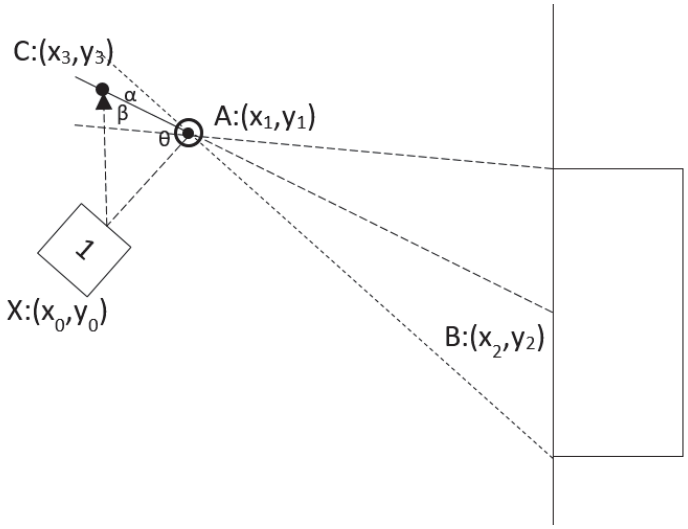


Figure 5.5: The attacking position

If d_1 is greater than the $max(length, width)$ of the agent, then putting an attractive force at the point C and converting the artificial potential field generated by the ball to a repulsive force, the agent can move towards the attacking position if it is not yet in the attacking area.

5.2.2 Finite State Machine for Attacker Intelligence

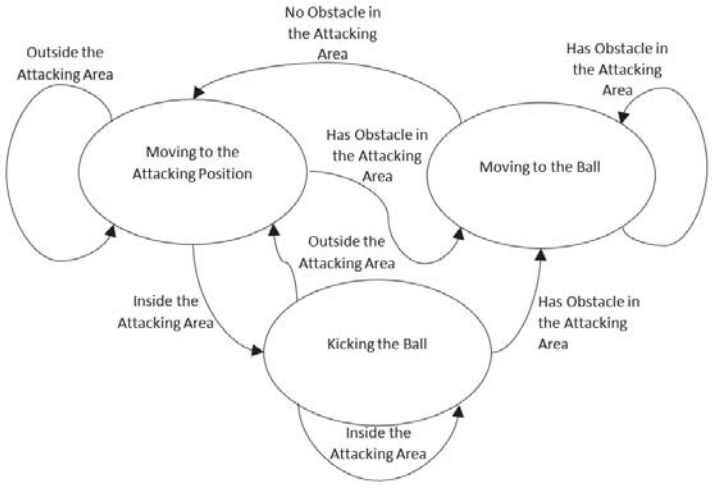


Figure 5.6: The states and transition conditions of the attacker

Since the dynamic attacking position is introduced, there will be multiple behaviors of the attacker: moving toward the attacking position, moving toward the ball and kicking

the ball. The only condition that may change the state to moving towards the ball is any obstacle that is placed inside the attacking area. There is a higher priority that the attacker must push the ball away from the obstacle first. If the attacking area is clear, the attacker will move into the attacking area first, then switch the target point to the ball again and accelerate to the max speed to kick the ball. As soon as the ball is kicked, the attacker should try to keep attacking, the state will be switched depend on the new status of the ball and the attacking area.

Figure 5.6 shows the finite state machine of the attacker.

5.3 Goal Keeper Strategies

Different from the attacker that will keep attacking, the keeper should stay in front of the goal, waiting for any possible threat and protecting the goal by blocking the ball. The behavior of the keeper will be more complex than the attacker.

5.3.1 Defensive Blocking Position

Similar to the attacker, the keeper should have the ability to find out a blocking position for the coming attacker. If the ball is near the penalty box, the keeper need to move to the blocking position. Figure 5.7 illustrates the definition of the blocking position E which is determined by the current parking position F and the segment AD, where A is the position of the ball, and D is the centre of the goal. The segment FE is perpendicular to the segment AD and E is the perpendicular foot.

The perpendicular foot could be found by equations 6.6 and 6.7 in the future chapters. If the foot is inside the goal, then the blocking position should be switched to the intersection point of the segment AD and BC, which will be the point D.

5.3.2 Parking

If the keeper is far away from the goal, it should have the ability to move back to the front of the goal. Furthermore, the keeper has no need to move around if the ball is far away from the goal. That is, the behavior of parking contains two parts: moving back to the gate and waiting. The condition of switching between these two parts is the distance between the keeper and the goal: by setting the centre of the goal as the attractive point, the keeper can move back to the goal automatically if the current behavior selection is parking, and waiting at the goal if the distance to the goal is near enough.

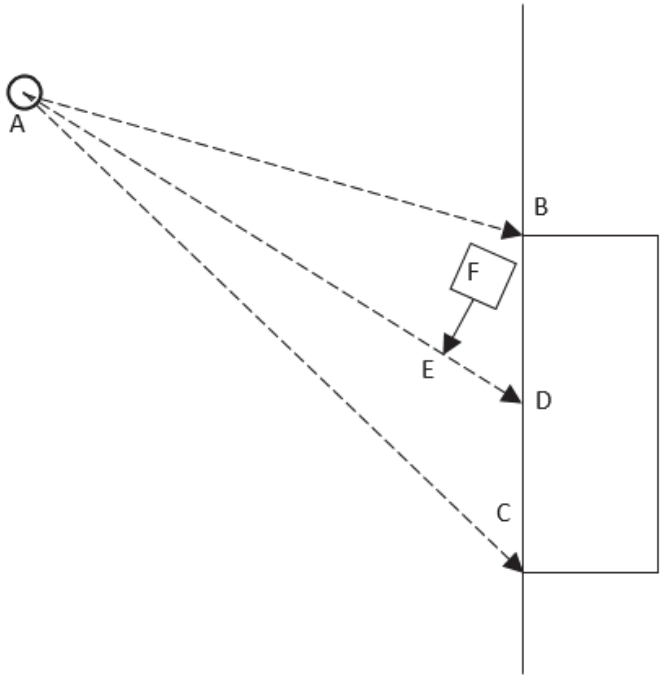


Figure 5.7: Moving to the blocking position E for coming attacking

5.3.3 Finite State Machine for Keeper Intelligence

There are three major behaviors of the keeper: blocking the ball, pursuing the ball and parking. Slightly different from the attacker, the condition of the transitions depend on the distance between the ball, the keeper and the goal separately.

The condition which has the highest priority is when the ball is close to the goal, which will order the keeper to push it away from the goal immediately. All the states will switch to pursuing the ball and push it away. However, in most cases, the keeper only need to switch between the case of blocking and parking.

If the keeper has moved to the blocking position, there is no need to switch back to parking, even though the ball is far away from the penalty box. Moving to the predicted blocking position is always a safe behavior that can protect the goal.

There are two conditions of switching to the behavior of moving to the blocking position: the ball is moving close to the penalty box or after a pursuing defense. After the keeper has arrived at the blocking position, it will check the current status of the ball and determine if it needs parking or not. If the situation is safe, the keeper will move to goal and waiting around the centre of the goal until the ball is close to the penalty box or the goal.

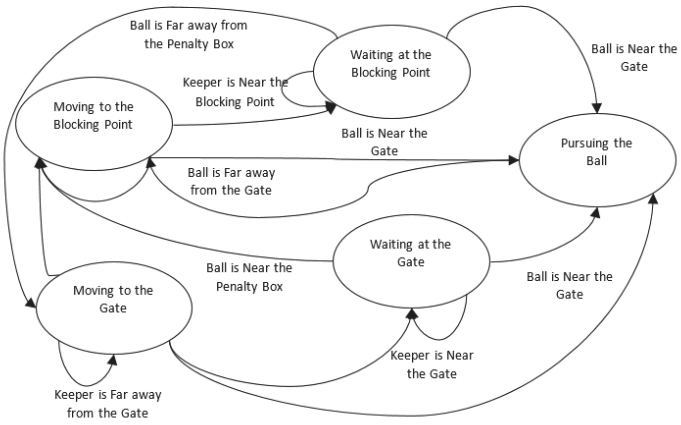


Figure 5.8: The states and transition conditions of the keeper

Figure 5.8 demonstrates the finite state machine of the keeper. When the keeper changes the target, a path re-planning will happen, otherwise the keeper will move along the current path and then wait around the target point.

Chapter 6

Implementation of the Simulation Platform

The simulation platform is used to process and to demonstrate the experiments and explorations of all the algorithms. In this research, the simulation platform is developed by c++11 with OpenGL ES 4.0. It could be compiled and run on different platforms like Win32 and OSX. It also uses the libraries GLFW, GLEW, GLM and SOIL, which are all the open source libraries under different license.

6.1 Implementing System Dynamics

For the purpose of examining the performance of the robotics navigation algorithms, the mechanics simulation should be established based on the Newton's laws of motion.

Discrete Model

Due to the nature of digital computers, the discrete model of the newton's laws of motion is compulsory. Considering the formulas of newton's second law:

$$\begin{cases} \vec{F} &= m\frac{d\vec{v}}{dt} = m\vec{a}, \\ \vec{J} &= \int_{\Delta t} \vec{F} dt. \end{cases} \tag{6.1}$$

In the computer, there is no actual continuous procedure. That is, during a given interval Δt, there must be at least one variable that is considered as constant. In

the discrete model, the acceleration is the selected constant value. Then, the discrete formulas are

$$\begin{cases} \vec{v}(t_1) & = \vec{v}(t_0) + \vec{a}\Delta t, \\ \vec{s}(t_1) & = \vec{s}(t_0) + (\vec{v}(t_0) + 0.5\vec{a}\Delta t). \end{cases} \tag{6.2}$$

The Rigid Body Simplification

Because the search field is a 2D-plane, most of the objects on this plain could be simplified to 2D polygons crossed by multiple segments, which could be represented as a series of points $P = \{p_1, p_2, ...p_n\}$.

In order to simplify the complexity of the computing, and also considering the simplification of the search field and obstacles, rectangle, as one of the most simple polygon, is preferred to represent a directed rigid body.

A rectangle could be described by a set of parameters, as it is shown in the figure 6.1: the centre point $P = (x_0, y_0)$, the width w, the height h and the direction θ.

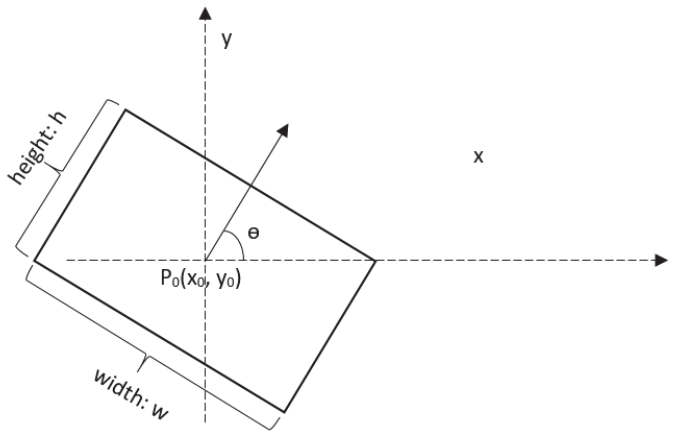


Figure 6.1: Use 5 parameters to describe a rectangle

Most of the objects could be represented by multiple rectangles, which could be overlapped to each other. However, under some of the scenarios, for example, a wall without height, we have no need to use rectangles. Therefore, for the case of a wall, a segment $W = \{P_0, P_1\}$ can represent it completely.

Furthermore, a circle is another simple but important component of the simplification, which is described by two parameters: the centre point $P = \{x_0, y_0\}$ and the radius r.

6.1.1 Collision Detection

Collision detection is the most significant feature of the mechanics simulation. Baber introduced the basic method of collision detective modularization in 2006 [53]. In the 2D-space, the objects are simplified to polygons or circles. Then, the collision is indicated by any overlap.

Since rigid bodies may not overlap, the collision detection procedure is to figure out it after the motion of the latest interval if there is any object overlapped.

Because of the various types of rigid bodies, the detection should be computed separated.

Rectangle - Rectangle Collision

The most common case of a collision is rectangle - rectangle collision. Suppose that there are two rectangles $R_1 = \{x_1, y_1, h_1, w_1, \theta_1\}$ and $R_2 = \{x_2, y_2, h_2, w_2, \theta_2\}$, let the maximum radius of a rectangle is $r_m = \sqrt{(\frac{w}{2})^2 + (\frac{h}{2})^2}$, and the distance between the centre point of the rectangles is $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$. It is obvious that the the condition $d \leq r_{m1} + r_{m2}$ is a consequence of a collision. This could be used as the necessary condition to filter the cases of non-collision.

However, the actual collision only happens if there is an intersection of the edges from the different rectangles. As a result, further computation is required.

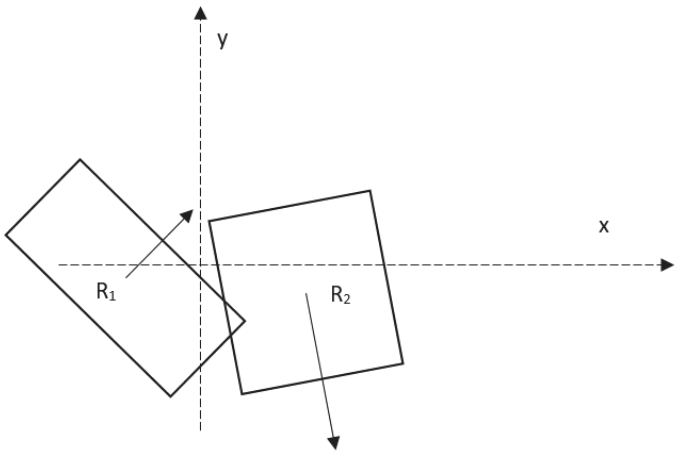


Figure 6.2: Two rectangles are overlapped

Figure 6.2 shows that a collision happened between two rectangles. The sufficient and necessary condition of the collision is that there is an intersection between the edges

from different rectangles. The modified Cohen-Sutherland algorithm could be applied here to detect the intersection for rectangles at any angle.

Rectangle - Wall Collision

Since the wall is simplified into a segment, the collision between rectangle and the wall could be detected by the modified Cohen-Sutherland algorithm directly.

Rectangle - Circle Collision

The collision detection between a circle and a rectangle is also the problem of computing the distance from a point to the rectangle. Assuming the given rectangle is $R = \{x_1, y_1, w, h, \theta\}$, and the given circle is $X = \{x_2, y_2, r\}$, according to the figure 6.3, there is $A = (x_1, y_1)$, $B = (x_2, y_2)$, $dist = \sqrt{(y2 - y1)^2 + (x2 - x1)^2}$ and then $\phi = \arctan(\frac{y_2 - y_1}{x_2 - x_1})$, $\psi = \phi - \theta$.

Since the segment BC is orthogonal to the edge of the rectangle, the length of BC is

$$BC = dist \cdot cos(\psi) - l. \tag{6.3}$$

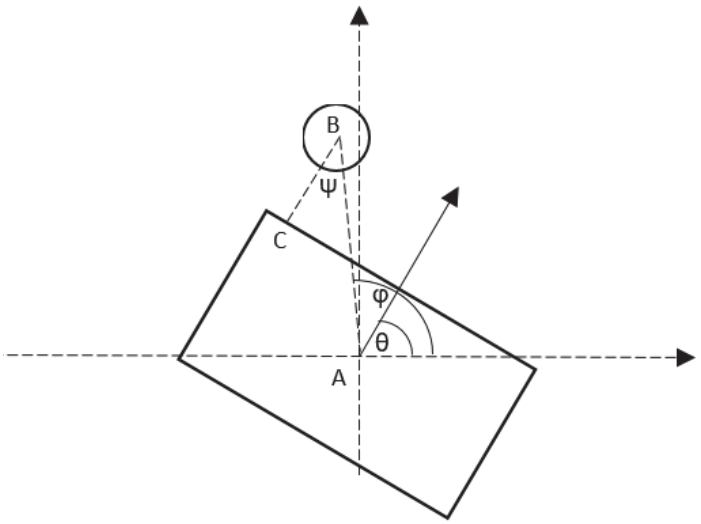


Figure 6.3: The distance between a circle and a rectangle

The rest of the work is to determine the value of l in the equation 6.3. Considering the figure 6.4, the diagonals of the rectangle divideds the space into four sections, and the angle $\rho = \arctan(\frac{w}{h})$. Then, comparing the value of ψ and ρ can figure out if $l = \frac{h}{2}$ or

$\frac{w}{2}$:

$$l = \begin{cases} \frac{h}{2} & \psi \in (\rho, \pi - \rho) \cup (\pi + \rho, 2\pi - \rho), \\ \frac{w}{2} & \psi \neq k\pi \pm \rho \ where \ k \in \mathbb{N}. \end{cases} \tag{6.4}$$

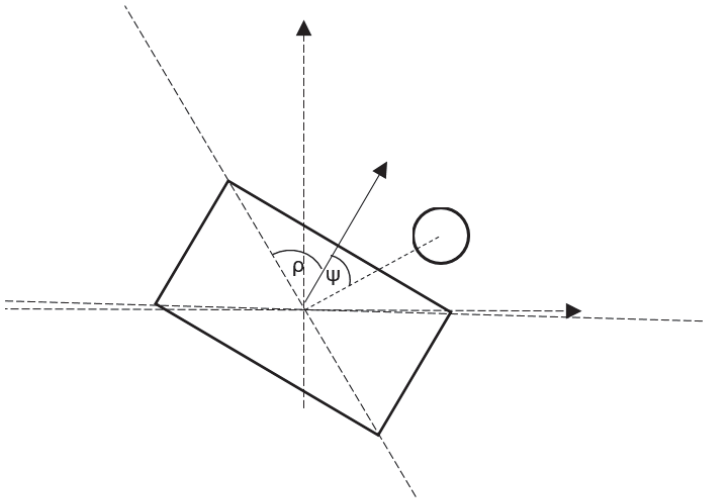


Figure 6.4: Determine which quadrant the circle belongs to

The last special case is $\psi = k\pi \pm \rho$, where the distance from point to rectangle is

$$d(P, R) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} - \frac{\sqrt{w^2 + h^2}}{2}. \tag{6.5}$$

Due to the radius of the circle, the final result is the distance minus the radius, and a collision happened if the result is equal or less than 0.

Circle - Wall Collision

Similar to the circle-rectangle collision detective, the circle-wall collision could be simplified to a point-segment distance computing. Assuming the given point is $P_0 = (x_0, y_0)$, and the end points of the segment is P_1, P_2 and $x_1 < x_2$, then there are

$$\begin{cases} a & = y_2 - y_1, \\ b & = x_1 - x_2, \\ c & = y_1(x_2 - x_1) + x_1(y_1 - y_2). \end{cases} \tag{6.6}$$

Then, the perpendicular foot P_f through P_0 is

$$P_f = \begin{cases} x_f & = \frac{b^2x - aby_0 - ac}{a^2+b^2}, \\ y_f & = \frac{a^2y - abx_0 - bc}{a^2+b^2}. \end{cases} \tag{6.7}$$

if $x_f < x_1$ or $x_f > x_2$, the distance from the point P_0 to the segment $\{P_1, P_2\}$ is the distance between the point P_0 and corresponding end point, otherwise it is the distance from the point P_0 to the perpendicular foot P_f.

Also, the radius should be subtracted from the distance for the final result.

Circle - Circle Collision

Circle-circle collision is the most simple case. Assuming the given circles are $C_1 = \{x_1, y_1, r_1\}$ and $C_2 = \{x_2, y_2, r_2\}$, simply compare the distance and the sum radius of the two circles. if $\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \leq r_1 + r_2$, then there is a collision.

Wall - Wall Collision

The wall-wall collision detection is to find out the intersection point between two segments. Assuming there are two segments $S_1 = \{P_1, P_2\}$ and $S_2 = \{P_3, P_4\}$, then similar to the previous section, the general format of the line could be calculated by the equation 6.6. Therefore, $L_1 = (a_1, b_1, c_1)$, $L_2 = (a_2, b_2, c_2)$.

Let $r_1 = a_1x_3 + b_1y_3 + c_1$, $r_2 = a_1x_4 + b1y_4 + c1$, which are the distances from point P_3 and P_4 to L_1, and $r_3 = a_2x_1 + b_2y_1 + c_2$, $r_4 = a_2x_2 + b2y_2 + c2$, then, the condition $r_1 \neq 0 \,\&\, r_2 \neq 0 \,\&\, r_1r_2 > 0$ can indicate that there is no intersection between these two segments. Also, $r_3 \neq 0 \,\&\, r_4 \neq 0 \,\&\, r_3r_4 > 0$ has the same conclusion.

The intersection point of two segments could be found by the following way: let $r = a_1b_2 - a_2b_1$, the two segments are co-interlinear if $r = 0$, otherwise

$$\begin{cases} x_i = \frac{|b_1c_2 - b_2c_1| + 0.5r}{r} \frac{b_1c_2 - b_2c_1}{|b_1c_2 - b_2c_1|}, \\ y_i = \frac{|a_2c_1 - a_1c_2| + 0.5r}{r} \frac{b_1c_2 - b_2c_1}{|b_1c_2 - b_2c_1|}. \end{cases} \tag{6.8}$$

6.1.2 The Force and the Impulse Direction

If the collision detection results true between two objects, a impulse must be applied to push the objects away from each other.

According to Newton's third law of motion, the forces have same magnitude and opposite direction, that is, there is only need of find out of one them, and the other could be calculated by simply inverse the direction of the force.

The fundamental idea of calculating the force is to figure out which object is the active one. When a collision happened, the direction is dependent on the active object but not the passive object. For instance, considering the case illustrated in the figure 6.5, and assuming the rectangle at left is the active one, so that the direction of the force on the right rectangle is same as the moving direction.

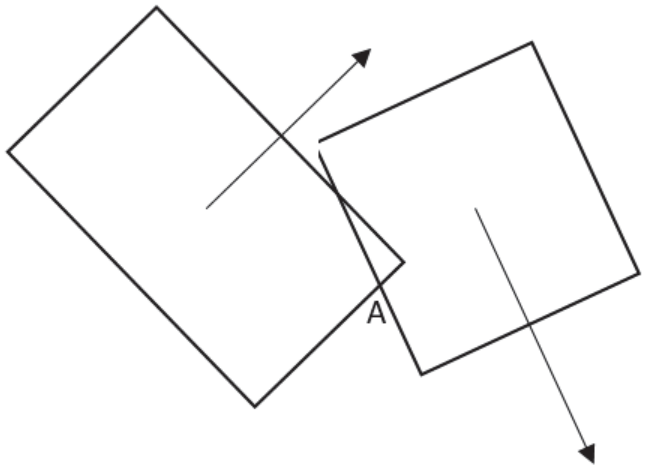


Figure 6.5: The left object hits the right one

Sometimes the two objects may hit each other at same time. It is also essential to figure out if the collision is one-way or not. In order to resolve that, the velocities of the objects could be divide into the x-component and y-component along the corresponding axis.

First, assuming the two given objects are $O_1 = \{x_1, y_1, vx_1, vy_1\}$ and $O_2 = \{x_2, y_2, vx_2, vy_2\}$, then comparing the direction of vx and vy separately. If any of them have different direction and are moving towards each other, then both of the objects are the active. Otherwise only the one which has higher magnitude of velocity components is the active one.

Similar to the collision detection, there are also difference cases of the force direction calculation. If both of the objects are active, the force direction is the net force from each.

Friction

The friction is also an important factor that affect the motion. Since the direction of friction is always opposite to the velocity, it could be considered as a constant reduction of the velocity magnitude.

Due to the discrete model, when friction is applied to an object, the magnitude of the velocity will be

$$|\vec{v}(t+1)| = max(|\vec{v}(t) + \frac{\vec{f}}{m}\Delta t|, 0). \tag{6.9}$$

Also, since both friction and velocity are vectors, the calculation of the friction could be divided into x and y components separately.

6.1.3 Applying the Artificial Potential Field

Other than the collision detection, the artificial potential field is another component that needs the mechanics simulation. According to the equation 2.4 on the page 29, when the object is affected by the artificial potential field, a virtual force will be applied to move the object, like a real force. For a source object which may generate an artificial potential field that affects target objects, there are three factors that are involved: the distance between the source and the target, the direction of the force, and the magnitude of the force.

The distance between objects could be found during the collision detection phase. The direction and the magnitude of the virtual force depend on the generator.

When collisions happen, the virtual force from the artificial potential field is meaningless because one of the major purposes of the artificial potential field is to avoid the obstacles. A non-zero collision force also indicates a failure of avoiding obstacles, so that an impulse of collision should be applied instead of the virtual force.

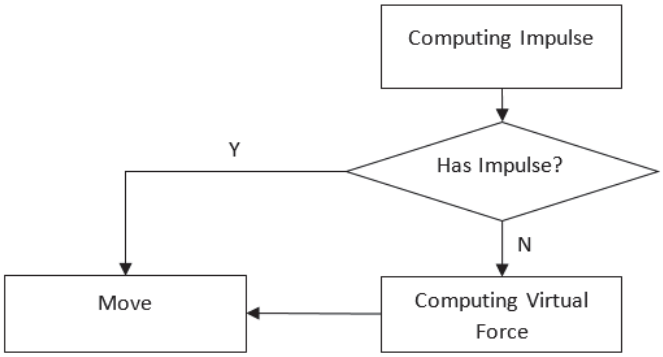


Figure 6.6: To determine if need to compute virtual force

6.2 Implementation Details of Optimal Search Algorithms

6.2.1 Grid mapping and Dynamic Cell Size

When using search algorithms in real problem, one of the most fundamental problem is to avoid the error of floating point computations. While comparing the position of current node and the goal, there might be a small error at the end of the decimal part.

Mapping an area to a grid-based search field can convert the comparisons between nodes from floating point to integer. However, due to the unknown size of searching area and the factor from the artificial potential field, the length of the cell's edges should have a proper range.

Assuming that the position of the start point and goal point are $P_s = \{x_1, y_1\}$, $P_g = \{x_2, y_2\}$ which is a rectangle, and the edge of the rectangle will be divided into at least one segments, then

$$g_i = 1 + [\frac{|i_2 - i_1|}{k}] \qquad where\ k \in \mathbb{N}^+. \tag{6.10}$$

The density of the cell in the search field is determined by the value of k. In other words, the size of the cell is dynamic. When the start position is far away from the goal, the accuracy of the path is less important than the computation speed. On the other hand, if the the start and the goal are close, the path with high accuracy can have a better navigation result.

After the mapping, the location of vertices on the search field is $L_s = (s_i, s_j)$, so that the start node will be marked as $(0, 0)$, and the goal node will be marked as (g_x, g_y).

As a result, the position of the vertex s, and distance between vertices s, p could be represented as:

$$\begin{cases} step_x & = \frac{x_2 - x_1}{g_x}, \\ step_y & = \frac{y_2 - y_1}{g_y}, \\ Position(s) & = \{x_1 + s_i \cdot step_x, y_1 + s_j \cdot step_y\}, \\ Distance(s,p) & = \sqrt{((s_i - p_i) \cdot step_x)^2 + ((s_j - p_j) \cdot step_y)^2}. \end{cases} \tag{6.11}$$

6.2.2 Use of the Artificial Potential Field

Since the position of the vertices vary, the magnitude from the artificial potential field should be computed in real-time. However, according to the equation 4.3 on page 61, the magnitude depends on how the generator implemented and the minimum distance to the nearest object. The computation have to access all the objects that may generate the artificial potential field.

In order to reduce the complexity, an extra data structure should be used to store the magnitude of known vertices. For instance, relationship collections like $Dictionary < Location(s), Magnitude(s) >$ could be a proper choice.

6.2.3 Multi-Linked List

In order to reduce the running time of the search algorithms, the multi-linked list could be used in the search algorithms which can reuse the existing data from the expanded nodes and generate the path directly.

During the search, when a new node s' is visited, the parent node s be determined immediately until the node s' is expanded. That is, in some of the cases, there might be multiple nodes that have the same successor s'.

6.3 Architecture Design of the Simulation Platform

In order to implement the discrete model of the mechanics, which is the basis of the simulation of robotics behaviors, a suitable architecture is essential, due to the requirement of real-time visualization and interaction. The architecture of computer games is one of the most appropriate.

Different from the event-driven architecture, the game architecture has an advantage that the low-delay of rendering. Furthermore, it involves the power of the graphics

cards that reduce the workload of CPU. The CPU can concentrate on the computing of the simulation jobs, while the visualization is carried out by the GPU.

6.3.1 Main-loop and FPS Limits

While implementing the discrete model of the mechanics simulation, the fundamental component is an infinite loop with accurate time stamps, which is also known as main-loop. In each iteration, the time difference is updated by the time stamp generator, and all the events from the previous iteration must be handled for the preparation. After that, all the objects will have their collision checked to determine the forces applied on each one, then they will be moved separately and drawn. The last step is to generate new events depend on the after effect of moving.

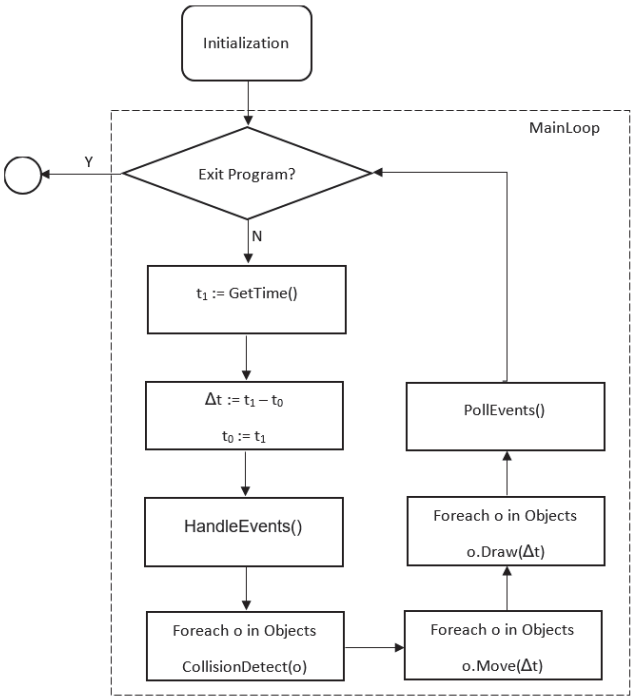


Figure 6.7: The main-loop of the simulation

The figure 6.7 illustrates the procedure and the key steps of the main-loop. Sometimes, a Sleep() might be called after the procedure PollEvents(), which aim is to reduce the workload of the system by limiting the refresh rate. Suppose that the refresh rate requirement of the simulation is 60fps, which means rendering a frame each 16ms. During the run-time if Δt is smaller than 16, then the call of Sleep(16-Δt) can restrict the time spend of each iteration.

6.3.2 Structure of the Rigid Body Management

The management of all the rigid body objects is done by using a manager instance, which is a singleton pattern that the instance is unique during run-time, in the simulator. All the rigid body instances must register themselves to the manager with a distinct ID. In c++, *this* pointer will be the best ID of the instance since it is unique. In other words, the manager instance maintains a collection of the pointers of base class whose name is RigidBodyBase, and all the rigid body object classes must inherit the base class to implement the pure virtual methods declared in the base class.

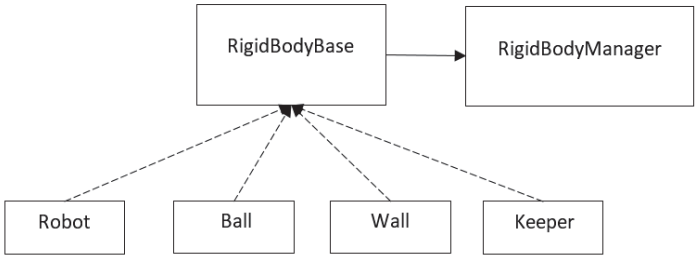


Figure 6.8: The class diagram of rigid body management

The figure 6.8 shows an example of a diagram of the rigid body manager. Then, in order to implement the singleton instance, the RigidBodyManager class need a private constructor with a public static method to process the access and instantiate work. The following code is a sample of implementing a static singleton pattern, which will instantiate during the start phase at run-time.

```
1  class RigidBodyManager{
2  public:
3          static RigidBodyManager& getInstance(){
4                  return RigidBodyManager::instance;
5          }
6  private:
7          static RigidBodyManager instance;
8          explicit RigidBodyManager();
9          RigidBodyManager(RigidBodyManager&) = delete;
10 };
11
12 RigidBodyManager RigidBodyManager::instance;
```

Sometimes, for the purpose of re-initializing and save memory, dynamic singleton is also used. The instantiation of the dynamic singleton will be processed at the first call

during the run-time.

```
class RigidBodyManager{
public:
        static RigidBodyManager* getInstance(){
          if(RigidBodyManager::instance == nullptr){
                RigidBodyManager::instance =
                        new RigidBodyManager();
          }
          return RigidBodyManager::instance;
        }
private:
        static RigidBodyManager* instance;
        explicit RigidBodyManager();
        RigidBodyManager(RigidBodyManager&) = delete;
};

RigidBodyManager* RigidBodyManager::instance = nullptr;
```

Since *this* pointer of all the rigid body instance is the preferred ID of each object, the collection that manages the rigid bodies could be a set, which has the best balance of speed and space.

By adding this pointer to the manager class within the constructor of the base class RigidBody, all the instances will be managed by the manager, and there is no need to delete outside. The following code demonstrates a sample of use manager for auto disposing:

```
class RigidBody{
public:
        RigidBody();
        virtual ~RigidBody(){}
        virtual void move(double)=0;
};
class Ball:public RigidBody{
public:
        Ball():RigidBody(){     }
        virtual ~Ball(){    }
        virtual void move(double dt){   }
};
```

```
class RigidBodyManager{
```

```
2         friend class RigidBody;
3 public:
4         static RigidBodyManager* getInstance(){
5                 if(!RigidBodyManager::instance){
6                         RigidBodyManager::instance =
7                                 new RigidBodyManager();
8                 }
9                 return RigidBodyManager::instance;
10        }
11        static void releaseInstance(){
12                delete RigidBodyManager::instance;
13                RigidBodyManager::instance = nullptr;
14        }
15 private:
16        static RigidBodyManager* instance;
17        explicit RigidBodyManager(){
18
19        }
20        RigidBodyManager(RigidBodyManager&) = delete;
21        set<RigidBody*> rigids;
22        void addRigidToCollection(RigidBody* r){
23                this->rigids.insert(r);
24        }
25
26
27
28        void clearAll(){
29                for(auto p : rigids){
30                        if(p != nullptr){
31                                delete p;
32                        }
33                }
34        }
35        ~RigidBodyManager(){
36                clearAll();
37        }
38 };
39
40 RigidBodyManager* RigidBodyManager::instance = nullptr;
41
42 RigidBody::RigidBody(){
```

```
43          RigidBodyManager::getInstance()
44              ->addRigidToCollection(this);
45  }
```

Due to the undefined behavior of incomplete type while calling the destructor of the derived class from a pointer of the base class, the order of the code is important. The definition of RigidBodyManager::clear() must be placed after the declaration of the destructors to indicate to the compiler that there is an inherited relationship. Otherwise, undefined behavior of the compiler may cause an unexpected result during run-time.

6.3.3 Messages and Event Handling

The message module is used to communicate between the other modules. A global singleton message manager class is used to handle the messages. In the practice, this module is provided by the framework GFLW.

The module registered themselves to the message controller with a callback function as the event handler. During the procedure of PollEvents() illustrated in the figure 6.7, the message manager traversal all the collected message during the iteration, and call the registered event handlers to process the message.

In the message manager, there are two collections. First is the EventList which collects all the events generated during the iteration, the other is the ListenerManager which type is $Dictionary < EventType, Dictionary < ID, Listener >>$. When a new module register the listener to the message manager, it must provide the ID, the type of event to listen and the listener call back. Then, the manager will check if it is has been registered to determine adding the new listener or replacing the existed one.

Figure 6.9 illustrates the entire procedure of PollEvents(). Generally, the event handler will not return a value of *true* to finalize the dispatching of the event. However, in some of the special case, it still has the possibility to make the event exclusive.

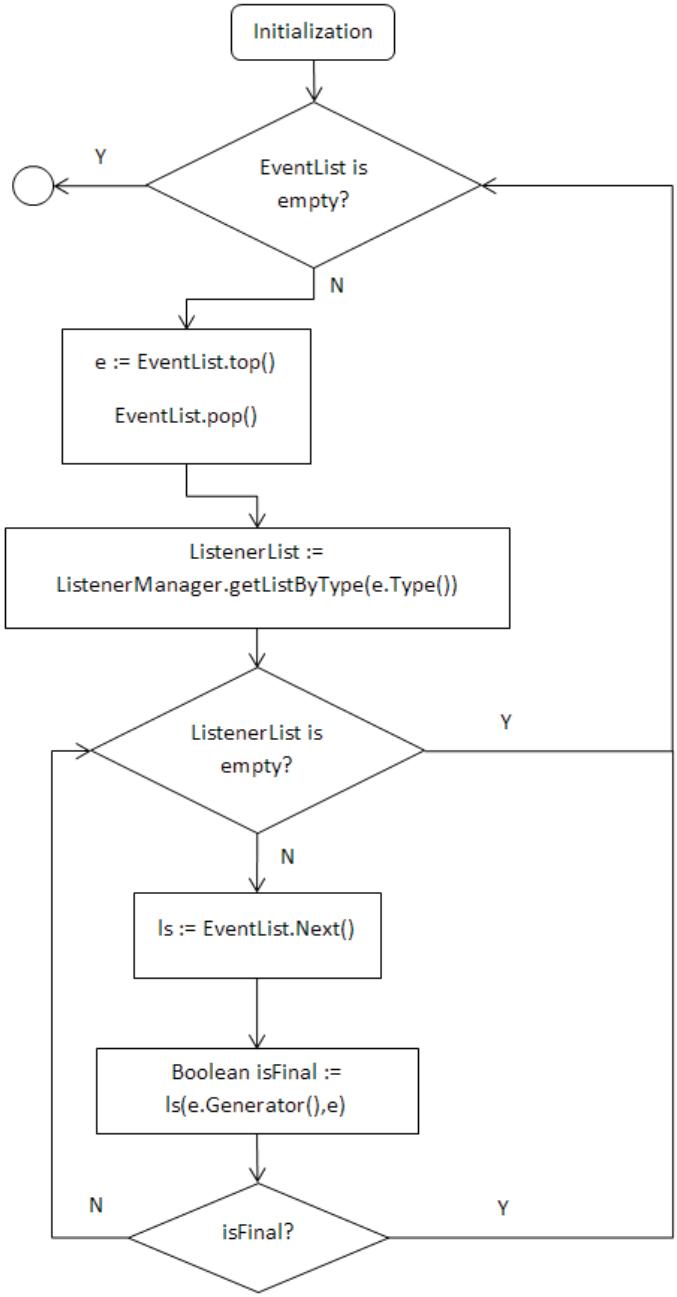


Figure 6.9: The procedure of poll event

Chapter 7

Experiments

7.1 Experiment of the Optimal Searches with the Artifical Potential Field

7.1.1 Platform of the Search Algorithm Experiment

The platform was developed using Visual C# with .Net Framework 4.5.2, running on Windows 7 sp1 64bits. All the result are from the computer with Intel i7 6700k, 32G Memory, single Thread without Hyper-Threading.

There are various cases selected to explore the performance of the introduced algorithms in the previous chapters. Although the robot soccer game is a major problem domain of this research, some of the cases are extended beyond the realm of the robot soccer game to discover the potency of the algorithms in the area of real world navigation problems.

7.1.2 Definitions and the Statistics

Overall, the statistics of the experiments is concentrated the runtime, the path length and the safety of the path found by the different algorithms. The previous two are same as that in the preliminary experiments which indicates the general performance of the search algorithms, and the third one is to examine if the path has a proper distance from the obstacles, so that when the agent is moving along the path, the possibility of hitting the obstacle will be minimized.

7.1.3 Safety Factor of the Potential Field

According to the equation 2.4, 4.1, 4.3 and 4.4, there will be different performance if using the safety factor (SF) or not. For the search algorithm, while the safety factor (SF) is involved, the research should have a tendency where the way-points are keeping away from the obstacles, and more close to the cells have lower magnitude.

In the experiment, there will be the comparison between the result of using the safety factor or not.

Generator of the Artificial Potential Field

The repulsive force generator of the artificial potential field is: assuming the minimum distance from the point s to the nearest obstacle is $dmin(s)$, the threshold of the max radius is r, then the magnitude $m(s)$ of the artificial potential field at the point s

$$m(s) = \begin{cases} +\infty & dmin(s) < 0, \\ \frac{k}{1+e^{dmin(s)-\frac{r}{2}}} & , \\ 0 & dmin(s) > r. \end{cases} \tag{7.1}$$

The value of the constant k and r are depend on the size of the field. The variant of the value will affect the performance of the artificial potential field.

7.2 Experiments of Search Algorithms with the Artificial Potential Field

In the experiment of the optimal search algorithms with the artificial potential field (APF-Optimal Search), the cases from the preliminary experiments are reused. By using a different algorithm, the performances varies as well. In order to simplify the presentation, the search algorithms used in the following experiments without specification are referred to the APF-Optimal Search, such as the APF-A* Search and the APF-Theta* Search.

The algorithm selected for the line-of-check in the Theta* Search is the modified Bresenham's Line Algorithm.

Similar to the legends in the preliminary experiments, the black cells are the blocked that cannot be visited. The white cells are accessible, green is the start node and cyan is the goal node. The yellow cells are visited at least once during the search, orange

cells are those in the expanded list, and the red cells are the way-points. In order to demonstrate the path found by the Theta* Search more intuitive. The blue segments are used in the case of the Theta* Search to identify the actual path.

7.2.1 Comparison Between the APF-A* and the Original A*

Different from the original Optimal Search, the APF-Optimal Search has an extra factor in the fcost which is the Safety Factor (SF). According to the equation 4.3 and 4.4, the SF will change the result of the search algorithms if and only if there is any obstacle near the path found by the orignial optimal search algorithms.

Figures 7.1 and 7.2 show the most simple case that there is no different between the original A* Search and the APF-A* Search since there is no obstacle nearby. However, the result changed a lot even when there is only one obstacle place in the middle, as it is shown in the figure 7.3 and 7.4: the inserted obstacle in the middle changes the distribution of the magnitude, illustrated in the figure 7.5. Then, the path found by the APF-A* Search keeps a proper ditance away from the obstacle to ensure the safety of the path.

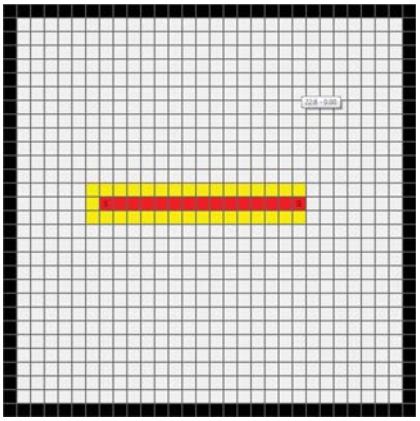


Figure 7.1: The original A* Search

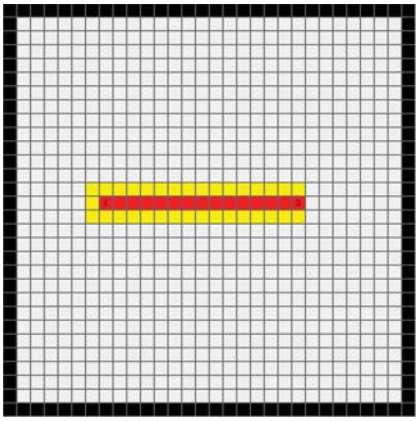


Figure 7.2: The APF-A* Search

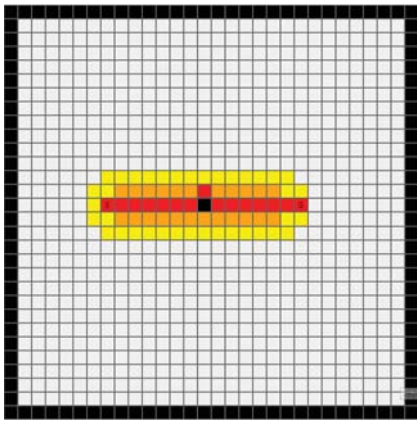


Figure 7.3: The original A* Search with one dot in the middle

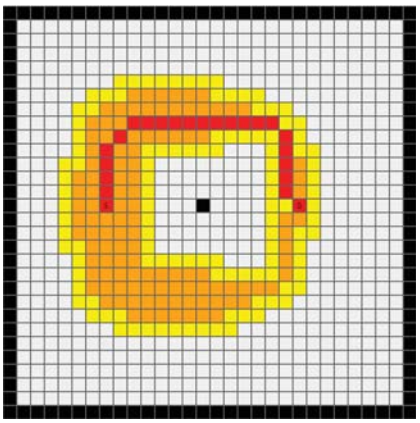


Figure 7.4: The APF-A* Search with one dot in the middle

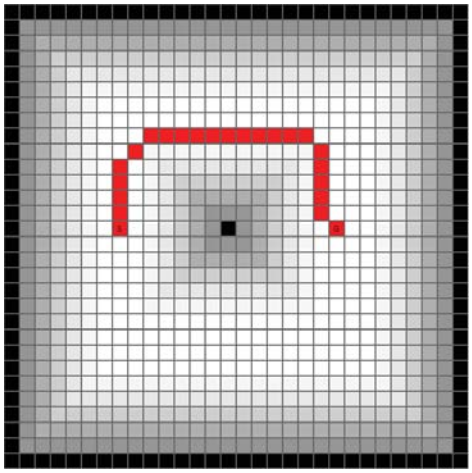


Figure 7.5: The magnitude distribution when the obstalce is placed in the middle

This comparison shows the efficacy of the APF-Optimal Search with SF. The next experiments will explore the performance of both the APF-A* Search and the APF-Theta* Search in different scenarios, sometimes using SF.

7.2.2 One Big Obstacle in the Centre, Size 20x20

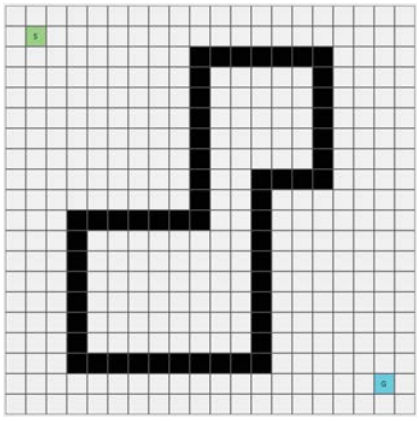


Figure 7.6: A big obstacle in the centre of the field

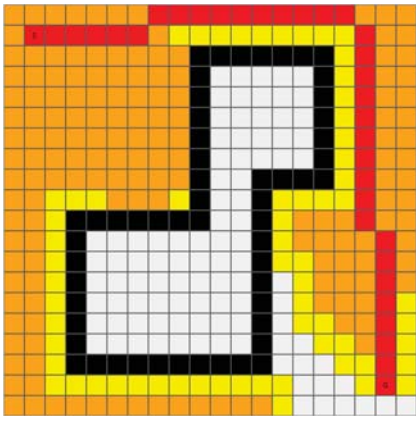


Figure 7.7: Path found by APF-A* Search without SF

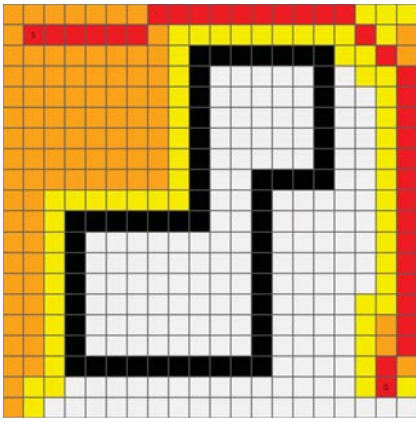


Figure 7.8: Path found by APF-A* Search with SF

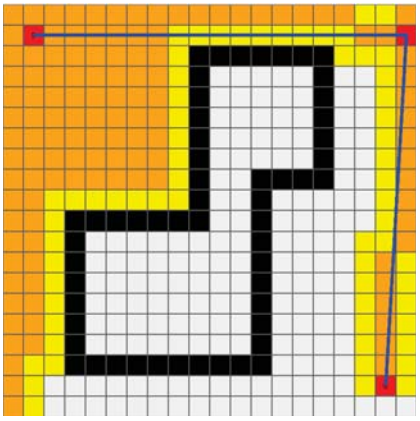


Figure 7.9: Path found by APF-Theta* Search without SF

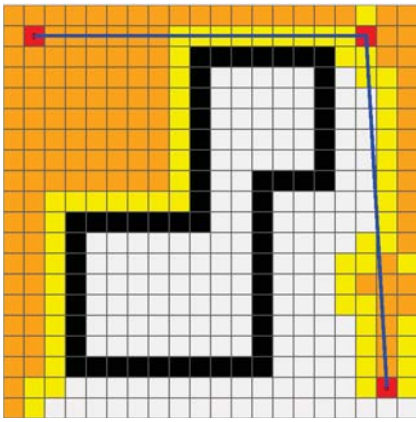


Figure 7.10: Path found by APF-Theta* Search with SF

Table 7.1: Statistics of the Performances of APF-Optimal Searches with or without SF, in the case of Big Obstacle

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	2.01046	±0.01822	721	186	34.243
APF-A* SF	1.21830	±0.00756	492	123	35.071
APF-Theta*	1.25210	± 0.00998	494	120	35.029
APF-Theta* SF	1.27460	±0.00801	510	126	33.029

Since there is no wall around the field, both of the results from the APF-A* Search under different conditions find out the path to the goal while away from the obstacle. The one using SF has a better runtime, while the other one has a better path length.

Both the results of the APF-Theta* Search reflect the concern discussed in the previous chapters: the second way-point at top-right corner indicates that the algorithm has the tendency of keep away from the obstacle. However, due to the line-of-sight check, the start node which has a visual to the second one, purges all the intermediate nodes on the path, so that the first part of the path has is much closer to the obstacle than the APF-A* Search.

The time performance of the APF-Theta* Search is still better than the APF-A* Search without SF. Nevertheless, comparing with the APF-A* search with SF, the APF-Theta* Search has no advantage of the runtime.

7.2.3 Four Medium Obstacles, Size 20x20

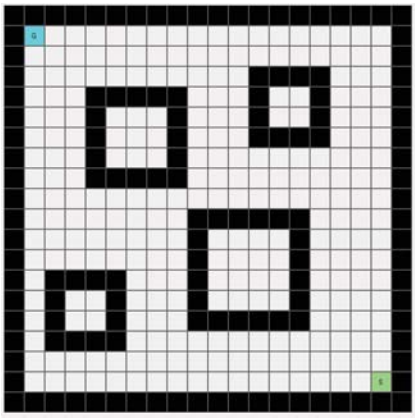


Figure 7.11: Four medium obstaces

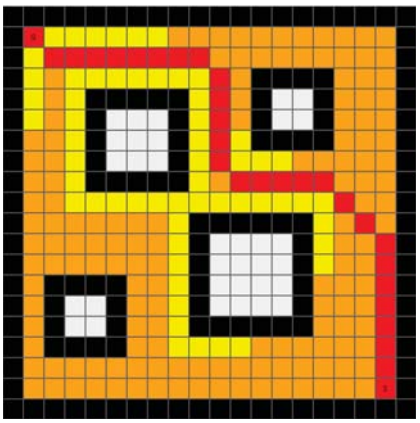


Figure 7.12: Path found by APF-A* Search without SF

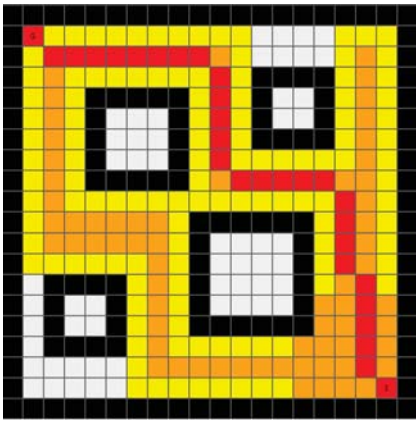


Figure 7.13: Path found by APF-A* Search with SF

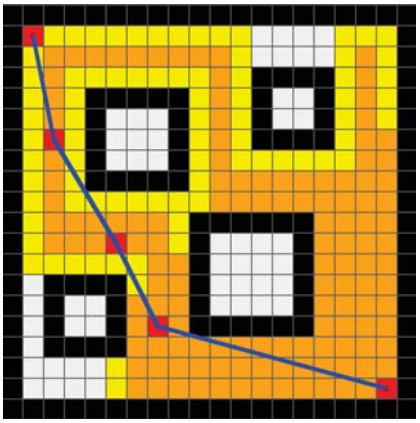


Figure 7.14: Path found by APF-Theta* Search without SF

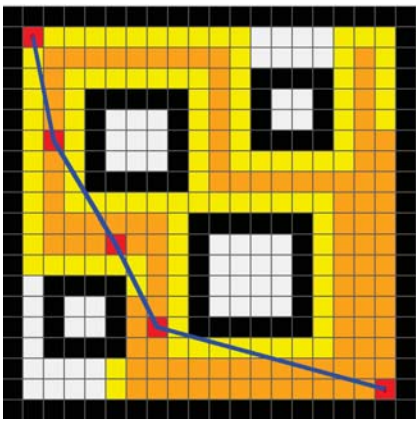


Figure 7.15: Path found by APF-Theta* Search with SF

Table 7.2: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Four Medium Obstalces

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	2.00167	±0.02283	646	174	30.485
APF-A* SF	1.43622	±0.01922	520	93	30.485
APF-Theta*	1.72943	±0.02305	569	133	26.804
APF-Theta* SF	1.51287	±0.00977	540	109	26.804

The path found by the APF-A* Search under different condition are different, shown in the figures 7.12 and 7.13. The APF-A* Search with SF has an impressive result which avoids all the obstacles and has the shortest running time.

The performances of the APF-Theta* Search in this case are still in the middle. Although the way-points also have a proper distance away from the obstacles, similar to the previous case, the APF-Theta* Search cannot guarantee the entire path has a safe distance to the obstacle. The SF does not affect the results of the APF-Theta* Search, but increase the speed by reducing visited and expanded count.

7.2.4 Five Small Obstacles, Size 20x20

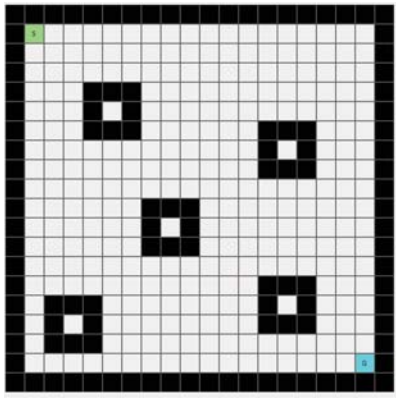


Figure 7.16: Five small obstacles

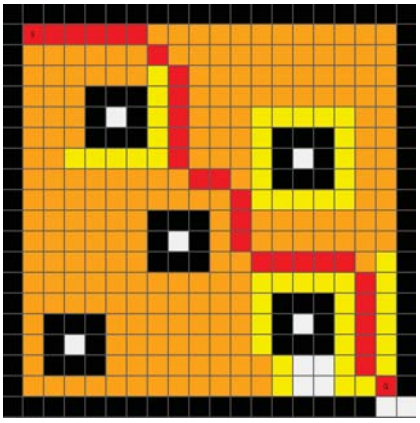


Figure 7.17: Path found by APF-A* Search without operator factor

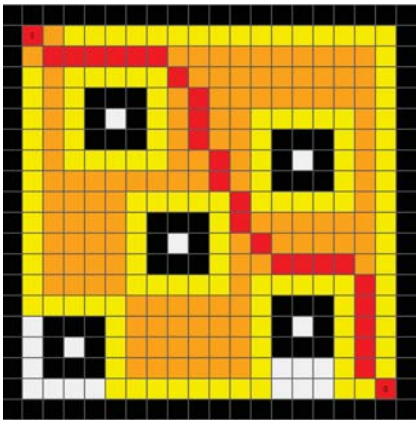


Figure 7.18: Path found by APF-A* Search with operator factor

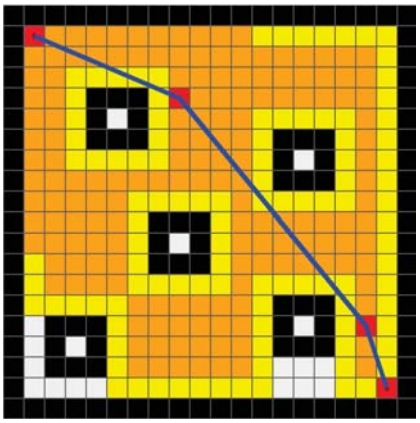


Figure 7.19: Path found by APF-Theta* Search without operator factor

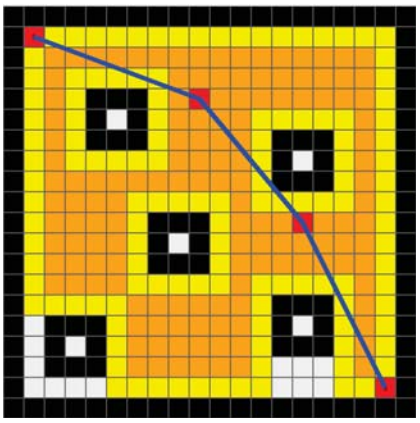


Figure 7.20: Path found by APF-Theta* Search with operator factor

Table 7.3: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Five Small Obstalces

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	2.62274	±0.01029	861	228	29.899
APF-A* SF	2.20710	±0.00965	755	143	28.728
APF-Theta*	2.25890	±0.00767	773	162	24.991
APF-Theta* SF	2.14002	±0.01942	752	141	25.299

In this case, the SF reduces the visited count and expanded count that increases the speed. Also, with the SF, the APF-A* Search finds out the safest path as well. The second way-point selected by the APF-Theta* Search with SF also has a higher distance to the obstacle than the one without SF, as a result, the path found by the APF-Theta* Search under different conditions are different.

7.2.5 Random Dots, Size 30x30

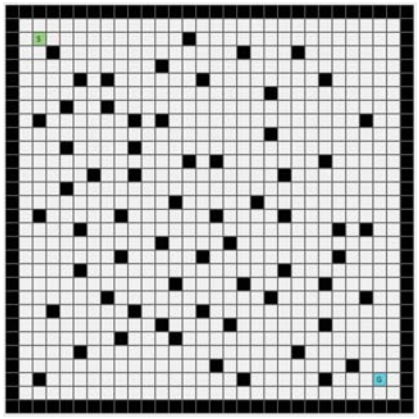


Figure 7.21: A set of dots placed randomly

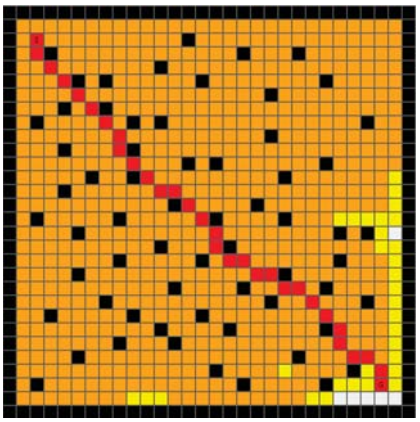


Figure 7.22: Path found by APF-A* Search without SF

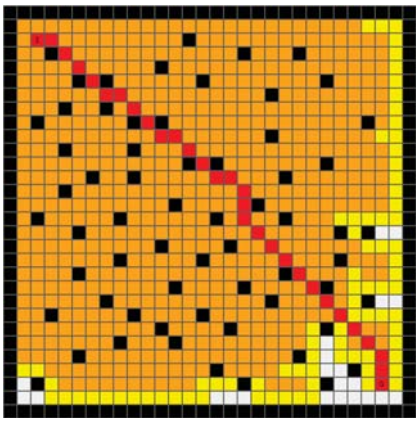


Figure 7.23: Path found by APF-A* Search with SF

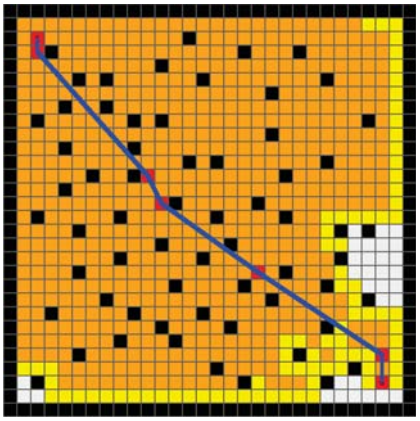


Figure 7.24: Path found by APF-Theta* Search without SF

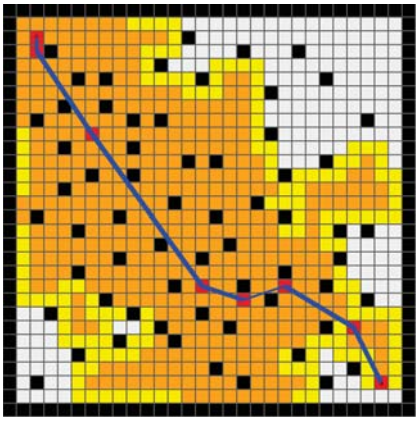


Figure 7.25: Path found by APF-Theta* Search with SF

Table 7.4: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Random Dots

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	8.76389	±0.03597	2408	684	38.284
APF-A* SF	8.32279	±0.03205	2277	618	37.698
APF-Theta*	7.97601	±0.02710	2248	609	36.697
APF-Theta* SF	5.57725	±0.02136	1635	413	38.440

In this case, the APF-A* Search without SF fails to find out the safest path and has the worst runtime. The APF-A* Search with SF partly finds out a safer path at the last sections, but is still not satisfied.

On the other hand, the performance of the APF-Theta* Search with SF is distinct, most of the way points have a proper distance away from the obstacles, and the average runtime is much smaller than the other three.

7.2.6 Four Walls with a Wiggled Lane, Size 20x20

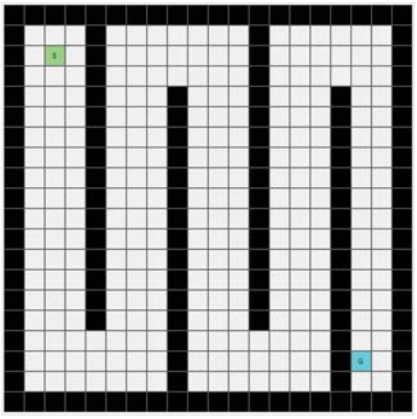


Figure 7.26: Four walls and a wiggled path

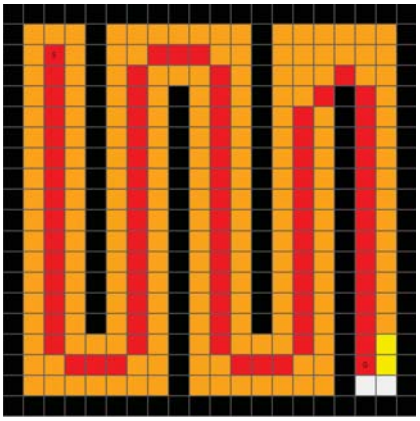


Figure 7.27: Path found by APF-A* Search without SF

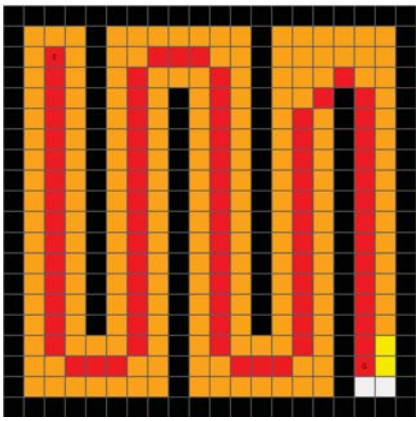


Figure 7.28: Path found by APF-A* Search with SF

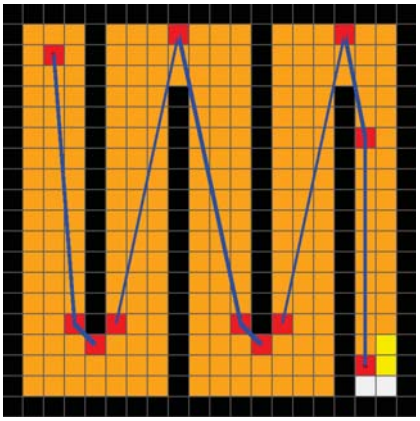


Figure 7.29: Path found by APF-Theta* Search without SF

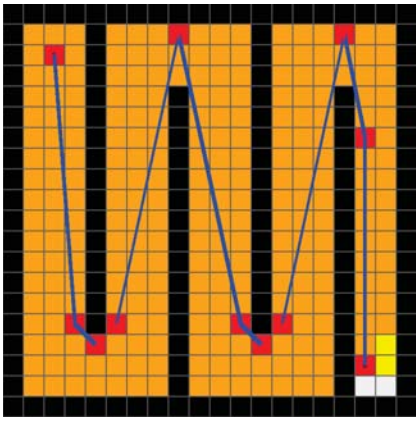


Figure 7.30: Path found by APF-Theta* Search with SF

Table 7.5: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Four Walls with a Wiggled Lane

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	3.38960	±0.01350	771	260	82.728
APF-A* SF	3.37563	±0.01297	771	260	82.728
APF-Theta*	2.72877	±0.01247	771	260	77.748
APF-Theta* SF	2.73419	±0.01088	771	260	77.748

The APF-A* Search in this case find out the safest path, while the performance of the APF-Theta* Search is not satisfactory. Although the average speed of the Theta* Search is still shorter than the APF-A* Search, the way-points are too close to the obstacles. Also, the SF does not affect the final results.

7.2.7 A Series of Walls with a Lane in the Middle, Size 30x30

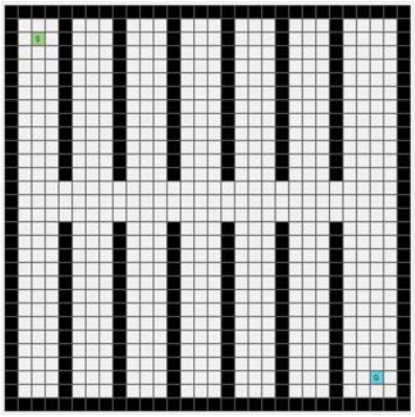


Figure 7.31: A series of walls with a lane in the middle

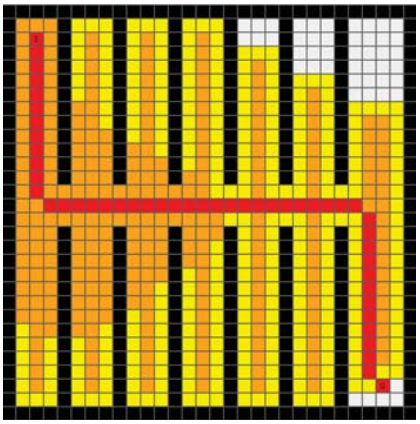


Figure 7.32: Path found by APF-A* Search without SF

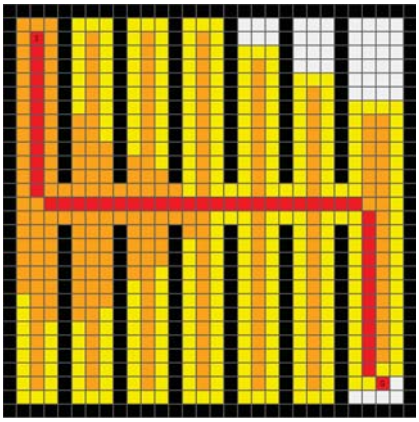


Figure 7.33: Path found by APF-A* Search with SF

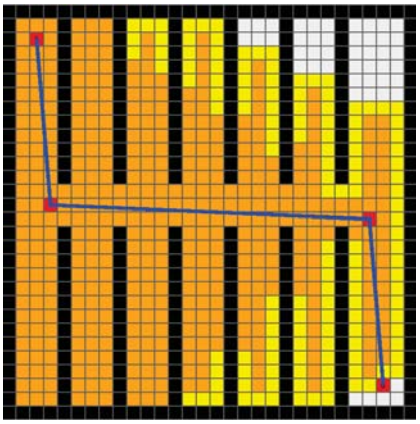


Figure 7.34: Path found by APF-Theta* Search without SF

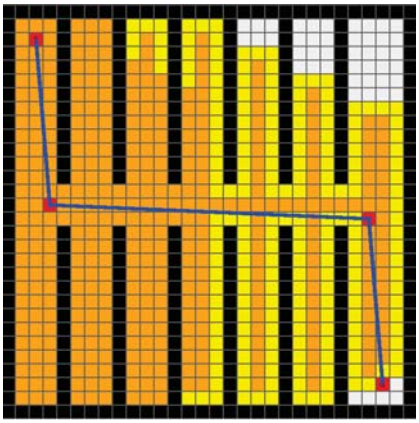


Figure 7.35: Path found by APF-Theta* Search with SF

Table 7.6: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Walls with a Lane in the Middle

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	5.09582	±0.03651	1490	313	48.243
APF-A* SF	4.90887	±0.02578	1470	295	48.243
APF-Theta*	5.72170	±0.02347	1649	454	47.105
APF-Theta* SF	5.14074	±0.02812	1574	394	47.105

The performance of the APF-A* Search in this case is better than the APF-Theta* Search. Also, the paths find by both the algorithms have the perfect balance between the length and the safety.

7.2.8 Maze, Size 30x30

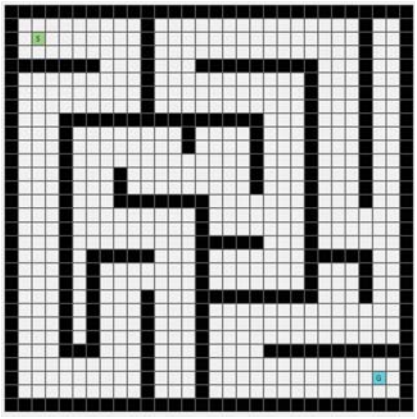


Figure 7.36: The maze which is the most complex terrian

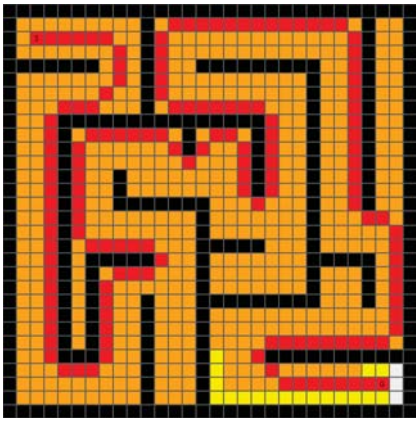


Figure 7.37: Path found by APF-A* Search without SF

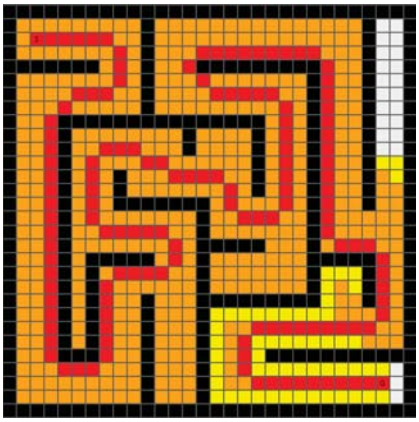


Figure 7.38: Path found by APF-A* Search with SF

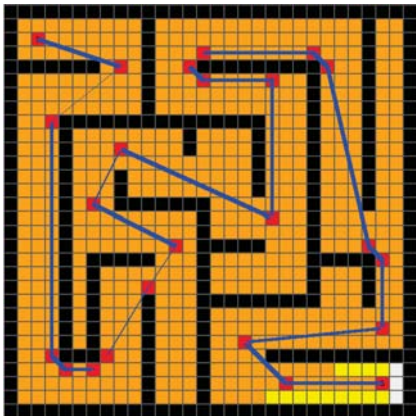


Figure 7.39: Path found by APF-Theta* Search without SF

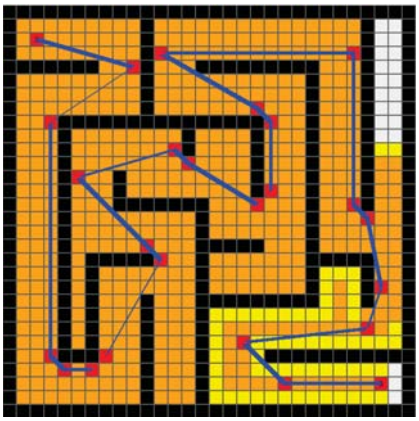


Figure 7.40: Path found by APF-Theta* Search with SF

Table 7.7: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Maze

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	56.7973	±0.64618	1825	599	155.598
APF-A* SF	56.3173	±0.67461	1732	542	146.770
APF-Theta*	33.3802	±2.23005	1831	604	135.547
APF-Theta* SF	36.822	±2.25468	17362	539	138.257

As for the most complex terrain, the performance of the APF-A* Search without SF in this case is bad. Most of the way-points are all close to the obstacles except the first and the last sections. With the SF, the performance of the APF-A* Search is much better, although there are still several sections that are too close to the walls.

For the APF-Theta* Search, some of the way-points are not well placed. Also, an abnormal result here is the average runtime of the APF-Theta* Search, which is shorter than the APF-A* Search, but the expand count of APF-the Theta* Search is bigger.

The SF also takes advantage of reducing the number of expanded nodes.

7.2.9 Potential Well, Size 30x30

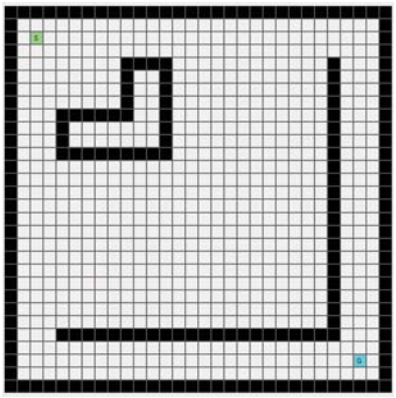


Figure 7.41: The potential well that the agent will be trapped in the middle

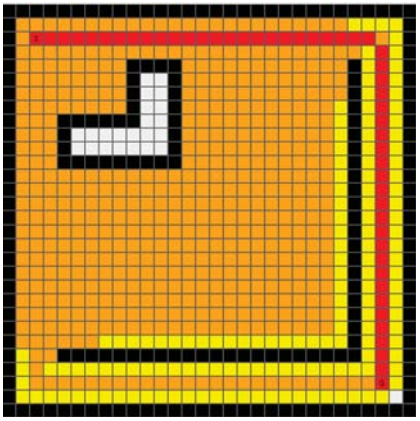


Figure 7.42: Path found by APF-A* Search without SF

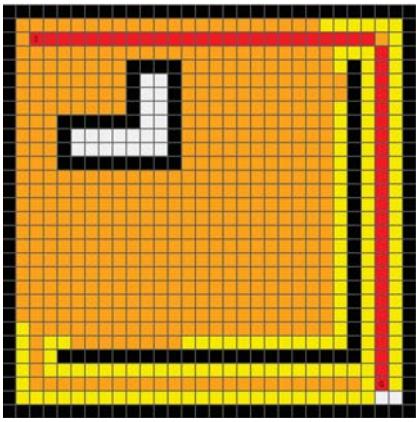


Figure 7.43: Path found by APF-A* Search with SF

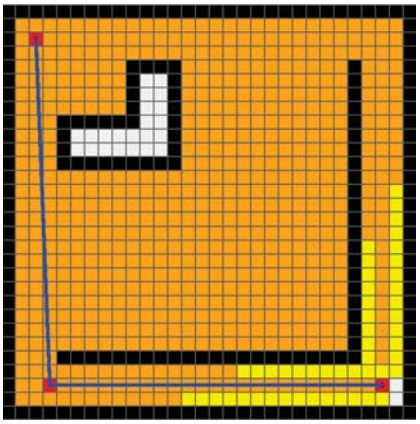


Figure 7.44: Path found by APF-Theta* Search without SF

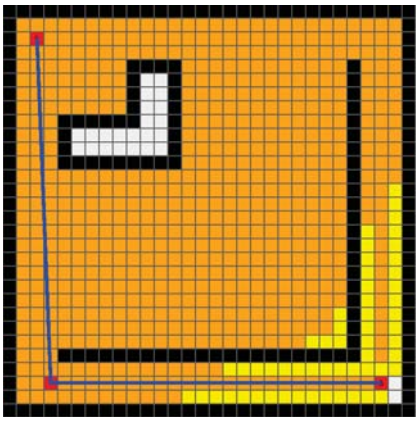


Figure 7.45: Path found by APF-Theta* Search with SF

Table 7.8: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Potential Field

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	8.23637	±0.04975	2263	546	49.414
APF-A* SF	8.24693	±0.03812	2249	541	49.414
APF-Theta*	8.17853	±0.03765	2352	638	49.020
APF-Theta* SF	8.01983	±0.03364	2345	631	49.020

The performances of all the algorithms are almost the same.

7.2.10 Trap, Size 30x30

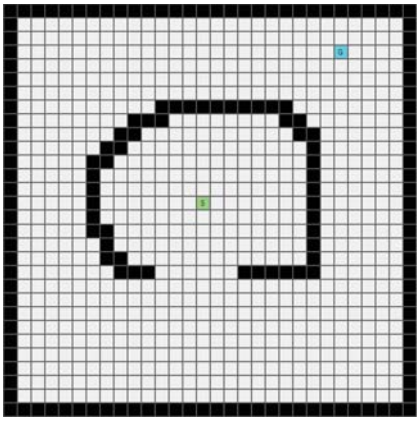


Figure 7.46: The trap where the goal is at the opposite direction of the exit

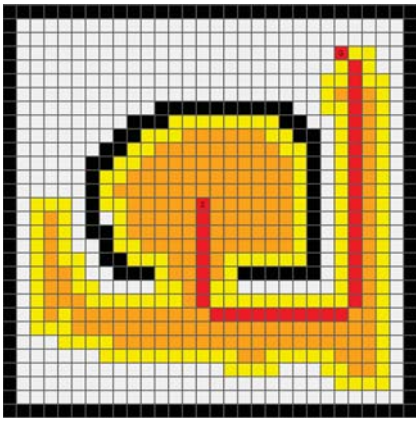


Figure 7.47: Path found by APF-A* Search without SF

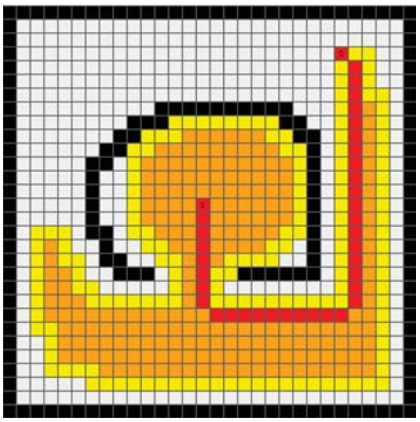


Figure 7.48: Path found by APF-A* Search with SF

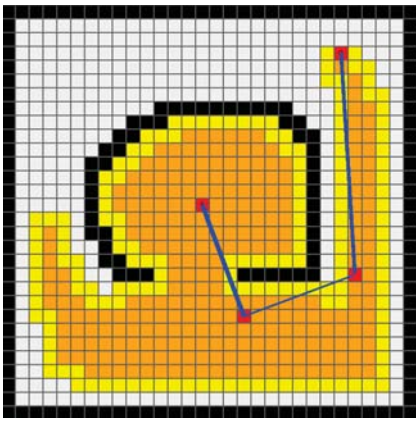


Figure 7.49: Path found by APF-Theta* Search without SF

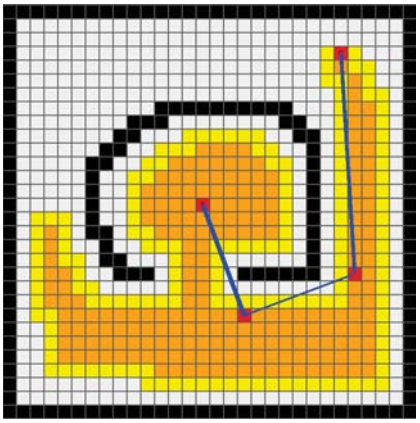


Figure 7.50: Path found by APF-Theta* Search with SF

Table 7.9: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Trap

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A*	3.39526	±0.01358	1121	227	37.243
APF-A* SF	3.65828	±0.03069	1200	250	37.243
APF-Theta*	4.08862	±0.01970	1326	281	33.119
APF-Theta* SF	3.22848	±0.01140	1080	220	33.119

In this case, when the algorithm consider the SF, the cells inside the trap are not totally visited, but the cells outside the trap have much bigger number of visited nodes. That reflects the purpose of the SF that giving lower gcost for the nodes have similar distance but lower magnitude, which makes the search result has a tendency of staying away from the obstacles.

The APF-Theta* Search with SF has the best result in this case.

7.2.11 GNRON Case

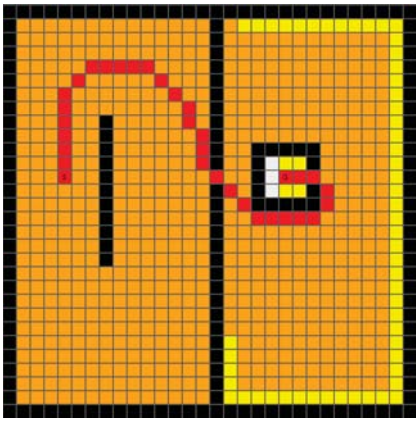


Figure 7.51: Path found by APF-A* Search with SF

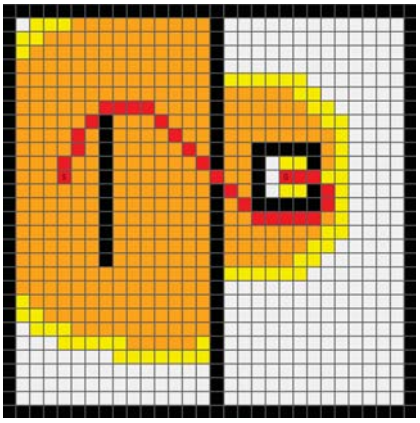


Figure 7.52: Path found by Original A* Search

This case is to illustrate the limitation of the SF in the case of GNRON: there is wall in the middle of the field, the size of the only gate is 1 cell, which means all the paths found by any search algorithm must have an intermediate node at that cell. Also, the procedure of the search could be divided into two part: from start node to the gate node, and from the gate node to the goal node.

Furthermore, another gate is place right before the goal node, which size is also 1, where is the GNRON problem of the Artificial Potential Field.

The next figures illustrate the result of the original A* Search and the APF-A* Search with SF, and the distribution of the magnitude.

Comparing with figures 7.51 and 7.52, it is clear that the APF-A* with SF visited all the cells on the field, due to the high magnitude at the gate, they have higher magnitude which causes higher fcost that also lower priority. As a result, the search algorithm visited most of the nodes first, and then went back to area close to the goal.

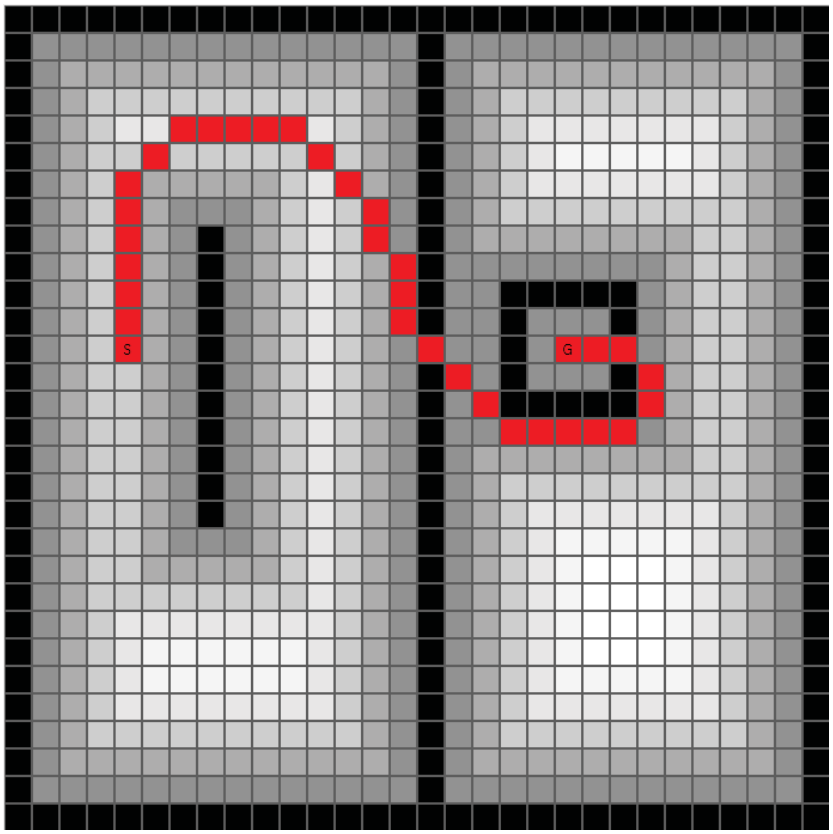


Figure 7.53: Magnitude distribution

7.2.12 Summary

The experiments showed the performance of different optimal search algorithms in various environments. In general, the optimal searches with the artificial potential field illustrate the ability of finding the proper path that balance the safety and the distance.

Although the line-of-sight will consume extra processing time, the Modified Bresenham's Line Algorithm doesn't reduce the performance of the Theta* Search, where the overall performance of the APF-Theta* Search on runtime is better than the APF-A* Search for the lower count of the visited and the expanded nodes. Also, the strength of the APF-Theta* Search in the area of finding the shortest path is exposed as well.

The performance of the safety factor (SF) also improves the algorithms. In most cases, there is a reduction of the runtime when SF is involved.

The limitation of the algorithm is also explored in the special case: Although the GNRON problem is resolved by the optimal search, with the potential field having GNRON problem, the node around the goal will have a lower priority due to the high magnitude, that causes the search algorithm to visit most of the other nodes first reducing the speed.

Moreover, the limitation of the APF-Theta* Search is exposed as well. Although the way-points are having a proper distance away from the obstacles, the APF-Theta* Search cannot guarantee the the path segment between adjacent way-points is far away from the obstacle.

7.3 Experiment of the Different Performances of the Artificial Potential Field Generator

In the previous section, the efficacy of the artificial potential have been illustrated. The following experiments are explored the performance of the different generators. The case of the potential well will be used.

Same as the previous one, the radius r is 5, and the magnitude at the point s is marked as $m(s)$, the minimum distance from the node s to the nearest obstacle is $dmin(s)$.

The double threshold is applied as well. For all the test cases, there are $t_1 = 1$ and $t_2 = r$, so that if $dmin(s) < 1$ then $m(s) = \infty$, and if $dmin(s) > r$ then $m(s) = 0$

7.3.1 The Distribution of the Magnitude Using Different Generators

The following experiments shows the distribution of the magnitude under the generator of the linear function, the hyperbola function and the sigmoid function. The gray scale illustrates the magnitude, where the darker cells have lower value.

Linear Functions

The generator selected for the case of linear function and the demonstration of the magnitude distribution is:

$$m(s) = 2r \cdot (-dmin(s) + r). \tag{7.2}$$

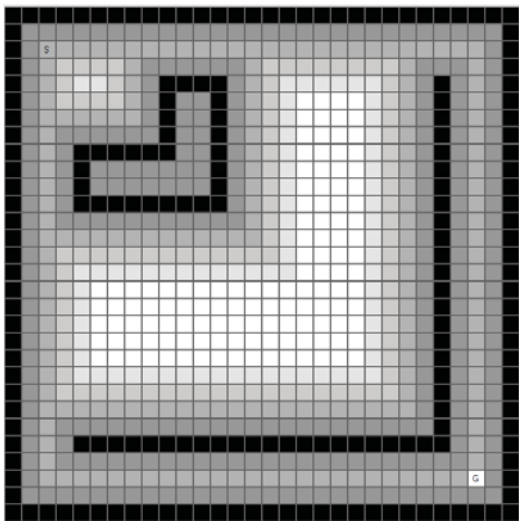


Figure 7.54: Distribution of the linear function

Hyperbola Functions

The generator selected for the case of hyperbola function to the power of -2 and the demonstration of the magnitude distribution is:

$$m(s) = 2r \cdot dmin(s)^{-2}. \tag{7.3}$$

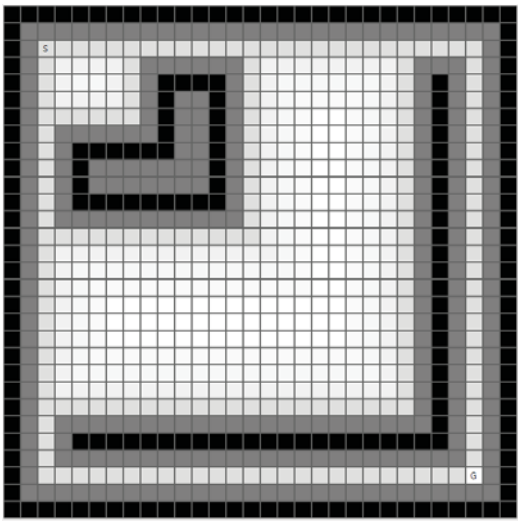


Figure 7.55: Distribution of the hyperbola function to the power of -2

The generator selected for the case of hyperbola function with the power of -1 and the demonstration of the magnitude distribution is

$$m(s) = 2r \cdot dmin(s)^{-1}. \tag{7.4}$$

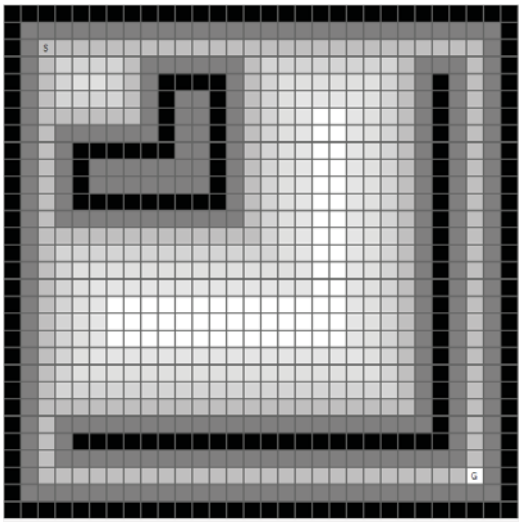


Figure 7.56: Distribution of the hyperbola function to the power of -1

Sigmoid Functions

The generator selected for the case of sigmoid function and the demonstration of the magnitude distribution is:

$$m(s) = \frac{10r}{1 + e^{dmin(s) - 0.5r}}. \tag{7.5}$$

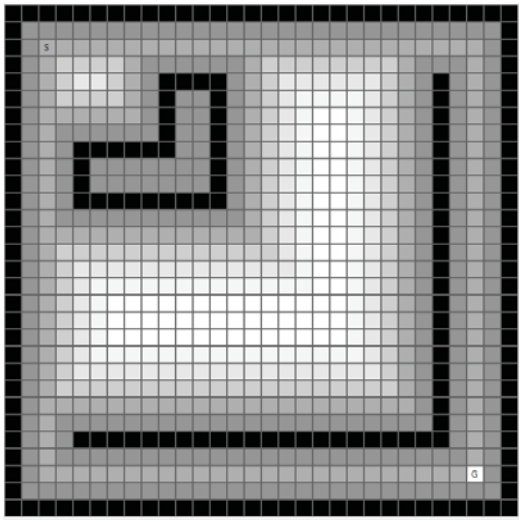


Figure 7.57: Distribution of the sigmoid function

Comparison between the Functions

Figure 7.58 illustrates the comparison between the functions. The value on the x-axis is $dmin(s)$, and the value on the y-axis is the magnitude $m(s)$. For the same dmin(s), the difference between the sigmoid function and the linear function is not big, but the performance of the hyperbola function is more different. With the reduction of the dmin(s), the magnitude increased dramatically, as it is shown in the figure 7.55 and 7.56, the cells which distance to nearest obstacle greater than 1 have similar magnitude.

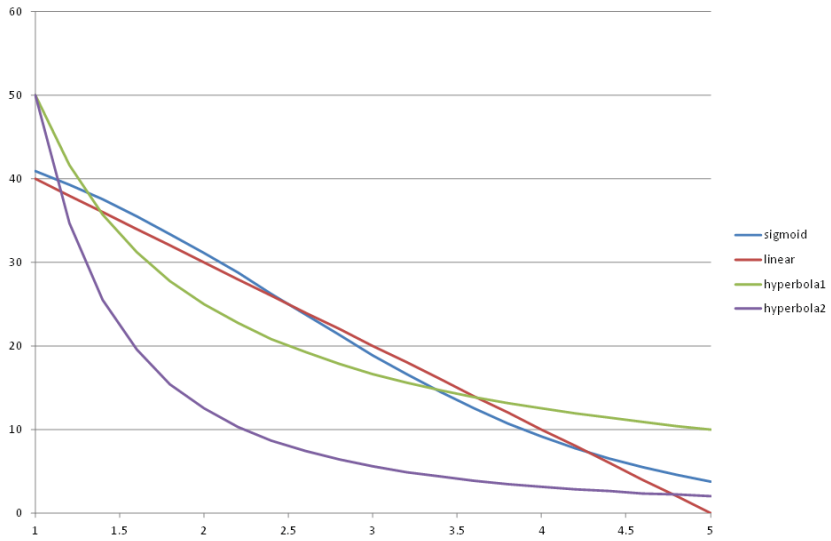


Figure 7.58: Comparison between the functions

7.4 Comparison with Related Researches

The experiments focus on the comparison between the path found by the optimal search with artificial potential field and the path found by other algorithms from related searches. Due to the different experimental circumstances, the comparison will only concern the path itself. The environments are replicated by the experiment platform.

7.4.1 Experiments

The GNRON

In the research by Ge and Cui in 2000 [51], a GNRON case is exposed with a solution of the new potential function, shown in the figure 7.59:

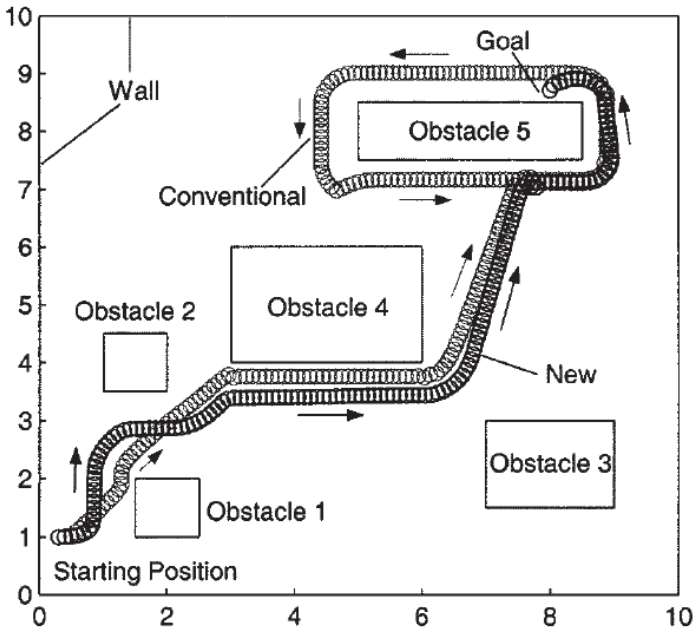


Figure 7.59: The scenario of GNRON problem from the study in [51]

The performance of using optimal search algorithm with artificial potential field are exposed in the following figures:

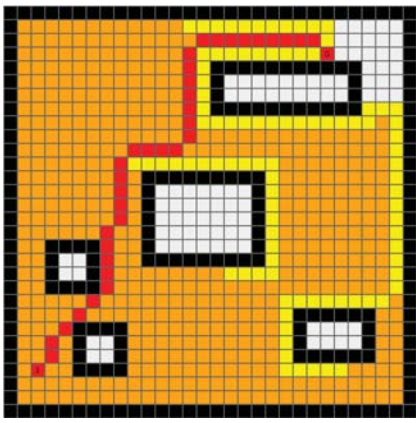


Figure 7.60: The performance of the APF-A* Search without SF

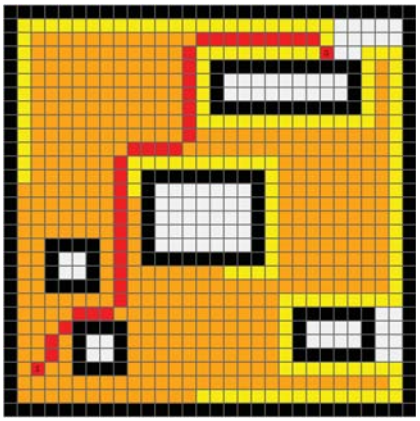


Figure 7.61: The performance of the APF-A* Search with SF

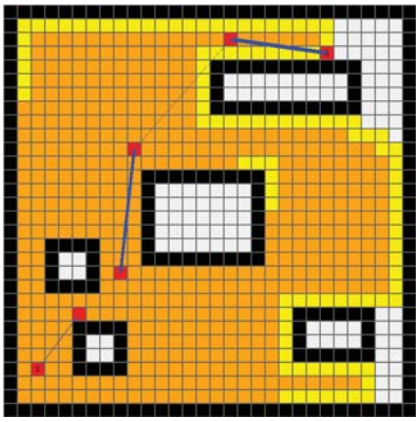


Figure 7.62: The performance of the Theta* Search without SF

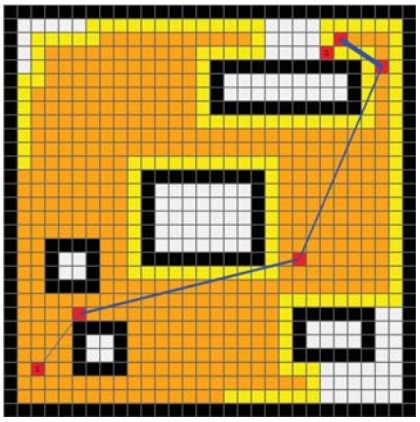


Figure 7.63: The performance of the APF-Theta* Search with SF

Table 7.10: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of Replicated Environment from the Study by Ge & Cui in 2000

Type	Visited	Expanded	Path Length
APF-A*	1976	505	40.14
APF-A* SF	1925	453	41.31
APF-Theta*	1891	472	36.00
APF-Theta* SF	1837	430	41.74

All the algorithms have found the proper path successfully. The performance of the APF-Theta* Search with SF is quite similar to the original study, and the other three have the tendency of selecting different a path.

The Shortest Path

The research by Li, Yamashita, Asama & Tamura in 2012 introduced an improvement of using the artificial potential field with regression search to reduce the length of the path [5]. The sample illustrated in the research is shown in figure 7.64, where the red line plotted the shortest path found by their algorithm.

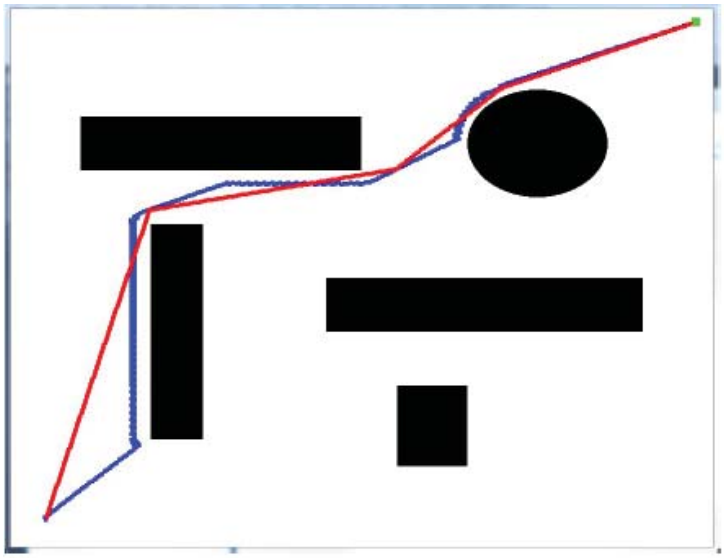


Figure 7.64: The path planning demonstration from the study in [5]

The replication of the experiment environment and the result are exposed in the following figures:

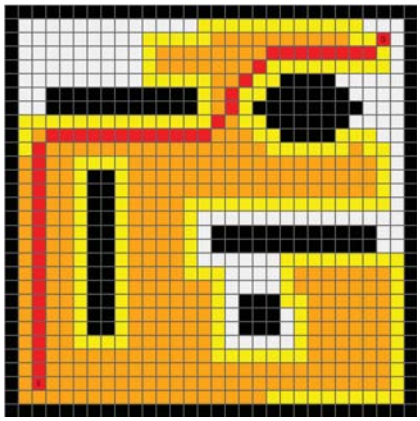


Figure 7.65: The performance of the APF-A* Search without SF

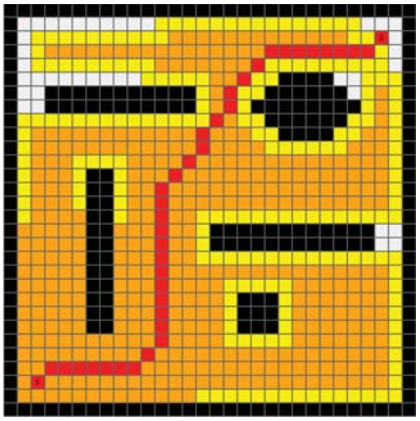


Figure 7.66: The performance of the APF-A* Search with SF

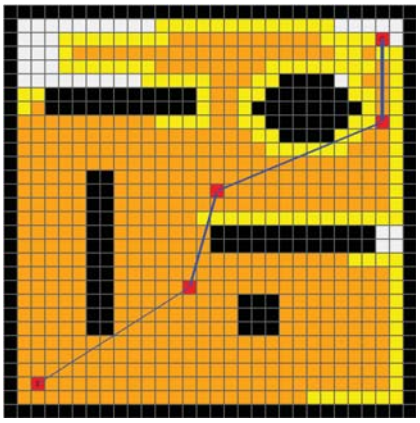


Figure 7.67: The performance of the APF-Theta* Search without SF

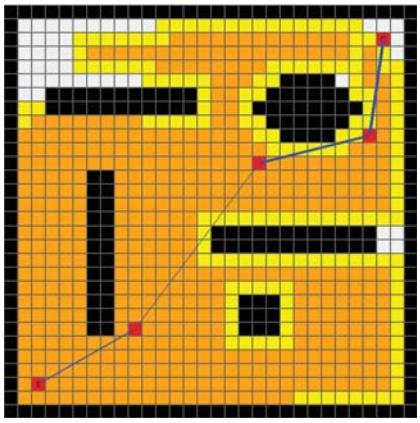


Figure 7.68: The performance of the APF-Theta* Search with SF

Table 7.11: Statistics of the Performances of APF-Optimal Searches with and without SF, in the case of the Replicated Environment from the Study by Li, Yamashita, Asama & Tamura in 2012

Type	Visited	Expanded	Path Length
APF-A*	1681	371	45.90
APF-A* SF	1992	446	42.97
APF-Theta*	2057	509	39.32
APF-Theta* SF	2023	478	38.38

All the algorithms demonstrate the ability of finding the proper path. In the APF-Theta* Search, the mechanism of the line-of-sight ensures that the path is always the shortest.

7.5 Experiment of the Performances of the Robot Behaviors

The experiments are to examine the performances of the attacker and the keeper. First, the path planning under different algorithms, the target pursuing and the obstacle avoidance will be tested. Then, for the attacker, finding the proper attacking point and the re-planning for moving target will be examined. For the keeper, the abilities of both of the parking and the blocking will be tested. Furthermore, the experiments also include the path planning using optimal search with the artificial potential field in the simulation with a complex scenario.

Although the robot soccer simulation platform is used as the environment of the experiments, some of the cases may not happen in the real robot soccer game. However, since the major purpose of the experiment is to discover the performance of the selected algorithms under different circumstances, the cases of complex terrains are still explored.

7.5.1 Platform of the Robotics Soccer Simulation

The platform was developed in c++11, using OpenGL ES 4.0. In order to collect accurate routime measurements, the experiment simulator is running on Mac OSX 10. All the results are run by ae CPU of Intel i7 4790k, 16GB Memory, single Thread without Hyper-Threading.

The size of the playing field is the same size of the robot soccer game. The figure 7.69 illustrates the aerial-view of the entire field.

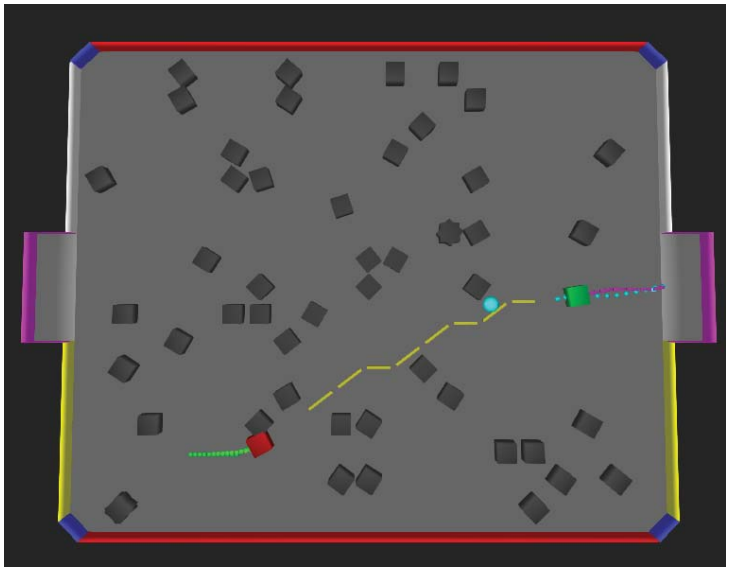


Figure 7.69: Sample performance of the Robots

The red box with a green tail is attacker, and the yellow dashed-line is the path found by the search algorithms. The trail demonstrate the actual path of the attacker.

Similarly, the green box with a cyan tail is the keeper, and the magenta dashed-line is the path found the search algorithms.

The dark-gray boxes are the obstacles, and all the walls are obstacles as well.

The figure 7.69, the keeper had just pushed the ball away from the gate and turning back to the gate for parking, while the attacker is about to pursue the ball for further attacks.

7.5.2 Experiments of the Performance

The General Performance

The general performance include the target pursuit and the obstacle avoidance. For the target pursuit, the experiments examine the running time of path-finding and visualize the path. For the obstacle avoidance, it counts the time of hitting obstacles when the agent is moving towards the target.

The Performance of the Attacker

The performances of the attacker include the attack point finding and the switching between the states. The attacker should have the ability to determine if it is necessary to use the attacking position or to pursue the target directly, and then updating the path accordingly.

The Performance of the Keeper

The performance of the keeper include the blocking efficiency and the parking. When the ball is near the penalty box and the goal, the keeper should have the ability to protect the goal by blocking the ball, and then parking back.

7.5.3 Experiment Cases and Results

The algorithm used for line-of-sight is the modified Cohen-Sutherland algorithm, because the grid is not static, and therefor it is hard to map the obstacles.

Potential Well

The initial position of the robot is in the middle of the C-Block where is a potential well. The ball is placed at the right-bottom side. Both the APF-A* Search and the APF-Theta* Search find out the path to the ball, as illustrated in figures 7.70 and 7.71. However, due to the internal drawback which has been discussed in the previous section, the path found by the APF-Theta* Search is much closer to the obstacle than the APF-A* Search.

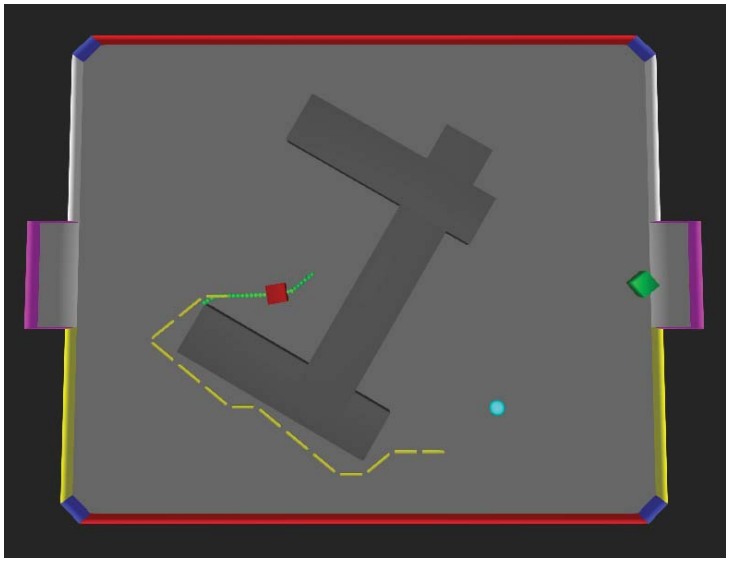


Figure 7.70: The path found by the APF-A* Search with SF

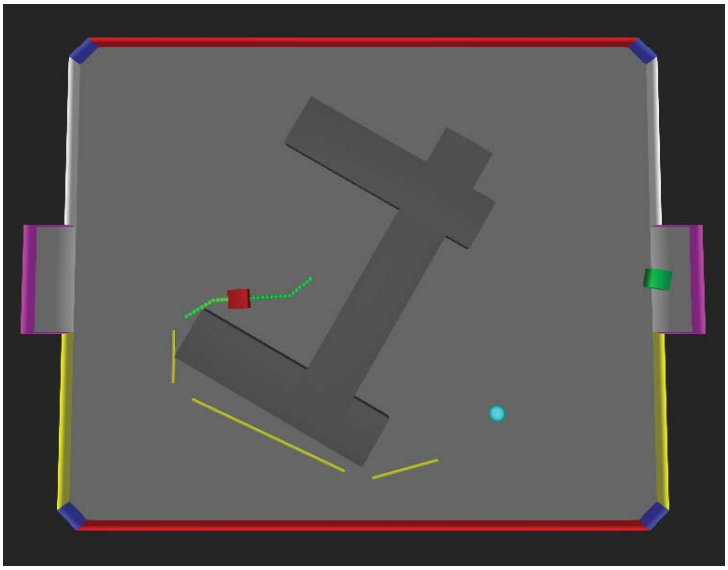


Figure 7.71: The path found by the APF-Theta* Search with SF

Table 7.12: Statistics of the Performances of APF-Optimal Searches with SF, in the case of the Potential Well running on the Simulator

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A* SF	6.0374	±0.0657	703	203	20.198
APF-Theta* SF	9.1448	±0.0131	727	212	19.137

The average runtime of the APF-Theta* Search is much longer than the APF-A* Search, where the differences of number of visited and expanded nodes are not that big. The different selection of the line-of-check might be a reason for the lower speed.

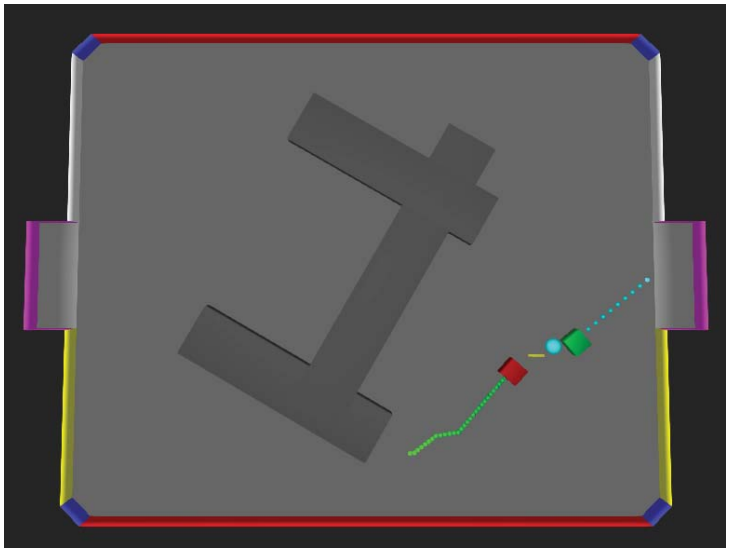


Figure 7.72: The attacking and the defense

The visualization of the re-planning of the attacker and the blocking of the keeper is shown in figure 7.72. After arriving at the attacking position successfully, the attacker keeps finding the new attacking point by re-planning the path (the yellow dashed line), and the keeper moves toward the ball immediately while the ball is near the gate.

The corresponding video has been uploaded to youtube, which demonstrates the performance of path-planning by the search algorithm and the motion control.

(link: https://www.youtube.com/watch?v=viIR07i8hB8)

Multiple Walls with a Lane in the Middle

The purpose of this case is to test the safety of the path. The path found by the APF-A* Search and the APF-Theta* Search are illustrated in figures 7.73 and 7.74 separately. In these cases, the APF-Theta* Search has a better performance than the previous one. Also, both algorithms navigate the keeper to the attacking position but not the ball directly.

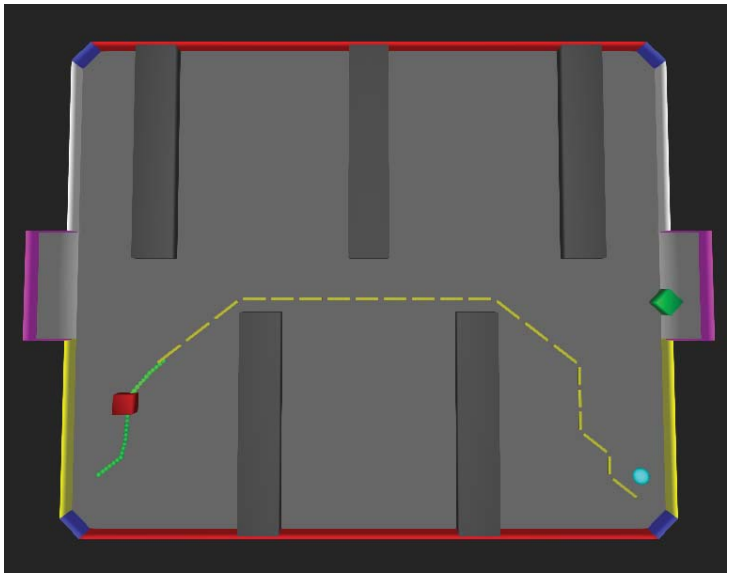


Figure 7.73: The path found by the APF-A* Search with SF

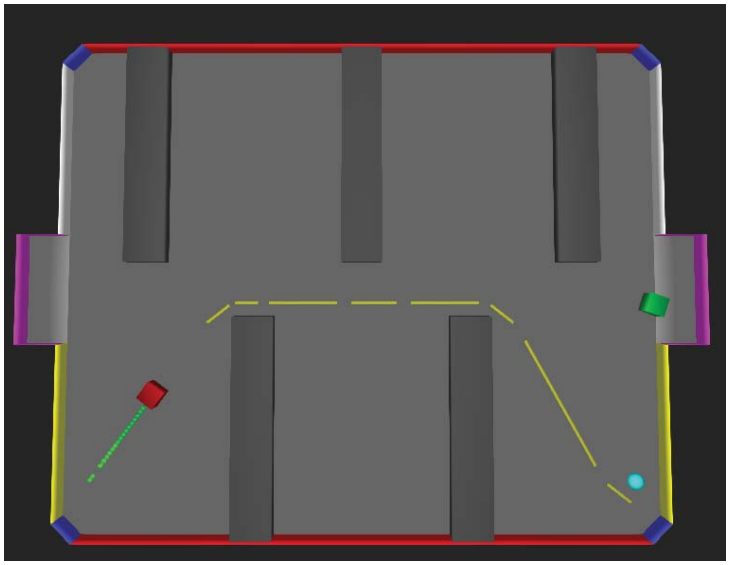


Figure 7.74: The path found by the APF-Theta* Search with SF

Table 7.13: Statistics of the Performances of APF-Optimal Searches with SF, in the case of the Multiple Walls with a Lane inthe Middle running on the Simulator

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A* SF	8.7465	±0.2553	1057	309	28.346
APF-Theta* SF	12.7534	±0.0859	1102	328	26.943

The speed of the APF-Theta* Search is still over 10ms. Considering the fps requirement of at least 30, which is 33ms for each iteration of the mainloop, this speed is not satisfactory. The speed advantage of the APF-A* Search is not as big as the previous one as well.

The corresponding video that demonstrates the performance in this case has been uploaded to youtube.

(link: https://www.youtube.com/watch?v=EDQBqg-0dyE)

GNRON

The GNRON is opposite to the potential well, where the magnitude of the artificial potential field around the goal position is much higher than the other area.

The path found by the algorithms are almost the same, since the APF-Theta* Search has purged the intermediate way-points, it is much clearer than the APF-A*, but will not affect overall result. Also, the goal of the path is the attacking position but not the ball.

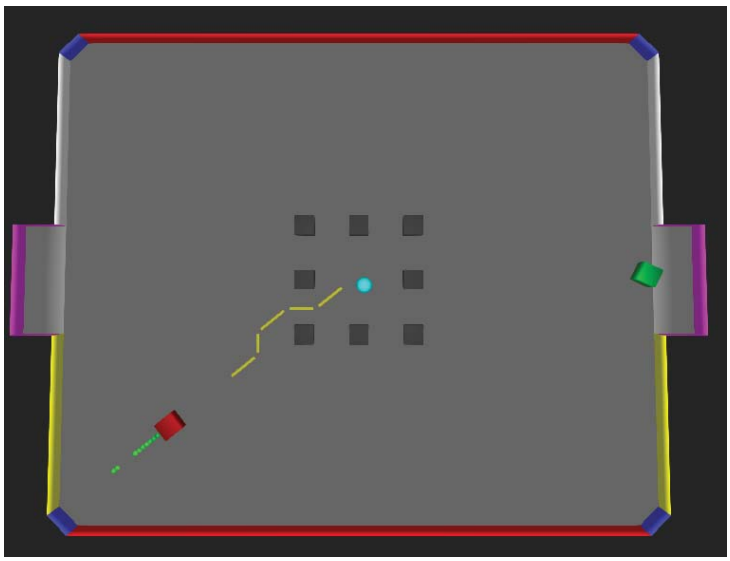


Figure 7.75: The path found by the APF-A* Search with SF

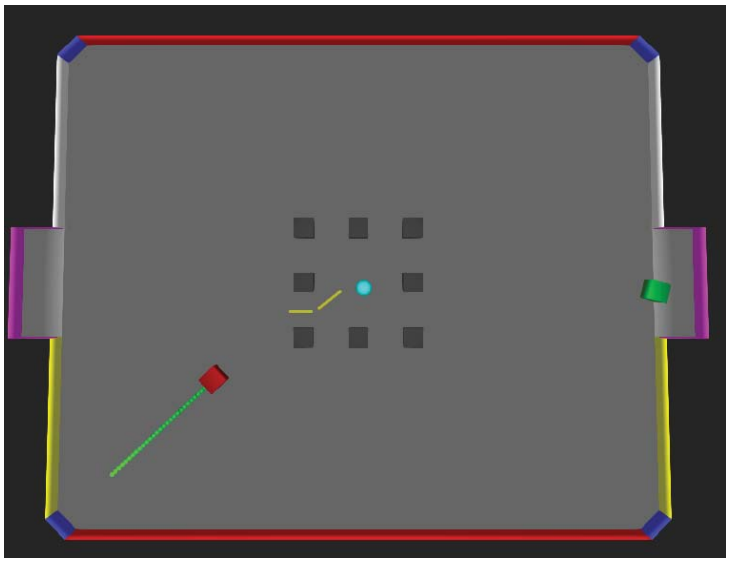


Figure 7.76: The path found by the APF-Theta* Search with SF

Table 7.14: Statistics of the Performances of APF-Optimal Searches with SF, in the case of the GNRON problem running on the Simulator

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A* SF	4.4543	±0.0832	501	130	11.408
APF-Theta* SF	6.6503	±0.0446	519	136	11.115

In scenarios that have fewer and smaller obstacles, the performance of the Search algorithms is better. The APF-A* Search still have the advantage of the runtime which is almost 50% faster than the APF-Theta* Search.

The link of the corresponding video of this case on youtube is:

https://www.youtube.com/watch?v=6N2it81MrXY

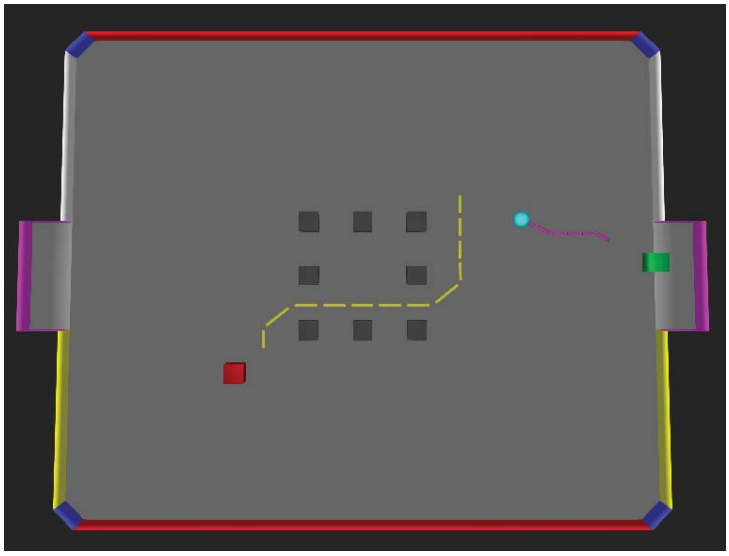


Figure 7.77: The defense of the keeper

Figure 7.77 demonstrates the behavior of the defense and the path planning by the APF-A* Search, which went through the trap and kept a safety distance away from the obstacles.

Cross Blocks

The case of cross blocks has more potential wells: in each quadrant, there is a potential well that may trap the agent inside it.

Both the APF-A* Search and the APF-Theta* search find out a proper path for attacking.

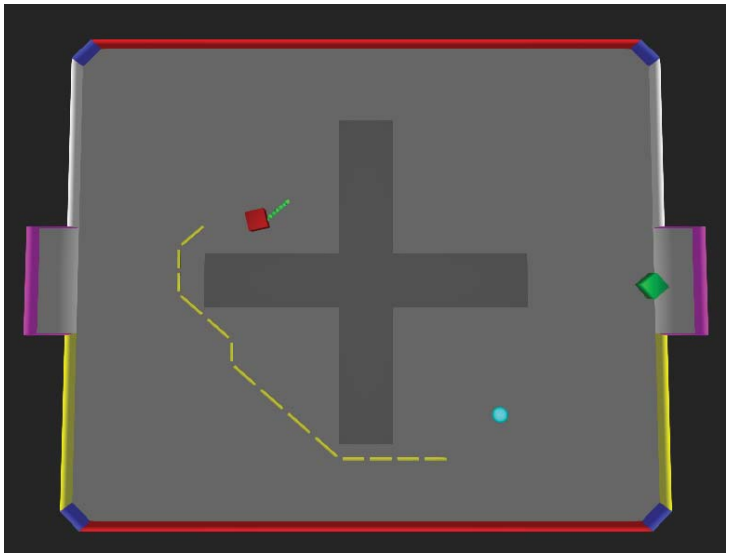


Figure 7.78: The path found by the APF-A* Search with SF

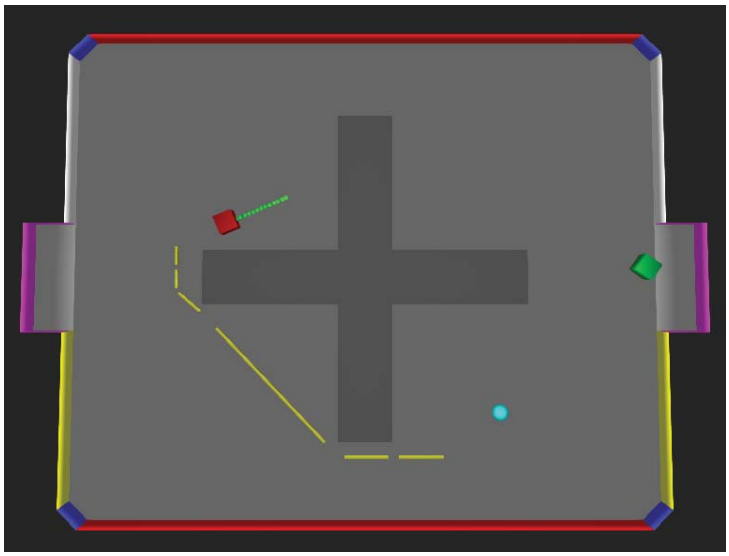


Figure 7.79: The path found by the APF-Theta* Search with SF

Table 7.15: Statistics of the Performances of APF-Optimal Searches with SF, in the case of the Cross Blocks running on the Simulator

Type	Speed(ms)	Standard Deviation(ms)	Visited	Expanded	Path Length
APF-A* SF	7.4619	±0.2334	949	264	19.396
APF-Theta* SF	10.9900	±0.1688	964	270	18.834

The average runtime of the APF-Theta* Search breaks 10ms again in this experiment.

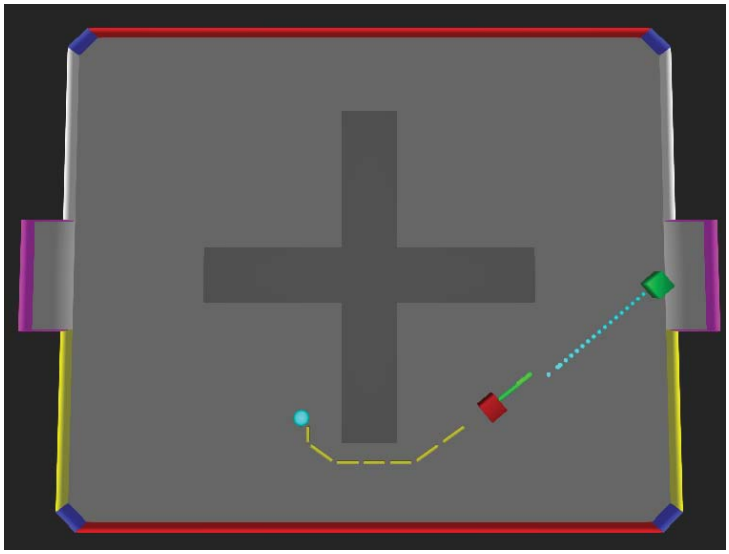


Figure 7.80: The re-planning of the attacker after the keeper push the ball away

Figure 7.80 demonstrate a re-planning after the ball is push away by the keeper for defense.

Random Obstacles

In this case, obstacles are generated and placed at random place with random direction on the field. The size of the obstacles are exactly the same as the robot, which is 7x7. The purpose of this case is to explore the relationship between the number of the obstacles and the runtime of the Theta* Search.

Figure 7.81 demonstrates an example of placing 50 random obstacles on the field and a path found by the APF-Theta* Search.

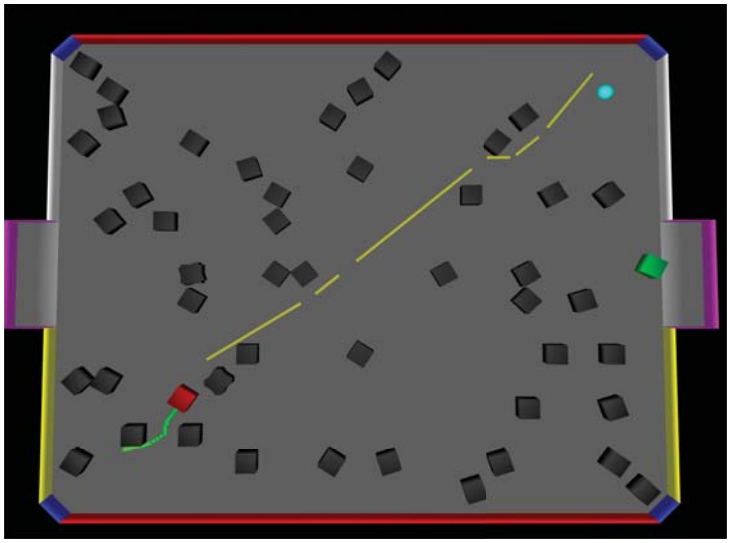


Figure 7.81: 50 Random obstacles and the path planned by the APF-Theta* Search

Table 7.16: Statistics of the Performances of APF-Optimal Searches with SF, in the case of the Random Obstacles running on the Simulator

Obs Count	Speed(ms)	Standard Deviation(ms)	Visited	Expanded
10	12.5387	±0.0393	1332	365
20	14.8016	±0.0715	1219	351
30	16.1277	±0.0426	1096	327
40	18.2074	±0.0209	1052	322
50	19.4733	±0.0240	974	302

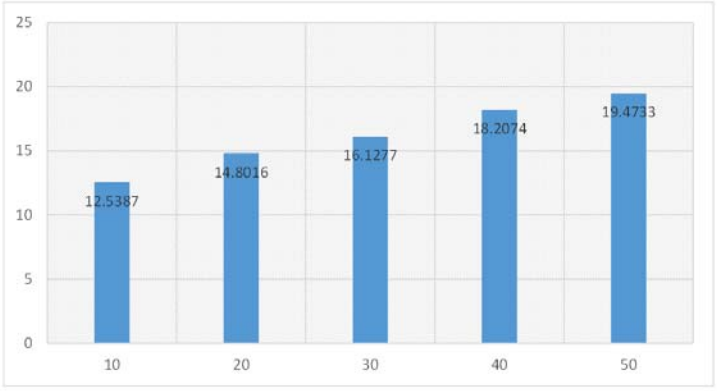


Figure 7.82: The average running time and the obstacle count

Figure 7.82 and table 7.16 illustrate the relationship between the obstacle count and the speed as a linear relationship. With the increase number of the obstacles, the line-of-sight check should access more obstacles while visiting the new node.

7.5.4 Summary

In this section, the performance of the robot soccer players in various environment are discussed and examined. Overall, the performance of optimal search with the artificial potential field and safety factor is satisfactory. Both the attacker and the keeper can avoid the obstacles when pursuing the target position.

Due to the different methods used for line-of-sight checking, the performance of the APF-Theta* Search is quite different from the experiments in the section 7.1. The need for the traversal of all the obstacles consumed much more time than visiting a limited number of cells or vertices. As a result, the runtime of the Theta* Search is not good in some of the scenarios, where the planning and re-planning time are over 10ms.

Another limitation of the APF-Theta* Search is exposed as well: despite the way-points having a proper distance away from the obstacle, the segment between the way-points is sometimes too close to the obstacles and that causes a collision. The paths found by the APF-A* Search have a better tolerance when the path is near the obstacles.

The behaviours of the robot soccer players are also exposed. The attacker illustrates the ability of finding a proper position and angle to kick the ball toward the goal, while the keeper shows the reaction when the ball is moving toward the goal.

7.6 Analysis of the Experiment Results and Discussion

7.6.1 The Performance of the APF-Optimal Search Algorithms

In general, the performance of the search algorithms are satisfied in the area of speed for the path planning. In the simple cases, the average runtime of the APF-A* Search is lower than 8ms, which can support the real-time for two agents at same time under the FPS limit of 30. The speed of the APF-Theta* Search is not as good as the APF-A* Search but still acceptable. Since the mechanism of the simulator will not plan or re-plan the path at each iteration, it is still usable.

The biggest limitation of the APF-Theta* Search is that the purged intermediate way-points causes the path to not be safety enough. For the APF-A* Search, since the existence of the dynamic cell size, the way-points which have enough distance to the obstacles will ensure that the path between the neighbour way-points has a safe distance as well. However, there is no such an insurance in the APF-Theta* Search, that causes some of the path to be too close to the obstacles.

There are cases where the calculated path is too close to the obstacles, but this will not always cause a collision. In the experiments the agent could sometimes avoid the obstacles successfully. The collision happens in the following cases:

1. a way-point or the goal is too close to the obstacles
2. obstacles are moving faster than the path planning updates
3. the robot is making a sharp turn through a small gap
4. the robot is moving too fast and too close to the obstacles, not having enough safe distance to turn

7.6.2 The Performance of the Artificial Potential Field Generator

The difference between the performances of the linear function and the sigmoid function is not big. The hyperbola function has a sharp reduction of the magnitude which is not quite satisfactory. The mechanism of the double threshold performs a significant role in the artificial potential field, which reduces the computation workload and improve the performance of the artificial potential field.

As a suggestion, the linear function with double threshold is total fine to use, although it is not as smooth as the sigmoid function. An alternative solution is use multiple

linear function together with multiple thresholds, as that could be more smooth. The hyperbola function and all the other similar functions are not advisable.

7.6.3 The Performance of the Line-of-sight Algorithms

The experiments also examined the performance of the line-of-sight algorithms, and both of them have a good performance that assists the Theta* Search to purge the unnecessary intermediate nodes.

The modified Cohen-Sutherland algorithm requires a traversal of all the obstacles on the field to check the line-of-sight during the search, and that reduces the speed if the number of obstacles is increased.

7.6.4 The Effects of Using the Safety Factor on Performance

In most cases, the APF-Optimal Search with Safety Factor can improve the safety of the trajectory by maintaining a safe distance between the calculated way-points and the obstacles. By counting the number of way-points that are touching the obstacles (unsafe), the improvement of the safety using the APF-A* Search with SF is from 18.72% to 56.54%, as compared to the original A* search algorithm.

The following tables illustrate the comparison between the APF-A* Search with SF and the original A* Search. As it is mentioned above, the "Unsafe" way-point is defined as a way-point which is touching an obstacle. Lastly, the improvement of path safety is defined as the ratio of increment between the percentage of safe way-points found by the original A* Search, and the percentage of safe way-points found by the APF-A* Search with SF.

Table 7.17: Comparison Between Original A* and APF-A* SF, in the case of One Big Obstacle, according to the figures 3.4 and 7.8

Case	One Big Obstacle
Number of way-points, A*	32
Number of Unsafe way-points, A*	14
Percentage of Unsafe way-pints, A*	43.75%
Number of way-points, APF-A* SF	34
Number of Unsafe way-points, APF-A* SF	0
Percentage of unsafe way-pints, APF-A* SF	0.00%
Improvement of Safeness	43.75%

Table 7.18: Comparison Between Original A* and APF-A* SF, in the case of Four Medium Obstacles, according to the figures 3.7 and 7.13

Case	Four Medium Obstacles
Number of way-points, A*	23
Number of Unsafe way-points, A*	9
Percentage of Unsafe way-pints, A*	39.13%
Number of way-points, APF-A* SF	29
Number of Unsafe way-points, APF-A* SF	0
Percentage of unsafe way-pints, APF-A* SF	0.00%
Improvement of Safeness	39.13%

Table 7.19: Comparison Between Original A* and APF-A* SF, in the case of Five Small Obstacle, according to the figures 3.10 and 7.18

Case	Fivs Small Obstacles
Number of way-points, A*	5
Number of Unsafe way-points, A*	21
Percentage of Unsafe way-pints, A*	23.81%
Number of way-points, APF-A* SF	26
Number of Unsafe way-points, APF-A* SF	0
Percentage of unsafe way-pints, APF-A* SF	0.00%
Improvement of Safeness	23.81%

Table 7.20: Comparison Between Original A* and APF-A* SF, in the case of Random Dots, according to the figures 3.13 and 7.23

Case	Random Dots
Number of way-points, A*	29
Number of Unsafe way-points, A*	18
Percentage of Unsafe way-pints, A*	62.07%
Number of way-points, APF-A* SF	30
Number of Unsafe way-points, APF-A* SF	16
Percentage of unsafe way-pints, APF-A* SF	53.33%
Improvement of Safeness	18.72%

Table 7.21: Comparison Between Original A* and APF-A* SF, in the case of Four Walls with a Wiggle Lane, according to the figures 3.16 and 7.28

Case	Four Walls with a Wiggle Lane
Number of way-points, A*	68
Number of Unsafe way-points, A*	43
Percentage of Unsafe way-pints, A*	63.24%
Number of way-points, APF-A* SF	80
Number of Unsafe way-points, APF-A* SF	15
Percentage of unsafe way-pints, APF-A* SF	18.75%
Improvement of Safeness	54.75%

Table 7.22: Comparison Between Original A* and APF-A* SF, in the case of Walls with a Lane in the Middle, according to the figures 3.19 and 7.33

Case	Walls with a Lane in the Middle
Number of way-points, A*	44
Number of Unsafe way-points, A*	24
Percentage of Unsafe way-pints, A*	54.55%
Number of way-points, APF-A* SF	48
Number of Unsafe way-points, APF-A* SF	0
Percentage of unsafe way-pints, APF-A* SF	0.00%
Improvement of Safeness	54.55%

Table 7.23: Comparison Between Original A* and APF-A* SF, in the case of Maze, according to the figures 3.22 and 7.38

Case	Maze
Number of way-points, A*	112
Number of Unsafe way-points, A*	85
Percentage of Unsafe way-pints, A*	75.89%
Number of way-points, APF-A* SF	137
Number of Unsafe way-points, APF-A* SF	61
Percentage of unsafe way-pints, APF-A* SF	44.53%
Improvement of Safeness	56.54%

Table 7.24: Comparison Between Original A* and APF-A* SF, in the case of Potential Well, according to the figures 3.25 and 7.43

Case	Potential Well
Number of way-points, A*	48
Number of Unsafe way-points, A*	23
Percentage of Unsafe way-pints, A*	47.92%
Number of way-points, APF-A* SF	50
Number of Unsafe way-points, APF-A* SF	0
Percentage of unsafe way-pints, APF-A* SF	0.00%
Improvement of Safeness	47.92%

Chapter 8

Conclusion

This research explored various ways of integrating the Artificial Potential Field (APF) algorithm inside selected optimal search algorithms (APF-Optimal Search), as listed in Table 8.1. In order to utilise the any-angle path-planning Theta* algorithm, three types of line-of-sight detection algorithms were proposed, namely the simple LOS check, Modified Bresenham's Line algorithm and Modified Cohen-Sutherland algorithm. Furthermore, the proposed algorithms were applied to a multi-objective intelligent behaviour benchmarking test bed called the robot soccer game. The simulation platform developed in C++11, included a simple physics engine as well as collision detection techniques.

The main purpose of the integrative architecture is to resolve the limitations of the APF algorithm, such as the local minima problem and the Goal-Non-Reachable-with-Obstacle-Nearby (GNRON) problems reported in [2] [3] [23] [5] and [51]. This work contributes a novel integration technique whereby the Potential Field approach is used as an internal component of an optimal search algorithm that calculates the g-cost values of each visited way-point, while also considering the safeness of the robot. First, the optimal search algorithms were used for high-level path-planning considering the current position of the robot and the goal position. This approach ensures that we can avoid the most crucial local minima and the GNRON problems prohibiting APF from efficient path planning. Then, by including the concept of Safety Factor (SF), introduced and proved in Section 4.2.1, the optimal search algorithms were improved, allowing for the introduction of the hybrid APF-Optimal search algorithms. The hybrid approach can plan the path effectively, striking a balance between minimising the path length and the maximising the trajectory's safeness. Also, as one of the most important component developed in this research, the performance of the APF-Optimal Search with or without Safety Factor component (SF) are explored and examined in Section 7.1. The experimental results illustrate that the APF-Optimal Search with SF effectively

adjusts the safeness of the planned path, where the percentage of the count of the waypoints positioned close to the obstacle can be reduced between 18.72% up to 56.54% , as compared to the original optimal search algorithms results. The complete summary of results can be found in the tables 7.17 to 7.24. This trajectory safeness, however, increases the path length between 1.59% up to 16.18%, and the percentage of the reduction of the number of expanded states range from +19.68% to -230.5%. The overall running speed of the path planning is smaller than 10ms.

Various types of APF generator functions were investigated, such as the linear function, the hyperbola function and the sigmoid function, as can be found in Chapter 4. We analysed the proposed functions by comparing them with the related works in [51]. The performance of the different generator functions are exposed and explored in Chapter 7.

As an auxiliary component of the Theta* search, the results of the experiments in this research show that for cell-based/grid-based path-planning, the most effective technique found was the proposed Modified Bresenham's Line Algorithm. It has a lower level of complexity among the other techniques, and is therefore faster to execute. On the other hand, with regards to vertex-based trajectory planning, the Modified Cohen-Sutherland Algorithm was found to be the most effective one, as it does not require any time-consuming mapping of the obstacles into the gridworld representation. Instead, obstacles are treated as vectors, requiring only a simple world coordinates transformation allowing for a speedy bit-wise operation for checking the line-of-sight between adjacent waypoints. The implementation details can be found in Section 4.4.

Overall, when the proposed hybrid algorithms were combined together using a finite state machine, to implement the multi-objective intelligent behaviours, the research was able to achieve implementing the intelligence for single robot behaviours include the goal keeper behaviour and the robot attacker/shooter behaviour. As observed in the experiments in Section 7.5, the performance of the APF-optimal search with the safety factor succesfully allowed both the attacker and the keeper to avoid the obstacles, while pursuing their target positions, kicking/shooting the ball, and positioning defensively. The robot behaviours are also demonstrated by the videos.

The links of the videos are listed following:

https://www.youtube.com/watch?v=6WJIT3zuInk
https://www.youtube.com/watch?v=6N2it81MrXY
https://www.youtube.com/watch?v=EDQBqg-0dyE
https://www.youtube.com/watch?v=viIR07i8hB8

Table 8.1: Summary of the Algorithms Tested and Developed

A* (Base algorithm for this research)		Grid-world Domain

Contributed Algorithms

APF-A*		Grid-world Domain
APF-A* SF		Grid-world Domain
Theta*	Simple LOS	Grid-world Domain
	Modified Bresenham's Line Algorithm	Grid-world Domain
APF-Theta*	Simple LOS	Grid-world Domain
	Modified Bresenham's Line Algorithm	Grid-world Domain
APF-Theta* SF	Simple LOS	Grid-world Domain
	Modified Bresenham's Line Algorithm	Grid-world Domain
APF-A* SF		Robot Soccer Domain
APF-Theta* SF	Modified Cohen-Sutherland Algorithm	Robot Soccer Domain

8.1 Future Works

8.1.1 Alternate Optimal Search Algorithms

Other than the A* Search and the Theta* Search, there are some other types of the search algorithm that could be considered like the D* Lite, Block A* and JPS. Different from the A* Search and the Theta* Search, these algorithms require a database before the actual search is started, and that means that the runtime of the initialization would be much longer than the A* Search and the Theta* Search. Also, the dynamic cell size could not be used which means for unknown terrain, finding out a proper size of the cell might be the biggest challenge when using these search algorithms. If a proper cell size could be found, with the ability of high efficient of re-planning, these algorithms have the ability of better performance than the A* Search and Theta* Search.

8.1.2 Decision Making

In the research, the decision making is still using the finite state machine that cannot cover all the possible scenarios during the game play. A possible solution is to use the reinforcement learning for decision making.

The decisions include: finding out a proper threshold for the artificial potential field generator, finding better attacking point and pushing direction if there is no line-of-sight between the ball and the goal, and deciding the need of using search algorithms

to re-planning since the runtime of the search algorithms is not small.

8.1.3 Use of Fuzzy System with Artificial Potential Field

Fuzzy system is a fast and smooth solution for the robot navigation. It has the potential of combining with the artificial potential field for the navigation. The biggest challenge of using the fuzzy system with artificial potential field might be the definition of the fuzzy rules, where the rules might be changed during the game-play. The reinforcement learning or the neural network might be used here as well to identify the change of the rules and make the system stable.

8.1.4 A* Search in Vector Space

In the previous chapter, the discussion of the angular A* Search is using A* Search in the vector space. Different from the A* Search in the 2D/3D Euclidean space, the definition of the distance between vectors for proper heuristic is much harder. An option is to use Dijkstra's Algorithm which has no heuristic, but the speed of the Dijkstra algorithm is much slower than other optimal searches.

Bibliography

[1] Sun, F., & Han, S. (2016, July). A flight path planning method based on improved artificial potential field. In Computer, Information and Telecommunication Systems (CITS), 2016 International Conference on (pp. 1-5). IEEE.

[2] Yang X, Yang W, Zhang H, et al. A new method for robot path planning based artificial potential field[C]//Industrial Electronics and Applications (ICIEA), 2016 IEEE 11th Conference on. IEEE, 2016: 1294-1299.

[3] Liu, Y., & Zhao, Y. (2016, August). A virtual-waypoint based artificial potential field method for UAV path planning. In Guidance, Navigation and Control Conference (CGNCC), 2016 IEEE Chinese (pp. 949-953). IEEE.

[4] Van Toll, W., & Geraerts, R. (2015, September). Dynamically Pruned A* for replanning in navigation meshes. In Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on (pp. 2051-2057). IEEE.

[5] Li, G., Yamashita, A., Asama, H., & Tamura, Y. (2012, August). An efficient improved artificial potential field based regression search method for robot path planning. In Mechatronics and Automation (ICMA), 2012 International Conference on (pp. 1227-1232). IEEE.

[6] Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. Numerische mathematik, 1(1), 269-271.

[7] Firmansyah E R, Masruroh S U, Fahrianto F. Comparative Analysis of A* and Basic Theta* Algorithm in Android-Based Pathfinding Games[C]//Information and Communication Technology for The Muslim World (ICT4M), 2016 6th International Conference on. IEEE, 2016: 275-280.

[8] Wang, H., Zhou, J., Zheng, G., & Liang, Y. (2014, November). HAS: Hierarchical A-Star algorithm for big map navigation in special areas. In Digital Home (ICDH), 2014 5th International Conference on (pp. 222-225). IEEE.

[9] Hart, P. E., Nilsson, N. J., Raphael, B. (1968). A formal basis for the heuristic

determination of minimum cost paths. IEEE transactions on Systems Science and Cybernetics, 4(2), 100-107.

[10] Skiena, S. (1990). Dijkstra's algorithm. Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley, 225-227.

[11] Barbehenn, M. (1998). A note on the complexity of Dijkstra's algorithm for graphs with weighted vertices. IEEE Transactions on computers, 47(2), 263.

[12] Madan, L., Bhushan, K. A. B. BRESENHAM'S LINES ALGORITHM.

[13] Pochmara, J., Grygiel, W., Koppa, R., & Kaminski, K. (2013, June). Mobile Robot Platform for Real-Time Search Algorithms. In Mixed Design of Integrated Circuits and Systems (MIXDES), 2013 Proceedings of the 20th International Conference (pp. 615-620). IEEE.

[14] Newman, W. M., Sproull, R. F. (1979). Principles of interactive computer graphics. McGraw-Hill, Inc..

[15] Daniel, K.; Nash, A.; Koenig, S.; and Felner, A. 2010. Theta*: Any-angle path planning on grids. JAIR 39:533-579.

[16] Fernandes, E., Costa, P., Lima, J., & Veiga, G. (2015, March). Towards an orientation enhanced astar algorithm for robotic navigation. In Industrial Technology (ICIT), 2015 IEEE International Conference on (pp. 3320-3325). IEEE.

[17] Yap, P. K. Y., Burch, N., Holte, R. C., Schaeffer, J. , 2011. Any-Angle Path Planning for Computer Games. AIIDE.

[18] Duchon, F., Babinec, A., Kajan, M., Beno, P., Florek, M., Fico, T., & Juri?ica, L. (2014). Path planning with modified A star algorithm for a mobile robot. Procedia Engineering, 96, 59-69.

[19] Goodrich, M. A. (2002). Potential fields tutorial. Class Notes, 157.

[20] Thorpe, C., Matthies, L. (1984, September). Path relaxation: Path planning for a mobile robot. In OCEANS 1984 (pp. 576-581). IEEE.

[21] Felner, A., Zahavi, U., Holte, R., Schaeffer, J., Sturtevant, N., and Zhang, Z. (2011). Inconsistent heuristics in theory and practice. Artificial Intelligence, 175(9-10), 1570-1603.

[22] Yujin Robotics, Co., Ltd. (2003). Robot Soccer: YSR-A System Manual. Yujin Robotics, Co., Ltd. Korea.

[23] Tan, J., Zhao, L., Wang, Y., Zhang, Y., & Li, L. (2016, August). The 3D Path Planning Based on A* Algorithm and Artificial Potential Field for the Rotary-Wing Flying Robot. In Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2016 8th International Conference on (Vol. 2, pp. 551-556). IEEE.

[24] Wang, Y., & Chirikjian, G. S. (2000). A new potential field method for robot path planning. In Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on (Vol. 2, pp. 977-982). IEEE.

[25] Li, G., Tong, S., Lv, G., Xiao, R., Cong, F., Tong, Z., ... & Asama, H. (2015, October). An Improved Artificial Potential Field-based SImultaneous FORward Search (Improved APF-based SIFORS) method for robot path planning. In Ubiquitous Robots and Ambient Intelligence (URAI), 2015 12th International Conference on (pp. 330-335). IEEE.

[26] Tang, L., Dian, S., Gu, G., Zhou, K., Wang, S., & Feng, X. (2010, July). A novel potential field method for obstacle avoidance and path planning of mobile robot. In Computer Science and Information Technology (ICCSIT), 2010 3rd IEEE International Conference on (Vol. 9, pp. 633-637). IEEE.

[27] Batten, C. (2005). Control for Mobile Robots. Maslab IAP Robotics Course.

[28] Eslami, A., Asadi, S., Soleymani, G. R., & Azimirad, V. (2014). A real-time global optimal path planning for mobile robot in dynamic environment based on artificial immune approach. GSTF Journal on Computing (JoC), 2(1).

[29] Rezaee, H., & Abdollahi, F. (2012, July). Adaptive artificial potential field approach for obstacle avoidance of unmanned aircrafts. In Advanced Intelligent Mechatronics (AIM), 2012 IEEE/ASME International Conference on (pp. 1-6). IEEE.

[30] Nazario-Casey, B., Newsteder, H., & Kreidl, O. P. (2017, March). Algorithmic decision making for robot navigation in unknown environments. In SoutheastCon, 2017 (pp. 1-2). IEEE.

[31] Wang, Y., Wang, J., & Yin, S. (2009, October). An Object-Based Path Planning Using Grids-Potential Fields for Intelligent Robot. In Genetic and Evolutionary Computing, 2009. WGEC'09. 3rd International Conference on (pp. 150-153). IEEE.

[32] Shi, H., Sun, C., Sun, X., & Feng, T. (2006). Chaotic potential field method and application in robot soccer game. In Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on (Vol. 2, pp. 9297-9301). IEEE.

[33] Mendonca, P., & Goodwin, S. (2015, December). C-Theta*: Cluster Based Path-Planning on Grids. In Computational Science and Computational Intelligence (CSCI), 2015 International Conference on (pp. 605-608). IEEE.

[34] McIntyre, D., Naeem, W., & Zhang, C. (2016, June). Cooperative mapping and exploration using counter-rotational potential fields. In Signals and Systems Conference (ISSC), 2016 27th Irish (pp. 1-7). IEEE.

[35] Sadeli, A. M. I., Prihatmanto, A. S., & Rijanto, E. (2014, November). Design and implementation of heuristic intelligent attack algorithm in robot soccer system small size league. In System Engineering and Technology (ICSET), 2014 IEEE 4th International Conference on (Vol. 4, pp. 1-7). IEEE.

[36] Manalu, F. R. G. (2014, October). Double target potential field: Planning the movement on passing and kicking the ball in soccer robot. In Information Technology and Electrical Engineering (ICITEE), 2014 6th International Conference on (pp. 1-5). IEEE.

[37] Choi, S., Lee, J. Y., & Yu, W. (2010, December). Fast any-angle path planning on grid maps with non-collision pruning. In Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on (pp. 1051-1056). IEEE.

[38] Mu?oz, P., & Rodriguez-Moreno, M. (2012, September). Improving efficiency in any-angle path-planning algorithms. In Intelligent Systems (IS), 2012 6th IEEE International Conference (pp. 213-218). IEEE.

[39] Gyorgy, A., & Harmati, I. (2009, August). Motion planning algorithms for tactical actions in robot soccer. In Control Conference (ECC), 2009 European (pp. 4445-4450). IEEE.

[40] Gupta, G. S., Demidenko, S., & Messom, C. (2008, May). Obstacle avoidance in a collaborative environment. In Instrumentation and Measurement Technology Conference Proceedings, 2008. IMTC 2008. IEEE (pp. 991-996). IEEE.

[41] Hai-hua, W., & Dong-liang, L. (2010, July). Optimal obstacle avoidance path planning in robot soccer. In Environmental Science and Information Application Technology (ESIAT), 2010 International Conference on (Vol. 1, pp. 748-750). IEEE.

[42] Raja, P., & Pugazhenthi, S. (2012). Optimal path planning of mobile robots: A review. International Journal of Physical Sciences, 7(9), 1314-1320.

[43] Xu, X., Xie, J., & Xie, K. (2006). Path planning and obstacle-avoidance for soccer robot based on artificial potential field and genetic algorithm. In Intelligent Control

and Automation, 2006. WCICA 2006. The Sixth World Congress on (Vol. 1, pp. 3494-3498). IEEE.

[44] Dolgov, D., Thrun, S., Montemerlo, M., & Diebel, J. (2010). Path planning for autonomous vehicles in unknown semi-structured environments. The International Journal of Robotics Research, 29(5), 485-501.

[45] Yao, J., Lin, C., Xie, X., Wang, A. J., & Hung, C. C. (2010, April). Path planning for virtual human motion using improved A* star algorithm. In Information Technology: New Generations (ITNG), 2010 Seventh International Conference on (pp. 1154-1158). IEEE.

[46] Silva, M. O., Silva, W. C., & Romero, R. A. (2010, October). Performance analysis of path planning techniques based on potential fields. In Robotics Symposium and Intelligent Robotic Meeting (LARS), 2010 Latin American (pp. 115-119). IEEE.

[47] Sgorbissa, A., & Zaccaria, R. (2012). Planning and obstacle avoidance in mobile robotics. Robotics and Autonomous Systems, 60(4), 628-638.

[48] Aggarwal, S., Sharma, K., & Priyadarshini, M. (2016, March). Robot navigation: Review of techniques and research challenges. In Computing for Sustainable Global Development (INDIACom), 2016 3rd International Conference on (pp. 3660-3665). IEEE.

[49] Khatib, O. (1986). Real-time obstacle avoidance for manipulators and mobile robots. The international journal of robotics research, 5(1), 90-98.

[50] Ruchti, J., Senkbeil, R., Carroll, J., Dickinson, J., Holt, J., & Biaz, S. (2011). UAV Collision Avoidance Using Artificial Potential Fields Technical Report# CSSE11-03. Technical Report, Auburn University, Auburn, AL.

[51] Ge, S. S., & Cui, Y. J. (2000). New potential functions for mobile robot path planning. IEEE Transactions on robotics and automation, 16(5), 615-620.

[52] Willms, A. R., & Yang, S. X. (2006). An efficient dynamic system for real-time robot-path planning. IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics), 36(4), 755-766.

[53] Baber, R. (2006). Rigid body simulation (Doctoral dissertation, University of Warwick).

[54] Lifen, L., Ruoxin, S., Shuandao, L., & Jiang, W. (2016, August). Path planning for UAVS based on improved artificial potential field method through changing the repulsive potential function. In Guidance, Navigation and Control Conference (CGNCC), 2016 IEEE Chinese (pp. 2011-2015). IEEE.