Collision-free path finding for dynamic gaming and real time robot navigation

Roopam Bamal
Computer Science, UiT Tromsø, Tromsø, Norway
Email: roopam.bamal@uit.no

***Abstract*—Collision-free path finding is crucial for multi-agent traversing environments like gaming systems. An efficient and accurate technique is proposed for avoiding collisions with potential obstacles in virtual and real time environments. Potential field is a coherent technique but it eventuates with various problems like static map usage and pre-calculated potential field map of the environment. It is unsuitable for dynamically changing or unknown environments. Agents can get stuck inside a local minima incompetent in escaping without a workaround implementation. This paper presents efficient and accurate solutions to find collision free path using potential field for dynamic gaming and real time robot navigation. A surfing game in two testing environments with a Gamecar and a physical robot called Robocar is created with dynamic and solid obstacles. Sensor like proximity, line and ultrasonic are used along with the camera as different agents for path finding. The proposed intelligent agent (IA) technique is compared with other path planing algorithms and games in terms of time complexity, cost metrics, decision making complexity, action repertoire, inter-agent communication, reactivity and temporally continuous. It traverses for 135 meters(m) in 55.8 seconds(s) covering 20 goals and 419.3 m in 8.7 minutes while avoiding 10 local minimas successfully. Proposed technique shows comparable results to path finding with techniques using neural networks and A^* algorithm. Experimental results prove the efficiency with run time overload, time complexity and resource consumption of the proposed technique.**

***Index Terms*—Path finding, Potential field, Local minima, Robocar, Gamecar.**

I. Introduction

Path finding is an important problem while working with navigation for mobile robotics. The basic agenda for path finding algorithms is to search the most optimal path in an efficient manner. Work in this field started in the second half of last century but started to grow rapidly in the 80's work [1]. Many algorithms in the past 20 years with development and improvement in this area. Path finding is the process of avoiding collision into the obstacles by a moving robot. Path finding algorithms provide this efficiency by calculating the most optimal path to the goal so that the agent can move along that path, avoiding obstacles if there are any, to reach its goal destination to perform its designated actions/tasks. The faster the path is calculated, the faster the agent can fulfill its purpose.

Artificial intelligence (AI) is used in various types of games in today's technical era. A detailed survey for AI agent based games and types is provided [2]. Adventure games involve the players to move towards designation with solving puzzles and clearing upcoming levels of provided story line. Strategy games could be a mixture of fantasy, mythical, science fiction and recreation of battle history, where player is in-charge of military troops, weapons, game plans against the opponents.A sub-goal graph method in eight neighbor grid map games to find the shortest path with low memory is proposed [3]. Action games are most popular category which extends from post-apocalyptic Gamecar based carnage to conquering an alien horde. Role playing games are adventurous games involving things like dungeons or dragons. Similarly there are many other AI games as well. The proposed game environment transpire under adventure games.

In this paper an intelligence agent (IA) based collision free path finding technique is proposed which is applicable to any static, dynamic, virtual or real time robot navigation. A multi-agent AI surfing game is designed and implemented with Gamecar traversing through a virtual dynamic gaming environment to visit several goals and adding game points while avoiding various solid and dynamic obstacles in its path within a certain time period until its final destination is reached. The technique also has to account for the local minima problem to get out of local minima traps. A self-driving robotic car is constructed which can run while finding a collision free path for real time implementation. The novelty of the proposed technique is to use same technique with similar agents for both real-time and virtual environments. Robocar and a surfing game are created as examples to prove the efficiency and accuracy of the proposed technique

The outlook of Gamecar and Robocar is shown in Figure 1

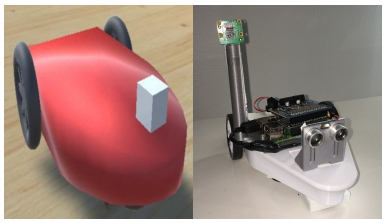


Fig. 1: Gamecar and Robocar, respectively.

This paper is structured as follows. Section II presents the related work in potential field illustrating the major problems with existing work. Proposed techniques are presented along with the materials used in creating car robot in Section III.

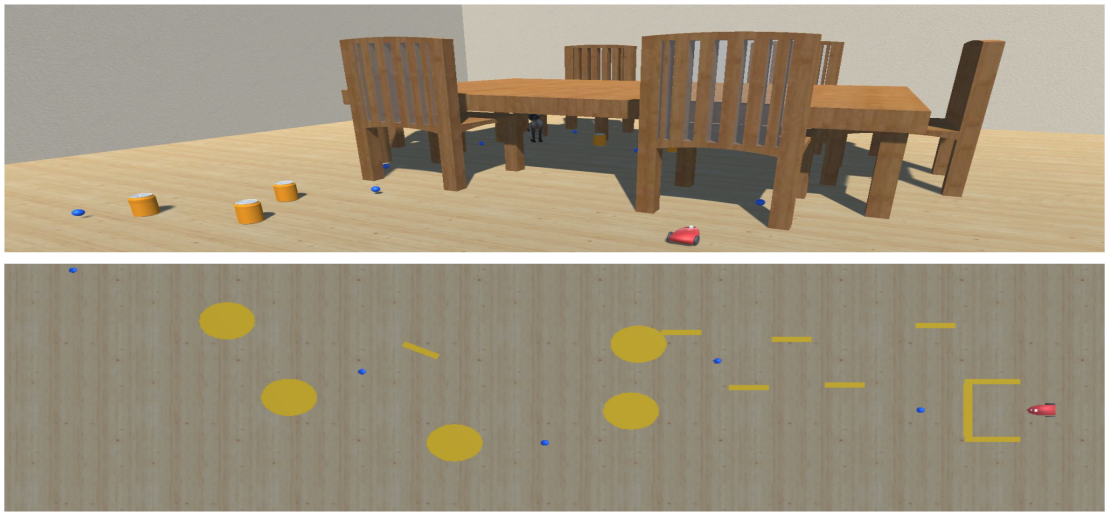


Fig. 2: Dining table environment and Test scene map in Unity 3D.

Implementation of Gamecar and Robocar is presented in section IV. Section V presents experimental results and discussions about the contributions of the paper. Finally, section VI presents conclusions and future works.

II. Related Work

Path finding can be explained in two different ways. The first approach is where a collision free path is calculated while the robot already have the knowledge of the environment. The solution to such cases are evaluated by providing highly complex scenarios. Rate of failure of such algorithms is high, because of the changing environment models or even the uncertainties of the robot itself. Rapidly exploring Random Trees (RRT) Potential Field methods and Dijkstra or the A^* algorithms fall under this category. A^* is most commonly used in path planning but is not good for dynamic set ups and static map path finding techniques can also cause delays and game freeze until the optimal path is found.Jump points were introduced in grid maps as certain expanded nodes by [4] for improving A^*. [5] proposed low time complexity for path finding with A^* technique but increased the cost metrics. [6] also, explored the limitations of A^* with low time complexity using its flooding behaviour. AA^* [7] gave collision-free path planning algorithm with high performance working with multi-agents. It finds the shortest path but not time efficient for all the possible scenarios. Path finding on maps was optimized in [8] with A^* and IDA^*. It explores the problem of non re-expansion of dead-ends and repair for IDA^*. [9] introduced a goal oriented path finding algorithm using neural network for dynamic environment. It tackled the problems with rigid unrealistic movements. Second approach is based on sensors i.e. reactive way of finding path. The robot does not have the prior knowledge of environment map and deals with the unknown situations. Decision of manoeuvre is taken in the moment based on various factors fed to the robot by sensors. The Velocity Obstacles approach, the Virtual Field Histogram (VFH) [10] and its modification VFH+, the last two based on the Potential Field (PF) method and Dynamic Window Approach are the few examples. The proposed technique is inspired by the combining both above stated approaches of path finding and provides a solution to the robots with AI.

Potential field was introduced [11] as a real-time collision avoidance module in the start, but it was later extended to motion planning. Robotic motion is influenced by the force field or an artificial potential field. Motion policy control law is akin to gradient descent on the potential function. In 2D, the gradient of a function f is defined as:

$$\bigtriangledown_f = \left(\frac{\partial_f}{\partial_x}\hat{x} + \frac{\partial_f}{\partial_y}\hat{y}\right)$$

The largest value of the derivative and the gradient will point in the same direction is checked i.e. the highest growth rate of the f value. The minimum of the function is also determined while searching the opposite direction. Similar approach is followed by Potential field. The biggest problem of the approach is the lack of performance guarantees with the robot as it can become trapped in local minima [12]. Robots are generally modeled as a 2-D collection of point agents, where the agents perform pair wise repulsive-attractive interactions, phase transitions, emergent patterns and self-organizing coherent patterns. Some implementations are done by making meta agents [13], structural potential functions and virtual leaders for object avoidance and desired pattern formation. Such techniques can lead to the local minima problem [14]. Approaches such as Local minima avoidance and Local minima escape (LME) [15] are used earlier for resolving this issue, which modifies the goal position by modifying the potential field. Navigation functions, solenoidal field and harmonic functions methods are good examples of these approaches. But it could not be practical for partially known

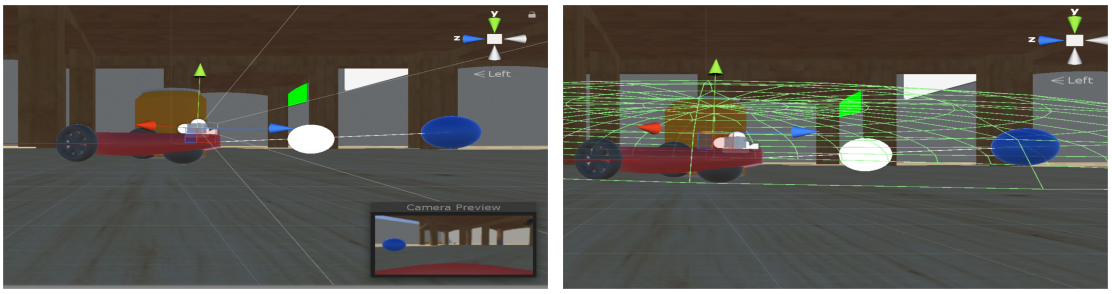


Fig. 3: Direction vectors of Gamecar with respect to camera sensor agent and proximity sensor agent.

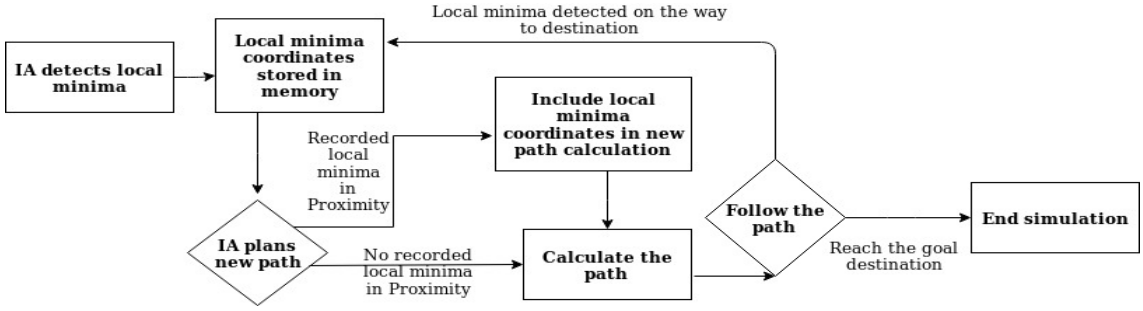


Fig. 4: The flowchart for the local minima work around logic that is used in the virtual system

environments and the real-time motion planning in dynamic. The LME approach design the algorithms to escape out of the local minima region. In the cases of several fields or multi-level potential field method, different resolution potential maps could be produced. The major disadvantage that still exist is that the agent can come back again to a previously visited local minimum configuration, which lead to the failure of the algorithm. The Straight Line Select method was introduced as well, by combining the straight line method and random walks, to find a new direction for a robot. This approach is again unsuitable for dynamic or for a real-time motion planning. In the virtual obstacle method, a virtual obstacle is added each time the agent faces a concave shaped obstacle, while making a virtual polygon is made inside the concavity. This leads to the drawback of the heuristic selection of the polygons line lengths, failing with poor choice of polygon length and destroying the performance.

The aim is to create an efficient architecture, design and implementation for path finding using potential field, while eliminating the possible drawbacks of the existing algorithms in the dynamic virtual environment and real-time environment.The major problems that this implementation provides solutions for are:

- Collision free path finding: The AI has to avoid obstacles along the way to its goal destination in the environment.
- Dynamic path planning: The AI has no predefined path calculated towards its goal. Path planning has to be done at runtime continuously with the help of sensors.
- Local minima problem: Workaround for the local minima problem of Potential Field technique has to be implemented in order for the AI to move out of the local minima areas.

III. Proposed Technique and Materials used

In the virtual implementation, a two-wheel Gamecar is created using a custom-made 3D model and Unity Game Engine's own physics components. This Gamecar, using the motors on its two wheels, is able to move forwards, backwards, steer left, steer right and brake. However, the Gamecar can only do one of the motor functions at a given time. This is done to provide more precise control of the Gamecar. Without those limitations the Gamecar behaved in a far more unpredictable manner which made controlling it extremely difficult. The Gamecar is controlled by proposed technique. Gamecar traverses a given set of goal destinations within a certain time limit. If it can visit all of the destinations within the given time period, without any collisions with the obstacles in the environment, the simulation run is considered a success.

Two distinct environments were created to run simulation instances for the IA. The first environment is a more real life like environment with depictions of real-life objects as the solid and dynamic obstacles in the dining table environment as shown in Figure 2. The second environment is filled with more challenging and synthetic obstacles to highlight the workarounds. Worst case scenarios are created like in the test scene from Figure 2 by deliberately introducing direct local minima situations like "U" loops.

In order for the Gamecar to detect its surroundings, a virtual proximity sensor agent was designed. This proximity sensor agent can detect nearby objects in any horizontal direction

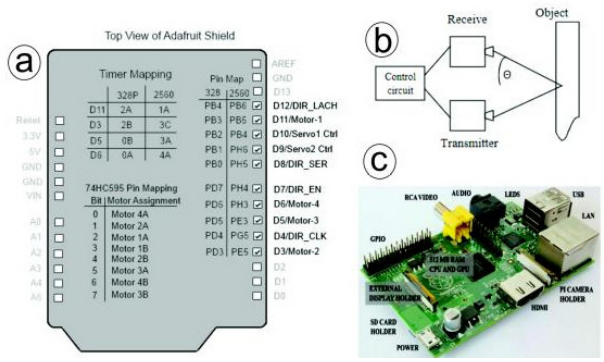


Fig. 5: a) Raspberry pi motor shield pin-structure. b) Ultrasonic sensor work-flow diagram. c) Raspberry pi outline.

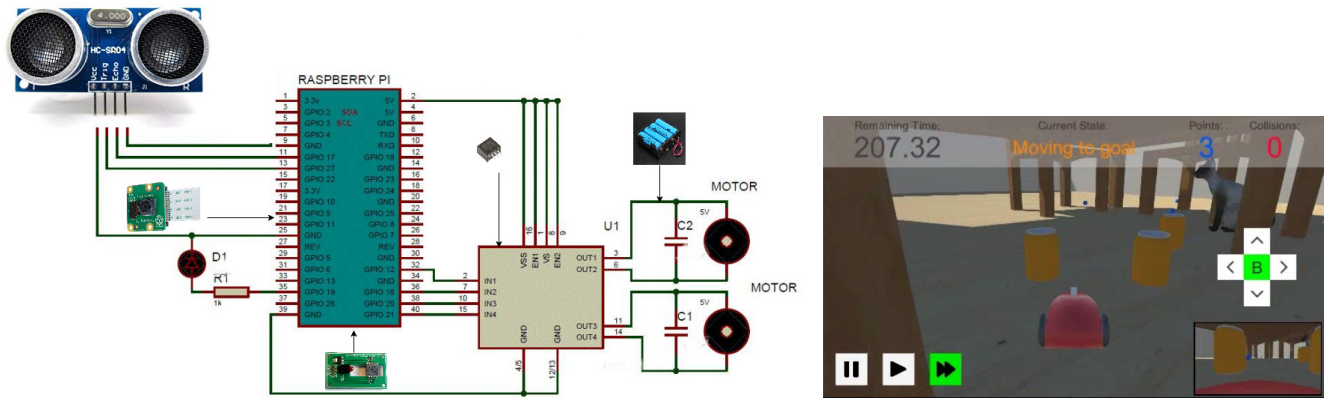
from the Gamecar. This agent is able to find the distance and the closest point from the Gamecar to all of the obstacles within its proximity. Direction of movement of IA is calculated after every move in similar mannner. Figure 3 displays behind the scene forward movement decision by direction vectors of Gamecar with IA as, camera sensor agent's line of path is clear without obstacles to the goal (blue ball) and the goal is in the coverage area of proximity sensor agent. The agent contributes to the proposed technique in finding a collision free path away from the obstacles in proximity towards its goal destination. This is done by the following steps:

1) Calculating a normalized direction vector from the Gamecar's current coordinates to the goal destination coordinates and multiplying it with 2.
2) Determining if a detected obstacle is within a critical distance for each obstacle that was detected at that point in time.
3) Summing up the distances from the Gamecar to all of the solid and dynamic obstacles within the critical range.
4) Calculating the inverse direction vectors from the critical obstacles to the Gamecar.
5) Summing up all of the inverse direction vectors.
6) Calculating the average distance and normalizing it to the range 0 to 1.
7) Multiplying the direction vector sum with the calculated average distance value.
8) Adding the normalized direction vector of the goal (from Step 1) and the final direction sum (from Step 7) to the current coordinates of the Gamecar to determine an arbitrary destination for the IA to follow to reach its goal destination.

This custom path finding technique is heavily influenced by the Potential Field [11] method. In this custom technique the goal destination influences the arbitrary direction of the path to point towards itself with a multiplier to make it more influential, while the obstacles try to point the vehicle away from themselves to provide a collision free path for the IA to follow. This does however come with some of the problems with Potential Field algorithm as well. The local minima problem affects the custom path finding technique as well. To work around this issue, a logic that follows the flowchart that can be seen on Figure 4 was designed where, when the IA detects a local minima, it marks that area virtually as an obstacle which it must escape in order for it to progress towards its goal destination. This design choice leads to an issue where the IA, as it creates these virtual obstacles, can cause more local minima areas to appear in the environment. For most problem cases this issue does not cause a major loss of functionality as the IA can still try to escape the new local minima areas using the same method.

Phyical System Design of the model of the Robocar is dependent on the working of its components acting as various agents. Each agent has a specific functionality and play a significant role.

- Motor shield: Pin structure of the Adafruit shield is as shown in Figure 5 a). The two DC motors for the model are connected and are controlled by the L293D chip. The L293D is a 4-channel monolithic integrated motor driver chip of high voltage and high current. L293D has two pins.
- Raspberry pi: is a single board, tiny credit card sized computer with the components shown in Figure 5 c). In this model Raspberry Pi 3 is used that includes a new Broadcom BCM2837 SoC with a 64-bit processor, a 1.2GHz quad-core ARM Cortex-A53 CPU.
- Ultrasonic sensors: The transceivers, used for both sending and receiving, also called transducers. The sensor evaluates the attributes of a target by interpreting the echoes from radio or sound waves respectively. In this project, they are used to detect the distance of obstacle in front of Robocar and helps in avoiding the collision.The workflow of the sensor is as shown in Figure 5 b).
- Line sensor: DPI (Dots Per Inch) to describe the number of little blobs of color (dots) the sensor can fit in a linear inch is up to 4000 dpi and 3 settings of dpi for better control. This line sensor enables the lift off distance of less than 3mm. Maximum acceleration, maximum tracking speed or the maximum linear velocity which the sensor will accurately record and maximum polling varies till 20G, 60 ips (Inches Per Second) and up to 1000Hz,



(a) Work-flow of the Robocar.

(b) Game environment with camera perspective.

Fig. 6: Work-flow of Robocar and game arena for Gamecar

respectively.

- Camera: Camera is fully Compatible with many Raspberry Pi cases with 5MP omnivision (5647 camera module) clicking pictures of 2592 $*$ 1944 resolution, which supports video having 1080p @ 30fps, 720p @ 60fps and 640 $*$ 480p 60/90 recording.
- Python: Most of the code is written in Python 3, a general purpose, high level programming language.

IV. Implementation

Virtual System Implementation: The virtual system was implemented in Unity Game Engine using C# as the programming language. The two-wheel Gamecar is created using a custom 3D model for the body of the car and its wheels. The wheels main behaviour is provided by the WheelCollider component of Unity Game Engine. The WheelColliders provide basic wheel physics with publicly accessible functions and variables to control torque of the wheels. On top, the Wheel Collider component act as one of the agents and a custom C# script was written which provides the Gamecar with ways to control the inputs for the WheelColliders. The inputs being go forwards and backwards, steer left and right and brake. The IA uses this script to basically move Gamecar around the environment. Another agent i.e. virtual proximity sensor, was implemented by again creating a C# script which provides obstacle detection and optimal path finding functionalities. The obstacles in the environment are detected using agents called Colliders provided by the Unity Game Engine. These Colliders are used to determine the collision areas for different objects in the environment. Whenever a new obstacle is detected in proximity, it is added to a priority list of collisions to further analyze during the path finding phase. When that obstacle is no longer in proximity, it is removed from the list to avoid unnecessary and unrealistic path finding calculations. Path finding functionality was implemented using a public function that returns a 3D vector to determine the arbitrary destinations coordinates in the environment. This function takes the Gamecars current coordinates and its goal destinations coordinates then calculates the arbitrary path with respect to its current obstacles in its proximity.

Finally, a C# script for the proposed technique was written. This script has references to both agent scripts that controls the wheels and the agent script that provides proximity obstacle detection. Gamecar has control over its wheels and can communicate with its proximity sensor at any given moment. Gamecar has three states: Idle state, Goal state and Destination state. In the surfing game, 100m after the last goal from the first in first out list of the goals is called the destination state. There could be any number of goals dispersed randomly in the game environment, Gamecar picks the nearest goal and have to reach all the goals before designation. Gamecar goes into Idle state if it has finished its traversal around the environment successfully, if it failed to traverse all goals and destination in the environment within the given time period or if it has nothing to do at all (default state). Gamecar goes into Goal state if it has goals and destination that it still has to visit. Gamecar is fed the destination coordinates before it can start traversing the environment. During the Goal state, Gamecar communicates with all of its agents including proximity sensor to find the arbitrary path to its current goal destination. The path is followed by giving commands to the wheel agents to move and steer towards the arbitrary destinations. As Gamecar is moving along the path, it continuously communicates with the wheel manager agent script and its proximity sensor agent to check its velocity and check if there are any obstacles directly in front of it. If its current velocity is higher than a given safety threshold and there is an obstacle directly in front of it, the IA immediately pulls the brakes until a safer velocity has been reached thus avoiding a head on collision. Gamecar also checks for local minima areas during its traversal and if it detects one, it immediately pushes the local minima coordinates to its proximity sensor agent which can then include that area while calculating the optimal path. In turn, allowing the IA to get Gamecar out of local minima traps if there are any in the current environment. The game arena of Gamecar is shown in Figure 6 b).

Physical System Implementation: Robocar was assembled using the components mentioned in Section III and the physical system scripts for all the agents including ultrasonic sensor agent and camera sensor agent were written in Python 3 programming language. The Python scripts are executed on the Raspberry Pi 3 which is running Raspian OS. The physical model working and architecture of Robocar is as shown in Figure 6 a).

V. Experimental Results and Discussions

Two separate test cases for the two systems were done separately. For the virtual system, simulation runs for both its environments were executed and observed. The AI was able to successfully finish both runs without any collisions and within the time limit. The physical system was also tested on a smaller scale by putting an object directly in front of it to make it move around the obstacle without any collisions. The physical Robocar was able to finish its testing successfully as well.

These results show that it is possible to implement a path finding technique where there is dynamic environment. IA game based Gamecar is tested for different maps and environments. Following are the few advantages of the proposed game and technique:

Maps: It can be implemented to many number of narratives, environment and maps. Example for the maps is shown in Figure 2.

Goal-oriented behaviour: Environments based on the proposed technique will have IA working to fulfil the goals. Fulfilling the goals and reaching the destination is the main motive of agents.

Time distribution: Time is accountable and low according to the distance covered. Number of goals also impact the time, as distance is increased. Figure 7 presents the variation of time and distance, when the number of goals covered by the Gamecar is increased in Dining table environment of Figure 2 with dynamic 39 obstacles like cans, table legs, chair legs and a cat. Time is directly proportional to distance and number of goals.

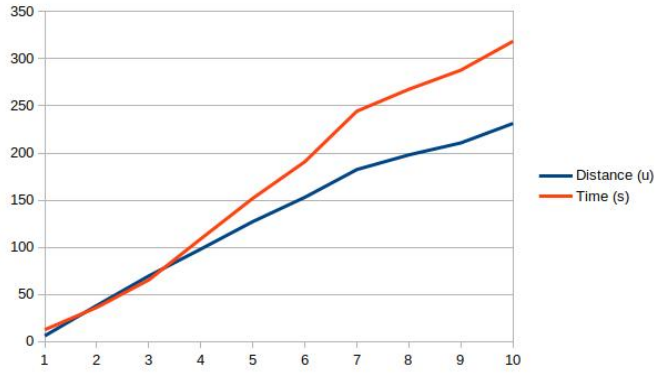


Fig. 7: Observations of Gamecar without collisions in Dining table environment with varying ”number of goals” on x-axis.

Autonomy : The agents in the Game car work on their own in gathering the information and proceeding towards the goals. Table I provides results for Test scene map shown in Figure 2 with dynamic 12 obstacles in yellow color and 5 goals(balls) in blue color. Increase in the number of local minimas, directly increase the distance and time.

TABLE I: Observations of Gamecar in Test scene from start point to designation with 5 goals.

Distance(m)	Time(s)	Local Minima
139.29	84.239	0
150.879	140.765	1
176.150	162.835	1
165.218	180.028	1
190.746	198.380	2
181.986	212.868	2
197.062	224.809	2
194.399	210.570	2
214.717	246.319	3
279.089	362.383	6
419.319	522.326	10

High performance: Autonomy agent nature insures that there is no performance sacrifice in the game engine interface and the algorithm. Table II illustrates various popular games while analyzing their agent characteristics.

TABLE II: Comparison and evaluation of various goal-oriented and situatedness games for their agent characteristics.

Games [2]	Meta agent sup. sup.	DMC	Action rep.	Inter agent Comm. Vis.	React.	Temp. cont.
Unreal Tournament	None	High	High	High	High	High
Black/ White 2	Low	High	High	Low	Low	High
Half Life 2	None	Low	High	High	High	Low
Warcraft 3	High	Low	High	Low	Low	High
Supreme Commander	High	Low	High	Low	Low	High
Tomb Raider: Anniversary	None	Low	Low	High	Low	Low
World of Warcraft	None	Low	Low	Low	Low	Low
Gamecar	High	Low	High	Low	High	High
Robocar	High	High	High	Low	High	High

- Meta agent supervision (Meta agent sup.): When all of the agents of the system report to one of the agents and that agent agent has the decision making power, that agent is called as meta agent. Most of the games need meta agent supervision like Warcraft 3 and Supreme Commander because each level of hierarchy of agents have different complexity levels. Gamecar and Robocar have high meta supervision as, proximity sensor agent and camera sensor agent reports are need to make the decision of the movement.
- Decision Making Complexity (DMC): Quality of decisions fall under this mechanism. Some of the agents in the game that follow deterministic finite state machines

TABLE III: Path finding techniques in video or robotic games stating comparable parameters.

Techniques	Topology	Agent type	Environment	Typification	Cost Metric	Time Complexity
A^* and IDA^* algorithms [8]	Undirected, uniform-cost, square grid maps with diagonal movement	Single Agent	Static	Game development	ALTBestp, Manhattan, and ALT heuristics	$A^* + ALTBestp$ $O(nlog_n)^{1/7} IDA^* + ALTBestpfaster$
Improved A^* algorithm [5]	Undirected, uniform-cost, square grid maps without diagonal movement	Single Agent	Static	Game development	Manhattan	$O(nlog_n)^{1/2}$
A^*, HPA^* and JPS algorithms [4]	Undirected, uniform-cost, square grid maps with diagonal movement	Single Agent	Static	Game development	Manhattan	JPS algorithm $O(nlog_n)^{1/10}$
SUB, BlockA^*, CPD -full/mbm, JPS-offline /online, PDH, PPQ and Tree [3]	Undirected, uniform-cost, square grid maps with diagonal movement	Single Agent	Static	Game development	–	SUB algorithm $O(nlog_n)^{1/100}$
A^*, FS, PBS and PRS algorithms [6]	Undirected, uniform-cost, square grid maps with diagonal movement	Multi-Agent	Real-time	Game development	Manhattan and Euclidean	PRS algorithm $O(nlog_n)^{1/10}$
AA^* algorithm [7]	Visible graph	Multi-Agent	Dynamic	Robotics development	Euclidean	–
Potential Field	Undirected, uniform-cost, square grid maps with diagonal movement	Multi-Agent	Real time/ Dynamic	Game/ Robotic development	Euclidean	$O(n)^2$

are never good in doing action substitutes. Proposed technique in Gamecar and Robocar follows non-deterministic finite state machine to perform better as Black and White 2 does. DMC for Robocar is high as there are high chances of randomly occurring dynamic damaging solid obstacles as compared to non damaging virtual dynamic obstacles of Gamecar.

- Action repertoire (Action rep.): It provides the agents with variety in to-do-action options. This plays a significant role in decision making for the agent. Action repertoire in dynamic or real time set ups is highly recommended.
- Inter agent Communication Visibility (Inter agent Comm. Vis.): The level of autonomy is directly proportional to the communication between the agents. Tomb Raider: Anniversary have a variety in autonomy levels at hierarchies.
- Reactivity (Rect.): Reactivity is the ability to respond in useful reactive responses. In real time or dynamic game systems, the situation tends to change rapidly. So, it requires high reactive time without a simple program with a bunch of percepted action rules or a production-rule system. Gamecar and Robocar, both have high reactivity, to avoid collisions.
- Temporally continuous (Temp. Cont.): Dynamic or real time environment based agents in pursuit of its own agenda, sense the surroundings and therefore, act on it over time. As, all the action effect, the future sense. Robocar and Gamecar work on high temporally continuous software.

Debugging: There are no pre-conditions and post-conditions in the game. The technique can be modified to support the debugging and development tools for the agents with the following challenges:

- Resource management: Meta-agent starts off by gathering local resources for the direction and movement in the proposed technique. Management of the resources becomes vital part of collision free path.
- Uncertainty in Decision making: Initially, at the start of the game or movement by the cars, there is no awareness of the obstacles or the map. A plausible hypotheses is made according to the information from the agents and actions are taken accordingly.
- Spatial and temporal reasoning: Understanding temporal relations of reactivity and decisions is utmost important in real or dynamic environment based games Static and dynamic terrain. It should be clear why the decision is made with reason.
- Real-time planning: Dynamic environment could be hostile like Robocar could encounter any sort of obstacle on its way towards the designation which can make it collide or stop, if the abstractions is not found to allow it to conduct forward searches its manageable abstract space to provide the resultant solution.

Comparison with existing techniques: Table III shows various techniques with single or multi-agents, static or dynamic/ real-time environment and specific typification from game development and robotic development. '–' is used for unknown results. Most of these techniques work as undirected behaviour at uniform cost in square grid maps with or without diagonal movements while AA^* algorithm work with visible graph. The comparison is between the algorithms being cost effective and low time complexity. Proposed technique is cost efficient with euclidean cost metric. Time complexity is calculated for each N goals with M proximity objects,path away from the obstacles is calculated, where if L is the number of local minimas, path away from the local minimas is also calculated. So, T (time taken) is assessed as the following:

$$T = N * (M + L)$$

which, gives time complexity for the proposed technique as:

$$O(n)^2$$

'n' being the number of goals of the real time environment. Another comparison is done in Table IV with [9] along time taken by the neural network technique and the proposed technique. Neural network technique varies a lot in time with the number of layers, as it needs to find an appropriate number of layers each time for each target set. The proposed technique for the same map and static environment with static goals in 3 takes, showing different decision making each time with slight difference in distance and time. Experimental results prove that the proposed technique gives better results than the best layer combination of the neural network technique.

TABLE IV: Comparison with [9] and Gamecar in test scene with no obstacles

Algorithm	Goals	Time(s)
Neural network with 2 hidden layer neurons	20	230.1
Neural network with 3 hidden layer neurons	20	141.6
Neural network with 4 hidden layer neurons	20	83.5
Neural network with 5 hidden layer neurons	20	99.5
Neural network with 6 hidden layer neurons	20	101.9
Gamecar travelling 135.428m distance	20	56.5
Gamecar travelling 135.055m distance	20	55.8
Gamecar travelling 135.205m distance	20	56.9

Design: Architecture implementation and design of the proposed Gamecar and Robocar are very efficient, low on resource consumption, low-cost, flexible and better time complexity.

VI. Conclusion with Future Work

Path planning or finding is still an open area of research. In dynamic gaming systems path finding and collision avoidance becomes an important aspect to keep the game entertaining. The robotic development along with the game with similar technique is an untouched zone in existing related world. As, physical implementation is very different from the virtual with many foreign factors coming into play.

In this paper, a path finding technique is introduced with multi-agents for the movement, avoiding obstacles if there are any, to reach its goal destination to perform its designated actions/tasks. A strong and coherent technique is applied in a car game and a physical car robot. Proposed technique is effective and accurate in avoiding obstacles, while avoiding the biggest problem of local minima. Experimental results prove that the setup is re-configurable, extensive, flexible, affordable, accurate and simple to use. Comparison with existing techniques is done, showing better results than methods using neural netwoks and A^* algorithm. Surfing game was the test case for the technique to prove the effectiveness and accuracy in the performance but it can be applied anywhere as illustrated by a minor prototype of Robocar.

Proposed IA technique can be expanded and used in the drones to find and deliver onto particular designation. Moreover, a stronger physical car robot can be designed to move in the snow, water and loess. This robot could be big in size with large battery power and capable of performing various functionalities like deploying parts and collecting data from the observational units for the DAO project working in arctic tundra for helping in preserving the ecosystem [16].

Acknowledgment

This project was carried out in UIT The Arctic University of Norway in the project group with Yigit Candundar. A special thanks to Øystein Tveito, who helped in engineering of the Robocar.

References

[1] T. Lozano-Pérez and M. A. Wesley, "An algorithm for planning collision-free paths among polyhedral obstacles," *Communications of the ACM*, vol. 22, no. 10, pp. 560–570, 1979.

[2] S. Yildirim and S. B. Stene, "A survey on the need and use of ai in game agents," in *Proceedings of the 2008 Spring simulation multiconference*. Society for Computer Simulation International, 2008, pp. 124–131.

[3] T. Uras, S. Koenig, and C. Hernández, "Subgoal graphs for optimal pathfinding in eight-neighbor grids," in *Twenty-Third International Conference on Automated Planning and Scheduling*, 2013.

[4] D. D. Harabor and A. Grastien, "Online graph pruning for pathfinding on grid maps," in *Twenty-Fifth AAAI Conference on Artificial Intelligence*, 2011.

[5] D. Harabor and A. Botea, "Breaking path symmetries on 4-connected grid maps," in *Sixth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2010.

[6] S. Brand and R. Bidarra, "Multi-core scalable and efficient pathfinding with parallel ripple search," *computer animation and virtual worlds*, vol. 23, no. 2, pp. 73–85, 2012.

[7] D. Šišlák, P. Volf, and M. Pechoucek, "Accelerated a* trajectory planning: Grid-based path planning comparison," in *19th International Conference on Automated Planning and Scheduling (ICAPS), Thessaloniki, Greece, Sept*. Citeseer, 2009, pp. 19–23.

[8] T. Cazenave, "Optimizations of data structures, heuristics and algorithms for path-finding on maps," in *2006 IEEE symposium on computational intelligence and games*. IEEE, 2006, pp. 27–33.

[9] R. Graham, H. McCabe, and S. Sheridan, "Neural networks for real-time pathfinding in computer games," *The ITB Journal*, vol. 5, no. 1, p. 21, 2004.

[10] J. Borenstein and Y. Koren, "The vector field histogram-fast obstacle avoidance for mobile robots," *IEEE transactions on robotics and automation*, vol. 7, no. 3, pp. 278–288, 1991.

[11] O. Khatib, "The potential field approach and operational space formulation in robot control," in *Adaptive and Learning Systems*. Springer, 1986, pp. 367–377.

[12] Y. Koren and J. Borenstein, "Potential field methods and their inherent limitations for mobile robot navigation," in *Proceedings. 1991 IEEE International Conference on Robotics and Automation*. IEEE, 1991, pp. 1398–1404.

[13] A. Håkansson and R. L. Hartung, "Autonomously creating a hierarchy of intelligent agents using clustering in a multi-agent system." in *IC-AI*, 2008, pp. 89–95.

[14] J. Barraquand, B. Langlois, and J.-C. Latombe, "Numerical potential field techniques for robot path planning," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 2, pp. 224–241, 1992.

[15] M. Peng, N. K. Gupta, and A. F. Armitage, "An investigation into the improvement of local minima of the hopfield network," *Neural networks*, vol. 9, no. 7, pp. 1241–1253, 1996.

[16] I. Raïs, J. M. Bjørndalen, P. H. Ha, K.-A. Jensen, L. S. Michalik, H. Mjøen, Ø. Tveito, and O. Anshus, "Uavs as a leverage to provide energy and network for cyber-physical observation units on the arctic tundra," in *2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS)*. IEEE, 2019, pp. 625–632.