**Available Online at www.ijcsmc.com**

**International Journal of Computer Science and Mobile Computing**

**A Monthly Journal of Computer Science and Information Technology**

**ISSN 2320–088X**

***IJCSMC, Vol. 3, Issue. 12, December 2014, pg.70 – 78***

**RESEARCH ARTICLE**

# Efficient Route Finding and Sensors for Collision Detection in Google's Driverless Car

**Jayaraman Seshan**
(Student, MCA IITM, India)
E-mail: jayram.d3@gmail.com

**Sandhya Maitra**
(Associate Professor, IITM, India)
E-mail: msan324@gmail.com

***Abstract: The research is aimed at creating an effective and efficient route planning algorithm for an autonomous car with sensors and cameras that work together. Despite several research attempts in the realm of path planning the process nevertheless suffers from inefficiencies such as owing to Pot holes, Heavy rain, parking space needed, etc. We propose a hybrid algorithm to plan the shortest and cost efficient path/route from source location of a car to the destination location traversing through intermediate routes. The method focuses on collision avoidance in case of obstacles with prior knowledge of occurrence. Hence the algorithm gives the optimal path with respect to the efficient distance (cost) between two successive nodes in the traversal path. We propose to combine several popular algorithms for path planning such as Dijkstra's algorithm, Euclidean distance, Approximation algorithm and A* algorithm. Considering the advantages and limitation of them with jotting down key areas of particular algorithm and its use, we arrive at solution with a flow graph.***

## Introduction

### 1.1 Background

**Driverless cars** will join the web as the biggest revolution of the early 21st century [6]. An autonomous vehicle is capable of fulfilling the human transportation capabilities of a traditional car. As an autonomous vehicle, it is capable of sensing its environment and navigating without human input. Autonomous vehicles sense their surroundings with such techniques as radar, lidar[1], GPS, and computer vision. Advanced control systems interpret sensory information to identify appropriate navigation paths, as well as obstacles and relevant signage. Google Chauffeur is the software that is used for Google's cars.

### 1.2 Problem Statements

It is a project by Google that involves developing technology for autonomous cars. This is the most emerging technology and being tested on-road for safety purpose. We all want to know about how it works on street and imagining without a driver! Is a citizen safe?

What technology is being used and how it functions?

How will it sense an object around a car at a particular distance and how will it react to it?

What will be its speed limit if it encounters a nearby vehicle or a person?

To find the desired route from source to destination.

All these queries will be researched and the solutions will be made with pros and cons in the end of this paper.

**1.3 Brief on Topic**

**Comparison**

The Google Driverless car is similar to any other car except the following features added to it such as it can steer itself while looking out for obstacles, it can accelerate itself to the correct speed limit and it can stop and go itself based on traffic conditions[3]. Most importantly it can take passengers to their destination safely, legally and comfortably.

Driver-less cars have components that performs their functions to get accurate readings and precision thus avoiding any collisions and inappropriate route selection as well. Components being used in autonomous cars are:

Google Maps: For road information

Hardware Sensors: For real time environment conditions.

Artificial intelligence: To provide real time decisions.

Google map plays a vital role in figuring out the following things like speed limits, upcoming intersections, traffic report, nearby collisions and directions.

Hardware Sensors include:

Lidar
Video Camera
Position estimator
Distance sensor

The following table depicts the working and use of each sensor and camera involved in driverless cars.

### 2.1 Sensors and Camera installed in Google driverless car

Table I

Table describing the sensors and cameras mounted in a driver-less car.

| S no | Sensor/Camera | Position | Quantity | Purpose |
|---|---|---|---|---|
| 1. | Lidar (sensor) | Top of car | 1 | This is a sensor installed to sense the environment surrounding the car. It rotates 360 degree and cover up the distance of 60 meters around a car. |
| 2. | Video Camera | Rear-view mirror of car | 1 | This video camera is essential in detecting traffic lights, read road signs, distance from other vehicles , pedestrians and other obstacles. |
| 3. | Position Estimator(sensor) | Left rear wheel | 1 | It determines the exact position of a car on a map along with its motion. |
| 4. | Radar/Distance sensor | 3- front bumper & 1-rear bumper | 4 | To reduce speed of a car by sensing nearby obstacles. (Cruise control systems.) |

These sensors and camera go hand in hand. If one gets affected the other fails to respond or give incorrect information. This could show a drastic change in case of failure. In addition to this, there is an antenna to catch GPS (global positioning system) signals from satellites and are combined with other readings to give out the accurate readings about positioning of a car. There is always prior knowledge of routes attained from Google maps regarding a user want to go from a particular source to its destination. Alternative routes are also gathered from land and transport authority as they give regular feeds about road blockage, road under construction, weather change from climate reports, heavy traffic information etc.

All these information are gathered and an appropriate path selection is made before starting from its source point. The camera that is mounted at rear-view mirror is often used to detect objects as column blocks and other moving obstacles as well.

Lidar sensor is known as "light detection and ranging" [1].This is basically a technology based up on remotely sensing nearby area of a vehicle, covering up 360 degree ranging 0- 60 meters from the car. It produces high resolution images and calculates the distances from source to destination by measuring the time that it takes to reach. It can examine everything / sense its surroundings using ultraviolet, visible and infrared lights.

All the sensors and camera work together. If any one of the sensor gets affected the other fails to respond its functionality thus results in incorrect information. To avoid such problems, prior knowledge of routes is gathered from Google maps and land and transport authority. In case of road blockage, site under-construction, weather change situations an alternate route is provided.

### 2.3 Path Tracking and Visual Tracking using fuzzy logic

According to Autonomous vehicles, path tracking and visual tracking plays a vital role. It involves both sensing and control components. It is all based on the previous path that was recorded or computed earlier using path planner (verified via Google maps) which later on senses in real time environment to determine the exact position and orientation of the vehicle with respect to the path that has been tracked. It has the ability to track moving objects, still objects, and types

of objects such as walls, pedestrians, lines etc. It is greatly dependent on efficient sensors installed in Google's driverless car (autonomous vehicle). In path tracking velocity commands are assumed to be constant. We'll be discussing about the application of fuzzy logic that is needed in certain areas of autonomous vehicles for improvement.

In real time environment, applying fuzzy logic application in autonomous vehicles creates problems such as inaccurate measurements of display regarding road signs, traffic lights etc, imprecise perception about the environment due to lack in environmental information from maps and authority, errors and rules/laws to be followed from an experienced driver and maintain flexibility , inputs and outputs gathered from human-driving experiences with vehicle for path tracking and attempting to improvise it.

**Efficient Route finding:**

Path tracking and visual tracking plays a vital role. It involves both sensing and control components. Google Map Senses in real time environment to determine the exact position and orientation of the vehicle with respect to the path that has been tracked [4].

The following diagram depicts the computation of explicit path tracking:

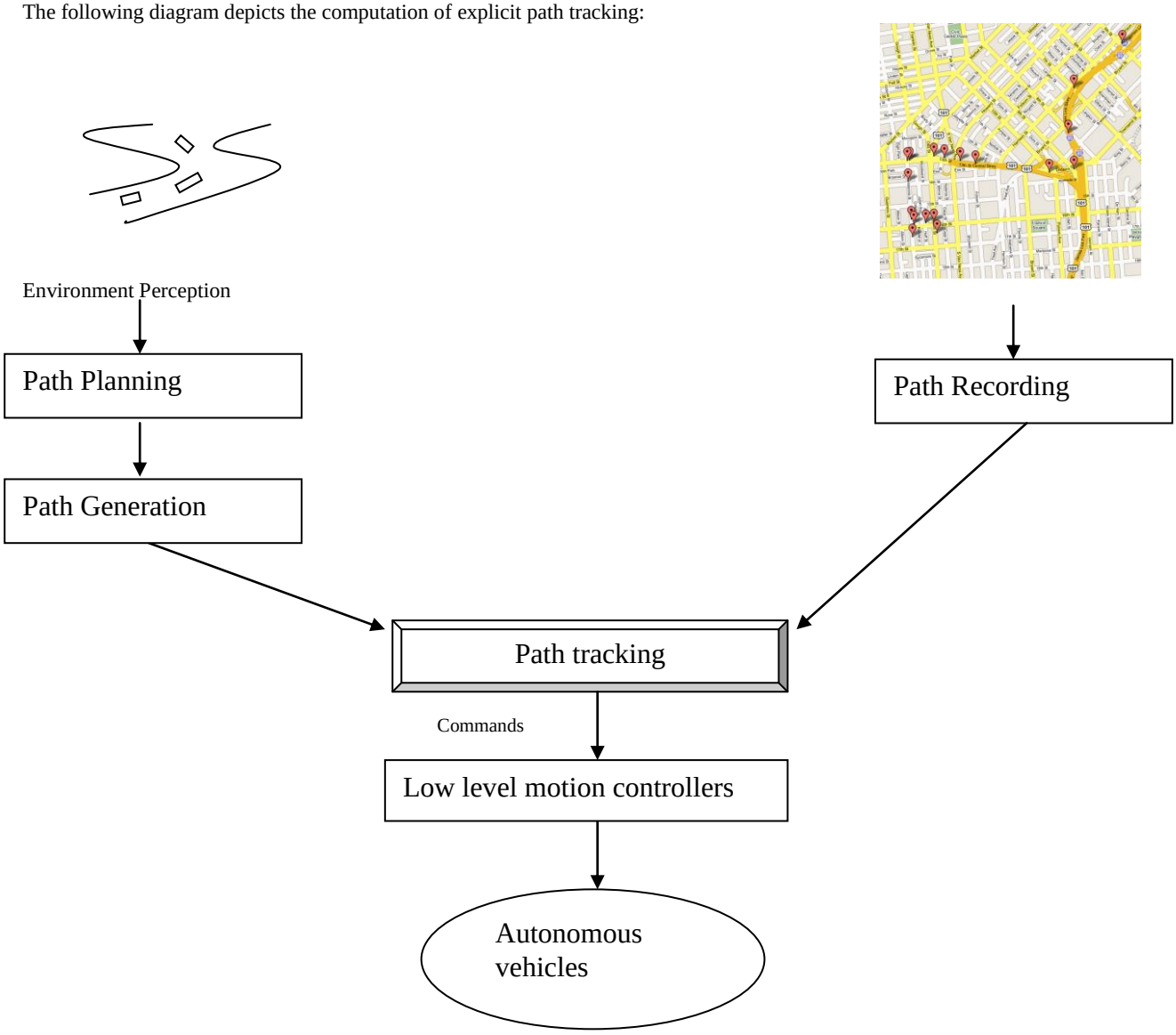



Fig. 1

The diagram explaining about the path planning and path recording that is attained from environment perception through land and transport authority and from Google maps respectively. After path is planned, it is generated and combined with prior knowledge achieved via path recording. This path is being tracked and sent to low level motion controllers via input commands readable format. Finally, the autonomous vehicle receives the final route to follow on-road.

### 3.1 Route finding algorithm in Google Driverless Car

The algorithms used are essentially for optimizing path cost and its length. It follows a series of steps and arrives at a solution. The algorithm being used is to find an optimal route from source station (where driverless car is positioned) towards its final destination by traversing intermediate route locations. If we follow our Google maps[4], it is all framed as network structure where every site can be reached through multiple ways even if one route fails due to some instances such as (road under construction, road blocked due to riots, heavy traffic jam etc). Nowadays everyone wants to reach to their destination as soon as possible within the time duration with respect to the distance from source location. The algorithm emphasizes on finding the shortest route on a directed graph as a model to implement this on real world[5].

With the help of sensors and camera mounted in a Google driverless car, we can get the visual appearances of our surroundings (road signs, traffic lights, GPS navigation, etc) and sensors to detect obstacles. Prior knowledge about destination route (multiple) is necessary through Google maps about different ways to reach from source to destination. This is done via GPS signals received through satellites.
We find the various algorithms that can be selected for efficient route finding and are described as follows:

Dijkstra's Algorithm

Euclidean Distance

Approximation Algorithms

A* Algorithm

### The basic aim for an algorithm based on path finding involves following areas to be considered

Optimizing path cost and its length (using Greedy algorithm).
Finding the shortest route on a directed graph.
Route finding in a network topology.
Computes minimum total cost path in non-negative edge weighted graph.
Solve problem more quickly and efficiently.

#### 3.2.1 Dijkstra's Algorithm

Solves single source shortest problem in a graph with non- negative edge weights [1]. Assigns an identifier to its previous nodes so that it is easy to backtrack the route it has taken. The selected Dijkstra's algorithm finds the shortest path from source to destination involving intermediate routes in a network topology. This algorithm uses local optimization at every phase and then concludes to final optimization hence termed as greedy algorithm. Now the big question arises is :

**Why we have taken Dijkstra's algorithm?**

It is the most basic and widely used algorithm for path planning systems.
Computes minimum total cost paths from initial location node to all the other possible nodes in non-negative edge weighted graph and assigns an identifier to its previous nodes so that it is easy to backtrack the route it has taken.
It is convenient to solve shortest path for directed graph in a network topology.

**How to proceed with this algorithm:**

According to this algorithm, we take :
Graph nodes = road intersection
Graph edge = roads

Now the edge weights/cost is assigned based on distance or time to reach the location.
The algorithm works like this, a source and destination location is fixed where the algorithm terminates once the car reaches to the destination location. The least cost between two nodes is mandatory[4].

**3.3 Pseudo-code:**

| |
|---|
| **Find the possible paths from source location (node) to destination (node).** |
| **Set Current node as ⇨ tarting from source node.** |
| **Set Visited node as ⊘** |
| **Update the nodes & previous nodes as the route follows towards its final destination from current location . ⇨** |
| **If there exist an unvisited node then set ⇨ that node with least cost value and continue step 3.** |

**Example**: Considering a situation where we want to travel from source node A to Destination F with its distance (km) recorded between every two nodes USING Dijkstra's Algorithm.

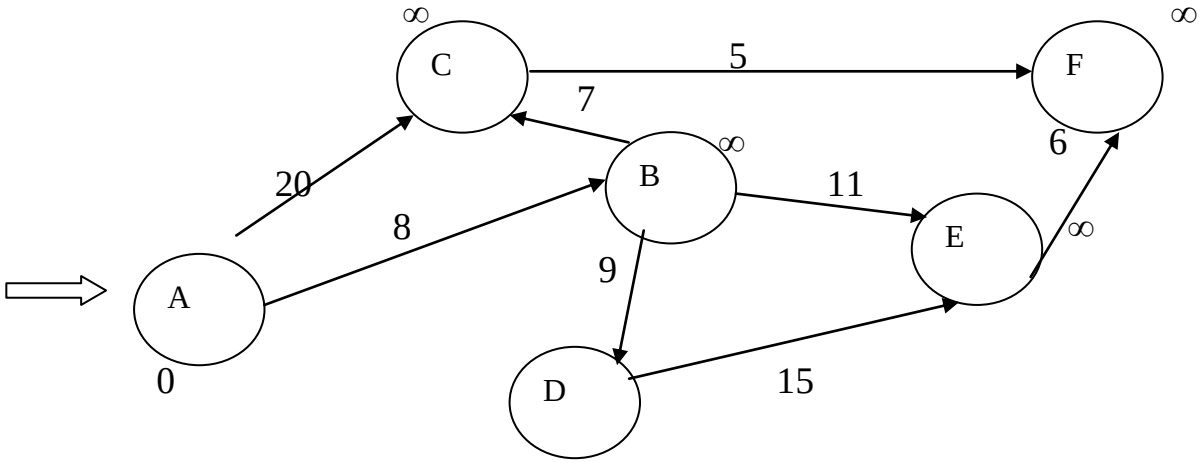
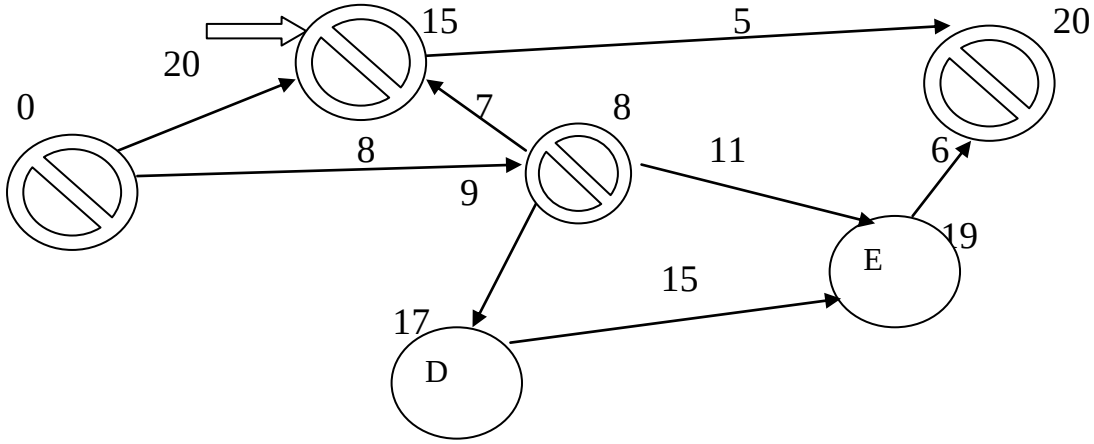



Fig 2.1

Fig 2.2



Hence shortest path from A to F is calculated = 20

**Problem with Dijkstra's Algorithm:**

The problem arises as it takes time to give out a solution. It becomes inefficient as the real world structure [2] existing is far too complex and there are many other algorithms that gives out more optimal solution. There are simpler functions that give out the lower bound estimation of the graph distance between a source node and destination node.

**Criteria for an algorithm:**

Now the edge weights/cost is assigned based on distance or time to reach the location.
The algorithm works like this, a source and destination location is fixed where the algorithm terminates once the car reaches to the destination location. The least cost between two nodes is mandatory.

**3.2.2 Euclidean Distance[5]**

The minimum distance between any node and the destination node is the Euclidean distance between two corresponding intersection (nodes).

It finds the shortest path between two intersections (nodes) that don't get intersected by any other intersection (node).

Including Euclidean distance algorithm in finding the shortest path along with Dijkstra's algorithm for precision in a 2- dimensional environment.

In most of the path planning algorithms, Euclidean Distance is used as simpler and easy to evaluate distance between two nodes.

**3.2.3 Approximation Algorithm**

Approximation Algorithm is used to find more graph nodes on/ near graph edges and results out the approximate solution for optimization problem. (3-Dimension)
**Cons:**

It is optimal to a small constant factor termed as constant factor approximation algorithm with a factor of 2. (covering up unvisited nodes is twice).

Expensive due to large input size.

**3.2.4 Demand of Speed-up Algorithm**

There are many functions or algorithms that meet the quality of solving problems more quickly, accurately and precisely.
Popular speed-up over standard Dijkstra's algorithm that are working with shortest path finding are:

Euclidean Distance

A* algorithm

**3.2.5 A* Algorithm**

It makes use of easy and simpler methods to prune out unnecessary routes by comparing minimal possible path distance from a given node.
This is improvised algorithm to Dijkstra's algorithm and is used for specified application such as route finding for vehicles where the distance can be evaluated ahead of time.

It Increases the efficiency.

It uses path distance from source node to destination node with least distance measure and identifying potential nodes.

The reason behind using this algorithm due to following factors described below[5]:

Gives least cost path distance from given node hence results as good estimation of total path length. Increased Performance and so far found as best in comparison to other algorithms as it gives out quick results. It follows Best First Search algorithm and finds least cost path from initial node to destination keeping a priority queue for alternate path as well. It always finds a solution if exists.

**Cost function:**
Sum of 2 functions:
Past route-cost function: It tells the known distance from initial node to current node.
Future route-cost function: It tells quick/forecast results of distance from any node to destination goal.

**Drawback of A* Algorithm**
Memory Requirement: An entire open list must be saved, A* algorithm is severely space limited.
Security issues: Hacks can be done in prior knowledge information that may result into incorrect route.
Sometimes there are errors in estimated path cost.

### 4.1 Comparison of A* Algorithm with Dijkstra's Algorithm illustrating with an example:

Starting with A* Algorithm which needs a source node and a destination node and Gives least cost path distance from given node hence results as good estimation of total path length. It has high performance and gives quick result. The following network directed graph with weights/cost on each edge and estimated path cost from a given node to its goal or destination point.

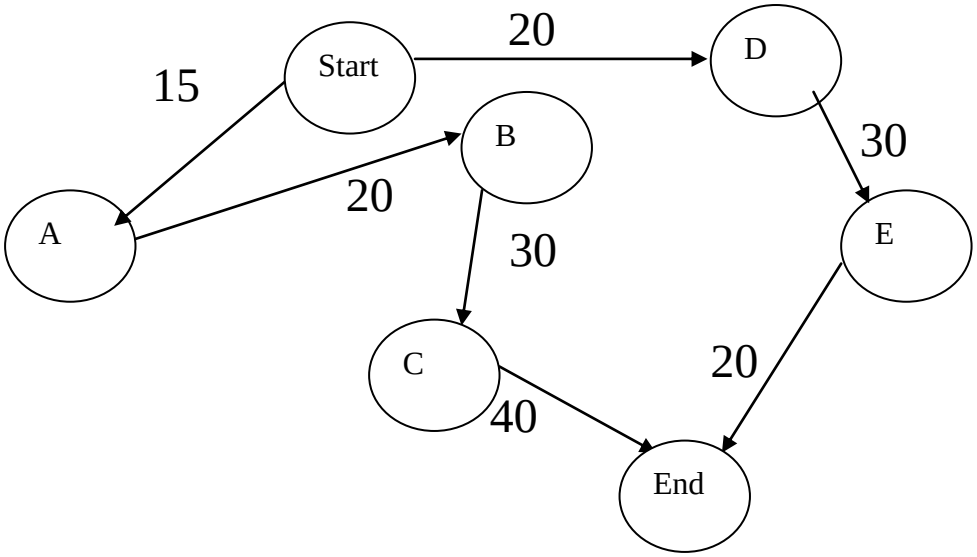



Fig 3

Now we are given estimated path cost of:

H(A)=40, H(B)=20, H(C)=40, H(D)=45 and H(E)=20

Estimated cost is considered to be distance/cost in straight line that connects from current node to its destination.
G (A)=15, G(B)=35, G(C)=65, G(D)=20, G(E)=50

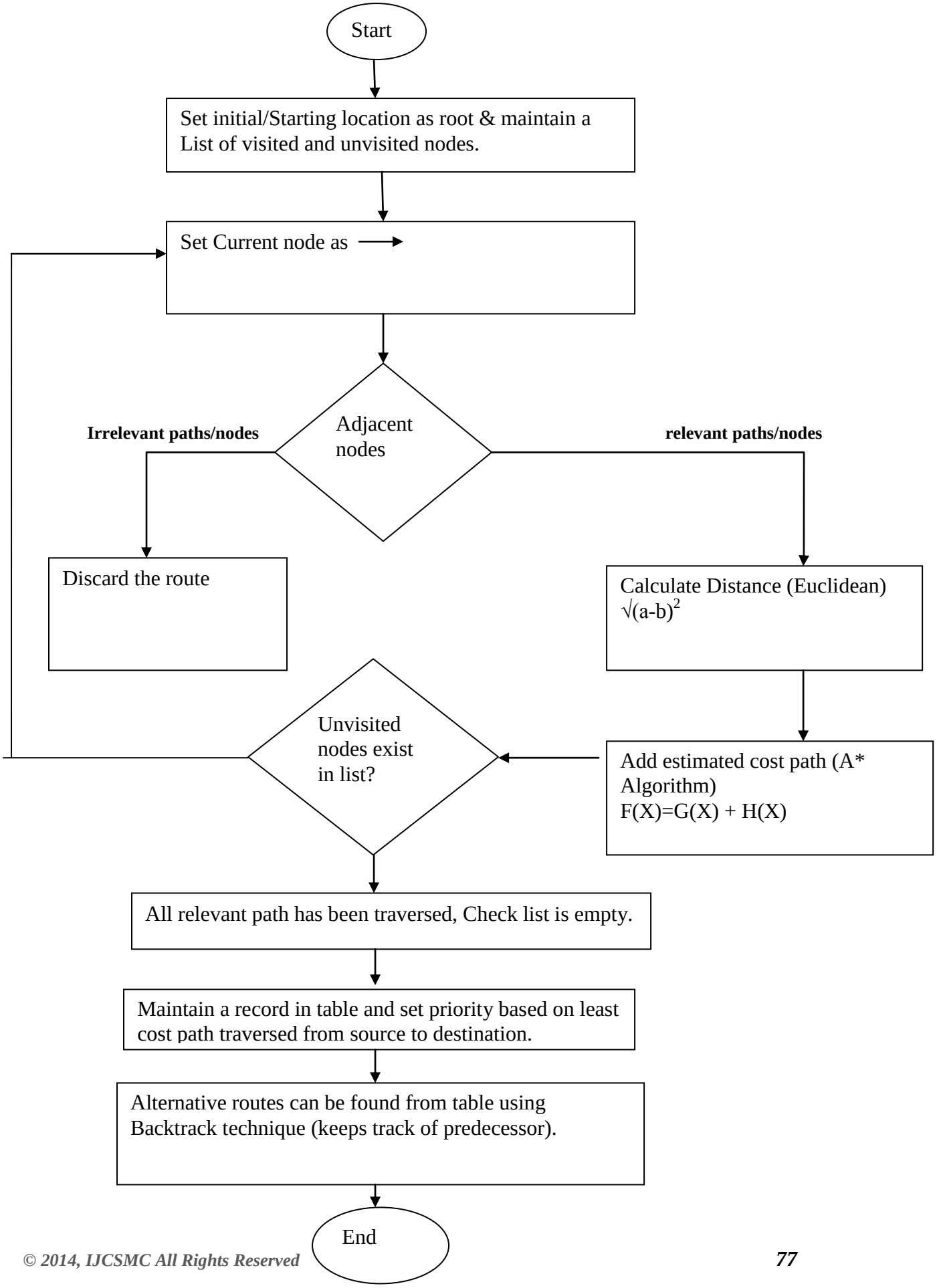
To calculate total least cost function: F(X)= G(X) +H(X)
we get, F(A)=55, F(B)=55, **F(C)=105,** F(D)=65, **F(E)=70 ;** therefore least cost value is 70 from start to end node.

**Proposed Algorithm (Hybrid)**
Steps to be followed:

1. Start from initial/starting point and make it as root and create a list of nodes that are either visited or unvisited. (Size= number of relevant nodes that lead to destination).
2. Initially the starting node is set to current node and list contains all other nodes as unvisited.
3. Trace the current node's neighbors (adjacent nodes) and calculate the distance from Current node to its neighbors. (using Euclidean formula $\sqrt{(a-b)^2}$)
4. Add estimated path cost to its value. After traversing mark them as visited also update the list and store its path cost. (Using A* algorithm formula F(X)=G(X) + H(X) Cost function considering approximation algorithm to discard irrelevant route nodes).
5. Select the least cost path value of the visited node that is in list towards the destination node and store its path selection.
6. Repeat Step 3 & 4 till all nodes that are relevant towards the destination point are visited.
7. Highlight the selected path (least cost and highly efficient) and insert it into the table.
8. End the process once Total least cost distance is achieved from source to destination.
9. Backtrack the entire path selection, which is easy because at every step of calculation, each node keeps a track of its predecessor.

**Flow Graph:**



## Conclusion

Serves the purpose of efficient route finding through hybrid algorithm proposed in an autonomous vehicle.
Easier to navigate through routes in addition, accurate and precise information of path planning can be determined by hybrid algorithm.
Sensors and camera work side by side that helps in detecting obstacles and help in reaching to destination in a safe and comfort manner.
Reverse check or backtracking, increased performance of algorithm due to estimated path cost from intermediate nodes help in route selection process more quickly and efficient.
In case of road blockage or any other crises, the algorithm can select an alternative route from the prior list with second best path that can be traversed.

## References

[1] Aaron J. Dalton, Autonomous path planning with remote sensing data, 2008
[2] GeeksforGeeks, Algorithms for shortest path, 2011
[3] Steven S. Skiena, The Algorithm Design Manual, Second Edition, 1998
[4] Johann Borenstein, Optimal Path algorithm for Autonomous Vehicles, Vol. 16, 1987, pp. 297-309
[5] Anel A. Montes, Network Shortest path application for optimum track ship routing, Kindle Edition, 2012
[6] Imran Sarwar Bajwa & M.J Arshad, Autonomous Robot Framework for Path Finding and Obstacle Evasion, Vol. 1, 2011