**Chapter 2**

**2.1 Background**

**Flying cars**

A flying car is a type of personal air vehicle or roadable aircraft that provides door-to-door transportation by both ground and air. In the 20th century when airplanes were invented the idea of flying cars came to many scientists. They started building many prototypes, but all their attempts ended in failure. Flying cars were just a science fiction where you can only see them in movies and cartoons. Scientists didn’t stop thinking about a way to make them real and available for everyone and finally in the present many prototypes passed the tests successfully and we will see flying cars in commercial in the next few years it’s no more a science fiction.

**Dynamic path planning**

An Algorithm for planning an optimal path for self-driving vehicles from their starting point to a target point which satisfies the constraints of the vehicles performance and environmental constrains. The generated path grantees that the car will arrive safely with the lowest cost and shortest time all together. The path is continually recomputed as more information becomes available

**2.2 Related works**

There are different algorithms for path planning here are some of them:

**Dijkstra’s:** Is an algorithm to find the shortest path between nodes in a graph. The algorithm finds the shortest path from a node to all other nodes in the graph provided that the nodes are reachable from the starting node. In the case of finding the shortest path between two nodes the algorithm can be stopped when reaching the goal node. From the starting node Dijkstra’s algorithm searches all the neighbor nodes and put the costs in a table in ascending order then visite the node with the smallest cost, Once we moved to the node with the smallest cost we started to search all its neighbor nodes and calculate their costs by summing the cost of the edges that lead to the node we are checking from the previous node then update the table again and visit the node with the smallest cost. We keep doing this until we discover all the graph at this point we know the shortest path from any node in the graph to any other node no need to run the algorithm again except if anything in the graph was changed.

**A\*:** This algorithm was first described in 1968 by Peter Hart, Nils Nilsson and Bertram Raphael.It’s used in path finding between two nodes in the graph. The difference between A\* algorithm and Dijkstra’s algorithm are that in Dijkstra’s we calculate the cost between two nodes with only one function which is the actual cost between the two nodes but in A\* we use two function one which is like the cost function in Dijkstra’s algorithm and the other one is the approximate cost from this node to the goal node. The total cost of each node in A\* algorithm is calculated by the summation of the two functions. The real difference is that A\* search only expands a node if it seems promising. It only focuses to reach the goal node from the current node, not to reach every other node. It is optimal, if the heuristic function is admissible. So, if the heuristic function is good to approximate the future cost, then you will need to explore a lot less nodes than Dijkstra’s. We can consider Dijkstra’s as a special case of A\* where the heuristic function is equal to zero.

**D\* Lite:** is an algorithm used in dynamic path planning between two nodes in the graph. The term D\* comes from the term Dynamic A\* because the algorithm behaves like A\* except that the cost functions can change as the algorithm runs. The incremental search methods reuse information from previous searches to find solutions to similar problems much faster than is possible by solving each search task from scratch. The difference between D\* and A\* is that D\* don’t need to recalculate the whole path when finding an obstacle, it just updates the path with the new information however A\* when finding an obstacle starts to calculate the path from the beginning.

**Chapter 3**

**Design and implementation of a java-based simulation**

**3.1 Proposed solution general approach**

Flying cars are not science fiction anymore in the next few years flying cars will be available in the market anyone can buy them. For safety and comfortability, we proposed Aviation control system for within City Flights. The system plans the path of cars from starting cell to the goal cell avoiding any accidents and to make the experience comfortable and enjoyable. No need for a pilot to drive the flying car or even getting a flying license for yourself, the cars are driven by the system from the starting point to the destination all what you need is to set your destination and just relax the system will do the remaining work. The system takes the city map as an input and when you set your destination the system plans the shortest path form your current place to your destination place avoiding any obstacles taking into considerations car’s and passenger’s safety, time taken to arrive to destination and the amount of fuel in car’s tank. The path of the car is planned dynamically step by step while the car is moving. The system sends information about the surrounding environment to the cars so that all the cars are aware of the surrounding environment to avoid any obstacles and to decide the correct place to go in its next step. The system is provided with safety measures and safety protocols to be activated in case of any problem to grantee fully safety of passengers.

**3.2 Design iterations**

We developed Aviation control system for within City Flights. The city is represented as a 3D grid of cells each cell is 1km3. The cell can be either blocked cell or empty cell. The car can land in any ground cell that has no running car on it. Multiple cars can join the system as soon as their passengers set their destinations. The system uses D\* lite algorithm to find the shortest path for any car between any two points in the city. The D\* lite algorithm handle one car at a time it finds the shortest path between the starting cell and the goal cell dynamically as the car is moving, so we modified the D\* lite algorithm to be able to handle multiple cars at the same time, so all the cars can reach their destinations when they are all running at the same time. The car starts in the starting cell, by default the starting cell is blocked whenever the destination is set, if the starting cell is already blocked the car wait until the starting cell is unblocked as no two running cars can be in the same cell. The car starts checking all its neighbor cells to know which cell provides the least cost to the destination cell, the check is done by D\* lite algorithm. If one cell is blocked, we remove it from neighbor cells while checking. If all the neighbor cells are blocked the car slows down and keep checking for its neighbor cells until one is unblocked. When the car finds the least cost cell it blocks it and then start moving towards it. When the car reaches this cell, it unblocks the starting cell, then start checking all the neighbor cells of its current cell until it finds the least cost one then blocks it and move towards it, when reaching it unblocks the previous cell and start checking all neighbor cells again. This process is continuously repeated until the car reach the destination cell. When the car land successfully in the destination cell the car park and unblocks the destination cell as now there is no running car in this cell. Many constrains were added to the algorithm to grantee both car’s and passenger’s safety. The car keeps checking for its fuel to grantee that there will be enough fuel to reach its destination and avoid any fuel shortage. There is an emergency landing system that can by activated either by the passengers in emergency cases or automatically if the system detects any need for that. The system checks the remaining fuel if it’s not enough to continue the journey safely the system activates the emergency landing protocol so that the emergency landing team can refill the fuel tank and let the car continue its journey. The system keeps checking for the car’s hardware if it detects any problem with it immediately activates the emergency landing protocol. When the emergency landing protocol is activated the car blocks all cells needed in its shortest landing path. If any cell is blocked due to the presence of a car on it the system notifies all cars in those cells to increase their speed, then after they pass the cells are automatically blocked to secure the safe landing as soon as possible. When the car pass any of the cells needed for emergency landing automatically this cell is unblocked until it reaches ground cell it parks safely and unblock the ground cell as there is no running car on it. The system notifies the flying cars emergency team with the exact cell of the emergency landing car and with the reason why the emergency landing is activated so that the team can deal with the problem. After the problem is fixed the car can rejoin the system again and continue to its destination cell.

**3.3 Java implementation**

A simulation of the system is implemented in java. After the program is run a text file is loaded to the system containing information about all cars. The first line contains number of cars that should be loaded and city dimensions, then each line contains starting cell, destination cell and fuel tank of each car. After all information is loaded cars are created then run. Cars are represented as an object. Each car run on a separated thread so that the simulation happens simultaneously. Each car has its own ID and contains fuel tank object. In the fuel tank object, you can decrement the fuel tank or check for the remaining fuel. Each car has space object where the cells are stored with their cost information relative to the starting cell and goal cell. The space object contains HashMap to map each cell to its information. The cell is represented as an object where the x, y and z dimensions of the cell are stored. All cars have access to two volatile objects Map and report. The map object helps the car to know whether any of its neighbor cells are blocked, Also the car can check for any cell whether it’s inside the city or not. The report object is where the car logs any action happens in the journey so that at the end of the journey there is a detailed report of every action the car did during its journey like moving from a cell to another cell, activating its emergency landing protocol or re-plane its path because all its neighbor cells are blocked. While the system is running you can add any car to the system by using the console. When a cars’ thread is run the car calls a method called find path in this method the car starts to find its next cell then move to it and start searching again for the next cell until the car reach the final cell. If any problem occurred during the run of this method the method notifies another method to solve the problem. The car uses D\* lite algorithm to find the next cell it should move to. The D\*lite algorithm choses the next cell based on the heuristic cost function. The function consists of the summation of the cost to move from the current cell to this neighbor cell and the cost to move from this neighbor cell to the destination cell. The system has no GUI as there will be another simulation with a simulation software.

**3.4 java results**

**3.5 Challenges**

The most important challenge we faced is safety as we know there is no 100% safe system. We tried as much as we can to make the system 99% safe by adding a lot of safety protocols and emergency systems. We want every car to arrive safely in a short period of time throw the shortest path. Other big challenge we faced during our java-based simulation is threads concurrency, Simply the processor switches between threads randomly which causes a lot of problems as this is not how it works in the real life. In real life all the cars are run with constant speed but in the simulation some cars may run then stop in the air after a short period of time as the processor switches to another thread leaving one or two cells blocked until the processor goes back to this thread again which is not the case in the real life. So, to avoid this simulation error we will be heading towards a simulation software with the same algorithm and implementation to fix this problem also to give us visualization of the resulted work of the system.

