Navigation in Dynamic Envrionment

Chao Lin

March 16, 2017

Abstract

A* algorithm has been proved that it will provide optimal solutions in a static environment. We also want to introduce the A* algorithm to a dynamic environment where exists some moving objects which are unknown to the path planner, the agent. Since A* algorithm requires a perfect knowledge to the environment, we cannot directly implement A* in this situation. But we will enlarge the agent's knowledge to the map while moving towards the goals state by detecting the moving object, so that we can use A* to plan a new optimal path from the current state to the goal state if it is necessary. The definition of "optimal" means the path should be the shortest without colliding to the object. Our algorithm will accomplish two subgoals: collision prediction and path re-planning. The collision prediction determines whether a re-planning is necessary.

Contents

1 Introduction

Finding out a path from a start position to a goal position seems to be easy for us as human beings. For example, we know how to safely get to the school from our home. The definition of "safely" is that we will not collide with any cars driving on the road, or collide into any people walking on the sidewalk. Though a car and a person are some unpredictable elements that may suddenly occur when we are on the way to school, we have abilities to detect and avoid collisions. However, the task will be more challenging if we are asked to find out the shortest path from our home to school without colliding any obstacles. It is hard for us to tell whether the path we choose is the most optimal. Though in reality, we are not asked to find the most optimal path, we have to introduce an algorithm called path planning to solve this kind of problem.

Path planning is the approach of finding out the most efficient way from the start position to the goal position. The definition is abstract, but it becomes more concrete if we use a graph as structure to interpret the problem. Each position will be represented by the vertex in the graph. Two vertices are linked together if

the represented positions are adjacent to each other. In this interpretation, we can define the path planning problems as the approaches of "finding a sequence of state transitions through a graph from some initial state to a goal state, or determining that no such sequence exists" [15]. In our project, we will use the graph to interpret the problem. In most of cases, we are more interested in finding out the most optimal solution to the path planning problem.

People have been working on path-planning algorithms for decades. In fact, there are several established algorithms that can solve path searching problems, such a s A* algorithm. The basic idea behinds these algorithms is to recursively go to and expand the states that are adjacent to the current state until reach the goal state; or it has traversed all the states. The algorithms of path planning have been applied to variable situations especially in robotics and games.

A* Path Planning Algorithm

A* algorithm is a typical path planning algorithm, which uses a heuristic function [3] to estimate the cost of expanding a state. Note that heuristic function is always underestimated, which means the cost of a state calculated by the heuristic function will always be lower than the actual cost of that state. The cost is specified by the path length between two different vertices. [3] For each time the algorithm decides to choose a state to expand, it will always choose the state with cheapest cost. That means the estimated cost from the chosen state to the goal state plus the actual cost that has spent should be the lowest. Those states that have costs are more expensive than the current cost of next movement will be eliminated. Therefore, A* algorithm introduce an optimized path with fewer states expanded.

A* is widely implemented in video games, especially in real-time strategy(RTS) games. [17] "Auto-pathing system" in RTS games is the implementation of A* algorithm. For example, in League of Legends (LOL [7]), a computer controlled "agent" who is the path planner will solve the path planning problem. That is, when the player uses mouse click to control the character's motion, a path from the start state, the current position of the character, to the goal state, the position where the mouse is clicked, will be generated by the agent. Similar agent exists in other types of games, such as car racing games [16] where the agent will plan a path for the racing cars which are not controlled by the player, so that the cars can drive on the track by theirselves.

Restrictions in Dynamic Environment

However, A* algorithm has several restrictions [1]. For example, the agent should have perfect knowledge to the environment. That is, in the game, the agent knows which parts of the game map are invalid to step on. Meanwhile, the game map should be static, which means those spots that are invalid to step on will always be at the same position during the whole time. Meanwhile, no new invalid spots will be introduced at anytime.

We can assume a simple scenario which represents the dynamic environment. In a grid world (the X-axis is pointing to the right), some cells, we call them static cells, are always labeled as invalid cells, which cannot be stepped on. Some cells, we call them dynamic cells, are labeled as valid and invalid back and forth (i.e, at t_0, the cell (0, 0) can be labeled as valid grid; at t_1, it can be labeled as invalid; at t_2, it can be labeled as valid again). Now imagine that you are at cell (10, 10), and your goal is to go to cell (0, 0) as fast as you can without stepping on those invalid cells. Meanwhile, you only know about the static cells, and has no knowledge to the dynamic cells.

Then in this scenario, the path planned by A* algorithm may not be the optimized solution. That is, you may step onto the an invalid cell, which is a dynamic grid, while you are proceeding the planned path. It is because the agent does not know about the existence of the dynamic cells, and it assumes those dynamic cells are all valid cells.

Dynamic Environment in RTS games

In fact, in most RTS games, the situation is similar to the grid senario. In the game, the map is fixed, but there may exist some minions (see Figure 1 and Figure 2). The minions, particularly in LOL, are some map elements that they have certain width, length, and height. Players can gain money from killing these minions. Those minions are controlled by the computer, and will not interact with players. These minions will be roaming around the map along some random paths during the game. The characters controlled by players are allowed to collide with the minions. In order to simplify the representation of the minions, we

Figure 1: Warcraft III [6]

Figure 2: League of Legends [7]

can regard them as the dynamic cell. That is, at time t_0, the minion is at position p_0, then p_0 is now labeled invalid; at time t_1, the minion is at position p_1, then p_1 is now labeled invalid, and p_0 is labeled valid again.

Using the "auto-pathing system", which is discussed previously, we can see a collision between the character who is controlled by the player, and the minion when the character is moving along the path calculated by the "auto-pathing system". This observation supports that A* algorithm has to have full knowledge to the environment, and requires a static environment.

In fact, in many real situations, such as self-driving cars, require some path planning algorithm which works for dynamic environment. We will require the algorithm to produce an optimal path which avoids colliding into not only static obstacles, but also moving obstacles. We decided to develop a collision-free path planning algorithm based on A* algorithm.

Though A* algorithm has constraints, it has two major advantages:

1. A* can produce the optimal solution, which its optimization can be proved;
2. A* is efficient in terms of time efficiency.

We want to take the advantage of the A* algorithm, and apply to the dynamic environment, so that we have to satisfy the restrictions that A* contains. Therefore, we decided to introduce a "collision detection system" to enlarge the knowledge of the path planning agent to the environment. With the detection system, the A* path planning agent now has knowledges to the moving obstacles, so that it can plan an optimal path without colliding the obstacles.

In order to test our algorithm, We will model a 3-D world, and place a robot, our subject who will be implemented the path planning algorithm, in the world. The robot will be assigned to accomplish several tasks to examine the efficiency, in terms of run time and the times of collision, of our algorithm.

2 Related Work

Lumelski et al. [12] developed a naive path planning algorithm in dynamic environment. Their approach is to move a robot, in our case the character, directly to the goal state. That is, the robot will keeping moving straight forward towards the goal state, until it is stopped by some obstacle. Then the robot will move around the perimeter of the obstacle. Once a point which is nearest, in terms of the direct distance, to the goal state is found on the obstacle, then the robot starts to move towards the goal state again. This procedure will be repeated until the robot reaches the goal state.

Lumelski's algorithm can exhaust all the situations, but the solution produced by the algorithm may not be optimal in terms of path length. We can imagine a triangle ABC as an obstacle which the robot want to get around. Assume the robot hits the triangle at point A, the goal position is in the direction of $\overrightarrow{BC}$. Since the robot will always move around the perimeter of the obstacle in one direction, say it will moving clockwise, then it will move along AB and BC, instead of moving along AC. Therefore, the actual path length is larger than the ideal path length. More over, in some situation, the algorithm will never terminate. That is, the robot will keep moving around the perimeter of some obstacles and never reaches the goal state.

Since the travel time of Lumelski's algorithm is unstable, we will not develop our algorithm abased on that. Instead, we wish to develop our algorithm based on some algorithm that its optimization can be proved. A* algorithm, for example, may be a candidate.

Peter E. Hart et al. [3] have proved two important properties of A* algorithm. They proved that if a heuristic function, a function measuring the cost, is a lower bound on the true minimal cost from the current state to a goal state, then A* was admissible. That is, if there exists a path from the start state to the goal state, then it will find out a path with minimal cost. They further proved that "if the heuristic function satisfies something called the consistency assumption, then A* was the optimal" [4], which means the path introduced by A* can be proved to have the shortest path length.

D* [14] is a typical path planning algorithm based on A* algorithm which can provide an optimal solution in a dynamic environment. It needs a map of the environment, but the map can be incomplete, empty, or contains only partial information. The main idea of D* algorithm is to initially plan an optimal path. The robot follows the planned path until reaches an obstacle, and re-plan the path from the current state to the goal state. The optimality of D* has been proved. [13]

D* will re-plan a path until it finds that there is an obstacle on the state where is the next movement. That is, the robot will stop moving when reaches an obstacle, and will re-move again after the re-planning is done. Though the path planned by D* will be the shortest, we will waste time on planning the path since the robot will stop while the path planning agent is planning the path. Moreover, D* allows collision which we are not expecting to. Our algorithm will improve this situation, that the collision detection system will predict a collision in advance and notice the path planning agent to re-plan the path.

Maxim Likhachev et al. [11] developed a new graph-based re-planning algorithm, Anytime Dynamic A*(AD*), which is able to produce bounded suboptimal, bounded by the inflation factor ϵ, solutions in an anytime fashion. That is, the algorithm can produce a suboptimal solution at any time during the given computation time. The algorithm concentrates on decreasing a suboptimal bound on its solution and reusing previous search efforts as much as possible. As a result, AD* algorithm is able to repair its previous solutions incrementally once it encounters dynamic environment. In other words, AD* algorithm combines the benefits of anytime fashion and incremental planners to provide more efficient solutions that can be applied in a complex, dynamic search problem.

AD* has similar drawback that it will re-plan at anytime when the map is updated. That may cost a waste in time, since for some updates, there is no need to do the re-planning. Also, as the algorithm of AD* presented, it will re-plan the path only when the robot who is implemented AD* algorithm reaches the obstacle. So that though the path length will be optimal, the time will not be efficient.

We can consider our situation in another way. That is, we can plan a path from the character to the moving object once it is observed. Label the states in that path as non-reachable states. Update the new environment after states are labeled. Then according to the new environment, search for an optimized path from the start state to the goal state. This approach requires an algorithm to plan a path to a moving target. Mark Goldenberg et al. [8] introduced an improved moving-target search algorithm(MTS) that achieves a goal to plan a path from the start state to a dynamic goal state, which can change its position over time, by using heuristic search. The algorithm also solve the situation where the average speed of the target is slower than that of the agent.

Nevertheless, this approach requires multiple path planning at one time step. That is, the agent need to plan a path P_1 from the robot to the moving object first, and plan a new path P_2 from the robot to the goal state without overlapping P_1. Moreover, since the object is moving, a re-planning will always be proceeded until the object cannot be detected. Therefore, this approach will be extremely inefficient.

In fact, our situation are more like the situation that self-driving cars are facing. By drawing inspiration from Google self-driving car [2], we decide to introduce a "moving object detection system". In fact, studies to the detection system have been done. Mubbasir Kapadia et al. [9] designed a goal-directed navigation system that satisfies multiple spatial constraints, such as staying behind a building and walking along the walls, imposed on the path. But to our situation specifically, we can simplify the procedures of detection since, the agent has perfect knowledge to the map. The only thing it need to detect is the moving object.

3 Methods and Design

3.1 Proposed Algorithm

In this project, we proposed a path planning algorithm in a dynamic environment, which will plan a path from the start position to the goal position without colliding any moving object in the environment. Ideally, the planned path will be the optimal. That is, the time used to travel from the start position to the end position by walking along the path planned by our algorithm will be the shortest. We also look forward to receive zero collisions. In order to prove the optimization of the planned path, we planned to develop our algorithm based on A* algorithm. It is because the planned path introduced by A* algorithm can be proved to be the optimized solution.

Note that the path planning agent only has knowledges to the map where the character, who needs to be path planned, lives, but does not know the existence of the moving objects. We will satisfy the restrictions of A* algorithm by introducing a detection system, which will detect the surrounding area and enlarge the path planning agent's knowledge to the environment. In another word, we will divide it into two subgoals:

1. Collision prediction (see algorithm 2);
2. Path (re-)planning (see algorithm 3).

The agent first takes the start position and the assigned goal position as the parameters, as well as the map. It carries out the traditional A* algorithm and provides an initially optimized path. The character starts to proceed the path. The agent will notify a detecter, that is a plug-in for the character to "observe" the surroundings. Since the agent itself has no chance to know the existence of the moving object, we must import an external agent to help the path planning agent to gather the surrounding information. That external agent will be defined as the detecter, and we call it "collision prediction agent".

Once the collision prediction agent is activated, it will notify the hardware to detect the surrounding area. The detection hardware has the ability of determining the velocity of the moving object, as well as its moving direction, once the moving object is captured. These data will be send to the collision prediction agent. The agent will estimate a potential path of the object based on its velocity and moving direction. Then the agent compares the estimated potential path to the path which the character is currently proceeding, and see if paths have any intersection. If an intersection is found, then a collision is predicted.

After the collision is predicted, the collision prediction agent will update the map. That is, all the spots in the potential path will be labeled invalid in the map. We cannot simply mark the intersection spot as invalid since when the re-planned path may still have intersections with the potential path at other spots. The way we update the map ensures that the re-planned path will not have any overlaps with the potential path. The updated map will be sent to the path planning agent. Path planning agent will be notified to re-plan the path.

In order to avoid stopping the character while it is proceeding the path re-planning, the path planning agent will first estimate a start state based on the average time used for itself to plan a path, as well as the velocity and the moving direction of the character. Note that every time carrying out the planning, the time for planning will be record by the agent. An average time will be calculated as well. The actuarial start position will always be one step further than the estimated start position to ensure that the character has not past the start position. The goal state will be assigned after the start state is assigned. Then the agent will calculate a new path using A* algorithm. The agent will notify the character to practices the new path.

If, unfortunately, the character has past the new start position, it will be called to go back to that position first by reversing part of the original path. If the character has not reached the new start position, it will proceed the original path until it reaches the new start position.

3.2 Experiment Design

In our experiment, we will examine whether each subgoal is accomplished. Therefore, we compare the result from experiment group with that from control group. The path planner in the control group will use the traditional A* path planning algorithm, while the planner in the experiment group will use the proposed algorithm.

Algorithm 1 The proposed algorithm for path planning in dynamic environment

```
Use A* to calculate an optimized path and practice it
while moving to the goal position do
    Detect moving objects
    if a moving object is detected then
        Calculate its moving direction, velocity, etc. to predict if there will be a collision
        if there will be a collision then
            Set the current position as the start state
            Use A* to re-calculate a path
    else
        Stick with the original path
```

Algorithm 2 The proposed algorithm for collision prediction

```
if an object is detected at (x, y) then
    collect Data (i.e., velocity, direction, etc)
    calculate the path of the moving object and compare to the path of the character
    if there exists intersections between two paths then
        mark the positions in the calculated path of moving object as invalid states
        notify the path planning agent
```

Besides the experiment group and the control group, we will have two additional experiment groups, where each group will examine one of the subgoals(see Figure 3) That is, in each group we will only implement either collision prediction or path re-panning. Ideally, the proposed algorithm will perform the best in terms of time efficiency and number of collisions, but that may not be the case. It is possible that either collision prediction or path re-planning will have a high cost. These two additional experiment groups will helps us to determine which subgoal is more expensive.

3.3 Approach

The algorithm will be examined in a simulator. We choose to use Gazebo [5] since it allows us to build a 3-D world in the simulator. Gazebo also allows us to simulate different physical phenomenons. We will apply gravity to the environment so that the character and the moving object will not flow in the air. The physical collision will be simulated as well. Both the character and the moving object can collide to the walls, or they can collide into each other.

Two same simulated robots, the simulated robot of Pioneer 3-DX(p3dx), will be used to represent the character and the moving object. Since both of the character and the moving object are sharing the same properties, it is unnecessary to distinct from each other. But we will label different names to them when they are spawned. We will use ROS [10] to control the robots.

The character will be placed at a randomly chosen spot in the world, which will be regarded as the initial position. Note that, in oder to make the path planning question complex enough, we will manually create a list of start positions l_1 and a list of goal positions l_2. Each element in l_1 is pairwise far enough from all elements in l_2. That is, the length of the optimal path is greater than 15 steps. For each run, we will choose a start position and a goal position from these lists, respectively. A moving object will be randomly located as well. It will start to move after it is placed into the map.

Time will be recorded after the agent starts to calculate the path, and will be stopped when the character

Algorithm 3 The proposed algorithm for path re-planning

```
estimate a start position as the start state.
set the goal position as the goal state
re-plan the path
```

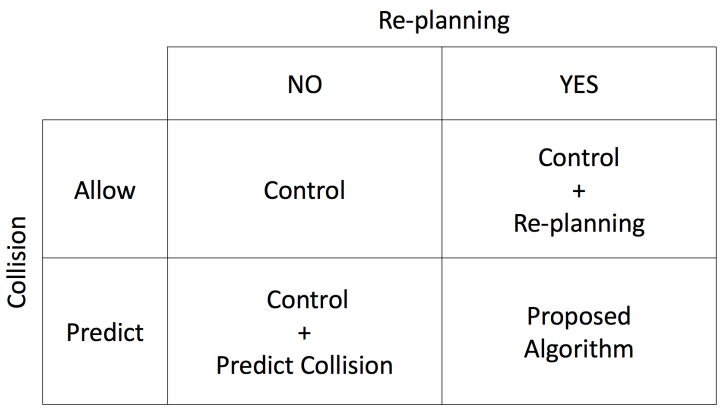

Figure 3: Experiment Design

has eventually reached the goal state. The time cost in total, including time for planning and the time for processing, will be evaluated.

If a collision happens, then the current test will be stopped. A "fail" will be reported as the result of the current test, and that test will terminate. Same procedures will be repeatedly practiced more than a hundred times for each experiment group, as well as the control group. in order to collect variable data sets.

The World
A 3-D world will be constructed by Gazebo. The size of the world is 20 units $\times 15$ units, so that the length of the optimal path can be ensured. Obstacles will be placed in the world. Note that these obstacles are cuboids with size 1 unit $\times$ 0.5 unit $\times 1$ unit, and will be placed randomly. The width of the corridor between two obstacles may not fits the size of the character. Therefore, the agent should take account of this as a condition while planning the path. Note that there are 20% of the world are walls. A world will be bounded by walls as well, so that both the character and the moving object cannot leave the world.

The world will be created by a script. It will be presented as a 2×2 matrix. We use some scripts to translate the matrix to a "world file", which Gazebo can use it to structure out the real world in the simulator.

Once the world is created, it will be fixed. Note that there is no need to change the map since the path planning agent has a perfect knowledge to the map. In another word, the differences in maps will not affect the ability of the agent solving the problem. It may only effect the motions of the moving object, which will not be examined in our experiment.

The Motion Rule
The character and the moving object share the same rule of motion. They can only move in their heading direction. In oder to make a turn, they have to change their heading direction first. They are allowed to move forward in the heading direction while they are turning theirselves. They cannot translate left or right directly. The reason we decide to make this rule is because neither in a game nor in real life can moving thing translate left or right directly. We want to make our experiment more realistic. So a turning will be observed as a curve.

The Character
The character will be assigned a max velocity and an acceleration. The acceleration will be constant before the character reached its max velocity. After the character reached its max velocity, the acceleration will be 0. The velocity cannot be negative. Our path planning algorithm will be plugged into the robot.

The character will start to move once a plan is available. The character will always moving along the planned path until a new path is introduced. Once it start to move, it will not stop, but may decelerate while it is turning.

The Moving Object
The moving object will be assigned will be assigned a max velocity and an acceleration as well. We model the moving object with some randomness in terms of moving direction. That is, the object will moving in a direction for a certain time and change the direction at some time step.

In order to achieve this goal, we will introduce an A* path planning agent to the object as well. The agent will first pick a position as a goal state, and plan a path. The object will move follow the path until it reaches the goal state. Then the agent will another position as a goal state repeat the procedures. Since the each goal state is randomly picked, the object is, overall, moving to a random place. But it will move along a fixed path, then its motion is not completely random. Note that a complete randomness is impossible to predict.

The object will never know the existence of the character. There will not be any detecters who are looking for the character. Since our project is focusing on how to avoid colliding moving object, there is no need to let the object not collide to the character. That is, the object is allowed to collide into the character.

The Lidar, the Agents and the Planned Path
The p3dx robot has a built-in lidar node. We will use that lidar to detect the moving object. The lidar have abilities of detecting the velocity and moving direction of the moving object. These data will be recorded and send to the agent in oder to the predict a collision. The lidar will be activated once the character starts to move.

There are two agents. One is the collision prediction agent, another is the path planning agent. The collision prediction agent will take data collected from lidar as parameters, and proceeds the collision prediction algorithm. A output of a calculated path of the moving object will be returned. This outcome will be used to update the map if necessary. The path planning agent has full knowledge to the map, except knowing the existence of the moving object. The agent will use A* algorithm to plan the path.

The planned path will be translated into a chain of actions, so that the character can easily follow. The character will proceed each actions until it reaches the goals state. The actions will be updated once the orientation of the robot has been changed. For example, assume the original action is to go left two steps. Once the character has turned to left, then the action becomes going forward by two steps.

3.4 Evaluation

The efficiency of the algorithm is will be evaluated from three aspects.

Time efficiency
We will measure the time spent on planning and processing the optimized path for both experiment group and control group, as well as the time of other two intermedia experiment groups. We will compare the time of experiment group to that of the control group, and see if the experiment group spend less time on planning and processing the optimized path.

This is to check whether the new algorithm is optimized in terms of time. Ideally the time used for our new algorithm will be less than that used for A*. Meanwhile, a high cost in time efficiency implies excessive number of expanded states, as well as a longer path length.

We will also compare the time of experiment group with to of intermedia experiment groups. We want to verify that our proposed algorithm is optimal in terms of time. However, the time of one or both of the intermedia experiment groups may be less than that of the experiment group, since it may have a high cost in collision prediction and path re-planning.

Times of collision
The major goal of the algorithm is to avoid collision. Therefore, the number of collisions occurred becomes significant. If a collision is visually observed, that is, the experiment terminate at time t, we will define that there is a collision at time t. Since once the collision happens, a "fail" will be reported, we will count the number of "fail" in order to count the number of collisions. Optimistically, no collision should be detected.

Table 1: Summary of the Data

	Control	Re-plan only	Prediction Only	Proposed
$\overline{PlanningTime}$(sec)	$4.51E-05$	$4.24E-05$	$4.38E-05$	$4.41E-05$
$\overline{TravelTime(sec)}$	40.61907	41.25087	42.6072	50.965657
$\overline{Num.ofCollision}$	0.35	0.21	0.112	0.08

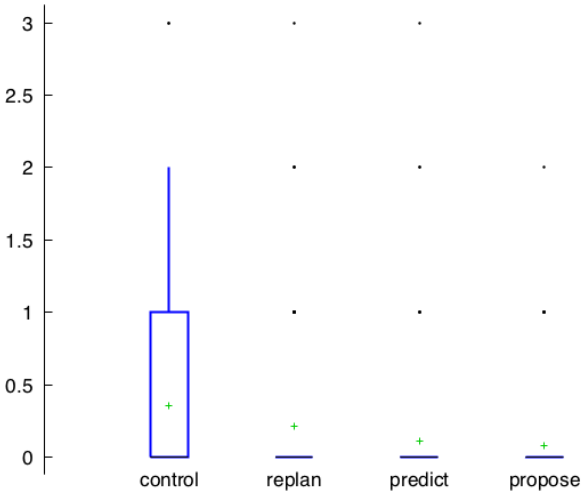

Figure 4: Distribution of Number of Collision

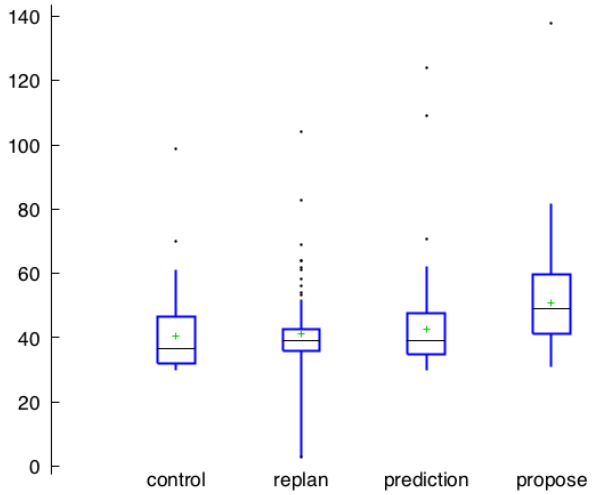

Figure 5: Distribution of Travel Time

Same procedures will be carried out for each groups.

Length of the Actual Path
Since the path will be printed out, it is easy to calculate the actual path. We will compare it to the length of idealize optimized path. If the length matches, then the actual path is optimized, otherwise, it is not. Same procedures will be carried out for each groups.

4 Result

We set up one control group, and there experiment groups. In the control group, only A* algorithm was implemented. In all of these four groups, we have collected the time of planning a path before moving, the total travel time, and the number of collisions. The planning time does not include the re-planning time. We collected these measures for 100 different runs of the simulation.

Table 1 shows the mean of the planning time, the travel time and the number of collision for each group. Figure 5 and Figure 4 show the distribution of travel time and the number of collision in each group.

The means of the planning time in each group are very small. That is because the size of the map we used for the experiment is not vary large, the path planning agent had only a few positions in the map it needed to deal with. The differences among the means of the planning time in different groups are very small as well. That was what we expected, because we were using the same A* algorithm for each group. Theoretically, the planning time for each group should be the same, but due to the hardware condition, there will be some slight differences. Therefore, we may not focus on the planning time in the discussion.

5 Discussion

Firstly, we focused on the group with proposed algorithm and the control group. The group with proposed algorithm has fewer collisions on average compared to that of the control group. We did a standard t-test on the number of collisions by setting the control group as the dependent variable and setting the group with proposed algorithm as the regressor. The p-value is 0.3097, which is larger than 5%. Therefore, the result is not statistically significant. We cannot conclude that the number of collisions in the group with proposed algorithm is decrease. We may need more data to further prove this.

We were not surprised to see this result. Since the mean of the number of collisions is less than 1 in both control group and non-control group, we can conclude that there are few collision happened during the experiment. Thus, even though the group with proposed algorithm had fewer collision in average, we may not conclude that the proposed algorithm has improved the number of collisions. And we need to collect more data to prove the statistical significancy.

We also compared the travel time between the control group and the proposed algorithm. The proposed algorithm takes 10 seconds more travel time on average, which suggests that the paths it chooses are longer than optimal. The difference is statistically significant ($p < 0.01$).

By reviewing the video record of our experiment, we found that the over predicting may be the causal of spending more time on traveling. During the experiment, the prediction collision agent was too "sensitive" that at most of time, it would predict a collision. And it would noticed the path planning agent to re-plan the path. That is, the robot changed its path in most of runs. Therefore, travel time increased. Since our collision prediction algorithm assumes the heading direction of the obstacle will not change once it is detected. Unless the heading of the obstacle is in the same direction as that of the robot, there must be a intersection between robot's path and obstacle's path, which will be regarded as a collision. Therefore, the collision prediction agent will over predict in most of time.

Besides doing t-test on the original travel time, we did a standard t-test on the normalized travel time. The way we normalized the travel time was to use the actual travel time we collected from the experiment divided by the ideal travel time. Since we know the starting position and the goal position in each run, we can use A* algorithm to calculate the ideal path, which will ignore the moving obstacle. Since we know the velocity of the robot in each run as well, we can calculate the ideal travel time. We did a standard t-test based on the normalized travel time by setting the control group as the dependent variable and setting the group with proposed algorithm as the regressor. The p-value is 0.0011, which means the result is statistically significant.

Secondly, we did same tests between other two non-control groups and the control group. That is, we did a standard t-test on the travel time by setting the control group as dependent variable, and setting the group with only path re-plan algorithm only. The p-value is 4.47×10^{-7}, which is less than 5%. That means the result is statistically significant. The p-value of the standard t-test on number of collisions between control group, the dependent variable, and the group with re-plan algorithm only, regressor, is 0.1536. That means the number of collisions is not statistically significant. The p-value of the standard t-test on normalized travel time between control group, the dependent variable, and the group with re-plan algorithm only, regressor, is 4.47×10^{-7} which means normalized travel time is statistically significant.

Note that, from table 1, we see a decrease in the number of collisions between the control group and the group with re-plan algorithm only. However, we are not expect to see this since the robot using re-plan algorithm only was allowed to collide the obstacle. The robot has no chance to avoid a collision. It should have a similar number of collisions to that of the control group. But because the obstacle was changing its moving path in each run, the chance of colliding the obstacle in each run is different. Therefore, it is possible to have less collisions even the robot does not have a collision prediction algorithm implemented.

The p-value of t-test on number of collisions between the control group, dependent variable, and the group with collision prediction algorithm only, regressor, is 0.8516. That means we cannot tell differences between the number of collisions in the control group and that in the group with collision prediction algorithm only. The p-value of t-test on travel time between the control group, dependent variable, and the group with collision prediction algorithm only, regressor, is 0.1554. That means we cannot tell differences between the travel time in the control group and that in the group with collision prediction algorithm only.The p-value of t-test on normalized travel time between the control group, dependent variable, and the group with collision prediction algorithm only, regressor, is 0.1554. That means we cannot tell differences

between the normalized travel time in the control group and that in the group with collision prediction algorithm only.

Further more, in order to see the differences among the non-control groups, we did standard t-tests among these groups. We set the group with proposed algorithm as dependent, and setting the other groups as regressor. The p-value of t-test on the number of collisions between the group with proposed algorithm and the group with re-plan algorithm only is 0.1690. The p-value of t-test on the number of collisions between the group with proposed algorithm and the group with collision prediction algorithm only is 0.3007. This shows that we may not be able to tell differences among the numbers of collisions in these non-control groups.

The p-value of t-test on travel time between the group with proposed algorithm and the the group with re-plan algorithm only is 6.46×10^{-9}. The p-value of t-test on travel time between the group with proposed algorithm and the group with collision prediction only is 6.6×10^{-5}. The p-values show that there are differences among the travel times in these non-control groups.

6 Conclusion and Future Work

The statistical analyses implies that we cannot conclude the proposed algorithm has reduced the number of collisions, even though the proposed algorithm has the lowest number of collisions in average. There is no evidence showing that either the re-plan algorithm or the collision prediction algorithm has reduced as well. We will run more experiments on our proposed algorithm, and collect more data, in order to further test the statistical significance of the number of collisions.

However, since the travel time of the proposed algorithm is statistically significant, we conclude that the proposed algorithm has a poor performance in terms of travel time. The robot using the proposed algorithm would spend more time in traveling than the robot using traditional A* algorithm would. Also, either the robot using re-plan algorithm only or the robot using collision prediction only would spend less time in traveling than the robot using the proposed algorithm would. This supports the fact that the proposed algorithm has the worst travel time.

In the future, we want to improve our algorithm to make it more compatible in terms of travel time. We will focus more on improving the collision prediction algorithm because we believe, for now, our collision prediction algorithm is too naive. Since our navigation algorithm is more likely to be used in the scenario where the robot is automatically traveling in a crowed of people, we may predict the destination of a walking person. Then we can use the predicted destination to predict that person's moving trail.

Moreover, we will increase the number of moving obstacles in the map to simulate a more realistic scenario. We may need to change the collision prediction algorithm since the algorithm we are using can only detect one obstacle, and predict its moving trail. Also, we will increase the size of the map because for now, the map size is too small that our algorithm can easily plan a path in a short time. But we want to check if the algorithm still can plan a path in a short time once the map gets larger.

In conclusion, we will upgrade our algorithm to make it more applicable to the real world. We look forward to implementing our algorithm into a robot so that it can travel in the dynamic environment safely and fast.

References

[1] Mark Goldenberg, Alexander Kovarsky, Xiaomeng Wu, and Jonathan Schaeffer. Multiple agents moving target search. *IJCAI*, 2003.

[2] Google. Navigating city streets, 2014.

[3] Peter E. Hart, Nils J. Nilsson, and Bertram Raphael. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on 4.2*, pages 100–107, 1968.

[4] Peter E Hart, Nils J Nilsson, and Bertram Raphael. Correction to a formal basis for the heuristic determination of minimum cost paths. *ACM SIGART Bulletin*, pages 28–29, 1972.

[5] Andrew Howard and Nate Koenig. Gazebo, 2002.

[6] Blizzard Entertainment Inc. Warcraft iii, 2003.

[7] Riot Games Inc. League of legends, 2009.

[8] Toru Ishida and Richard E Korf. Moving-target search: A real-time search for changing goals. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(6):609–619, 1995.

[9] Mubbasir Kapadia, Kai Ninomiya, Alexander Shoulson, Francisco Garcia, and Norman Badler. Constraint-aware navigation in dynamic environments. In *Proceedings of Motion on Games*, pages 111–120. ACM, 2013.

[10] Stanford Artificial Intelligence Laboratory. Robot operating system(ros), 2007.

[11] Maxim Likhachev, David I Ferguson, Geoffrey J Gordon, Anthony Stentz, and Sebastian Thrun. Anytime dynamic a*: An anytime, replanning algorithm. In *ICAPS*, pages 262–271, 2005.

[12] V Lumelsky and A Stepanov. Dynamic path planning for a mobile automaton with limited information on the environment. *IEEE transactions on Automatic control*, 31(11):1058–1063, 1986.

[13] Anthony Stentz. Optimal and efficient path planning for unknown and dynamic environments. Technical report, DTIC Document, 1993.

[14] Anthony Stentz. Optimal and efficient path planning for partially-known environments. In *Robotics and Automation, 1994. Proceedings., 1994 IEEE International Conference on*, pages 3310–3317. IEEE, 1994.

[15] Anthony Stentz. The focussed d* algorithm for real-time replanning. *IJCAI.*, 95, 1995.

[16] Jung-Ying Wang and Yong-Bin Lin. Game ai: Simulating car racing game by applying pathfinding algorithms. *International Journal of Machine Learning and Computing*, 2(1):13, 2012.

[17] Sule Yildirim and Sindre Berg Stene. A survey on the need and use of ai in game agents. In *Proceedings of the 2008 Spring simulation multiconference*, pages 124–131. Society for Computer Simulation International, 2008.

Appedix

Row data

Control				Control			
Run	Tplan (Sec)	Ttravel (Sec)	Collision	Run	Tplan (Sec)	Ttravel (Sec)	Collision
1	4.39E-05	30.994	0	51	3.60E-05	34.094	0
2	4.39E-05	59.128	1	52	3.22E-05	31.948	0
3	4.20E-05	52.928	1	53	4.39E-05	31.948	0
4	3.19E-05	31.948	0	54	3.60E-05	57.935	1
5	3.50E-05	47.922	1	55	3.19E-05	32.902	0
6	3.10E-05	38.866	0	56	3.12E-05	31.948	0
7	3.22E-05	31.948	0	57	3.29E-05	33.14	0
8	4.10E-05	46.968	1	58	0.0001311	33.855	0
9	3.19E-05	32.128	0	59	3.22E-05	31.948	0
10	3.00E-05	49.114	1	60	3.00E-05	30.994	0
11	3.22E-05	29.802	0	61	6.82E-05	61.035	2
12	4.20E-05	41.007	0	62	3.19E-05	30.994	0
13	4.20E-05	44.107	0	63	3.19E-05	33.14	0
14	4.22E-05	41.007	0	64	4.10E-05	58.889	2
15	4.10E-05	42.915	0	65	3.19E-05	48.875	1
16	3.50E-05	36.001	0	66	3.70E-05	36.954	0
17	4.20E-05	41.007	0	67	3.00E-05	32.901	0
18	3.19E-05	40.054	0	68	4.29E-05	31.948	0
19	9.80E-05	98.943	3	69	4.10E-05	45.061	0
20	4.10E-05	59.127	2	70	4.20E-05	41.008	0
21	3.60E-05	31.948	0	71	3.10E-05	31.948	0
22	0.0001509	54.121	1	72	3.19E-05	30.04	0
23	5.01E-05	31.948	0	73	3.10E-05	31.948	0
24	2.88E-05	32.187	0	74	5.48E-05	51.021	1
25	6.39E-05	33.14	0	75	4.22E-05	32.901	0
26	4.20E-05	51.021	1	76	2.98E-05	31.232	0
27	9.61E-05	33.855	0	77	4.10E-05	40.532	0
28	3.19E-05	30.041	0	78	6.29E-05	46.968	0
29	6.60E-05	45.061	0	79	4.20E-05	50.067	1
30	3.22E-05	30.994	0	80	3.19E-05	40.054	0
31	4.41E-05	31.948	0	81	4.10E-05	53.382	1
32	3.79E-05	40.054	0	82	3.19E-05	30.04	0
33	3.19E-05	33.14	0	83	4.39E-05	50.067	1
34	4.10E-05	41.961	0	84	4.10E-05	39.815	0
35	4.39E-05	30.994	0	85	4.41E-05	48.875	1
36	3.48E-05	35.047	0	86	3.19E-05	31.948	0
37	0.0001459	46.015	1	87	3.10E-05	32.901	0
38	6.79E-05	39.101	0	88	4.32E-05	57.935	1
39	4.41E-05	31.948	0	89	3.50E-05	37.908	0
40	5.41E-05	34.809	0	90	0.0001349	43.869	0
41	4.10E-05	33.14	0	91	4.60E-05	30.994	0
42	4.29E-05	41.008	0	92	6.89E-05	61.035	2
43	4.20E-05	41.008	0	93	2.81E-05	30.941	0
44	4.10E-05	34.809	0	94	3.19E-05	31.948	0
45	3.19E-05	31.948	0	95	4.39E-05	31.948	0
46	3.10E-05	30.994	0	96	4.10E-05	59.843	1
47	4.48E-05	41.961	0	97	4.20E-05	45.893	1
48	4.70E-05	33.855	0	98	4.20E-05	61.035	2
49	5.29E-05	42	0	99	4.48E-05	70.095	3
50	3.60E-05	33.855	0	100	4.10E-05	53.382	1

Figure 6: Row data for control group

replan only				replan only			
Run	Tplan	Ttravel	Collision	Run	Tplan	Ttravel	Collision
1	3.29E-05	54.127	1	51	3.31E-05	36.007	0
2	3.00E-05	36.007	0	52	3.50E-05	36.961	0
3	4.60E-05	46.021	1	53	3.89E-05	41.014	0
4	4.60E-05	46.021	0	54	3.29E-05	38.153	0
5	3.39E-05	43.16	0	55	3.31E-05	44.829	0
6	3.41E-05	34.815	0	56	4.20E-05	40.061	0
7	5.41E-05	68.909	2	57	3.91E-05	36.961	0
8	4.01E-05	40.061	0	58	5.01E-05	41.014	0
9	4.20E-05	37.915	0	59	3.50E-05	36.961	0
10	4.48E-05	37.2	0	60	4.32E-05	40.776	0
11	4.72E-05	50.789	1	61	3.91E-05	41.968	0
12	0.0001721	103.957	3	62	3.50E-05	41.968	0
13	4.29E-05	35.054	0	63	4.48E-05	45.067	1
14	3.19E-05	33.862	0	64	3.50E-05	39.107	0
15	4.70E-05	51.028	1	65	6.20E-05	64.141	0
16	3.41E-05	36.007	0	66	4.70E-05	31.001	0
17	3.39E-05	34.1	0	67	3.48E-05	39.107	0
18	9.20E-05	61.041	1	68	5.20E-05	61.995	2
19	3.39E-05	41.968	0	69	9.11E-05	46.975	1
20	4.82E-05	33.862	0	70	7.10E-05	82.976	2
21	3.29E-05	39.107	0	71	3.39E-05	35.054	0
22	4.29E-05	39.107	0	72	3.60E-05	36.961	0
23	2.88E-05	32.193	0	73	3.41E-05	35.054	0
24	3.41E-05	36.961	0	74	5.29E-05	64.141	1
25	3.29E-05	36.007	0	75	3.41E-05	37.915	0
26	3.91E-05	42.922	0	76	5.10E-05	2.922	0
27	3.41E-05	33.862	0	77	3.41E-05	36.961	0
28	6.29E-05	33.146	0	78	3.29E-05	37.2	0
29	4.39E-05	43.16	0	79	3.91E-05	37.915	0
30	3.89E-05	40.061	0	80	3.29E-05	34.1	0
31	3.29E-05	34.1	0	81	7.30E-05	45.067	1
32	5.98E-05	39.107	0	82	3.41E-05	36.961	0
33	3.39E-05	36.007	0	83	3.39E-05	38.153	0
34	3.70E-05	37.2	0	84	2.90E-06	35.054	0
35	3.41E-05	38.869	0	85	4.39E-05	39.107	0
36	3.41E-05	36.961	0	86	3.70E-05	37.915	0
37	3.89E-05	44.114	0	87	6.20E-05	41.014	0
38	3.39E-05	38.153	0	88	5.79E-05	39.107	0
39	3.39E-05	39.107	0	89	3.41E-05	43.875	0
40	3.41E-05	35.769	0	90	4.20E-05	37.2	0
41	6.41E-05	41.968	0	91	4.70E-05	31.954	0
42	3.39E-05	53.174	0	92	4.70E-05	58.18	1
43	3.00E-05	31.954	0	93	3.41E-05	33.862	0
44	6.10E-05	41.968	0	94	4.29E-05	40.061	0
45	3.39E-05	39.107	0	95	4.70E-05	40.061	0
46	3.10E-05	32.193	0	96	3.70E-05	39.107	0
47	5.41E-05	56.035	1	97	3.29E-05	37.2	0
48	3.00E-05	31.001	0	98	3.70E-05	51.982	1
49	3.41E-05	32.908	0	99	3.10E-05	32.908	0
50	3.29E-05	37.915	0	100	3.29E-05	41.014	0

Figure 7: Row data for group with re-plan algorithm only

predicion only				predicion only			
Run	Tplan	Ttravel	Collision	Run	Tplan	Ttravel	Collision
1	4.41E-05	70.916	2	51	3.50E-05	43.007	0
2	3.60E-05	35.153	0	52	3.10E-05	34.915	0
3	4.79E-05	36.107	0	53	5.20E-05	124.083	3
4	3.39E-05	32.054	0	54	3.50E-05	32.054	0
5	3.31E-05	39.908	0	55	0.0001371	52.796	0
6	3.41E-05	39.922	0	56	3.50E-05	36.107	0
7	3.79E-05	109.063	2	57	3.48E-05	37.061	0
8	3.29E-05	32.054	0	58	3.48E-05	34.2	0
9	3.50E-05	32.054	0	59	4.29E-05	34.2	0
10	9.51E-05	62.095	0	60	6.41E-05	48.028	0
11	3.41E-05	40.862	0	61	3.50E-05	42.054	0
12	6.10E-05	59.949	0	62	4.01E-05	39.922	0
13	4.82E-05	30.001	0	63	3.50E-05	36.107	0
14	3.50E-05	33.007	0	64	3.41E-05	34.915	0
15	3.41E-05	39.922	0	65	3.19E-05	30.862	0
16	4.39E-05	51.127	0	66	3.60E-05	38.014	0
17	4.39E-05	50.174	0	67	3.00E-05	40.001	0
18	3.41E-05	30.862	0	68	3.41E-05	44.915	0
19	3.41E-05	30.862	0	69	7.10E-05	42.783	0
20	3.60E-05	43.007	0	70	3.81E-05	33.961	0
21	3.79E-05	40.16	0	71	6.01E-05	60.902	1
22	4.51E-05	58.995	0	72	4.72E-05	52.796	0
23	3.41E-05	36.107	0	73	7.10E-05	42.067	0
24	4.48E-05	50.174	0	74	3.60E-05	32.054	0
25	5.58E-05	38.014	0	75	4.41E-05	44.928	0
26	6.10E-05	59.949	1	76	4.01E-05	43.961	0
27	3.41E-05	33.961	0	77	3.41E-05	30.862	0
28	5.79E-05	37.061	0	78	5.70E-05	37.061	0
29	4.82E-05	38.014	0	79	4.48E-05	48.028	0
30	3.19E-05	30.954	0	80	4.82E-05	50.889	1
31	3.39E-05	33.007	0	81	5.70E-05	43.975	0
32	3.91E-05	38.014	0	82	4.41E-05	45.882	0
33	4.20E-05	34.915	0	83	3.50E-05	48.028	0
34	3.60E-05	38.968	0	84	4.01E-05	39.922	0
35	3.70E-05	35.869	0	85	5.89E-05	34.915	0
36	5.89E-05	53.035	0	86	3.50E-05	33.007	0
37	3.41E-05	33.961	0	87	3.60E-05	44.879	0
38	3.29E-05	34.2	0	88	3.89E-05	37.061	0
39	4.70E-05	38.014	0	89	4.51E-05	47.789	0
40	4.41E-05	48.028	0	90	4.48E-05	36.107	0
41	5.29E-05	52.081	0	91	3.50E-05	34.915	0
42	4.79E-05	32.054	0	92	5.91E-05	46.836	0
43	4.98E-05	42.067	0	93	4.01E-05	46.836	0
44	3.50E-05	35.153	0	94	4.29E-05	48.982	0
45	4.51E-05	48.982	0	95	3.79E-05	38.968	0
46	3.10E-05	38.001	0	96	3.50E-05	34.915	0
47	3.48E-05	35.153	0	97	6.20E-05	41.114	0
48	3.48E-05	33.961	0	98	4.60E-05	49.935	0
49	5.39E-05	58.995	1	99	4.51E-05	43.961	0
50	3.48E-05	36.107	0	100	6.20E-05	36.107	0

Figure 8: Row data for group with collision prediction algorithm only

Proposed algorithm				Proposed algorithm			
Run	Tplan	Ttravel	collision	Run	Tplan	Ttravel	collision
1	4.51E-05	79.87	1	51	3.60E-05	44.107	0
2	4.48E-05	59.127	0	52	4.41E-05	49.829	0
3	5.48E-05	51.021	0	53	3.31E-05	42.915	0
4	4.20E-05	51.021	0	54	5.10E-05	41.961	0
5	4.20E-05	50.067	0	55	3.60E-05	46.0147	0
6	3.19E-05	35.007	0	56	3.29E-05	41.961	0
7	3.29E-05	41.961	0	57	3.19E-05	41.961	0
8	3.31E-05	33.961	0	58	4.48E-05	61.035	0
9	5.51E-05	51.021	0	59	3.29E-05	40.054	0
10	4.60E-05	60.081	0	60	4.51E-05	59.127	0
11	3.31E-05	56.028	0	61	4.29E-05	75.101	0
12	3.29E-05	49.114	0	62	3.60E-05	46.968	0
13	4.01E-05	41.961	0	63	4.41E-05	48.875	0
14	3.29E-05	37.908	0	64	4.51E-05	31.961	0
15	3.29E-05	35.007	0	65	7.51E-05	63.896	0
16	3.48E-05	40.054	0	66	4.51E-05	60.081	0
17	3.29E-05	41.007	0	67	4.29E-05	51.021	0
18	4.39E-05	60.081	1	68	4.48E-05	48.16	0
19	5.51E-05	55.074	0	69	3.29E-05	40.054	0
20	3.29E-05	41.007	0	70	4.48E-05	78.201	0
21	3.10E-05	38.146	0	71	3.39E-05	40.054	0
22	3.31E-05	41.961	0	72	4.51E-05	60.081	0
23	5.41E-05	58.889	0	73	2.81E-05	34.093	0
24	5.29E-05	51.975	0	74	4.51E-05	58.889	0
25	3.29E-05	40.054	0	75	3.19E-05	42.2	0
26	4.41E-05	58.889	0	76	6.70E-05	81.777	1
27	5.51E-05	51.975	0	77	3.29E-05	49.114	0
28	0.000221	61.988	0	78	3.19E-05	39.1	0
29	3.29E-05	41.961	0	79	4.51E-05	60.796	0
30	3.29E-05	40.054	0	80	5.22E-05	38.862	0
31	3.19E-05	38.915	0	81	4.41E-05	49.114	0
32	4.20E-05	33.915	0	82	3.41E-05	41.961	0
33	3.29E-05	56.028	0	83	3.29E-05	31.007	0
34	4.89E-05	52.213	0	84	3.29E-05	42.915	0
35	4.41E-05	59.843	0	85	4.10E-05	45.061	0
36	3.29E-05	138.044	2	86	3.22E-05	59.843	0
37	4.39E-05	62.942	0	87	4.39E-05	60.081	0
38	4.29E-05	60.081	0	88	4.51E-05	48.875	0
39	4.29E-05	68.187	1	89	3.19E-05	41.007	0
40	0.000104	57.935	0	90	3.31E-05	41.961	0
41	4.39E-05	60.081	0	91	3.89E-05	42.2	0
42	5.10E-05	58.889	0	92	0.0001349	58.889	1
43	4.41E-05	31.961	0	93	3.19E-05	41.961	0
44	3.31E-05	56.982	0	94	8.92E-05	67.949	0
45	4.51E-05	67.949	0	95	4.60E-05	43.153	0
46	3.19E-05	41.961	0	96	4.51E-05	60.081	0
47	4.51E-05	60.081	0	97	3.50E-05	47.922	0
48	3.19E-05	39.1	0	98	2.81E-05	34.809	0
49	4.39E-05	69.141	0	99	3.48E-05	40.054	0
50	3.29E-05	41.961	0	100	4.32E-05	66.995	1

Figure 9: Row data for group with proposed algorithm only

T-tests

1. T-test on travel time depend on control group:

Model 1: OLS, using observations 1–100
Dependent variable: control

	Coefficient	Std. Error	t-ratio	p-value
replan	0.491770	0.104086	4.725	4.47×10^{-7}
prediction	0.130263	0.0909668	1.432	0.1554
propose	0.263318	0.0783529	3.361	0.0011

Mean dependent var	40.61907	S.D. dependent var	11.29604
Sum squared resid	17902.93	S.E. of regression	13.58552
Uncentered R^2	0.899208	Centered R^2	-0.417218
$F(3, 97)$	288.4607	P-value(F)	3.48e–48
Log-likelihood	−401.2713	Akaike criterion	808.5427
Schwarz criterion	816.3582	Hannan–Quinn	811.7057

2. T-test on number of collisions depend on control group:

Model 1: OLS, using observations 1–100
Dependent variable: control

	Coefficient	Std. Error	t-ratio	p-value
replan	0.195380	0.135839	1.438	0.1536
predict	−0.0317988	0.169567	−0.1875	0.8516
propose	0.247746	0.242590	1.021	0.3097

Mean dependent var	0.350000	S.D. dependent var	0.672324
Sum squared resid	54.92090	S.E. of regression	0.752459
Uncentered R^2	0.036475	Centered R^2	-0.227283
$F(3, 97)$	1.224020	P-value(F)	0.305193
Log-likelihood	−111.9300	Akaike criterion	229.8601
Schwarz criterion	237.6756	Hannan–Quinn	233.0232

3. T-test on normalized travel time depend on control group:

Model 1: OLS, using observations 1–100
Dependent variable: control_n

	Coefficient	Std. Error	t-ratio	p-value
replan_n	0.491770	0.104086	4.725	4.47×10^{-7}
prediction_n	0.130263	0.0909668	1.432	0.1554
propose_n	0.263318	0.0783529	3.361	0.0011

Mean dependent var	1.329767	S.D. dependent var	0.369804
Sum squared resid	19.18737	S.E. of regression	0.444756
Uncentered R^2	0.899208	Centered R^2	-0.417218
$F(3, 97)$	288.4607	P-value(F)	3.48e–48
Log-likelihood	−59.34796	Akaike criterion	124.6959
Schwarz criterion	132.5114	Hannan–Quinn	127.8590

4. T-test on number of collisions depend on group with proposed algorithm:

Model 1: OLS, using observations 1–100
Dependent variable: propose

	Coefficient	Std. Error	t-ratio	p-value
predict	0.0730594	0.0702217	1.040	0.3007
replan	0.0776256	0.0560176	1.386	0.1690

Mean dependent var	0.080000	S.D. dependent var	0.307482
Sum squared resid	9.621005	S.E. of regression	0.313327
Uncentered R^2	0.037900	Centered R^2	-0.027885
$F(2, 98)$	1.930233	P-value(F)	0.150592
Log-likelihood	−24.83278	Akaike criterion	53.66556
Schwarz criterion	58.87590	Hannan–Quinn	55.77427

5. T-test on travel time depend on group with proposed algorithm:

Model 4: OLS, using observations 1–100
Dependent variable: propose

	Coefficient	Std. Error	t-ratio	p-value
prediction	0.450616	0.108083	4.169	6.6×10^{-5}
replan	0.717995	0.112903	6.359	6.46×10^{-9}

Mean dependent var	50.96566	S.D. dependent var	14.41254
Sum squared resid	30063.72	S.E. of regression	17.51493
Uncentered R^2	0.892750	Centered R^2	-0.461928
$F(2, 98)$	407.8762	P-value(F)	3.09e–48
Log-likelihood	−427.1891	Akaike criterion	858.3781
Schwarz criterion	863.5885	Hannan–Quinn	860.4868

Detailed Statistical Summary

1. Control Group:

Summary Statistics, using the observations 1–100

Variable	Mean	Median	Minimum	Maximum
TplanSec	4.5061e–05	4.1008e–05	2.8133e–05	0.00015092
TtravelSec	40.619	36.477	29.802	98.943
Collision	0.35000	0.0000	0.0000	3.0000

Variable	Std. Dev.	C.V.	Skewness	Ex. kurtosis
TplanSec	2.3067e–05	0.51191	3.1627	10.204
TtravelSec	11.296	0.27810	1.9028	5.7481
Collision	0.67232	1.9209	2.0619	4.0007

Variable	5% perc.	95% perc.	IQ Range	Missing obs.
TplanSec	3.0041e–05	9.7895e–05	1.1921e–05	0
TtravelSec	30.944	60.975	14.782	0
Collision	0.0000	2.0000	1.0000	0

2. Group with re-plan algorithm only:

Summary Statistics, using the observations 1–100

Variable	Mean	Median	Minimum	Maximum
Tplan	4.2396e–05	3.6478e–05	2.9018e–06	0.00017214
Ttravel	41.251	38.988	2.9220	103.96
Collision	0.21000	0.0000	0.0000	3.0000

Variable	Std. Dev.	C.V.	Skewness	Ex. kurtosis
Tplan	1.8160e–05	0.42835	4.1172	25.184
Ttravel	11.415	0.27671	2.2333	10.485
Collision	0.53739	2.5590	2.8823	8.7727

Variable	5% perc.	95% perc.	IQ Range	Missing obs.
Tplan	3.0088e–05	7.0703e–05	1.2875e–05	0
Ttravel	31.966	64.034	6.6765	0
Collision	0.0000	1.0000	0.0000	0

3. Group with collision prediction algorithm only:

Summary Statistics, using the observations 1–100

Variable	Mean	Median	Minimum	Maximum
Tplan	4.3802e–05	3.8981e–05	3.0041e–05	0.00013709
Ttravel	42.607	38.968	30.001	124.08
Collision	0.11000	0.0000	0.0000	3.0000

Variable	Std. Dev.	C.V.	Skewness	Ex. kurtosis
Tplan	1.4535e–05	0.33184	3.3159	16.461
Ttravel	13.564	0.31836	3.5228	16.643
Collision	0.44710	4.0646	4.5817	21.972

Variable	5% perc.	95% perc.	IQ Range	Missing obs.
Tplan	3.1996e–05	6.4027e–05	1.3113e–05	0
Ttravel	30.867	60.854	12.636	0
Collision	0.0000	1.0000	0.0000	0

4. Group with proposed algorithm:

Summary Statistics, using the observations 1–100

Variable	Mean	Median	Minimum	Maximum
Tplan	4.4117e–05	4.1962e–05	2.8133e–05	0.00022101
Ttravel	50.966	48.995	31.007	138.04
collision	0.080000	0.0000	0.0000	2.0000

Variable	Std. Dev.	C.V.	Skewness	Ex. kurtosis
Tplan	2.3218e–05	0.52629	5.3307	34.436
Ttravel	14.413	0.28279	2.4023	11.625
collision	0.30748	3.8435	4.0866	17.422

Variable	5% perc.	95% perc.	IQ Range	Missing obs.
Tplan	3.1948e–05	7.4697e–05	1.2159e–05	0
Ttravel	33.968	74.803	18.598	0
collision	0.0000	1.0000	0.0000	0