**Multi-robot Motion Planning Algorithm Using D\* ROS Implementation Report**

Zejian Zhou

1. SUMMARY

This report describes the process and the result of implementing multi-robot motion planning algorithms using D\*. The algorithm is implemented over ROS. The introduction part gives a short introduction to both algorithms and ROS. The simulation is implemented step by step.

1. INTRODUCTION

There are plenty of motion planning algorithms now, they all can provide the optimal path in a known map. Most of the motion planning algorithms assume that the environment is completely known before the robot begins its traverse[1]. Algorithms like A\* compute the minimum cost from the start position to the goal position. They cannot react to possible map changes.

However, robot working environment cannot always be completely known or the environment may not remain static during robots’ moving process especially in dynamic circumstances like a multi-robot system. In order to adapt to environment changes, there are 2 approaches. The first one is to run motion-planning algorithms every time the environment changes. The second one is developing a new algorithm to fit the changed environment. The first approach is clearly not efficient compared to the second one in most circumstances. The D\* algorithm is one of the dynamic motion planning algorithms.

The core concept of the D\* algorithm is to compute costs from the goal position until it reaches the start position. Every step of the D\* algorithm is designed to assure minimum cost to the goal position. If there are more than one robots in the working area, collisions must be avoided. Two effective ways to avoid collision are detour and velocity control. Detour should be considered first because it usually costs less time and energy than velocity control.

However, a detour cannot solve all multiple robots motion planning problems. If the map is extremely small and the number of robots is large, detour without velocity control may cause deadlock easily. Deadlock leads to motion planning failure. When deadlock happens, some robots need to wait for other robots. The priority of robots is decided by the importance of them.

The rest of the report is organized as follows. In section 3, there is a brief introduction about D\* algorithm. Section 4 describes the multi-robot motion planning algorithm implemented in this simulation. The simulation process and results are covered in section 5. Section 6 provides a discussion session concerned with improvement on the multi-robot motion planning algorithm.

1. D\* MOTION PLANNING ALGORITHM

When a robot moves from the start position to goal position, its path can be described as a list of states . Therefore, the path is .

These states are connected by directional arcs, each of which has an associated cost. [1]The goal position is also a state and let’s denote it as G. Each state has a pointer which points to the next state. G state’s pointer points to null because it is the last state. If two states are connected, the arc cost between them is calculated by arc function . The cost arc usually equals the Euclidian distance between those states.

D\* uses the OPEN list to store the states waiting to be visited. The OPEN list is sorted by the cost function . If a state is visited before and this state’s cannot be reduced, it will be put into the CLOSED list. While if a state hasn’t been visited before, it’s called a NEW state.

The key feature of the D\* algorithm is the RAISE state and the LOWER state. Every time we run the PROCESS-STATE function, a minimum in the OPEN list will be picked and is denoted as . Then the state who owns will be moved out of the OPEN list. We then denote the new minimum cost in the OPEN list as . It is obvious that is not possible. Therefore, there are two possibilities in total. The first one is , which is called the RAISE state. The second one is , which is called the LOWER state. If a state is lower state, it means it’s current cannot be reduced. But if a state is RAISE state, it means the of the first element in the OPEN list can be reduced. Hence it is necessary to check its neighbors to find out how to reduce current cost.

The PROCESS-STATE function can make sure that every state in the CLOSED list has already obtained the minimum cost to the goal position. Keep running the PROCESS-STATE function until the goal state G is in the CLOSED list. If the OPEN list is empty before G goes to the CLOSED list, it means no path can be found between the start state and the goal state.

The D\* algorithm uses the MODIFY-COST function to modify the path to adapt to map changes. After the original path was planned, the robot can start moving. If a new obstacle suddenly shows up and was detected by the robot’s sensors, it needs to call the MODIFY-COST function to add new obstacles on the map and call the PROCESS-STATE again. There’s no need to do the whole path planning again. That’ why D\* algorithm’s performance is much better in a dynamic environment.

1. MULTI-ROBOT MOTION PLANNING ALGORITHM

Multiple robots motion planning can be relatively expensive because of massive high dimensional matrix calculation. Therefore, we decompose the problem into two modules, path planning and velocity planning. Several assumptions as follows are in need for this algorithm:

1. Each robot has an assigned goal, and the robot knows its start and goal positions; [2]
2. Robots operate in either indoor or outdoor environments, and have a pre-defined map. Specifically, in the indoor environment, the map defines the static polygonal obstacles in the environment; in the outdoor environment, the map defines terrain elevation and traversability based on a grid representation of the terrain. [2]
3. Robots’ onboard sensors detect the discrepancy between the pre-defined map and the environment and revise the map online. [2]
4. Robots are equipped with communication devices so that they can broadcast messages to others, and the communication is reliable. [2]
5. A robot’s motion control layer tracks pre-assigned trajectories within a small margin of error. [2]
6. Robots move at constant fixed speeds.
7. Robots can switch instantaneously between a fixed speed and halting.[2]

These assumptions make the multi-robot motion planning problem much easier so that we can focus on the main problem.

There are 6 key steps in this algorithm. The first one is path planning. This is the step that every robot in the working field plans their own optimal path using the D\* algorithm. Though it can be done by the original D\* algorithm, we can still come up with a better cost function. The following cost function is recommended by Dr. Guo:

Where: is Euclidian distance, is the slop of terrain, is penalty on obstacles, is the penalty for turning, are parameters to control the weight of every aspect.

After the path is found, the second step is to broadcast path information across all robots. While receiving other fellow robots’ path, a collision check is made immediately. If more than one robot shares the same state at the same time, this state can be considered as a collision state in the coordination diagram.

Step 3 is constructing the coordination diagram. The coordinate diagram is defined by the result of the collision check. The dimension of the coordinate diagram is equal to the number of robots. Every collision state should be marked as an obstacle in related robots’ internal maps.

Step 4: Run D\* in the coordination diagram. Dr. Guo suggests a new cost function which is designed for the coordination diagram:

Where is penalty on obstacles, is the Euclidian distance in N-dimensional, is the sum of every robots’ idle time, is penalty on giving way to others.

While searching in the coordination diagram, the searching time can be extremely high if the dimension of the coordination diagram is high (i.e. large number of robots). However, the cost can be significantly lower if N can be pre-fixed.

The output of running D\* in the coordination diagram can be organized as velocity profile and performance index .

Step 5 is broadcasting and across robots via the communication system.

The final step is to compare all submitted by robots in the working area and find the minimum , choose the corresponding as the optimal solution for velocity.

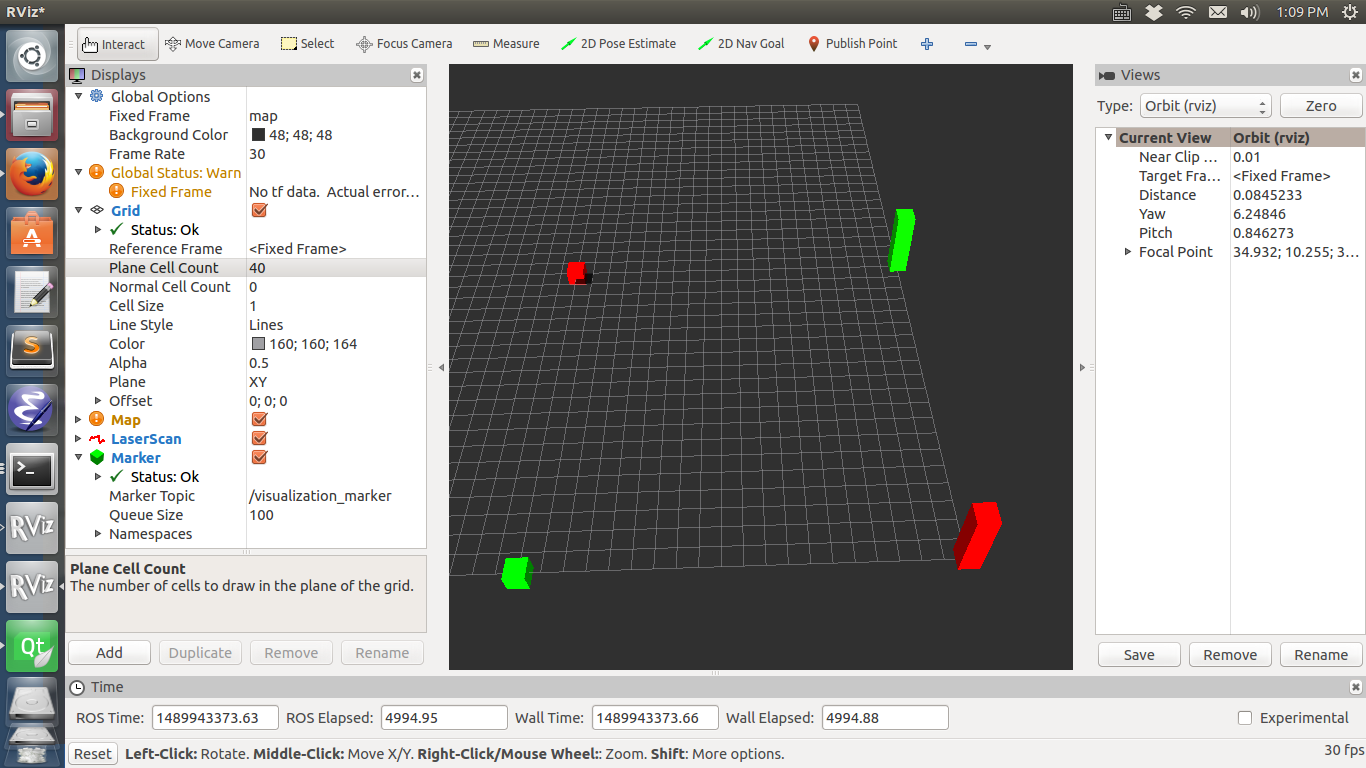
1. SIMULATION AND RESULTS

The simulation is implemented step by step as Dr. Guo suggested. It is running on a 32 bit Ubuntu 14.04 LTS. The robot simulation platform is ROS and GUI is rviz. This simulation uses C++ only.

1. Show the robot and the path in rviz

Although I can use turtlebot for the robot model, the implementation will become more difficult if I do so. In order to spend this limited week on the main problem, I decided to create 2 cubes as my robots. As figure 2 shows, the red cube is a robot called Jay and the green one is called Stella. Their targets are marked by 2 other larger cubes. The map is limited in a 20x20 grid map.

Jay starts from (0,0) and he is supposed to go to (20,20). Meanwhile, Stella starts from (20,0) and goes to (0,20). Jay’s path and goal are marked in red. Stella’s goal and path are marked in green.



*Figure* 1 *Simple map and robots show. Robot Jay and Stella are on the left side.*

*Their goals are on the right side. Red robot is Jay. Green robot is Stella*

1. Implementing single robot D\*

D\* algorithm consists of PROCESS-STATE function and MODIFY-COST function. The whole algorithm can be simplified as follows:

L1: While (OPEN is empty OR start state is in CLOSED)

L2: PROCESS\_STATE( )

L3:End

L4:Run the robot

L5: If (Detect environment change)

L6: MODIFY-COST( )

L7: While (OPEN is empty OR start state is in CLOSED)

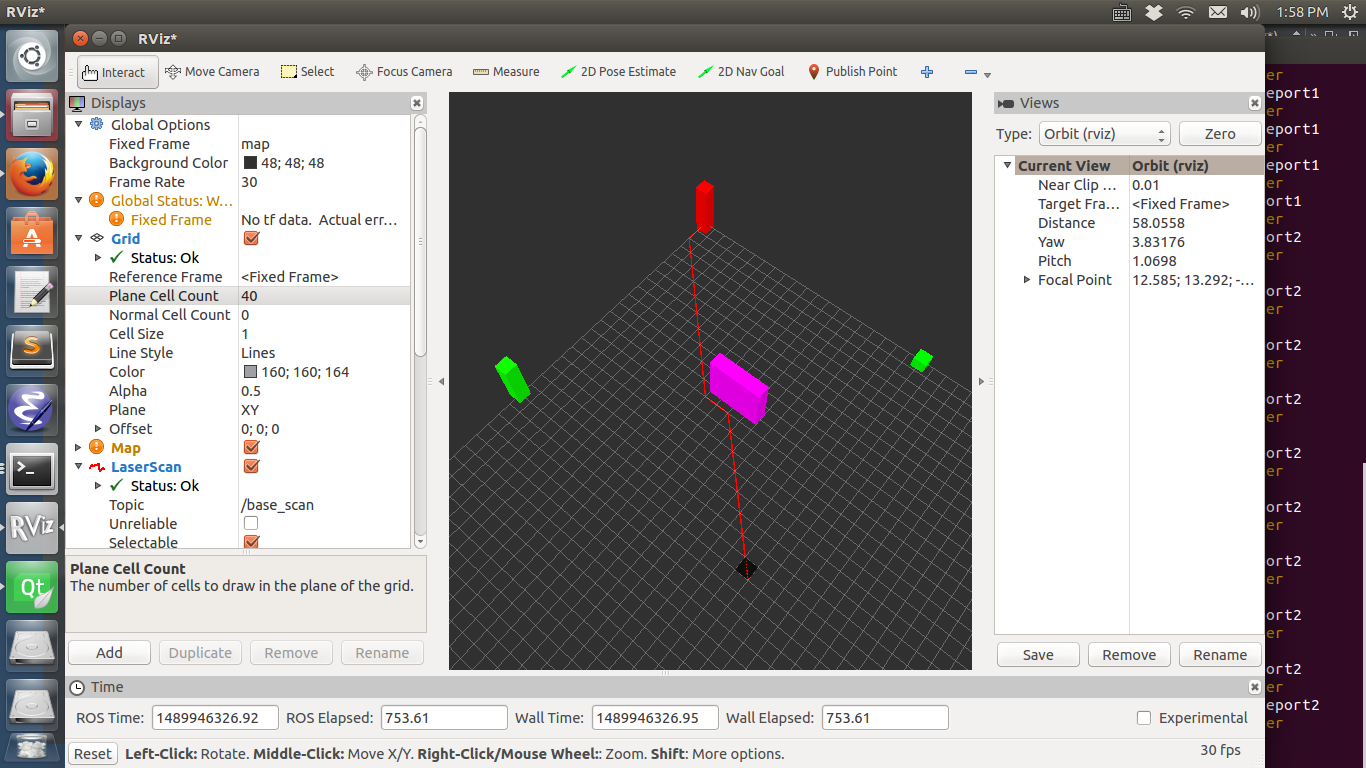
L8: PROCESS-STATE( )

L9: End

L10: End

L11:End

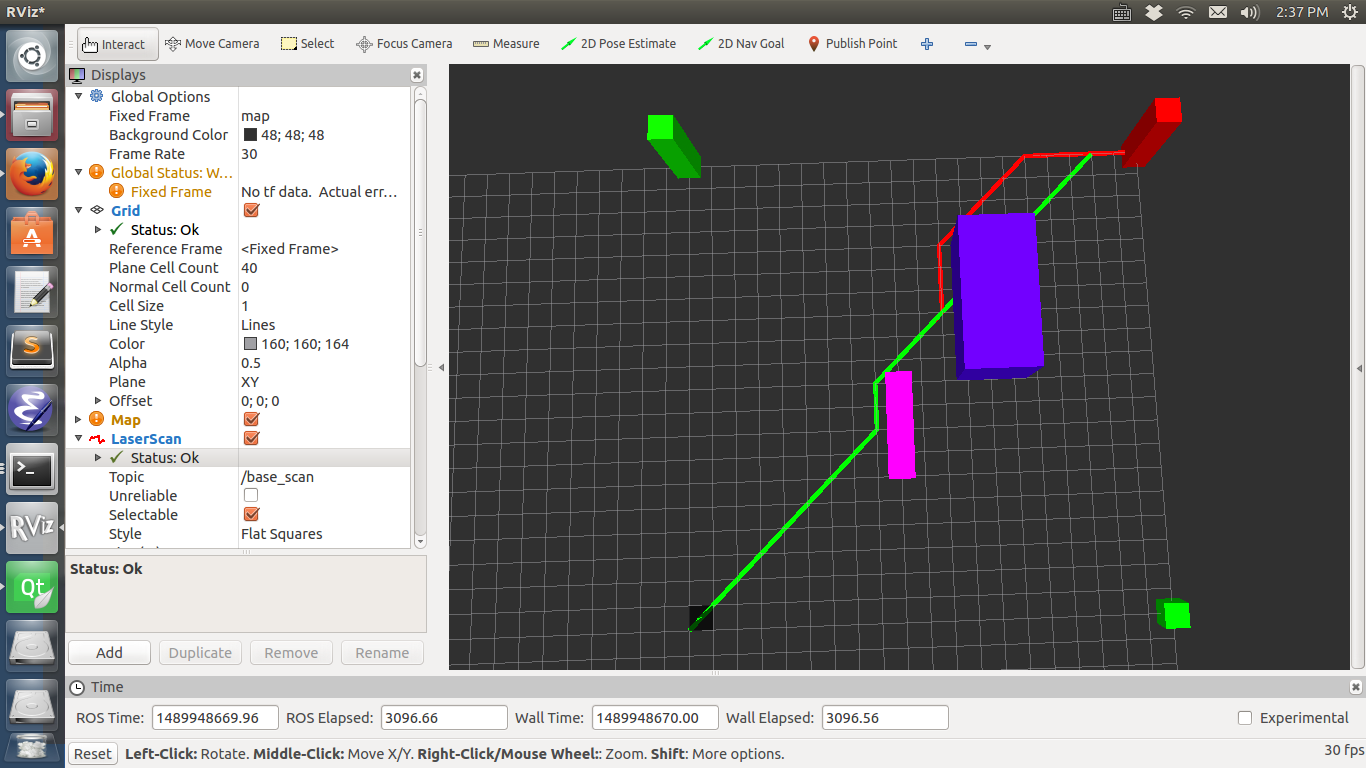
Figure 2 shows the result of the D\* motion planning in the static environment. The D\* algorithm code I use is written by myself. I designed three classes: Map, List, and Node corresponding to three layers of the map data structure: Map information, a sequence of states, and a single state. I use Euclidian distance as the cost function.



*Figure* 1 *The result of single robot D\* algorithm with static environment.*

*A pink obstacle is on Jay's way to his goal. Jay's path is marked with red line.*

Figure 3 shows the D\* algorithm’s performance in dynamic environment. Jay saw the pink obstacle and planned his path as the green line shows. When he finished path planning and started to move, the purple obstacle appeared out of nowhere. Maybe Jay’s sensors missed the purple obstacle or maybe it was dropped from sky, but Jay had to deal with it. He called MODIFY-COST function to mark that area as obstacles. Then he put the new obstacle area and its neighbors into the OPEN list and run the PROCESS-STATE function again and again until a new path came out. The new path is marked in red as figure 3 shows. He then proceeded to his goal successfully using his new path.



*Figure* 1 *D\* algorithm in dynamic environment. Green lines shows the original path before*

*the purple obstacle is added. Red line shows the path change after a new obstacle is added.*

1. Multi-robot motion planning

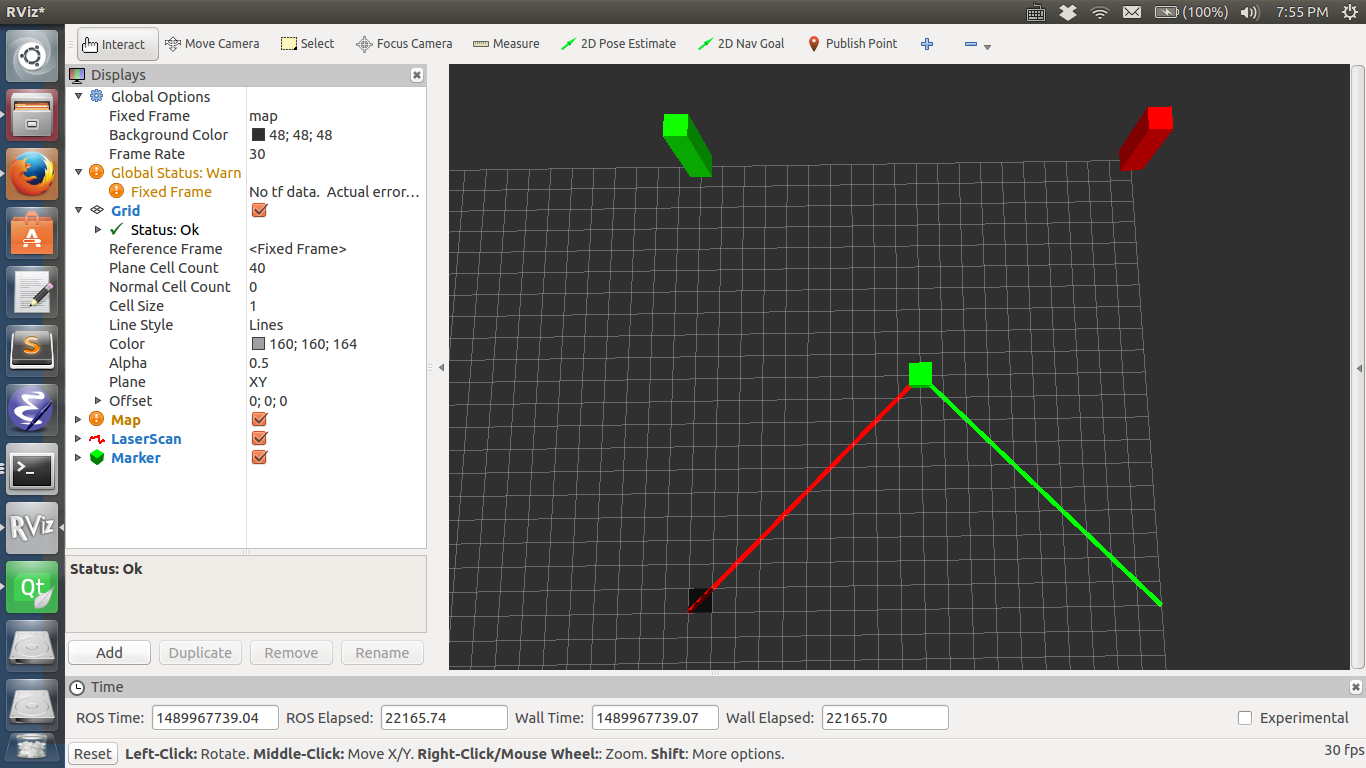
D\* in 3-dimension is much more difficult than in 2-dimension so I start with 2 robots.

Robot Jay(red) and Stella(green) are standing in the working area and they need to get their own goal positions as figure 1 shows. Jay and Stella both fits those assumptions we give in section 4. Before start to move, they need to run the D\* algorithm to get a optimal path. They run the D\* algorithm and tell each other about their own path.

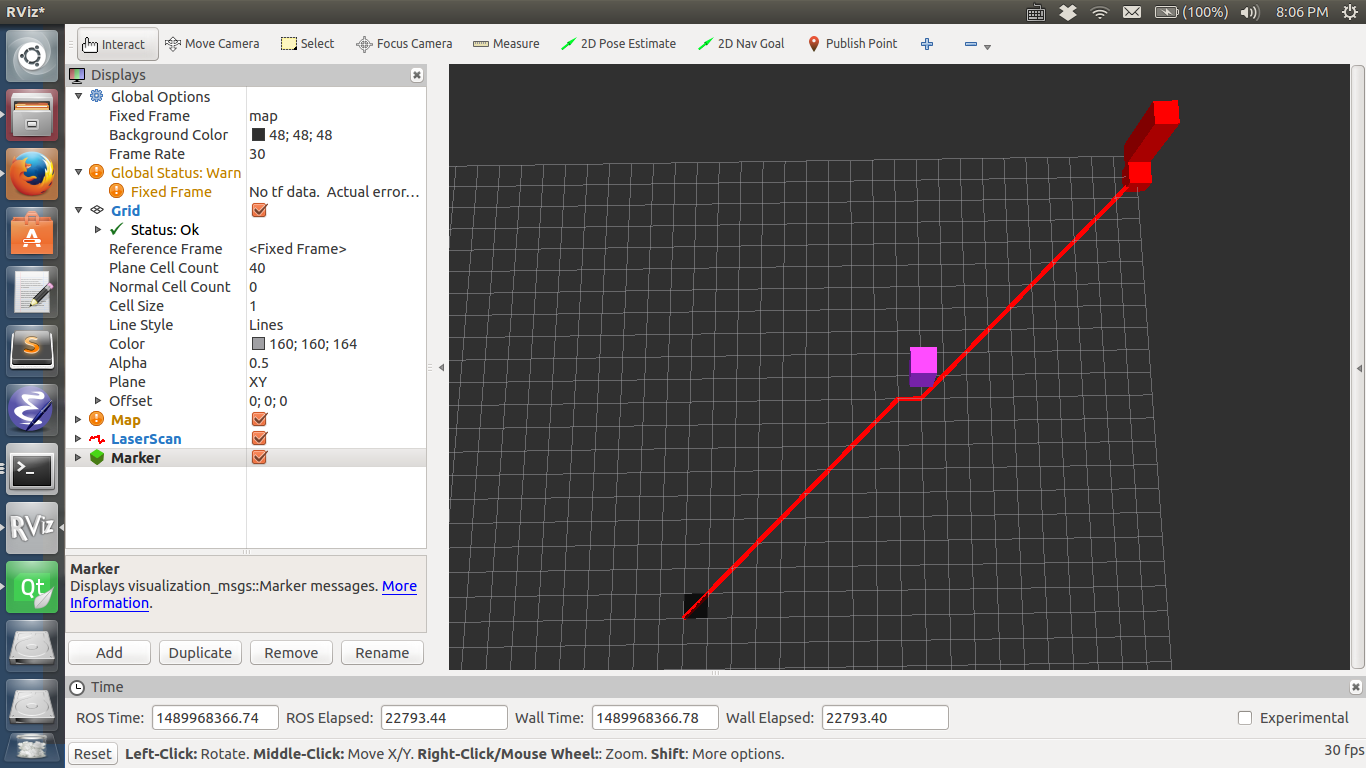
If Jay and Stella go to the same place at the same time, collision will happen. That is, if a node in Jay’s path shares the same x, y and time stamp with one node in Stella's path, this node is a collision node. Keep compare each node in Jay and Stella’s path can find all potential collision nodes. After all collision nodes are found, we mark those nodes as obstacles in the coordination diagram. The axis of the coordination diagram can either represent time elapsed or distance already walked.

Figure 4 shows that there is a potential collision node at (10,10), time stamp is 10th unit time. So they construct a coordinate diagram as figure 5 shows. Jay and Stella has the same priority and the cost function is unit time. The output of running the D\* in the coordination gram tells Stella to wait at the 9th unit time.

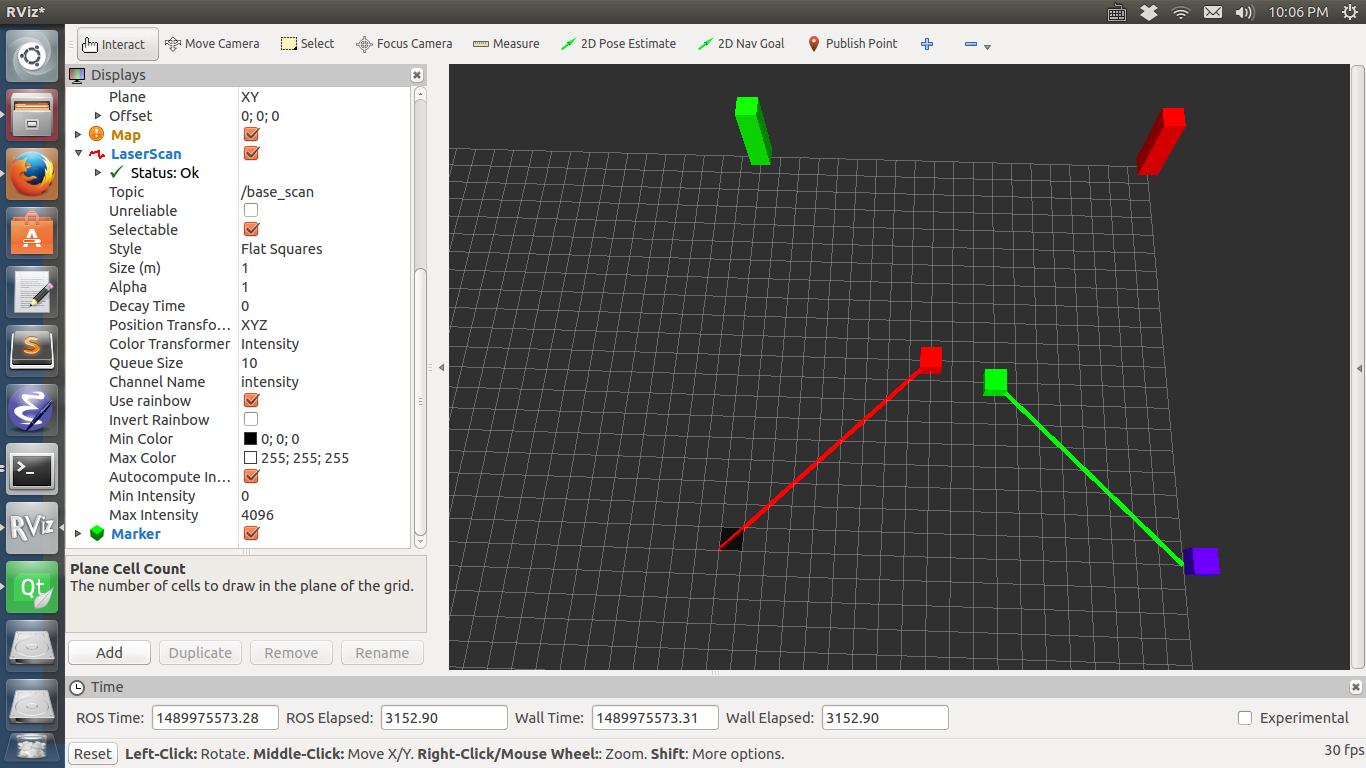
Figure 5 shows the result after running multi-robot motion planning algorithm. Collision is avoided successfully.



*Figure* 1 *Potential collision. Short green cube indicates the collision position. Red line is Jay's path. Green line is Stella's path.*



*Figure* 1 *Coordination gram constructed corresponding to figure 4. Vertical axis is Jay's time. Horizontal axis is Stella's time.*

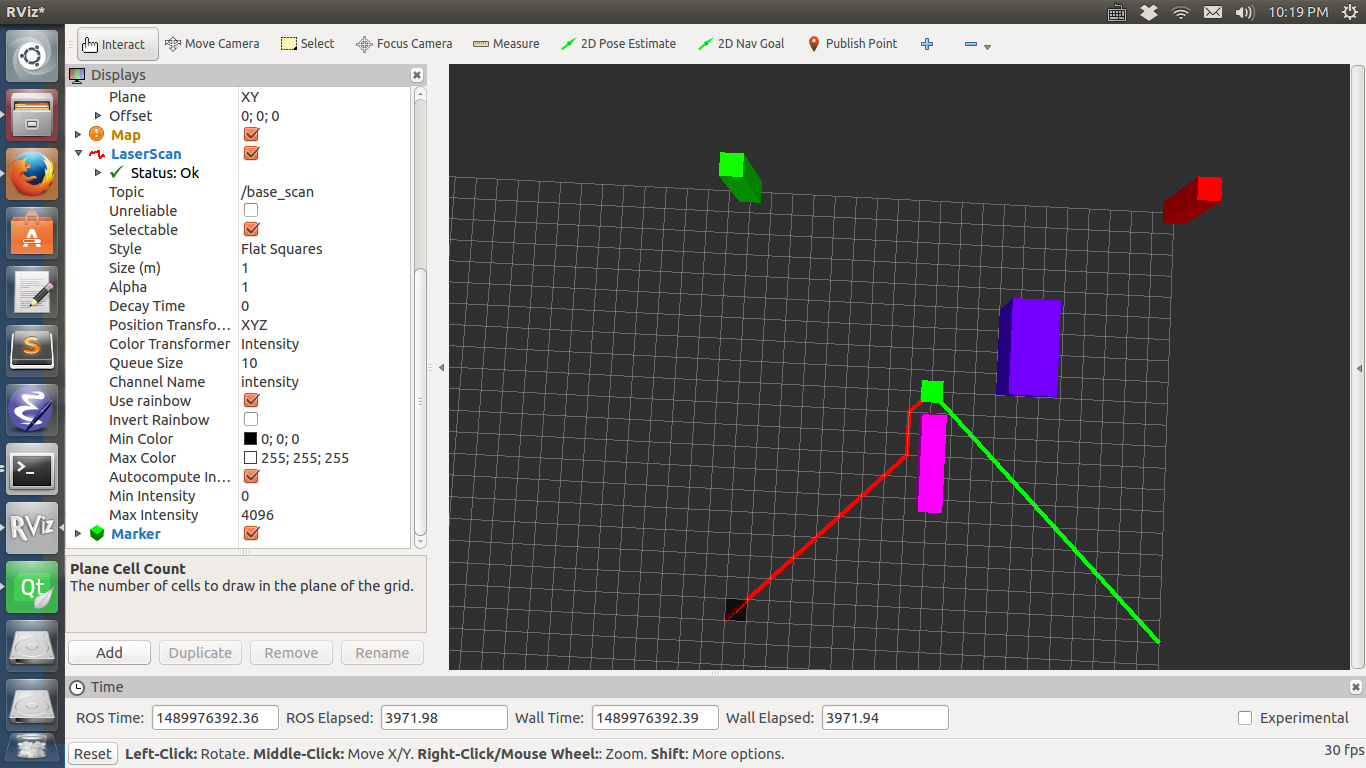


*Figure* 1 *Stella stops to avoid collision. Red one is Jay. Green one is Stella.*

1. Multi-robot motion planning with multiple obstacles

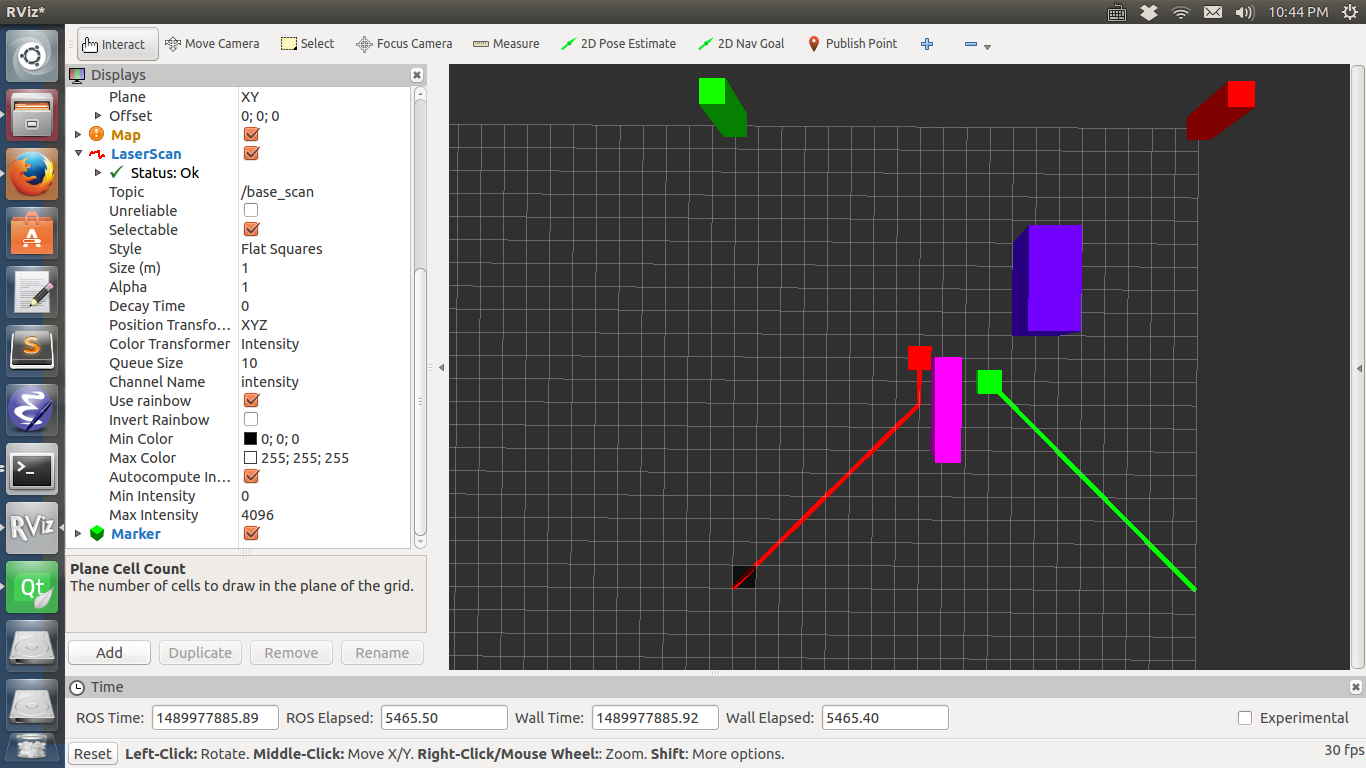
This simulation adds an obstacle on the map compared to last one. In this circumstance, Jay and Stella starts from the same position and goes to the same position but there will be several obstacles in the middle of the map.

Figure 7 shows the potential collision if Jay and Stella don’t run multi-robot path planning algorithm.

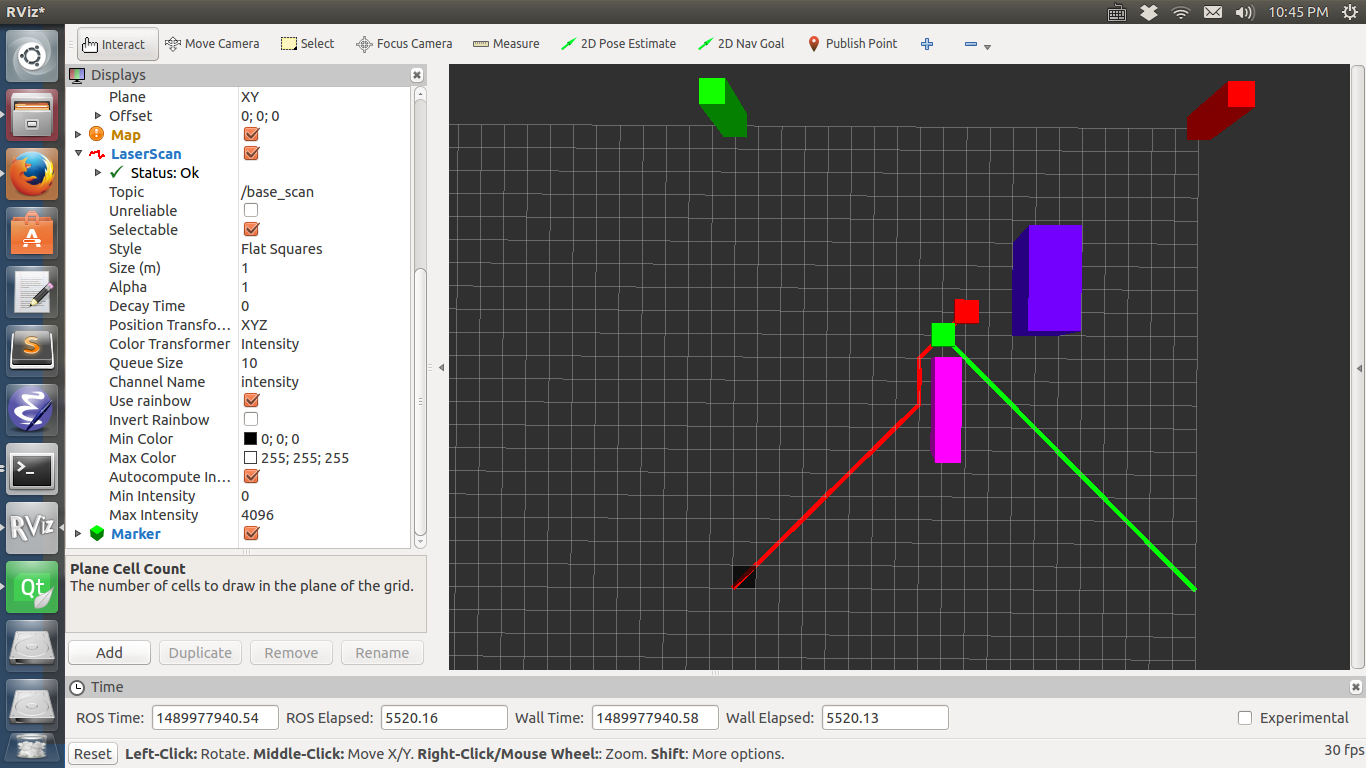


*Figure* 1 *Potential collision. Red one is Jay. Green one is Stella.*

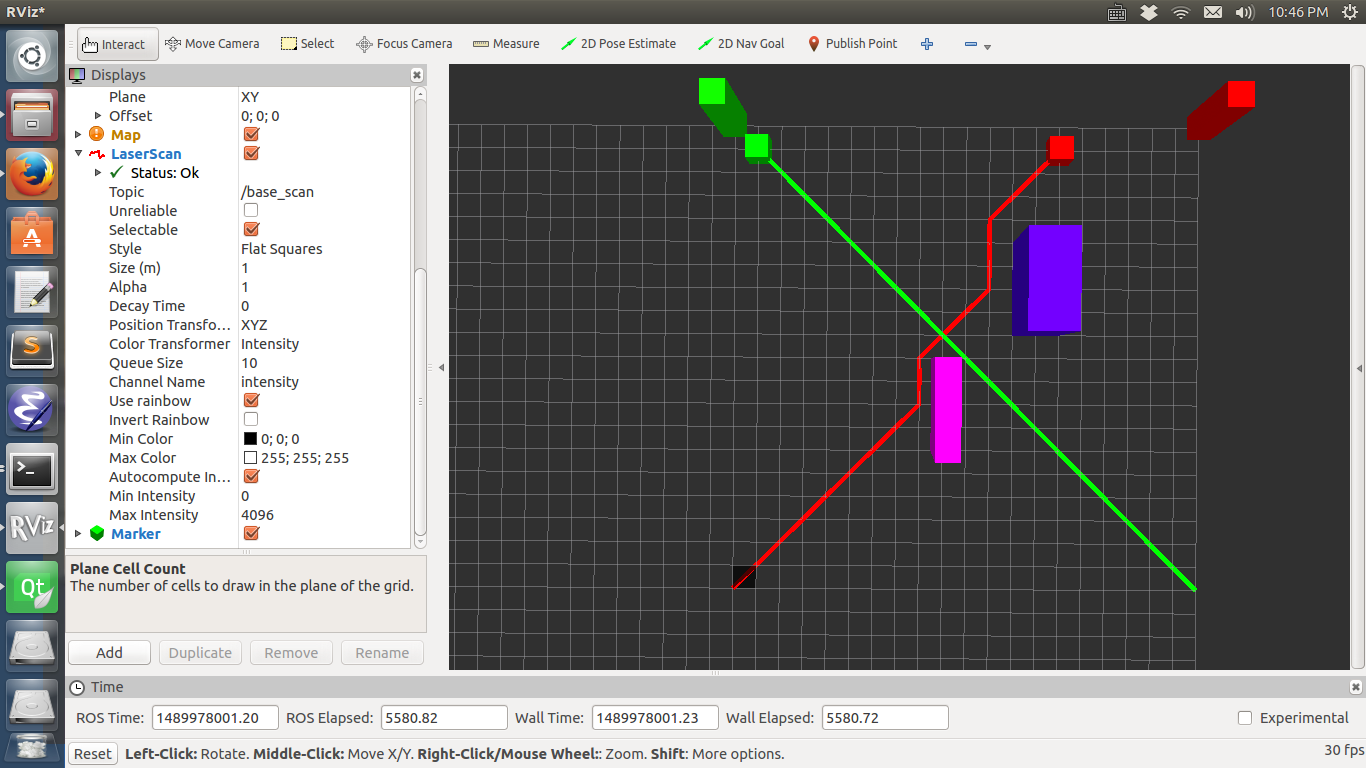
Figure 8, 9, 10 show the path after correction.



*Figure* 1 *Stella waits according to velocity profile*



*Figure* 1 *Collision is avoided*



*Figure* 1 *Carry on after collision is avoided*

*(5) 3 robots motion-planning-----D\* algorithm in 3-dimension*

*The key feature of 3 robots motion planning is to implement d\* algorithm in 3-dimention environment. I'll introduce how I implement d\* algorithm in 3-dimension first.*

*(how to ?)*

1. DISCUSSIONS

Velocity profile algorithm is good but is there any multi-robot motion planning algorithm which can combine detour and velocity control together? If so, that algorithm can be more efficient than control trajectory or velocity alone.

1. References

[1] Optimal and Efficient Path Planning for Partially-Known Environments, Anthony Stenz

[2] A distributed and Optimal Motion Planning Approach for Multiple Mobile Robots, Yi Guo