Environment Creation

The environment was created using the same technique used in the previous assignment but the only difference here is the tetrominoes are arranged randomly in the square box of size 15*15 square meters. The single box forms a single 'bush'. Similarly, N number of bushes are created in the environment based on the density provided. Each time when you execute the code new type of bushes are generated at random places.

Here the environment is of 250*250 matrix with open space represented as '0' and obstacles are represented either '0.1', '0.5', or '3'. These values depend on the state of the obstacle.

State of the Obstacles	**Background pixel value associated with the obstacles**
Extinguished	0.1
Intact	0.5
Burning	3

Table 1: state to value mapping

Please take a look in the below image which shows the environment for our fire truck to navigate. I have used grayscale image to save some unnecessary computational time. Color images can be simply created using RGB value at every pixel on the given environment 2D matrix.

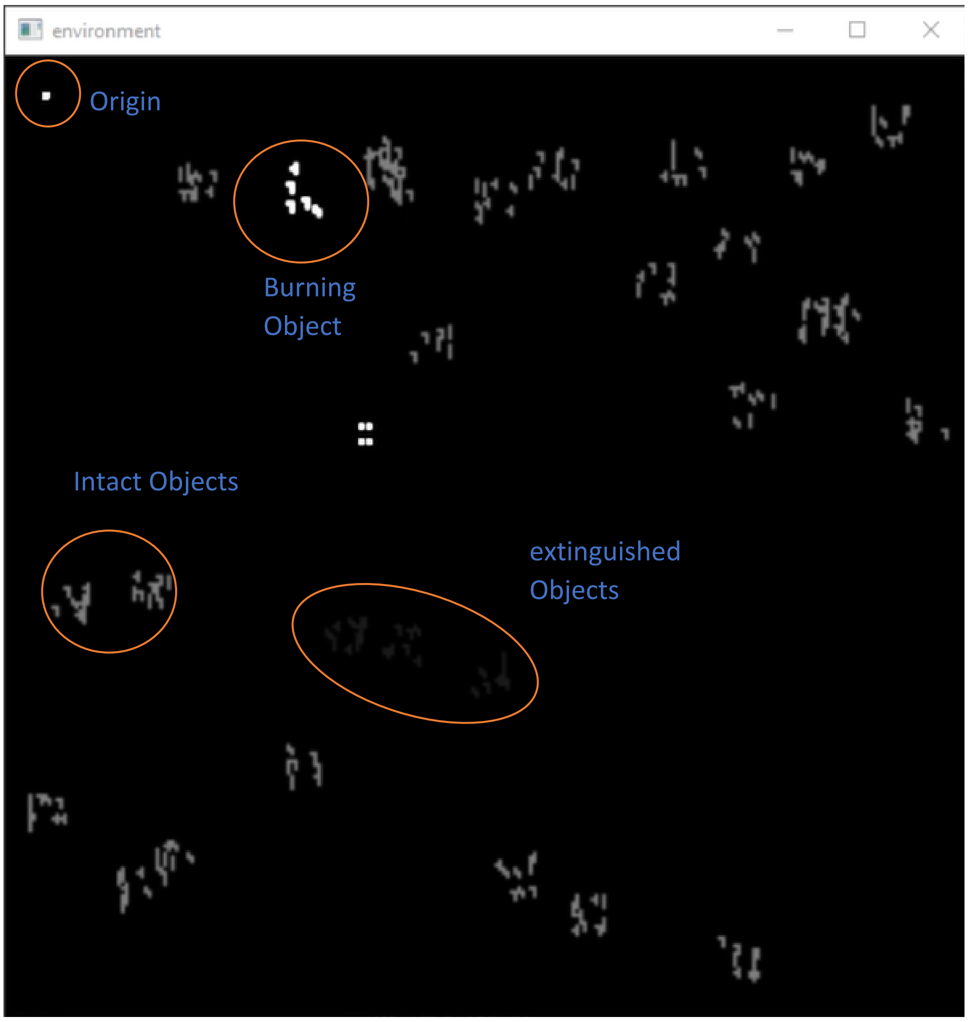

Figure 1: Environment in Grayscale image

General Wildfire Algorithm

- This algorithm provides base to implement any type of path planning algorithm for our firefighting mission.
- In the source code please look for files ..main…py.
- Psuedocode:-

```
Start_point ← predefined value
Total time ← 0 (system timer)
Time_20 ← 0 (system timer)
Time_60 ← 0 (system timer)

Put random bush on fire using → rand_fire() function
    time_60 ← 0
    update 'bush_at_fire'
While Total time < 3600 sec:
    If bush_at_fire == Null:
        No fire found!
    Else:
        Current_Bush_at_fire ← bush_at_fire
        Goal_point ← current_bush_at_fire
        Path ← Astar_planner(start_point, goal_point, environmet)
                    or
        path ← Prm_planner(start_point, goal_point, environmet)

        print the path and move the vehicle

        if Time_20 >= 20:
            put all neighbour at 15 meters of radius from
            'current_bush_at_fire' on fire
            Time_20 ← 0
        if Time_60 >= 60:
            put random bush on fire
            Time_60 ← 0

        If (path==Null):

            Extinguish the fire of current_node

        start ← path(end_value)
```

- Detailed explanation of Astar and Sampling-based Planners is given in the next page.

Combinatorial planner Algorithm (A*)

- Here I have used the standard A* algorithm to generate a path between two points in our environment.
- Note: A* and the wildfire algorithms have been executed parallelly. I have used threading/forking process to do this task.
- Initially the code generates a fire at a random point.
- That random point is fed as a goal_point to our A* planner. A* planner then generates a path from the predefined starting point and the goal_point considering the kinematics of the vehicle.
- Once the car reaches the goal state it extinguishes the fire on the current bush and moves to the nearer bush which is on fire.
- The code also keeps track of Burning bushes, Extinguished Bushes, and Intact Bushes.

Sampling Based planner Algorithm (PRM)

- Here I have used the standard Probabilistic Roadmap PRM algorithm to generate a path between two points in our environment. The algorithm is a modification of the PRM Algorithm mentioned on Github [1].
- The algorithm generated a random set of nodes (waypoints) from the environment and checks for object collision. For each node, it connects the current node to the nearer nodes. The connection between nodes is called edge and while connecting edges the collision check must be performed.
- Now we have a graph connection on which we can run planner algorithms such as Dijkstra's or A* algorithms to find the final path using waypoints. Here, I have used Dijkstra's Algorithm. Dijkstra's Algorithm takes the Kinematics of the Vehicle into account while generating the final path.
- In the Wildfire algorithm we just have to replace PRM and it'll work just fine.

Interesting Scenario

- I have used PRM as a Global Planner and A* star as a Local planner to generate the smooth path from start to the end-goal.

Results

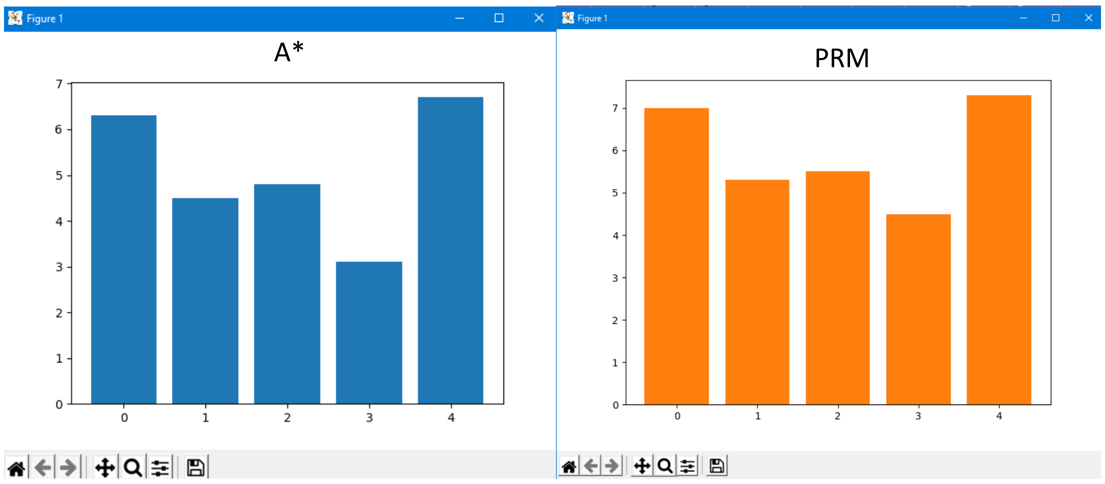

Figure 2: Extinguished to Burning Ration graph for 5 iterations

From the Figure 2, it is clear that PRM is a better choice for fighting fire.

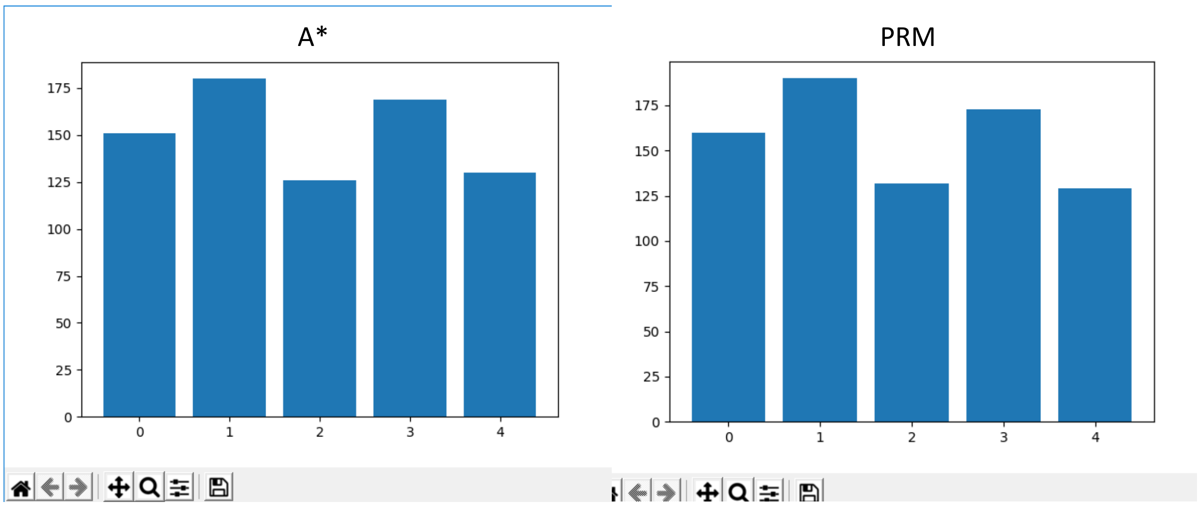

Figure 3: CPU Computation graphs for PRM and A*

From the Figure 3 we can say that PRM takes more computational power then A*.

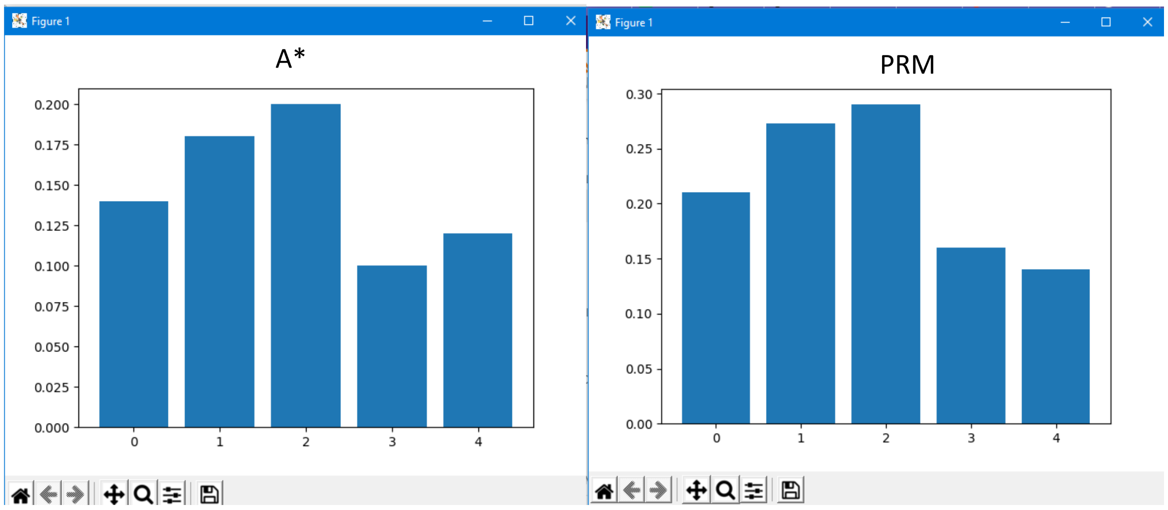

Figure 4: Intact to Total Ratio

References

1. https://github.com/AtsushiSakai/PythonRobotics/blob/master/PathPlanning/ProbabilisticRoadMap/probabilistic_road_map.py.