Multi-agent trajectory planning: A decentralized iterative algorithm based on single-agent dynamic RRT*

Ravi Teja Alapati
Worcester Polytechnic Institute
Worcester, Massachusetts

***Abstract*—I address the problem planning trajectories of multi-robot systems in an environment, while avoiding obstacles as well as other robots, adhering to a pre-defined clearance from each other. The proposed method uses the prune-and-graft approach for re-planning, which requires a maximum of n iteration for each robot to generate its collision-free trajectory. The entire method is decentralized with limited sharing of information (only pertaining to the shared space). I used a dynamic RRT* algorithm for this purpose undertaking a single-agent planning method.**

I. Introduction

Multi-robot systems have a profound interest in the robotics research domain over the last few years. Multi-agent systems have considerable advantage over their single robot configurations, even though they have a considerable computational power. In case of decentralized planning, such systems provide critical redundancy and may fault tolerance for completing the given problem. However, it is critical to ensure robust collaborative metrics, while developing planning algorithms for such systems. The multi-robot planning with the decentralized approach becomes a consolidated approach of collaboration between the individual systems, to avoid static obstacles as well as maintain a desired clearance distance from each other.

Collision free multi-robot trajectory planning has been previously addressed by multiple researchers. Real-time approach based on the concept of reciprocal velocity obstacle is presented by [1], while more heuristic-based graph-based approaches have been presented in [2]. In this approach, I attempt to implement the method proposed by [3], which relies on a sampling-based approach. The overall system consists of multiple agents, which are tasked with trajectory planning, while ensuring obstacle avoidance and inter-agent collision avoidance. I adopted a decentralized approach, in which the agents carry out the path planning individually, and broadcast a limited information to the other agents. A heuristic is then proposed which allows the agents to replan their trajectories to ensure collision-free final paths. For path planning, the agents employ Rapidly Exploring Random Trees (RRT*) [4], with a prune-and-graft approach [5]. This approach does not entirely replan the completed paths, but only on the overlapped sections. The degree to which each agent has to replan it's trajectory is termed as it's 'performance degradation', and this parameter should be communicated to the other agents. Agents with low performance degradation retain their paths, whereas the agents with higher performance degradation execute a replan-compare approach to generate an updated trajectory. This is repeated till the complete collision-free paths are obtained.

II. Method

The first step is that each agent develops its trajectory, z_i and the associated cost $J_i(z_i)$. This path is optimal in an independent sense, as the sampling-based RRT* algorithm is used to generate the path. Initially, the path of other agents is disregarded. Subsequently, an individual agent's path is broadcast to the other agents. The agents then check for collision with paths of other agents, by using the earlier heuristic for collision detection. If a collision is detected, the iterative process begins. All agents involved in a collision now replan their individual trajectories, considering all the other agents as dynamic obstacles. This can be done because of the fact that the information of their respective trajectories were shared previously. Using the same RRT* method, the new path z* and it's associated cost function J*(z*) is computed. The performance degradation of the agents are shared among each other, and the agent with the smallest performance degradation replans it's trajectory. The new trajectory of this agent is broadcast to all agents and the process continues iteratively till complete collision free paths are determined. The agent which changed it's trajectory in an iteration is not re-planned in further iterations. It takes maximum of n-1 iterations to obtain collision free paths for n agents. The above algorithm does not assume any priority order to replan the trajectories. It only uses the computed rubric of performance degradation for determining the re-plan order. The performance degradation measure for an agent i at iteration k is given by:

$$D_i^{(k)} = \frac{J_i(\mathbf{z}_i^{(k)}) - J_i^\star}{J_i^\star}$$

Every time a collision is detected, the agents involved in that collision have to replan their trajectories. One way to achieve this is to rebuild the tree from the root. But this is computationally very expensive. To avoid this, I used a prune and graft method that augments our RRT* algorithm implementation. When an agent detects a collision, the re-plan module is triggered. It detects the branches involved in the collision. All these branches along with their children are removed from the tree. Thus, only the section of the tree involved in the collision are removed and rest of the tree is still available. The re-planning is then performed with available tree using the same RRT* algorithm that was used for the initialization of the tree. Augmenting RRT* with prune and graft method avoids repeating the same computations and thus reduces the time taken by the algorithm considerably.

III. Implementation

This section provides the implementation details of the proposed algorithm. The algorithm was implemented in Python and the a simulation was performed using ROS and Gazebo. The exploration of the space is stored in the form of a tree and the cost of each node in the tree is given by the time taken by the agent to reach the node. The pseudo-code for Plan and Replan module is given in Algorithm 1 and 3. Algorithm 1 is the crux of the implementation and is used to plan a collision-free trajectory for a single agent. It takes the start and the goal point as an input and returns a safe trajectory. Following is a brief description of the functions used in the algorithm.

When a collision between any two agents is detected, the Replan module is triggered. This module takes the trajectories of the other agents and uses the Plan module to generate a collision free trajectory by considering other agents as moving obstacles. The first step in replanning is to prune the tree by removing any branch that was involved in the collision. This has the positive effect of saving time, memory, and avoiding repeating the same computations. More precisely, when an agent detects a collision with an obstacle, it prunes its own tree removing the branches that are involved in the collision and then starts regrowing the tree by grafting it with new edges. The details of prune algorithm is given in Algorithm 2. After that the pruned tree is rebuild again using Algorithm 1. Finally, the resultant collision free trajectory is returned. The plan and replan module together ensure a collision-free trajectory for a single agent in presence of other agents. These replanned trajectories are then checked for their performance degradation and the agent with the minimum performance degradation is given the task to update its trajectory. The whole procedure is repeated till collision-free trajectories for all agents are found. As mentioned earlier, the algorithm is run for a maximum of n iterations.

Subsequent to the iterations and visualization run on Python, the sameis simulated in Gazebo using the ROS framework. I obtained the final path for each of the n agents in the form of a text file from the python algorithm, which contains the time-stamped trajectory for each agent. The way points on the B-Spline for the robot path are generated every 0.2 seconds. I first created an initial project world with predefined robot start positions and the obstacles. I also separate namespaces for each of the Turtlebots, so that their */odom* and */cmdvel* messages can be commanded individually by the controller. With the initial world in place, the robots are spawned at their respective positions, and then proceed to follow their generated path. As discussed in the results section, the time-stamping obtained from the planner in Python was found established to be robust when executing the simulations in Gazebo.

Algorithm 1 Plan

```
Input: $start\_point, goal\_point$
Output: $trajectory$
  init($start\_point, goal\_point$)

  while $i < max\_iterations$ do
    $r\_node$ = get_random_point()
    $n\_node$ = get_nearest_node($r\_node$)
    $new\_node$ = step_ahead($r\_node, n\_node$)

    $neighbours$ = get_neighbours($new\_node$)
    set_parent($new\_node, neighbours$)
    rewire($new\_node, neighbours$)

    if $new\_node = goal\_point$ then
      $goal\_node = new\_node$
      $break$
    end if
  end while

  $trajectory$ = backtrack($goal\_node$)
```

Algorithm 2 Prune

```
Input: $other\_trajectories$
Output: $trajectory$
  for all $nodes$ in tree do
    if collision_detected($node$) then
      delete_node($node$)
    end if
  end for
```

Algorithm 3 Replan

```
Input: $start\_point, goal\_point, other\_trajectories$
Output: $trajectory$
  prune($other\_trajectories$)
  $trajectory$ = plan($start\_point, goal\_point$)
```

IV. Results

We first worked on two agent system, where we aim to obtain collision free paths for the agents in one iteration. In

Fig. 1, we can see the paths generated for agent 1 (blue) and agent 2 (red) through RRT* algorithm. These paths have collision at some point in time t. To obtain collision free paths, we first obtained re-planned paths for both the agents using prune and graft feature, as shown in Fig. 2. We calculate the performance degradation of both agents, and observe that it is less for agent 2 than that of agent 1. So agent 2 is assigned with its new re-planned trajectory and agent 1 will remain with its initial generated trajectory. Now the paths of the robots are collision-free.

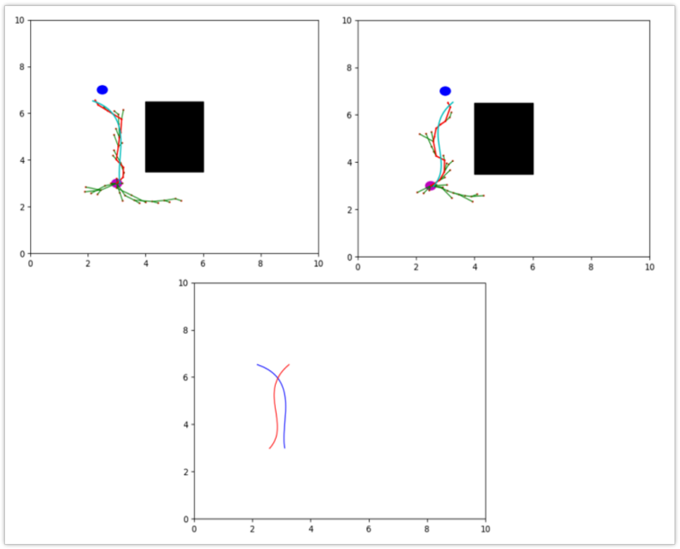


Fig. 1. **Two Agent System**. Top Left: Path generated for agent 1; Top Right: Path generated for agent 2; Bottom: Collision occurred at some time t

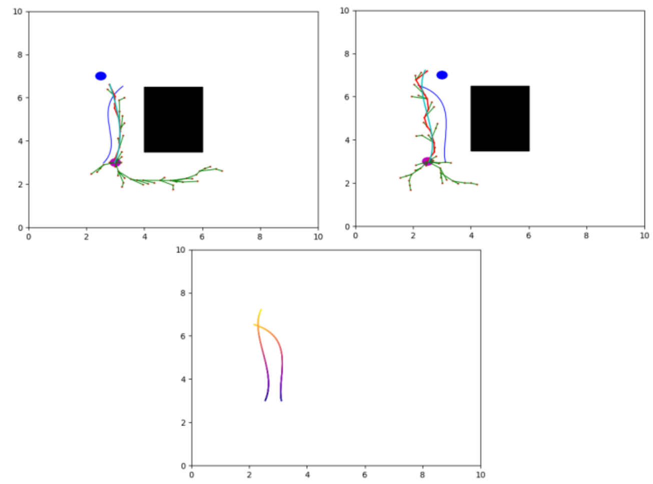


Fig. 2. **Two Agent System**. Top Left: Re-planning for agent 1; Top Right: Re-planning for agent 2; Bottom: Collision-free trajectories

In three agent system, it takes maximum of 3 iterations to find the collision free paths. In Fig. 3, we can see the initial paths obtained for agent 1 (red), agent 2 (green) and agent 3 (blue). All the three paths have collision at some pint in time with each other. Thus, in iteration 1, all the three agents re-plan their trajectories as shown in Fig. 4. Here, the performance degradation of agent 2 is less compared with others and thus agent 2 is assigned with its new re-planned trajectory, whereas agent 1 and 3 will be maintain their initial trajectories. The trajectory of agent 2 is not changed further and it will be considered as time-varying obstacle in the next iteration. In iteration 2, we re-plan trajectories for agent 1 and agent 3 as shown in Fig. 5. The performance degradation of agent 3 is less than that of agent 1 and thus the agent 3 is assigned with it's re-planned trajectory and agent 1 maintains it's initial trajectory. The final collision free paths are shown in Fig. 5.

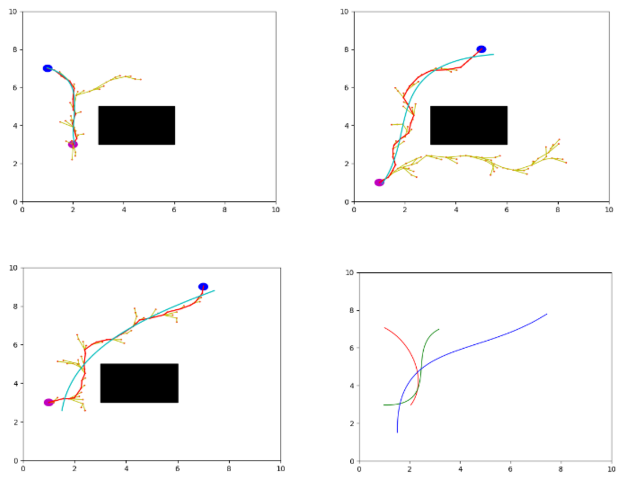


Fig. 3. **Three Agent System**. Top Left: Path generated for agent 1; Top Right: Path generated for agent 2; Bottom Left: Path generated for agent 3; Bottom Right: Initial paths obtained

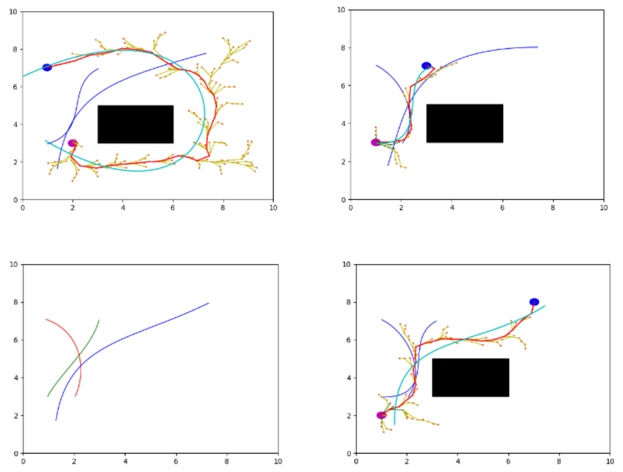


Fig. 4. **Three Agent System**. Top Left: Re-planning for agent 1; Top Right: Re-planning for agent 2; Bottom Left: Re-planning for agent 3; Bottom Right: Paths obtained after iteration 1

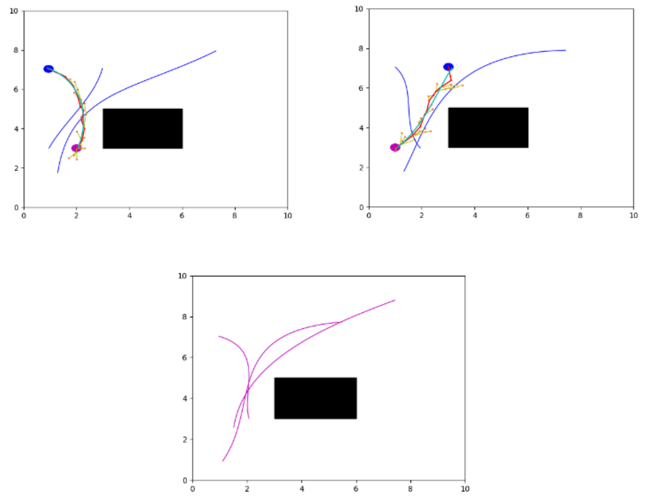


Fig. 5. **Three Agent System**. Top Left: Re-planning for agent 1; Top Right: Re-planning for agent 2; Bottom: Final collision free paths obtained after iteration 2

The ROS simulation snapshots are shown in Fig 6 to Fig 9. The sequence of images show how the robots, on account of following the generated trajectories avoid collision, while successfully reaching their goals. The agents are labeled with numbers in order to ease identification. The paths that the agents adhere to are the same ones that are shown from the Python simulations.

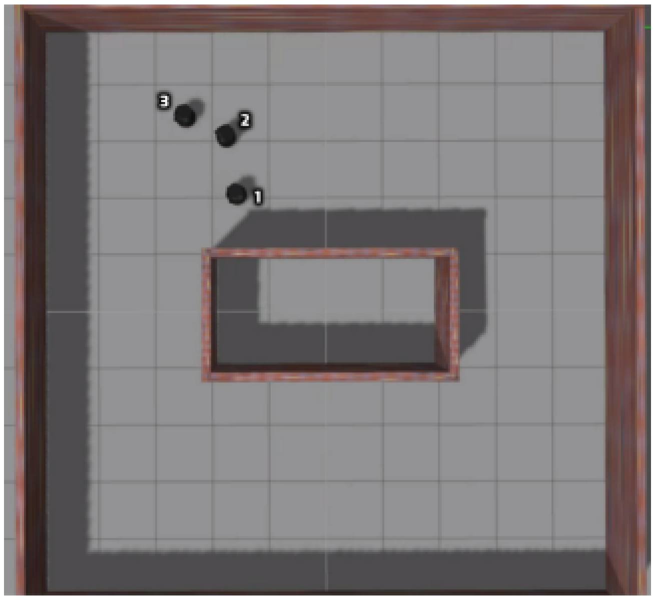


Fig. 6. Agents start from their initial configuration

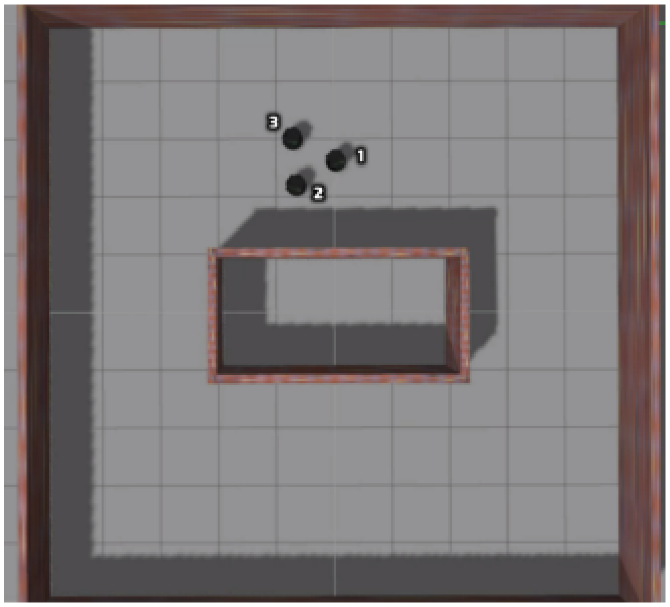


Fig. 7. Agents approach each other

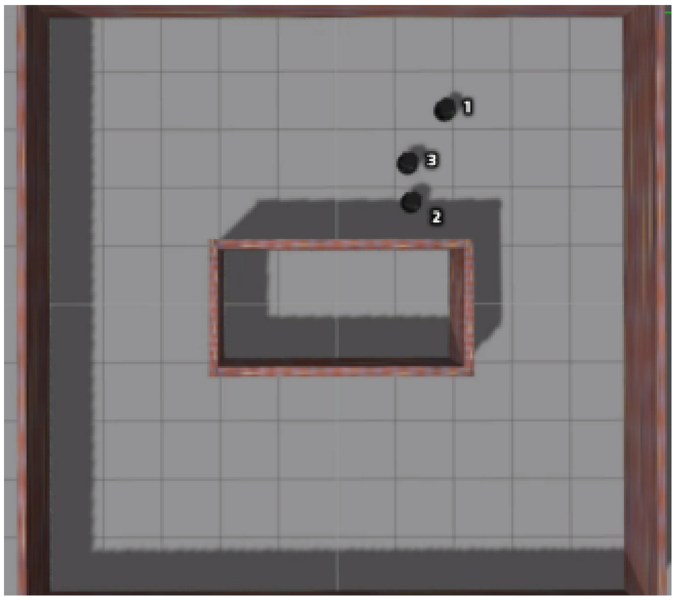


Fig. 8. Agents avoid collision by re-planning their path

V. Evaluation Metrics

A. Distance between the agents

The main aim of the algorithm is to avoid collisions between the agents in a multi-robot environment. We assign a safe distance where all the agents must maintain a minimum of this distance between each others. Fig. 10 shows a plot of the distance between robots over the time of simulation.

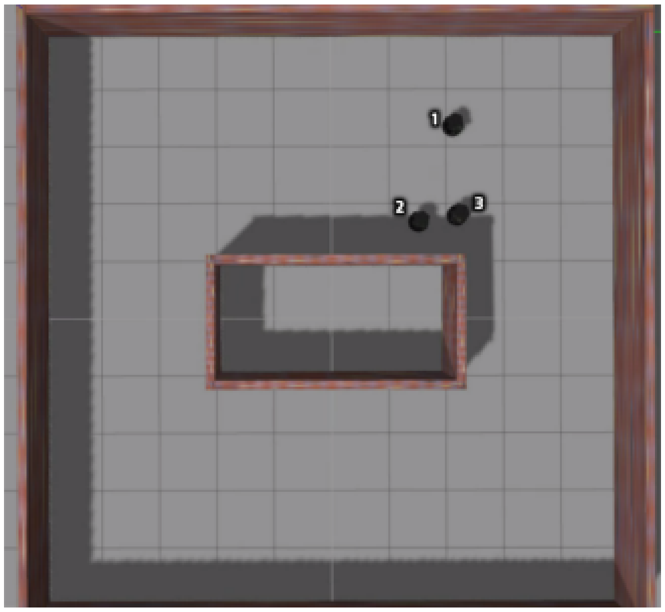


Fig. 9. Agents continue towards goal

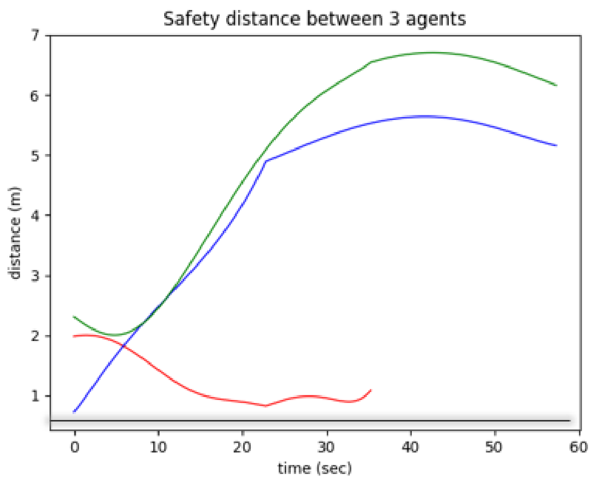
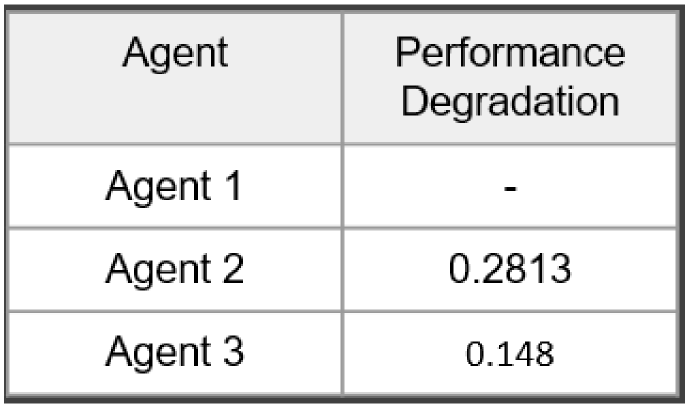


Fig. 10. **3-agent system**: distance between the first and the third agent (green), the first and the second agent (red), and the second and the third agent (blue). The black line defines the safety distance.

B. *Performance Degradation of the agents*

The performance degradation is a metric that shows the deviation of optimal paths. The initial trajectories found by the single-agent planner (RRT*) without considering other agents are optimal. Then the trajectories are replanned to avoid collision with other agents and thus the trajectories' optimality is decreased. The performance degradation of all the agents is calculated for the final trajectories compared to their initial trajectories and shown in Fig. 11.



Agent	Performance Degradation
Agent 1	-
Agent 2	0.2813
Agent 3	0.148

Fig. 11. **3-agent system**: Performance Degradation

VI. Scalability

The time taken for the 3-agent system is exponentially increased compared to that of the 2-agent system. Naturally, the time for planning the trajectories increases for more agents and also the time for replanning the trajectories increases. In the paper[3] I took reference from, it is mentioned that this algorithm is capable for upto 5 agents. I can conclude that for a system with more than 5 agents, the time to find the collision-free trajectories would be infeasible.

References

[1] J. Snape, S. J. Guy, J. Van Den Berg, and D. Manocha, "Smooth coordination and navigation for multiple differential-drive robots," in Experimental Robotics. Springer, 2014, pp. 601–613.

[2] J. Yu and S. M. LaValle, "Optimal multirobot path planning on graphs: Complete algorithms and effective heuristics," IEEE Transactions on Robotics, vol. 32, no. 5, pp. 1163–1177, 2016.

[3] P. Verbari, L. Bascetta, and M. Prandini, "Multi-agent trajectory planning: A decentralized iterative algorithm based on single-agent dynamic rrt," in 2019 American Control Conference (ACC). IEEE, 2019, pp. 1977–1982.

[4] S. Karaman and E. Frazzoli, "Sampling-based algorithms for optimal motion planning," The international journal of robotics research, vol. 30, no. 7, pp. 846–894, 2011.

[5] D. Ferguson, N. Kalra, and A. Stentz, "Replanning with rrts," in Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. ICRA 2006. IEEE, 2006, pp. 1243–1248.