Section 1. DMA-RRT

Each agent runs the individual and interaction components in parallel. The individual component is responsible for the path planning and bidding to be the next token holder, while the interaction component is responsible for listening for messages from other agents and updating its state accordingly.

We have implemented for you a CL-RRT API (Appendix 1), an API to interface with the Agent class (Appendix 2), and the interaction component (Algorithm 5 pseudocode below), so you can make use of their functionality and focus on implementing the higher level pseudocode of the individual components of DMA-RRT and the Cooperative DMA-RRT extension.

DMA-RRT Individual Component

Algorithm 4 DMA-RRT: Individual component

Initialize with p_0
`HaveToken` $\leftarrow$ *false* except for one randomly selected agent
while Agent is active **do**
    Grow CL-RRT (Algorithm 2), identify best path p_k^* satisfying all constraints (Algorithm 1)
    if `HaveToken` **then**
        $p_k \leftarrow p_k^*$
        `winner` $\leftarrow$ agent with best `bid`
        Broadcast `waypoints` of p_k and `winner` to team
        `HaveToken` $\leftarrow$ *false*
    else
        $p_k \leftarrow p_{k-1}$
        `bid` $\leftarrow (cost(p_k) - cost(p_k^*))$
        Broadcast `bid`
    end if
    $k \leftarrow k + 1$
end while

While our planner takes care of initializing the agents with an initial position and whether they are the token holder, your task will be to implement the body of the while loop outlined above, which comprises a single iteration of the individual process.

In a single iteration, the agent must first interface with the CL-RRT API to grow a tree from its current position, taking into account the other agents as dynamic obstacles. Once the optimal path is returned by the CL-RRT algorithm, if the agent holds the merit-based token, it must determine which agent has the most **potential path improvement (PPI)** and thus wins the right to replan its path in the next iteration. It must broadcast the winner to the rest of the team (see the Agent API in Appendix 2 for how to broadcast messages), along with the new plan it determined for itself in this iteration. If it does not hold the merit-based token, it must instead broadcast its PPI bid for a chance to replan in the next iteration. The PPI is calculated by comparing the cost of the new optimal path the agent would take if it got to replan to the cost of the path it is currently taking. Please see the CL-RRT API in Appendix 1 for how to retrieve the costs of different paths.

DMA-RRT Interaction Component

Algorithm 5 DMA-RRT: Interaction component

```
while Agent is active do
    Listen for messages
    if received waypoints and winner message then
        Simulate other agent's trajectory along waypoints, update constraints
        if agent is winner then
            HaveToken ← true
        end if
    end if
    if Received bid message then
        Update sender's bid
    end if
end while
```

While continuously running the individual process, each agent is also listening for messages from other agents in parallel, which we call the interaction component of DMA-RRT. We have implemented this functionality for you in a somewhat different way as part of the Agent class, but the pseudocode from the referenced paper is provided here. It is not important to know how this is implemented under the hood, but is sufficient to understand that every time an agent broadcasts a winner and waypoints or a PPI bid, all agents update their states to reflect this information.

Section 2. Cooperative DMA-RRT Extension

Cooperative DMA-RRT Individual Component

Algorithm 6 Cooperative: Individual component

Initialize with p_0
`HaveToken` ← *false* except for one predetermined agent
while Agent is active **do**
  Grow CL-RRT ignoring others' paths, identify best path p_k^*
  if `HaveToken` **then**
    if p_k^* conflicts with some agent j **then**
      Check emergency stops (Algorithm 7)
    end if
    if p_k^* conflicts with some other agent j' **then**
      p_k^* pruned to avoid conflict with agent j'
    end if
    Identify emergency stop nodes on p_k^*
    $p_k \leftarrow p_k^*$
    if agent j's plan was modified **then**
      `winner` ← agent j
    else
      `winner` ← agent with best `bid`
    end if
    Broadcast `waypoints` of p_k (with emergency stops) and `winner` to team
    `HaveToken` ← *false*
  else
    $p_k \leftarrow p_{k-1}$
    `bid` ← $(p_k.cost - p_k^*.cost)$
    Broadcast `bid`
  end if
  $k \leftarrow k+1$
end while

Algorithm 7 Emergency Stop Check

if `CheckEstops` **then**
  for all viable emergency stop nodes N_l in agent j's path **do**
    Find last safe stop node in p_k^* if agent j stops at N_l
    `TotalCost`$_l$ = cost of path ending at this node + agent j's cost to stop at N_l
  end for
  Select terminal node and N_l that minimize `TotalCost`$_l$
  if N_l is not agent j's original terminal node **then**
    Send `estop` message to agent j to stop at N_l
  end if
  if selected terminal node is not original terminal node **then**
    p_k^* pruned past new terminal node
  end if
else
  $p_k^* \leftarrow p_k^*$ pruned to satisfy all constraints
end if

Cooperative DMA-RRT Interaction Component

Algorithm 8 Cooperative: Interaction component

while Agent is active **do**
  Listen for messages
  if received `waypoints` and `winner` message **then**
    Simulate other agent's trajectory along `waypoints`, update constraints
    if agent is `winner` **then**
      `HaveToken` $\leftarrow$ *true*
    end if
  end if
  if Received `bid` message **then**
    Update sender's `bid`
  end if
  if Received `estop` message **then**
    Terminate p_k at node stop specified in `estop`
    `CheckEstops` $\leftarrow$ *false*
  end if
end while

Appendix 1. CL-RRT API

Algorithm 1 CL-RRT: Tree Expansion

Sample point x_{sample} from the environment
Identify nearest node N_{near} in tree
$k \leftarrow 0$
$\hat{x}(t+k) \leftarrow$ last state of N_{near}
while $\hat{x}(t+k) \in \mathcal{X}_{free}(t+k)$ and $\hat{x}(t+k)$ has not reached x_{sample} **do**
  Compute reference input $\hat{r}(t+k)$ from x_{sample}
  Compute control input $\hat{u}(t+k)$ from control law (1)
  Compute next state $\hat{x}(t+k+1)$ from propagation model (3)
  $k \leftarrow k+1$
end while
$N \leftarrow \hat{r}_{final}$
for each feasible node N produced **do**
  Update cost estimates for N
  Add N to tree
end for

Algorithm 2 CL-RRT: Execution Loop

1: $t \leftarrow 0, x(t) \leftarrow x_{initial}$
2: Initialize tree with node at $x_{initial}$
3: **while** $x(t) \notin \mathcal{X}_{goal}$ **do**
4:   Update current state $x(t)$
5:   $\hat{x}(t+\Delta t) \leftarrow x(t)$ propagated by Δt
6:   **while** *elapsed_time* $< \Delta t$ **do**
7:     Grow tree using Algorithm 1
8:   **end while**
9:   **if** no paths exist in tree **then**
10:     Apply safety action and goto line 19
11:   **end if**
12:   Select best path p^* from tree using cost estimates
13:   Recheck feasibility of p^* using Algorithm 3
14:   **if** rechecked p^* is feasible **then**
15:     Apply p^*
16:   **else**
17:     Goto line 9
18:   **end if**
19:   $t \leftarrow t + \Delta t$
20: **end while**

Algorithm 3 CL-RRT: Lazy Check

```
1: for each node $N_i \in p^*$ do
2:     while $\hat{x}(t+k) \in \mathcal{X}_{free}(t+k)$ and $\hat{x}(t+k)$ has not reached $N_i$ do
3:         Compute reference input $\hat{r}(t+k)$ from $N_i$
4:         Compute control input $\hat{u}(t+k)$ from (1)
5:         Compute next state $\hat{x}(t+k+1)$ from (3)
6:         $k \leftarrow k+1$
7:         if $\hat{x}(t+k) \notin \mathcal{X}_{free}(t+k)$ then
8:             Remove $N_i$ and all children from $p^*$, goto line 11
9:         end if
10:    end while
11: end for
```

Appendix 2. Agent API