Attachment 1(Improve ACO pseudocode)

```
function improved_aco_core(max_iter, num_ants)
    % parameter initialization
    A = 1.0; B = 1.0; C = 1.0; D = 1.0;    %The dynamic coefficients of α (N) and β (N)
    Q = 1.0;                                % Pheromone increment constant
    tau_min = 0.001; tau_max = 10;          % Upper and lower bounds of pheromone concentration

    % Main iteration loop
    for N = 1:max_iter
        % 1. Dynamically calculate heuristic parameters
        alpha_N = A * sin(N * pi / max_iter) + C;    % Importance of pheromones
        beta_N = B * cos(2 * N * pi / max_iter) + D; % Heuristic importance

        % 2. Calculate dynamic volatility coefficient
        rho_N = cos(N / max_iter);    % Iteration related volatility

        % 3. Ant path construction phase
        for ant = 1:num_ants
            % Use dynamic parameters to select the next node
            next_node = select_next_node(current_node, alpha_N, beta_N, tau, eta);
            % Note: The node selection function needs to be implemented separately
        end

        % 4. Pheromone update mechanism
        tau = (1 - rho_N) * tau;          % Pheromone volatilization
        tau = tau + delta_tau;            % Elite Path Enhancement
        % Pheromone concentration boundary processing
        tau(tau < tau_min) = tau_min;     % Lower bound truncation
        tau(tau > tau_max) = tau_max;     % Upper bound truncation
    end
end
```

Attachment 2(RRT-APF pseudocode)

```
function [tree, path] = RRT_APF_Planner(start_pos, goal_pos, max_iter)
    % initialization
    tree.nodes = {start_pos};
    tree.edges = [];
    goal_node = [];

    % Parameter settings
    step_size = 0.5;
```

```
    goal_thresh = 0.5;

    for iter = 1:max_iter
        % Sampling (biased target)
        if rand() < 0.8
            rand_point = goal_pos + randn(size(goal_pos))*0.1;
        else
            rand_point = rand(1,2)*10; % Assuming a map size of 10x10
        end

        % nearest neighbor search
        [nearest_idx, nearest_pos] = find_nearest(tree, rand_point);

        % Expand new nodes
        new_pos = nearest_pos + step_size*(rand_point-nearest_pos)/norm(rand_point-nea
rest_pos);
        % APF adjustment
        [F_att, F_rep] = APF(new_pos, goal_pos, obstacles);
        new_pos = new_pos + 0.2*(F_att + F_rep);

        % Collision detection and adding nodes
        if ~check_collision([nearest_pos; new_pos])
            new_idx = length(tree.nodes)+1;
            tree.nodes{new_idx} = new_pos;
            tree.edges = [tree.edges; nearest_idx new_idx];

            % Check to reach the target
            if norm(new_pos-goal_pos) < goal_thresh
                goal_node = new_idx;
                break;
            end
        end
    end

    % Extract path
    path = [];
    if ~isempty(goal_node)
        path = backtrack_path(tree, goal_node);
    end
end
```