% Constants

CELL\_RADIUS = 1000;

NUM\_USERS = 100;

NUM\_DRONES = 5;

CELL\_COUNT = 5;

INITIAL\_HEIGHT = 30;

MAX\_HEIGHT = 120;

HEIGHT\_STEP = 15;

SIMULATION\_TIME = 1; % in seconds

TIME\_STEP = 0.1; % in seconds

MAX\_SPEED = 0.5; % in meters per second

MOVE\_TOWARDS\_USERS\_PROBABILITY = 0.8; % Probability to move towards areas with high user density

% Generate drone and cell positions

drone\_positions = [2000, 2000; 4000, 2000; 6000, 2000; 4000, 4000; 6000, 4000];

cell\_centers = drone\_positions;

% Initialize height range

height\_range = INITIAL\_HEIGHT:HEIGHT\_STEP:MAX\_HEIGHT;

for h = height\_range

% Simulate user positions and calculate RSRP

all\_users = cell(NUM\_DRONES, 1);

all\_rsrp = cell(NUM\_DRONES, 1);

for i = 1:NUM\_DRONES

users = generate\_users(cell\_centers(i, :), NUM\_USERS, CELL\_RADIUS);

all\_users{i} = users;

rsrp = calculate\_rsrp(users, drone\_positions(i, :), h, 33);

all\_rsrp{i} = rsrp;

end

% Mobility and communication simulation

figure;

for t = 1:SIMULATION\_TIME

for i = 1:NUM\_DRONES

% Move users

users = all\_users{i};

new\_users = move\_users(users, cell\_centers(i, :), CELL\_RADIUS, MAX\_SPEED);

all\_users{i} = new\_users;

% Update RSRP

rsrp = calculate\_rsrp(new\_users, drone\_positions(i, :), h, 33);

all\_rsrp{i} = rsrp;

end

% Drone-to-drone communication

drone\_rsrp = calculate\_drone\_to\_drone\_rsrp(drone\_positions, h, 33);

all\_rsrp = transmit\_signal(drone\_rsrp, all\_rsrp);

% Move drones towards areas with high user density

for i = 1:NUM\_DRONES

users = all\_users{i};

drone\_pos = drone\_positions(i, :);

if rand < MOVE\_TOWARDS\_USERS\_PROBABILITY

% Compute centroid of user positions

centroid = mean(users);

% Compute direction vector towards centroid

direction = centroid - drone\_pos;

% Normalize direction vector

direction = direction / norm(direction);

% Compute new drone position by moving in the direction of the centroid

new\_drone\_pos = drone\_pos + direction \* MAX\_SPEED;

% Update drone position if it is within the cell radius

if sqrt((new\_drone\_pos(1) - cell\_centers(i, 1))^2 + (new\_drone\_pos(2) - cell\_centers(i, 2))^2) <= CELL\_RADIUS

drone\_positions(i, :) = new\_drone\_pos;

end

end

end

% Plot user positions and RSRP

clf;

for i = 1:NUM\_DRONES

users = all\_users{i};

rsrp = all\_rsrp{i};

scatter(users(:, 1), users(:, 2), 50, rsrp, 'filled');

hold on;

scatter(drone\_positions(i, 1), drone\_positions(i, 2), 100, 'kx');

end

colorbar;

title(sprintf('RSRP for Users in Different Cells (t = %d, Height = %d)', t, h));

xlabel('Cell Radius(m) ');

ylabel('Cell Radius(m)');

drawnow;

pause(TIME\_STEP);

end

end

% Function Definitions

% Generate random user positions within a cell

function users = generate\_users(cell\_center, num\_users, cell\_radius)

users = zeros(num\_users, 2);

for i = 1:num\_users

angle = 2 \* pi \* rand; % Random angle

radius = cell\_radius \* sqrt(rand); % Random radius within the cell

x = cell\_center(1) + radius \* cos(angle); % Compute x-coordinate

y = cell\_center(2) + radius \* sin(angle); % Compute y-coordinate

users(i, :) = [x, y];

end

end

% Calculate path loss between a user and a drone

function pl = path\_loss(user\_pos, drone\_pos, drone\_height)

distance\_2d = sqrt((user\_pos(1) - drone\_pos(1))^2 + (user\_pos(2) - drone\_pos(2))^2);

distance\_3d = sqrt(distance\_2d^2 + drone\_height^2);

pl = 20 \* log10(distance\_3d) + 20 \* log10(2.4 \* 10^9) + 20 \* log10(4 \* pi / 3 \* 10^-8);

end

% Calculate RSRP (Reference Signal Received Power) for each user

function rsrp = calculate\_rsrp(users, drone\_pos, drone\_height, tx\_power)

num\_users = size(users, 1);

rsrp = zeros(num\_users, 1);

for i = 1:num\_users

pl = path\_loss(users(i, :), drone\_pos, drone\_height);

rsrp(i) = tx\_power - pl;

end

end

% Move users randomly within a cell, considering cell boundaries

function new\_users = move\_users(users, cell\_center, cell\_radius, max\_speed)

num\_users = size(users, 1);

new\_users = users;

for i = 1:num\_users

angle = 2 \* pi \* rand; % Random angle

speed = max\_speed \* rand; % Random speed within the maximum speed

x = users(i, 1) + speed \* cos(angle); % Compute new x-coordinate

y = users(i, 2) + speed \* sin(angle); % Compute new y-coordinate

if sqrt((x - cell\_center(1))^2 + (y - cell\_center(2))^2) <= cell\_radius

% Check if the new position is within the cell radius

new\_users(i, :) = [x, y];

end

end

end

% Calculate RSRP (Reference Signal Received Power) between each pair of drones

function drone\_rsrp = calculate\_drone\_to\_drone\_rsrp(drone\_positions, drone\_height, tx\_power)

num\_drones = size(drone\_positions, 1);

drone\_rsrp = zeros(num\_drones, num\_drones);

for i = 1:num\_drones

for j = 1:num\_drones

if i ~= j

pl = path\_loss(drone\_positions(i, :), drone\_positions(j, :), drone\_height);

drone\_rsrp(i, j) = tx\_power - pl;

end

end

end

end

% Transmit signal between drones based on RSRP

function all\_rsrp = transmit\_signal(drone\_rsrp, all\_rsrp)

num\_drones = size(drone\_rsrp, 1);

for i = 1:num\_drones

for j = 1:num\_drones

if i ~= j

received\_rsrp = drone\_rsrp(j, i);

all\_rsrp{i} = max(all\_rsrp{i}, received\_rsrp);

end

end

end

end