```
%% Autonomous Keep Out Zone
% Colby Davis
% 12/18/25

clear;
clc;
close all;

%% SETTINGS
N_WORKERS = 8;
N_runs    = 500;


%% CPU WORKERS
if ~isempty(gcp('nocreate'))
    delete(gcp('nocreate'));
end
parpool('local', N_WORKERS);
addAttachedFiles(gcp(), {mfilename('fullpath')});

%% Orbit and sim settings
mu = 3.986004418e14;
Re = 6378e3;
h  = 500e3;
r0 = Re + h;
n  = sqrt(mu / r0^3);

dt   = 1.0;
Tend = 600;
t    = (0:dt:Tend).';
N    = numel(t);

% noise
sigma_pos = 1.0;    % 1 m position noise
sigma_vel = 0.01;   % 1 cm/s velocity noise

%% KOZ settings
Rkoz    = 500;
Rbuffer = 2000;
Tlook   = 60*60;

%% Defender fuel
dv0          = 30.0; % total fuel available
dv_warn_mag  = 0.8;
dv_block_mag = 1.2;

min_burn_interval = 0;  % (continous)

%% CW matrices
Phi_dt   = cw_phi(n, dt);
Gamma_dt = cw_gamma(n, dt);
Nlook    = round(Tlook/dt);

% Precompute lookahead
Phi_look = zeros(6,6,Nlook);
Phi_look(:,:,1) = Phi_dt;

for i = 2:Nlook

    Phi_look(:,:,i) = Phi_look(:,:,i-1) * Phi_dt;

```

```
end

Phi_stacked = reshape(permute(Phi_look, [1 3 2]), [], 6);

%% Monte Carlo outputs
success_asset_KOZ = false(N_runs,1);   % did intruder stay out of asset KOZ
threatening_run   = false(N_runs,1);   % would have entered asset KOZ without defender burns
blocked_threat    = false(N_runs,1);   % threatening and successfully blocked

min_asset_range_defended = zeros(N_runs,1);
min_asset_range_free     = zeros(N_runs,1);

defender_dv_used = zeros(N_runs,1);
block_used       = false(N_runs,1);
first_burn_time  = zeros(N_runs,1);

%% Plot history
intr_hist_all = zeros(6, N, N_runs);
def_hist_all  = zeros(6, N, N_runs);

%% Monte Carlo cpu aided
parfor run = 1:N_runs

    rng(run, 'twister');

    intr0_run = random_intruder_init();
    [kp_run, kd_run, amax_run, intr_dv0_run] = random_intruder_params();

    noise_seq = [sigma_pos*randn(3,N); sigma_vel*randn(3,N)];
    target_u  = rand(1,N);   % targeting decisions

    % Defender initial state
    patrol_radius    = 800 + 400*rand();
    patrol_angle     = 2*pi*rand();
    patrol_elevation = (rand()-0.5)*pi/3;

    def_x = patrol_radius * cos(patrol_elevation) * cos(patrol_angle);
    def_y = patrol_radius * cos(patrol_elevation) * sin(patrol_angle);
    def_z = 0;

    patrol_speed = 3.4;

    % Velocity direction toward intruder from defender position
    intr_pos0 = intr0_run(1:3);
    def_pos0  = [def_x; def_y; def_z];
    dir_to_intr = intr_pos0 - def_pos0;

    if norm(dir_to_intr) < 1e-9

        dir_to_intr = intr_pos0;

    end

    vel_dir = dir_to_intr / norm(dir_to_intr);

    def_vx = patrol_speed * vel_dir(1);
    def_vy = patrol_speed * vel_dir(2);
    def_vz = patrol_speed * vel_dir(3) * 0.1;

    def0_run = [def_x; def_y; def_z; def_vx; def_vy; def_vz];

```

```
123      %% intruder only, no defender, measure min range to ASSET
124      [min_asset_free_run] = simulate_intruder_only( ...
125          intr0_run, kp_run, kd_run, amax_run, intr_dv0_run, ...
126          Phi_dt, Gamma_dt, n, dt, N, noise_seq, target_u);
127
128      min_asset_range_free(run) = min_asset_free_run;
129      threatening_run(run) = (min_asset_free_run < Rkoz);
130
131      %% defender active, measure min range to ASSET
132      last_burn_time = -inf;
133
134      intr = intr0_run;
135      def  = def0_run;
136
137      dv_rem      = dv0;
138      intr_dv_rem = intr_dv0_run;
139
140      used_block  = false;
141      first_burn_t = inf;
142
143      intr_hist = zeros(6, N);
144      def_hist  = zeros(6, N);
145      intr_hist(:,1) = intr0_run;
146      def_hist(:,1)  = def;
147
148      % low pass filter
149      intr_filt  = intr0_run;
150      alpha_filt = 0.3;
151
152      min_asset_def = inf;
153
154      for k = 2:N
155
156          % Noise and filter
157          intr_meas = intr + noise_seq(:,k);
158          intr_filt = alpha_filt * intr_meas + (1-alpha_filt) * intr_filt;
159
160          % Predict intruder closest approach to ASSET from filtered estimate
161          [intr_states, rmin_pred, ~] = predict_range_and_rate(intr_filt, Phi_stacked, Nlook);
162
163          if rmin_pred > Rbuffer
164
165              mode = 0;
166
167          elseif rmin_pred > 1200
168
169              mode = 1;
170
171          else
172
173              mode = 2;
174              used_block = true;
175
176          end
177
178          % Defender burn
179          if mode ~= 0 && dv_rem > 0 && (k-1)*dt - last_burn_time >= min_burn_interval
180
181              dv_mag = dv_warn_mag;
182
183              if mode == 2
```

```
184
185                 dv_mag = dv_block_mag;
186
187             end
188
189             dv_mag = min(dv_mag, dv_rem);
190
191             dv_vec = choose_best_burn_gradient(def, intr_states, Phi_stacked, Nlook, dv_mag);
192
193             def(4:6) = def(4:6) + dv_vec;
194
195             dv_rem   = dv_rem - norm(dv_vec);
196
197             if first_burn_t == inf
198
199                 first_burn_t = (k-1)*dt;
200
201             end
202
203             last_burn_time = (k-1)*dt;
204
205         end
206
207         % Intruder controller
208         r = intr(1:3);
209         v = intr(4:6);
210
211         ax_ff = 3*n^2*r(1) + 2*n*v(2);
212         ay_ff = -2*n*v(1);
213         az_ff = n^2*r(3);
214
215         a_cmd = [ax_ff; ay_ff; az_ff] - kp_run*r - kd_run*v;
216
217         if target_u(k) < 0.96 % attacks 96% of time, other time coasts by
218
219             if norm(r) > 1e-9
220
221                 target_dir = -r / norm(r);
222                 a_cmd = a_cmd + 0.5 * target_dir;
223
224             end
225         end
226
227         an = norm(a_cmd);
228
229         if an > amax_run
230             a_cmd = (amax_run/an) * a_cmd;
231         end
232
233         dv_step = norm(a_cmd)*dt;
234
235         if intr_dv_rem > 0
236
237             if dv_step > intr_dv_rem
238
239                 a_cmd = a_cmd * (intr_dv_rem / dv_step);
240                 dv_step = intr_dv_rem;
241
242             end
243
244             intr_dv_rem = intr_dv_rem - dv_step;
```

```
245
246         else
247             a_cmd = [0;0;0];
248         end
249
250         % Propagate
251         def  = Phi_dt * def;
252         intr = Phi_dt * intr + Gamma_dt * a_cmd;
253
254         % Asset KOZ range
255         r_asset = norm(intr(1:3));
256
257         if r_asset < min_asset_def
258             min_asset_def = r_asset;
259         end
260
261         intr_hist(:,k) = intr;
262         def_hist(:,k)  = def;
263     end
264
265     intr_hist_all(:,:,run) = intr_hist;
266     def_hist_all(:,:,run)  = def_hist;
267
268     min_asset_range_defended(run) = min_asset_def;
269     success_asset_KOZ(run)        = (min_asset_def >= Rkoz);
270
271     defender_dv_used(run) = dv0 - dv_rem;
272     block_used(run)       = used_block;
273     first_burn_time(run)  = first_burn_t;
274
275     blocked_threat(run) = threatening_run(run) && success_asset_KOZ(run);
276 end
277
278 %% Results
279 n_threat = sum(threatening_run);
280
281 if n_threat == 0
282     threat_block_rate = NaN;
283 else
284     threat_block_rate = 100 * sum(blocked_threat) / n_threat;
285 end
286
287 fprintf("\nMonte Carlo Results (%d runs):\n", N_runs);
288 fprintf("Overall asset KOZ success rate (all intruders): %.1f%%\n", 100*mean(success_asset_KOZ));
289 fprintf("Threatening intruders (would enter KOZ without defense): %d (%.1f%%)\n", n_threat,
    100*mean(threatening_run));
290 fprintf("Threatening intruders successfully blocked: %d (%.1f%% of threatening)\n",
    sum(blocked_threat), threat_block_rate);
291
292 fprintf("Average min asset range (free): %.1f m\n", mean(min_asset_range_free));
293 fprintf("Average min asset range (defended): %.1f m\n", mean(min_asset_range_defended));
294 fprintf("Average defender dv used: %.3f m/s\n", mean(defender_dv_used));
295 fprintf("Block mode used in %.1f%% of runs\n", 100*mean(block_used));
296
297 finite_burns = first_burn_time(first_burn_time < inf);
298
299 if isempty(finite_burns)
300     fprintf("Median defender response time: n a\n");
301 else
302     fprintf("Median defender response time: %.1f s\n", median(finite_burns));
303 end
```

```
304
305 fprintf("99th percentile min asset range (defended): %.1f m\n",
    prctile(min_asset_range_defended, 99));
306
307 %% Plots
308 figure;
309 histogram(min_asset_range_defended, 30);
310 xline(Rkoz, 'r--', 'LineWidth', 2);
311 xline(Rbuffer, 'r-', 'LineWidth', 2);
312 xlabel("Minimum Asset Range (m)");
313 ylabel("Number of Runs");
314 title("Closest Approach to Asset (Defended)");
315
316 figure;
317 histogram(min_asset_range_free, 30);
318 xline(Rkoz, 'r--', 'LineWidth', 2);
319 xlabel("Minimum Asset Range (m)");
320 ylabel("Number of Runs");
321 title("Closest Approach to Asset (Free)");
322
323 figure;
324 histogram(defender_dv_used, 30);
325 xlabel("Defender dv Used (m/s)");
326 ylabel("Number of Runs");
327 title("Fuel Consumption");
328
329 figure;
330 pie([sum(success_asset_KOZ) sum(~success_asset_KOZ)]);
331 legend("Asset KOZ Enforced", "Asset KOZ Violated", 'Location', 'best');
332 title("Asset KOZ Enforcement");
333
334 %% Trajectory vid
335 run_to_plot = 2;
336 figure('Position', [100 100 900 900]);
337
338 vid = VideoWriter('KOZ_run_2.mp4','MPEG-4');
339 vid.FrameRate = 30;
340 vid.Quality = 100;
341 open(vid);
342
343 intr_traj = squeeze(intr_hist_all(:,:,run_to_plot));
344 def_traj  = squeeze(def_hist_all(:,:,run_to_plot));
345
346 for k = 1:5:N
347     clf;
348     hold on;
349
350     th = linspace(0, 2*pi, 100);
351     fill(Rkoz*cos(th), Rkoz*sin(th), [1 0.9 0.9], 'EdgeColor', 'r', 'LineWidth', 2);
352     plot(Rbuffer*cos(th), Rbuffer*sin(th), '--', 'Color', [0.9 0.7 0], 'LineWidth', 2);
353
354     trail_length = min(k, 150);
355     trail_start  = max(1, k-trail_length);
356
357     plot(intr_traj(2, trail_start:k), intr_traj(1, trail_start:k), 'Color', [1 0.5 0 0.4],
    'LineWidth', 2.5);
358     plot(def_traj(2, trail_start:k),  def_traj(1, trail_start:k),  'Color', [0 0.5 1 0.4],
    'LineWidth', 2.5);
359
360     plot([0 intr_traj(2,k)], [0 intr_traj(1,k)], 'r--', 'LineWidth', 1);
361
```

```
    plot(intr_traj(2,k), intr_traj(1,k), 'o', 'MarkerSize', 20, 'MarkerFaceColor', [1 0.3 0], 'MarkerEdgeColor', 'k', 'LineWidth', 2.5);
    plot(def_traj(2,k),  def_traj(1,k),  's', 'MarkerSize', 20, 'MarkerFaceColor', [0 0.5 1], 'MarkerEdgeColor', 'k', 'LineWidth', 2.5);

    plot(0, 0, 'p', 'MarkerSize', 30, 'MarkerFaceColor', 'g', 'MarkerEdgeColor', 'k', 'LineWidth', 3);

    current_asset_range = norm(intr_traj(1:3,k));
    def_intr_sep        = norm(intr_traj(1:3,k) - def_traj(1:3,k));

    axis equal;
    grid on;
    xlabel('Y (m)', 'FontSize', 13, 'FontWeight', 'bold');
    ylabel('X (m)', 'FontSize', 13, 'FontWeight', 'bold');

    max_extent = max([max(abs(intr_traj(1:2,1:k)), [], 'all'), Rbuffer*1.3]);
    xlim([-max_extent max_extent]);
    ylim([-max_extent max_extent]);

    title(sprintf('Run %d  Time %.0f s  Asset Range %.0f m  Def Int %.0f m', ...
        run_to_plot, (k-1)*dt, current_asset_range, def_intr_sep), ...
        'FontSize', 14, 'FontWeight', 'bold');

    if current_asset_range < Rkoz

        status_text  = 'KOZ VIOLATED';
        status_color = [1 0 0];

    elseif current_asset_range < Rbuffer

        status_text  = 'BLOCKING';
        status_color = [1 0.5 0];

    else

        status_text  = 'MONITORING';
        status_color = [0 0.7 0];

    end

    text(0.02, 0.98, status_text, 'Units', 'normalized', ...
        'FontSize', 16, 'FontWeight', 'bold', 'Color', status_color, ...
        'VerticalAlignment', 'top');

    hold off;
    drawnow;

    drawnow;
    frame = getframe(gcf);
    writeVideo(vid, frame);

end

close(vid);

%%

function dv_vec = choose_best_burn_gradient(def, intr_states, Phi_stacked, Nlook, dv_mag)

    def_states = reshape(Phi_stacked * def, 6, Nlook);
```

```
420
421     intr_pos = intr_states(1:3,:);
422     intr_ranges = sqrt(sum(intr_pos.^2, 1));
423     [~, idx_closest] = min(intr_ranges);
424
425     threat_position = intr_states(1:3, idx_closest);
426     threat_velocity = intr_states(4:6, idx_closest);
427
428     def_position = def_states(1:3, idx_closest);
429     def_velocity = def_states(4:6, idx_closest);
430
431     ideal_block_position = 0.7 * threat_position;
432
433     position_error = ideal_block_position - def_position;
434     velocity_error = threat_velocity - def_velocity;
435
436     guidance = 0.7 * position_error + 0.3 * velocity_error * 10;
437
438     if norm(guidance) > 1e-6
439
440         dv_vec = dv_mag * guidance / norm(guidance);
441
442     else
443
444         dv_vec = [0;0;0];
445
446     end
447 end
448
449 function [intr_states, rmin, rdot_min] = predict_range_and_rate(intr, Phi_stacked, Nlook)
450
451     intr_states = reshape(Phi_stacked * intr, 6, Nlook);
452     r = intr_states(1:3,:);
453     v = intr_states(4:6,:);
454
455     rr = sqrt(sum(r.^2,1));
456     [rmin, idx] = min(rr);
457
458     rhat = r(:,idx) / max(rmin, 1e-9);
459     rdot_min = dot(v(:,idx), rhat);
460 end
461
462 function [kp, kd, amax, intr_dv0] = random_intruder_params()
463
464     if rand() < 0.05
465
466         kp = 0.001; kd = 0.02; amax = 0.002; intr_dv0 = 3;
467
468     else
469
470         kp = 0.005 + 0.010*rand();
471         kd = 0.05 + 0.15*rand();
472         amax = 0.02 + 0.04*rand();
473         intr_dv0 = 8 + 12*rand();
474
475     end
476 end
477
478 function intr0 = random_intruder_init()
479
480     min_radius = 3200;
```

```
    max_radius = 4000;

    radius = sqrt(min_radius^2 + (max_radius^2 - min_radius^2) * rand());
    theta  = 2*pi*rand();

    r = radius * cos(theta);
    y = radius * sin(theta);
    z = 0;

    pos_vec = [r; y; 0];
    approach_dir = -pos_vec / norm(pos_vec);

    angle_variation = (rand() - 0.5) * pi/9;
    rotation = [cos(angle_variation) -sin(angle_variation) 0;
                sin(angle_variation)  cos(angle_variation) 0;
                0                     0                    1];

    vel_dir = rotation * approach_dir;

    speed = 0.4 + 0.5*rand();
    vx = speed * vel_dir(1);
    vy = speed * vel_dir(2);
    vz = 0;

    intr0 = [r; y; z; vx; vy; vz];

end

function Phi = cw_phi(n, dt)

    c = cos(n*dt); s = sin(n*dt);
    Phi = [4-3*c, 0, 0, s/n, 2*(1-c)/n, 0;
           6*(s-n*dt), 1, 0, -2*(1-c)/n, (4*s-3*n*dt)/n, 0;
           0, 0, c, 0, 0, s/n;
           3*n*s, 0, 0, c, 2*s, 0;
           -6*n*(1-c), 0, 0, -2*s, 4*c-3, 0;
           0, 0, -n*s, 0, 0, c];

end

function Gamma = cw_gamma(n, dt)

    c = cos(n*dt); s = sin(n*dt);
    Gamma = [(1-c)/n^2,          (2/n^2)*(n*dt - s),     0;
             (2/n^2)*(s - n*dt), (4*(1-c)/n^2 - 3*dt^2/2), 0;
             0,                  0,                      (1-c)/n^2;
             s/n,                2*(1-c)/n,              0;
             -2*(1-c)/n,         (4*s/n - 3*dt),         0;
             0,                  0,                      s/n];
end

function min_asset = simulate_intruder_only( ...
    intr0, kp, kd, amax, intr_dv0, Phi_dt, Gamma_dt, n, dt, N, noise_seq, target_u)

    intr = intr0;
    intr_dv_rem = intr_dv0;

    intr_filt  = intr0;
    alpha_filt = 0.3;

    min_asset = inf;
```

```
542
543     for k = 2:N
544
545         intr_meas = intr + noise_seq(:,k);
546         intr_filt = alpha_filt * intr_meas + (1-alpha_filt) * intr_filt;
547
548         r = intr(1:3);
549         v = intr(4:6);
550
551         ax_ff = 3*n^2*r(1) + 2*n*v(2);
552         ay_ff = -2*n*v(1);
553         az_ff = n^2*r(3);
554
555         a_cmd = [ax_ff; ay_ff; az_ff] - kp*r - kd*v;
556
557         if target_u(k) < 0.96
558             if norm(r) > 1e-9
559                 target_dir = -r / norm(r);
560                 a_cmd = a_cmd + 0.5 * target_dir;
561             end
562         end
563
564         an = norm(a_cmd);
565         if an > amax
566             a_cmd = (amax/an) * a_cmd;
567         end
568
569         dv_step = norm(a_cmd)*dt;
570         if intr_dv_rem > 0
571             if dv_step > intr_dv_rem
572                 a_cmd = a_cmd * (intr_dv_rem / dv_step);
573                 dv_step = intr_dv_rem;
574             end
575             intr_dv_rem = intr_dv_rem - dv_step;
576         else
577             a_cmd = [0;0;0];
578         end
579
580         intr = Phi_dt * intr + Gamma_dt * a_cmd;
581
582         r_asset = norm(intr(1:3));
583         if r_asset < min_asset
584             min_asset = r_asset;
585         end
586     end
587 end
588
```