Table of Contents

..... 1
Antenna Parameters 1
waveform Parameters 1
waveform Parameters for graphs 1
RRE parameters 2


```
classdef radarClass
    %Class to store radar parameters, constants, and requirements


    properties

        type %dewds 1 or dewds 2
        % Storage
        storage
        %Frequency
        dopMax
        dopAvg
```

Antenna Parameters

```
        antennaSizeX
        antennaSizeY
        numAntenna
        antennaSpin % in rpm
        rangeRes
        Ae
```

waveform Parameters

```
        TpTrack
        TpSearch
        PRISearch
        PRITrack
        freq
        lambda
        PRIPerDwell
        bandWidthTrack
        bandWidthSearch

        TDwellSearch
        TDwellTrack
        TfsSearchMin
        TfsSearchMax
```

waveform Parameters for graphs

```
        PRFAvgMin
```

```
        PRFMaxMin
```

RRE parameters

```
        PPeak
        Pt
        Pt_track
        duty_cycle
        Gain

        %Range and az/El coverage
        rangeTrack
        rangeSearch
        azCoverage
        elCoverageS %search
        elCoverageT %track
        solidAngleTrack
        solidAngleSearch
        beamWidthTrack
        beamWidthSearch
        nBeamsS
        nBeamsT

        %requirments
        R_warningTime % in seconds
        R_rangeResTrack %in meters
        R_rangeResSearch
        SNRmin_search
        SNRmin_track

        c = physconst("lightspeed");
        k = physconst("Boltzman");
        To = 290;


        %Losses
        Ls = 1;
        F = 1.413;

        %Results
        calcSNRTrack
        calcSNRSearch

    end

    methods

        function radar = radarClass(dewdsType, varFreqFlag)

            radar.rangeSearch = [30*10^3 300*10^3];
            radar.PPeak = 1*10^6;
%             radar.duty_cycle = .2; %needs to be fixed
```

```
            radar.antennaSizeX = 5;
            radar.antennaSizeY = 5;
            radar.azCoverage = 2*pi;
            radar.Ae = radar.areaEffective(radar.antennaSizeX);

            % enter in required values
            radar.R_rangeResTrack = 10;
            radar.R_rangeResSearch = 30;
            radar.R_warningTime = 5*60;
            radar.SNRmin_search = 10^(26/10);

            radar.TpSearch = (2.*radar.R_rangeResSearch)./radar.c;
            radar.bandWidthSearch = 1./radar.TpSearch;

            radar.type = dewdsType;
            if radar.type == "dewds1"
                radar.antennaSpin = 60;
                radar.numAntenna = 1;
                radar.freq = 0.5*10^9;
                radar.duty_cycle = radar.calc_duty_cycle();
            radar.Pt = radar.PPeak*radar.duty_cycle;



            elseif radar.type == "dewds2"
                radar.rangeTrack = [300 30*10^3];
                radar.numAntenna = 4;
                radar.TpTrack = (2.*radar.R_rangeResTrack)./radar.c;
                radar.bandWidthTrack = 1./radar.TpTrack;
                radar.freq = 1*10^9;
                radar.duty_cycle = radar.calc_duty_cycle();
                radar.Pt = radar.PPeak*radar.duty_cycle;
                radar.Pt_track = radar.PPeak*0.05 %duty_cylce for
 track

            end

        %              radar.freq = 1*10^9;
                radar.lambda = radar.c ./radar.freq;
                radar.dopMax = (2*500)/radar.lambda;
                radar.dopAvg = (2*200)/radar.lambda;
                radar.PRFMaxMin = (4*500)/radar.lambda;
                radar.PRFAvgMin = (4*200)/radar.lambda;

        end

        function res = rangeResFunc(radar, Tp)
            res = (radar.c/2)*Tp;
        end

        function duty_cycle = calc_duty_cycle(radar)
            tau     = (radar.R_rangeResSearch)./(2*radar.c);
            PRI     = (radar.rangeSearch)./(2*radar.c);
            duty_cycle  = max(tau./PRI);
```

```
        end

        function duty_cycle = calc_duty_cycle_track(radar)
            tau     = (radar.R_rangeResTrack)./(2*radar.c);
            PRI     = (radar.rangeTrack)./(2*radar.c);
            duty_cycle  = max(tau./PRI);
        end

        function angle = elAngle(radar, range, alt)
            angle = atan(alt ./range);
        end

        function PRI = PRI_calc(radar, range)
            PRI = 2*range/radar.c;
        end

        function SA = solidAngle(radar, el)
            SA = 2*pi*sin(el);
        end

        function BW = beamWidth(radar, freq, D)
            BW = 0.89*(radar.c./freq)./D;
        end

        function BC = beamCoverage(radar, solidAngle, theta3, phi3)
            BC = solidAngle./(theta3.*phi3);
        end

        function BW = bandWidth(radar, Tp)
            BW = 1./Tp;
        end

        function BWCalc = BWCalculation(radar, c)
            BWCalc = c/rangeRes;
        end

        function priMax = PRI_max(radar, f, vmax)
            priMax = (radar.c ./f)./(4*vmax);
        end

        function prf = PRF(radar, PRI)
            prf = 1./PRI;
        end

        function radar = GainCalc(radar)
            radar.Gain = 32400./(radar.beamWidthSearch.*...
                radar.beamWidthSearch*(180/pi)^2); %note this must be
    BW in degrees
        end

        function Pavg = Pavg(radar,  PRI, B)
         %pavg = Pt*nPulses./(Td*B);
         Pavg = radar.PPeak*(1./B)*(1./PRI);
        end
```

```
        function Pave_sweep = sweep_Pave(radar, Pt, dutyCyc)
            Pave_sweep = Pt.*dutyCyc;
        end
        function ae = areaEffective(radar, D)
            ae = D^2; %piazza post Ae = efficiency * A, efficiency = 1
 for our system;
        end

        function lhs = SNR_Track_LHS(radar, Pavg, G, Ae, Ls, F)
            lhs = Pavg*G*Ae / Ls*F;
        end

        function rhs = SNR_Track_RHS(radar, SNR, range, PRF, RCS)
            rhs = SNR*(4*pi)^2 * range^4 * radar.k * radar.To * PRF /
 RCS ...
                *1/Tfs;
        end

        function lhs = SNR_Search_LHS(radar, PAvg, Ae, F, Ls)
            lhs = PAvg.*Ae/(Ls * radar.To * F);
        end


        function rhs = SNR_Search_RHS(radar, dragon)
            RCS = dragon.RCSRange(1);
            rangeS = radar.rangeSearch(2);
            SNRmin = radar.SNRmin_search;
            num_pulse = SingBeamSNR(radar, RCS, SNRmin);
            Tfs = radar.time_range(num_pulse, max(dragon.speedRange));
            % choose max dragon speed
            rhs = SNRmin*4*pi*(rangeS^4/
RCS).*(radar.solidAngleSearch./Tfs);
        end

        function td = Td(radar, updateRate, NTargets)
            td = (updateRate*NTargets);
        end


        function radar = varFreq(radar, freq, vMax, vAvg)
            radar.freq = freq;
            radar.PRFMaxMin = radar.PRF(radar.PRI_max(radar.freq,
 vMax));
            radar.PRFAvgMin = radar.PRF(radar.PRI_max(radar.freq,
 vAvg));
            radar.beamWidthSearch = radar.beamWidth(freq,
 radar.antennaSizeX);

            if radar.type == "dewds2"
                radar.beamWidthTrack= radar.beamWidthSearch;
            end
```

```
        end

        function radar = SNRTrack(radar,RCS)
            radar = GainCalc(radar);
            radar.calcSNRTrack =
 radar.Pt_track.*radar.Gain.^2.*radar.lambda.*RCS(1).*1./
((4*pi)^3.* ...
                radar.rangeTrack.^4.*radar.k.*radar.To.*...
                radar.F.*radar.bandWidthTrack.*radar.Ls);
        end

        function radar = SNRSearch(radar,RCS)
            disp('stop')
            radar = GainCalc(radar);
            radar.calcSNRSearch = radar.Pt.*radar.Ae.*RCS(1).*...
                radar.TfsSearchMax./((4*pi).*radar.k.*radar.To.*...

 radar.F.*radar.Ls.*radar.rangeSearch.^4.*radar.solidAngleSearch);
        end



        function radar = time_range(radar,  num_pulse, maxspeedRange,
 dragons_Tracked)

            if radar.type == "dewds1"
                %dwell determined by spin rate and el coverage
                TDwell_Az = radar.beamWidthSearch /
 2*pi*radar.antennaSpin/60 ;

                numBeamsPerAz = ceil(radar.elCoverageS ./
radar.beamWidthSearch);
                TDwell = TDwell_Az/numBeamsPerAz;
                               if numBeamsPerAz > TDwell/
radar.PRISearch
                                   disp('Frequeny is TOO HIGH')
                               end

                radar.TDwellSearch = TDwell;
            else
                %look at dwell for different num of pulses
                TDwell = num_pulse.*(radar.PRISearch);
                radar.TDwellSearch = TDwell;
            end

            M = radar.solidAngleSearch;

            radar.TfsSearchMin = TDwell.*M/(radar.beamWidthSearch)^2;

            Crossrange_1beam =
 min(radar.rangeSearch)*(radar.beamWidthSearch);      % What is the
 crossrange distance of one beam? (theta/360 x circumference of 300m
 circle)
            threebeam_distance = Crossrange_1beam*3;
```

```
            % Tfs search has to been such that you can search the
whole
            % area before the dragon can move three beams
            dragonTravelTime = threebeam_distance / maxspeedRange;

            radar.TfsSearchMax = dragonTravelTime;
            %Tfs = [Tfs1 Tfs2];

            if radar.type == "dewds2"
                TDwell = radar.PRITrack*num_pulse;

                Crossrange_1beam =
min(radar.rangeTrack)*(radar.beamWidthTrack);      % What is the
crossrange distance of one beam? (theta/360 x circumference of 300m
circle)
                threebeam_distance = Crossrange_1beam*3;
   % Per spec, dragon not allowed to go more than three of these

                %number of cells to search
                Rosette = 25;

                revisit_time= dragons_Tracked.*TDwell.*Rosette;  % The
time it takes to revist a tracked dragon (25 is the rosette squares
for 3 beamwidths)

                disTraveled =  maxspeedRange.*revisit_time;

                if (disTraveled <= threebeam_distance)
                    radar.TDwellTrack = TDwell;
                    %radar.num_pulse = num_pulse;
                else
                    fprintf("Number of dragons is too high!")
                end

            end


        end
        function storage = calc_storage(radar, dragon)
            %#Range bins x #beams x #NumPulsesPerDwell).
            % Assume that complex values are stored as 2x8 byte
(double)
            % numbers in storage. (Note: 1 GB = 1024^3 = 2^30 bytes
            minTimeSearch   = abs(diff(radar.rangeSearch))./([200
500]);
            storage =
(2*8).*ceil(radar.nBeamsS).*ceil(abs(diff(radar.rangeSearch))./
radar.R_rangeResSearch)*1./ ...
                        2^30;
                    storage = storage.*minTimeSearch;
        end

    end
```

```
end
```

```
Not enough input arguments.

Error in radarClass (line 107)
            radar.type = dewdsType;
```

Published with MATLAB® R2017b