ASEN 6365 Final

Maya Greenstein
9 May 2023

Contents

1 Introduction

Multi-channel polarized ranging lidar (MPL) is an actively growing remote sensing field. Its main unique property compared to any standard lidar is its receiver-to-detector configuration. Figure 1 is an illustration of this configuration, showing the scattered light entering the receiver, a beam-splitter splitting that light into what is co-planar polarized and cross-planar polarized to the transmitted light, and then those split beams being detected by different photomultiplier tubes (PMT). Some MPLs contain more than two channels to more completely understand the polarization state of the received light.

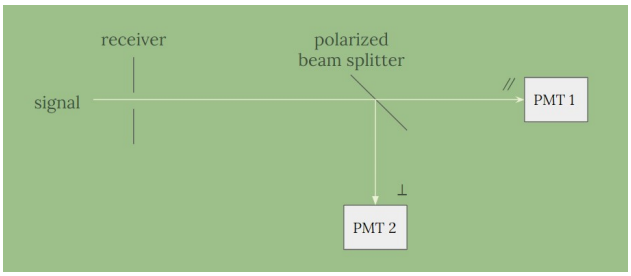


Fig. 1 Receiver Configuration

These lidar systems operate on the principle that different scatterers will affect polarization differently, and that, tracking that property, more than just the intensity of the light, will provide information about the scatterers. This project will focus on surface scattering, specifically how smooth vs rough surfaces affect polarization. Smooth surfaces are best described as specular reflectors and polarization preservers. This means that for a perfectly smooth surface with no surface impurities, all of the reflected light would follow a path with the same angle as the incident light, and the polarization of that reflected light would the same as it came in. Conversely, a rough surface will spread out the incident light in a diffuse manner, scattering it in all different directions. In this process, it will also depolarize the signal. The perfect example of this is called Lambertian scattering where the light is evenly distributed in all directions, and a perfect depolarizer is when none of the incident polarization is maintained. Figure 2 shows the physical properties of these two types of scattering surfaces. Any real surface will have properties that fall somewhere between the two extremes.

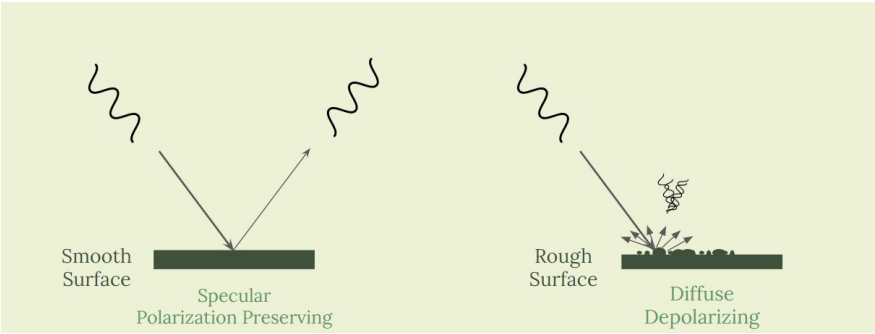


Fig. 2 Surface Polarization Properties

The applications of this include anything where it is desirable to understand the reflective polarization properties of the scatterer. Bathymetric lidar often uses these properties to distinguish between what is the water surface, and what is the bottom. The water surface acts as a pretty ideal smooth surface and will very consistently preserve laser pulse polarization. Additionally, bathymetric surfaces, which are often sandy or rocky will depolarize consistently. This makes the two easy two differentiate and thus a good candidate for MPL. Another candidate is vegetation classification. One study[2] used a multi-wavelength MPL to attempt to classify forest canopy makeup. They are also sometimes used to understand atmospheric aerosols[3] or aerosol effects on clouds[4]. This project will focus on the bathymetric applications.

2 Polarization Description

2.A Stokes Vectors

To describe the polarization state of light, a four-element matrix called a Stokes vector is used. The structure of this vector is described by Eq 1 where the first component, S_0, describes the overall intensity of the light the vector is describing, and the other three elements describe the state of polarization. S_1 is the horizontal and vertical polarization, S_2 is the 45° linear polarization and S_3 the circular polarization. In a normalized Stokes vector, the three bottom values will range from -1 to 1, while the first, intensity, will be between 0 and 1. For a fully polarized light state, $S_0^2 = S_1^2 + S_2^2 + S_3^2$, however after experiencing any kind of depolarization, $S_0^2 > S_1^2 + S_2^2 + S_3^2$. An example of a Stokes vector for a fully vertical-linear polarized light state is shown in Eq 2.

$$S = \begin{bmatrix} S_0 \\ S_1 \\ S_2 \\ S_3 \end{bmatrix} \tag{1}$$

$$S = \begin{bmatrix} 1 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{2}$$

2.B Mueller Matrices

With light now defined as a matrix, there needs to be a compatible way to describe its interactions. To do this, a 4x4 Mueller matrix is defined to attenuate the intensity and rotate the polarization states. A Mueller matrix is defined by Eq 3, all of the elements here will be defined by the type of interaction. The Stokes vector will change based on Eq 4 where S' is the output state of light after S experiences an interaction described by M.

$$\begin{bmatrix} m_{00} & m_{01} & m_{02} & m_{03} \\ m_{10} & m_{11} & m_{12} & m_{13} \\ m_{20} & m_{21} & m_{22} & m_{23} \\ m_{30} & m_{31} & m_{32} & m_{33} \end{bmatrix} \tag{3}$$

$$S' = M \cdot S \tag{4}$$

3 Stokes Vector Lidar Equation

Using the mathematical concepts of Stokes vectors and Mueller matrices, a lidar equation that includes polarization states can be crafted. This so-called Stokes vector lidar equation (SVLE) is shown in Eq 5. This particular form is describing the percentage of signal that will be returned per bin, where R is the range to that bin in meters.

The lone term on the right side of this equation is $\vec{S}_B$, the Stokes vector describing the background signal per range bin. The only Mueller matrix consideration for this vector is its interaction with the receiver, M_{Rx}. For the rest of the grouped terms, the equation from right to left essentially explains the path of the light from the transmitter to the receiver. $\vec{S}_{Tx}$ and M_{Tx} describe the transmitted light, where $\vec{S}_{Tx}$ is actually the vertical linearly polarized vector from Eq 2 and M_{Tx} accounts for receiver efficiencies and any offset angles. Next, the incident beam will propagate through the atmosphere, described here by $T_{atm}(\vec{k}_i, R)$. For current purposes, the atmospheric transmission of the reflected vector, $T_{atm}(\vec{k}_s, R)$, will be considered equivalent. The most significant term here, $F(\vec{k}_i, \vec{k}_s, R)$, is called the scattering phase matrix, as it is the dominant driver of any depolarization happening through scattering events. Lastly, there are the scalar terms. G(R) is the geometric phase function describing how much of the beam is actually in the receiver field of view, A is the receiver area, and ΔR is the size of the bin.

$$\vec{S}_{Rx}(R) = M_{Rx}[(G(R)\frac{A}{R^2}\Delta R)T_{atm}(\vec{k}_s, R) \times F(\vec{k}_i, \vec{k}_s, R)T_{atm}(\vec{k}_i, R)M_{Tx}\vec{S}_{Tx} + \vec{S}_B] \tag{5}$$

The first Mueller matrix to discuss is the scattering phase matrix. Depending on the type of scattering, this can include a few different elements. The most basic form, however, is simply a depolarization matrix defined by Eq

6 [5]. The first quantity to discuss is not present in this matrix, but the depolarization ratio δ is the resulting ratio of perpendicular to parallel polarized light in a light packet after being scattered Eq 7. The quantity d has a simple relationship with δ shown in Eq 8, qualitatively described as a measure of the likelihood for a scatterer to depolarize light. A scatterer with d=0 is fully polarization preserving, while one where d=1 is fully depolarizing.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1-d & 0 & 0 \\ 0 & 0 & d-1 & 0 \\ 0 & 0 & 0 & 2*d-1 \end{bmatrix} \tag{6}$$

$$\delta = \frac{I_{\perp}}{I_{\parallel}} \tag{7}$$

$$d = \frac{2\delta}{1+\delta} \tag{8}$$

When atmospheric scattering is considered, the Rayleigh backscatter coefficient will be used as a scalar multiple for this equation. This works out as that value, $\beta_{Rayleigh}$, is measured in units $m^{-1}sr^{-1}$. The $\frac{A}{R^2}$ term in Eq 5 has the units of sr and ΔR has units of m. However, when surface scattering, F will not be multiplied by a backscatter coefficient, but rather a surface reflectance with units of sr^{-1}. This means that the ΔR in Eq 5 must be omitted when considering surface scattering.

Another possible multiplier for this F matrix is used when the scatterer is a surface such as water where much of the signal will be transmitted through instead of reflected. For bathymetric lidar, this is, of course, an important element to consider. Using Fresnel's equations that describe how light travels through two different media, the matrix in Eq 9 is formed with the elements described in Eqs 10-12 and $\gamma_{Im} = 0$. The full scattering phase matrix for a water surface is formed by multiplying this matrix with the depolarization matrix from Eq 6.

$$\begin{bmatrix} \alpha+\eta & \alpha-\eta & 0 & 0 \\ \alpha-\eta & \alpha+\eta & 0 & 0 \\ 0 & 0 & \gamma_{Re} & -\gamma_{Im} \\ 0 & 0 & \gamma_{Im} & \gamma_{Re} \end{bmatrix} \tag{9}$$

$$\alpha = .5[\frac{tan(\theta_i - \theta_t)}{tan(\theta_i + \theta_t)}]^2 \tag{10}$$

$$\eta = .5[\frac{sin(\theta_i - \theta_t)}{sin(\theta_i + \theta_t)}]^2 \tag{11}$$

$$\alpha = -\frac{tan(\theta_i - \theta_t)sin(\theta_i - \theta_t)}{tan(\theta_i + \theta_t)sin(\theta_i + \theta_t)} \tag{12}$$

The transmission matrix is simply a diagonal matrix with the calculated transmission based on atmospheric density. The transmission value can be solved using the coefficient of extinction of the medium of propagation, α, as shown in Eq 13. Assuming the transmitter and receiver are co-linear, the Mueller matrices describing those will also be diagonal matrices, populated with their efficiencies. For the receiver with a perpendicular polarization, the receiver matrix will also account for that rotation.

$$T(R) = \int_0^R \alpha dR \tag{13}$$

4 Simulation

4.A Lidar Parameters

Some aspects of the lidar system that inspired this simulation is protected information, however, the properties integral to this simulation can be shared. One of the most important ones is the operating wavelength of 532nm. This is a fixed laser and cannot be tuned to any other frequency. It does not need to be as 532nm lasers are extremely common in both bathymetric and terrestrial mapping lidars due to the wavelength's penetration abilities in both the atmosphere and water. Due to its relatively short-ranging needs, and requirement to fly on a drone, it also operates at a low energy of $5\mu J$. The resolution is extremely high, time-tagging returns into bins of 25ps. Some other important parameters that will be referenced later are shown in Table 1. The transmitter and receiver efficiencies shown in the table account for all elements in the optical chain.

Component	A_r	η_{Rx}	η_{Tx}
Value	$2mm^2$	.01	.025

Table 1 Additional Lidar Parameters

4.B Process

The first step in the simulation was to define the operating conditions of the lidar. Boulder, CO was arbitrarily chosen as the location of operation, and the lidar was defined to be operating at an altitude of 40m looking down at a surface at

10m. The day was programmed to be the same day as the code operation.

With the setting defined, the atmospheric transmission properties at the chosen location and time could be calculated. To do this, the first step was to solve for the Rayleigh backscatter coefficient shown in Eq 14. The pressure and temperature values were estimated using the MATLAB atmosnrlmsise00 function. The phase function, Eq 15, was solved assuming 180° backscatter. Then those two values could be combined to solve for the total backscatter in Eq 16. Given the low altitude, two assumptions are made. The first is that any absorption is negligible, thus all extinction comes from scattering. The second is that aerosols are dominantly forward scattering, meaning only Rayleigh scattering contributes to extinction. This all provides that the total Rayleigh backscattering is equivalent to the total extinction. Going back to Eq 13, α can be replaced with the calculated β_T.

$$\beta_{Rayleigh}(\lambda, z, \theta) = 2.938X10^{-32}\frac{P(z)}{T(z)} * \frac{1}{\lambda^{4.0117}} \tag{14}$$

$$P(\theta) = .7629X(1 + .9324cos^2(\theta)) \tag{15}$$

$$\beta_T = \frac{4\pi}{P(\theta = \pi)}\beta(\theta) \tag{16}$$

The next step in the simulation was to define the scattering surfaces. Three surfaces were considered in this simulation, a rough opaque surface, a smooth opaque surface, and a water surface. Assuming a pointing angle of .001°and a water depolarization value of d=.02, the procedures described in Section 3 were used to define the scattering matrix. For both opaque surfaces, an albedo of .2 was used. For the rough surface, Lambertian scattering was assumed, assigning the BRDF of the function a value of $\frac{.2}{\pi}$ in units sr^{-1}. For the smooth surface, specular scattering was assumed, meaning all of the radiation is coming in the same direction rather than π directions. Thus the BRDF for the smooth surface was simply .2 sr^{-1}. To avoid too unrealistic conditions, the depolarization value assigned to the rough surface was .9, and for the smooth surface, .2.

4.C Results

With all of those properties defined, the SVLE could finally be solved for all of the propagation altitudes. The returned number of photons for each channel is shown in the resultant plots below. This value comes from simply multiplying the first element of the SVLE by the output photon count, return a percentage of those output photons that are received back. The results for opaque smooth and rough surfaces are shown in Figure 3 and for a water surface in Figure 4.

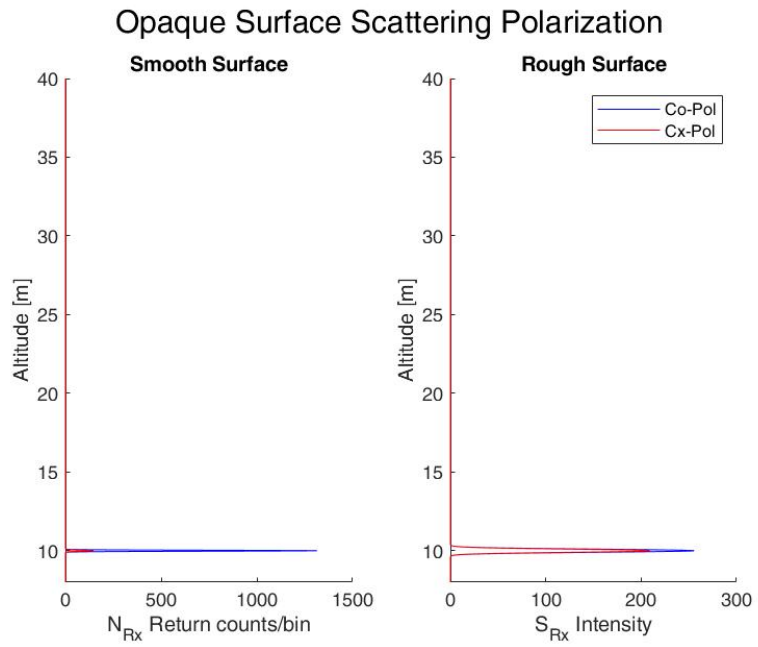


Fig. 3 Ground Scattering

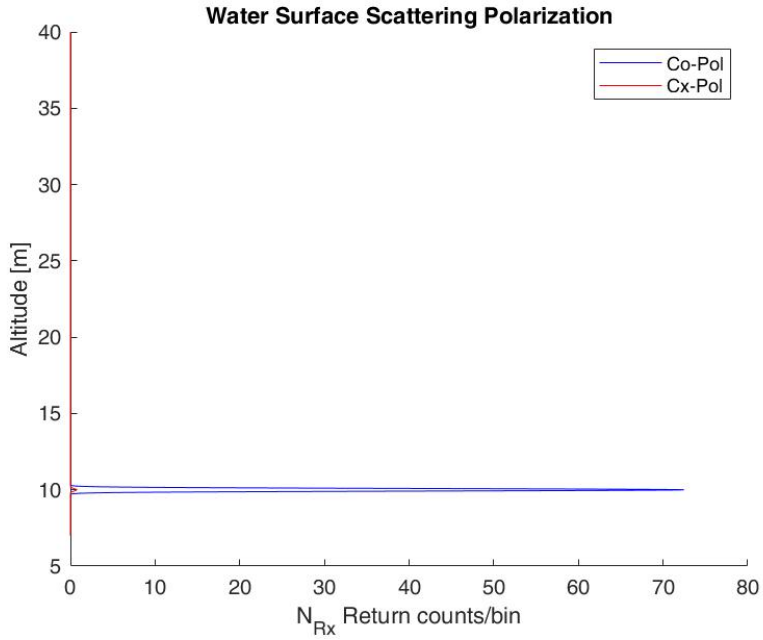


Fig. 4 Water Surface Scattering

The most important part of these results is the relative counts between the surfaces and between co and cross-pol channels. Beginning by looking at the different surfaces, the smooth surface dominates by an order of magnitude, followed by the rough surface, then the water surface. As discussed previously, much of the beam will transmit through the water surface, so seeing the least reflection from that makes sense. Followed by the rough surface which, despite

having the same albedo as the smooth surface, the reflected light will be scattered in all directions. The smooth surface which will solely scatter back towards the receiver.

Now, looking at the relative counts between the detector channels, everything behaves as expected. The smooth surface with its low depolarization constant is returning mostly in the co-polarized channel, with a few counts still in the cross-polarized. The rough surface with its high depolarization is still returning mostly co-polarized, but a very comparable amount cross. It is important to note that with a depolarization constant of 1, an equal amount of co and cross counts can be expected. Finally, for the water surface, there are barely any cross counts due to its near-ideal smoothness.

The overall numbers are quite small due to the small receiver area and efficiencies, however, after testing by increasing these values, they were determined to be logical. These numbers are for one shot, while in reality, the lidar would transmit many shots. Additionally, it is a single-photon-counting lidar, so receiving 100s of counts back from just one shot is plenty to trigger a return.

4.D Real Data Comparison

For some sort of real data reference/validation, here are two datasets depicting returns in different channels. In both data, the blue color shows returns in the co-pol channel and green in the cross-pol. The first images shown in Figure 5 and 6 were taken of coastal waters. In 5, a side view of this image is shown, where the top is the surface, and the bottom is the bathymetry. Already the difference in polarization distribution is abundantly clear. Figure 6 is included as well with the bathymetry cut out to further emphasize this difference.

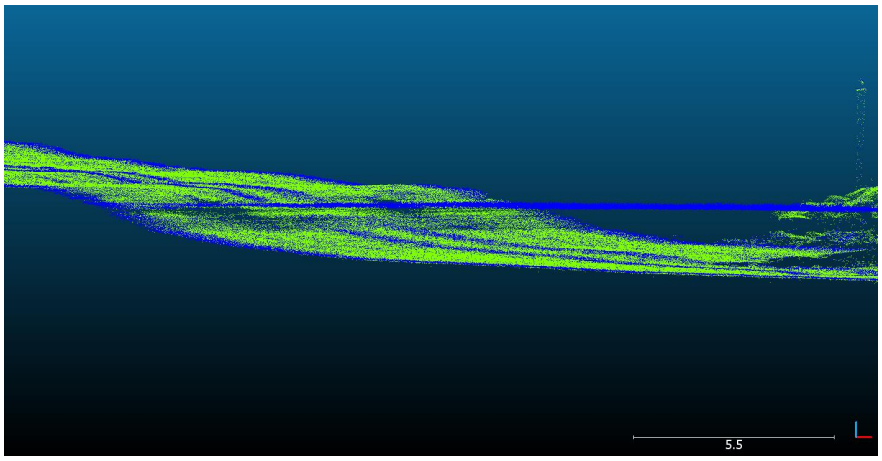


Fig. 5 Coastal Side View

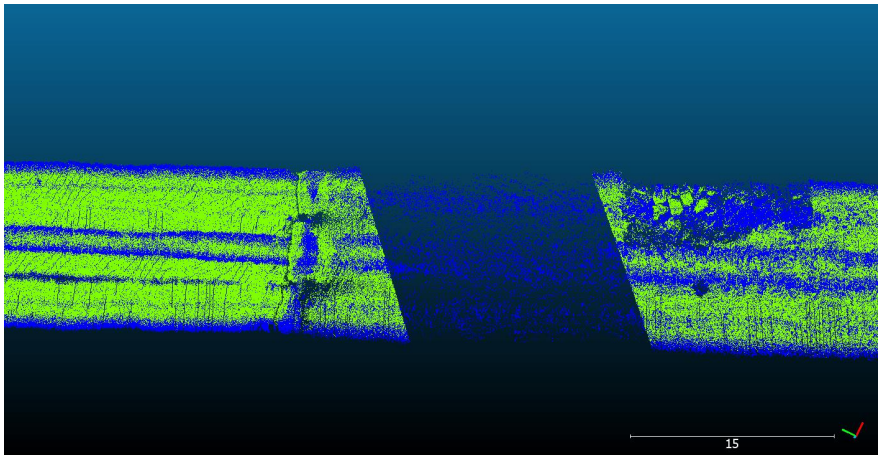


Fig. 6 Coastal Top View

Figure 7 and 8 come from a different topography data set. The image in 7 is, of course, taken with a camera, not a lidar, but it is an interesting comparison to the lidar data. One thing to note is the apparent properties of the white surfaces compared to the dark ones. White surfaces do exhibit higher reflectivity and generally more specular behavior than dark ones. This data set seems to imply that that will inherently lead to higher polarization-preserving properties.

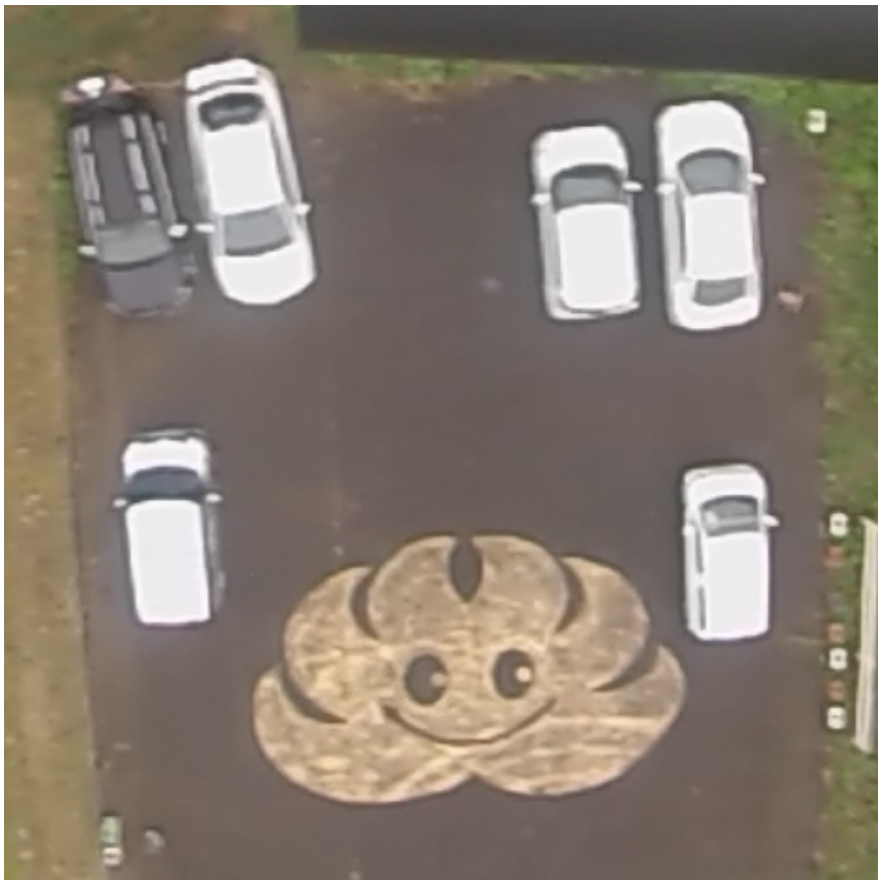


Fig. 7 Topographic Image

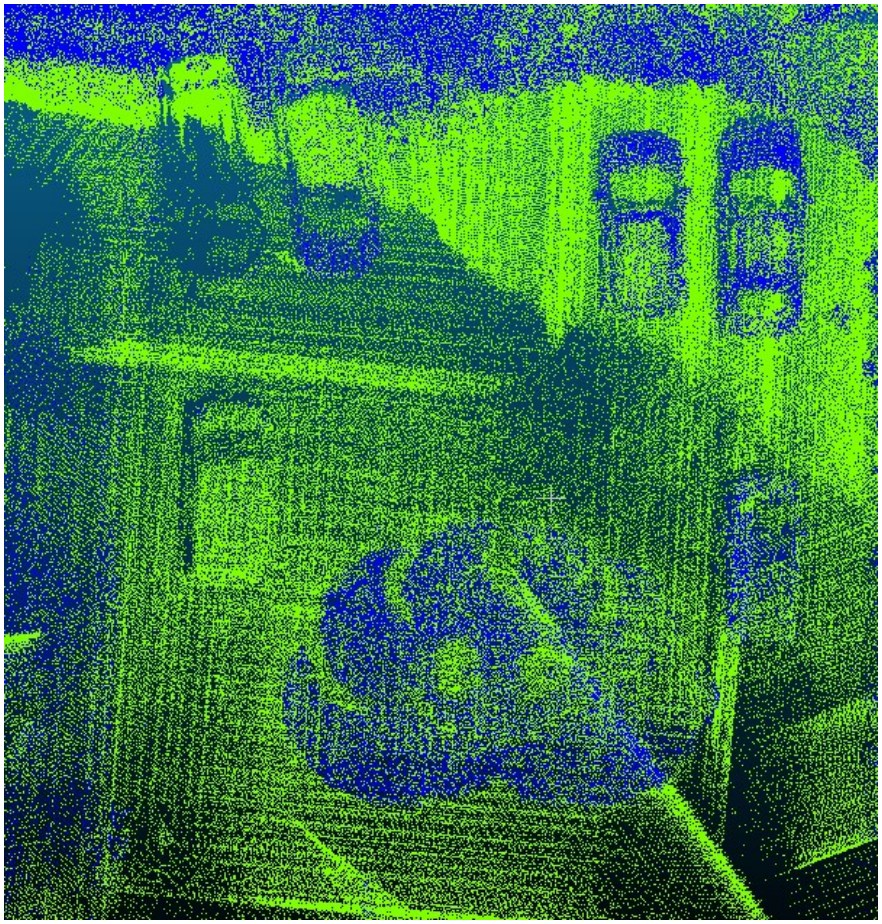


Fig. 8 Topographic Point Cloud

5 Further Work

There were a few elements that I would have liked to add to this project but was not able to given the time constraints. The first is actually adding background noise. The signals displayed in the results are purely from photon counts, and while they do illustrate what I was hoping for, real signals would of course have to account for and learn how to remove that noise. Adding noise would also add the ability to perform statistical analyses on the signal, a process that does not mean much without noise.

The other thing that could really improve the comprehensiveness of the program is adding scanning. The model lidar system is practically defined by its scanning capabilities. Without them, it would be impossible to capture any meaningful amount of mapping data. Scanning means hitting the surface at different incident angles, which will cause different reflectance properties, specifically on the water surface. And if that return is specular and sent in a direction away from the receiver, there's a good chance it will not be seen at all.

References

[1] Goldstein, Dennis H. Polarized Light. CRC Press, 19 Dec. 2017.

[2] Tan, Songxin, et al. “Polarized Lidar Reflectance Measurements of Vegetation at Near-Infrared and Green Wavelengths.” International Journal of Infrared and Millimeter Waves, vol. 26, no. 8, 25 July 2005, pp. 1175–1194, https://doi.org/10.1007/s10762-005-7276-3.

[3] Cottle, Paul Wesley. “Aerosol Type Analysis with Single Wavelength, Dual Polarization Elastic LIDAR.” T. University of British Columbia, 2016. Web. 7 May 2023. https://open.library.ubc.ca/collections/ubctheses/24/items/1.0225867

[4] Jimenez, Cristofer, et al. “The Dual-Field-of-View Polarization Lidar Technique: A New Concept in Monitoring Aerosol Effects in Liquid-Water Clouds – Case Studies.” Atmospheric Chemistry and Physics, vol. 20, no. 23, 8 Dec. 2020, pp. 15265–15284, https://doi.org/10.5194/acp-20-15265-2020.

[5] Matthew Hayman and Jeffrey P. Thayer, "Explicit description of polarization coupling in lidar applications," Opt. Lett. 34, 611-613 (2009)

[6] Barton-Grimley, Rory. Single Photon Counting Lidar Techniques and Instrumentation for Geoscience Applications. 2019.

[7] Wang, Jie, et al. “The Determination of Aerosol Distribution by a No-Blind-Zone Scanning Lidar.” Remote Sensing, vol. 12, no. 4, 13 Feb. 2020, p. 626, https://doi.org/10.3390/rs12040626.

Appendix A: Acknowledgements

A huge thank you to Dr Rory Barton-Grimley for his assistance with this project. His Ph.D. work provided guidance and inspiration for this project and taught me a lot about the SVLE. And when there was an issue with the equation that neither I, nor Kevin Sacca, nor Dr Jeffrey Thayer could figure out, he took time to meet with me and walk me through it.

Thank you to Jeff and Kevin as well, it was a very busy week but they both managed to find some time to help me out.

Table of Contents

ASEN 6365 Final Project

Maya Greenstein 05/01/23

```
clear; clc; close all;
```

Define Constants

```
c = 299792458; %[m/s]
kB = 1.380649e-23; %[(m^2kg)/(s^2K)]
h = 6.62607015e-34; %[m^2kg/s]
av = 6.0221408e23; %[1/mol]
ep0 = 8.854187817e-12; %[F/m]
me = 9.1093837e-31; %[kg]
e = 1.60217663e-19; %[coulumbs]
Ru = 8.314; %[kgm^2/(s^2Kmol)]

delta = .00365; % air depolarization ratio on central Cabannes line at
 532nm
D = 2*delta/(1+delta);
```

Solve Output Signal

```
lp = lidar_params; % load in potassium lidar parameters
Matrix = FindMuellerMatrices;

% Term 1: Number of laser shots out
NL = lp.pE/((h*c)/lp.wl);

% Term 3: Telescope params
A = (pi*(lp.D/2)^2); % primary mirror area
```

Rayleigh Counts from Alts

```
targetHeight = 10; %[m]
lidarHeight = 40; %[m]
% targetHeight = 2010; %[m]
% lidarHeight = 2040; %[m]

% Boulder lat/lon
lat = 40.016869;
lon = -105.279617;
```

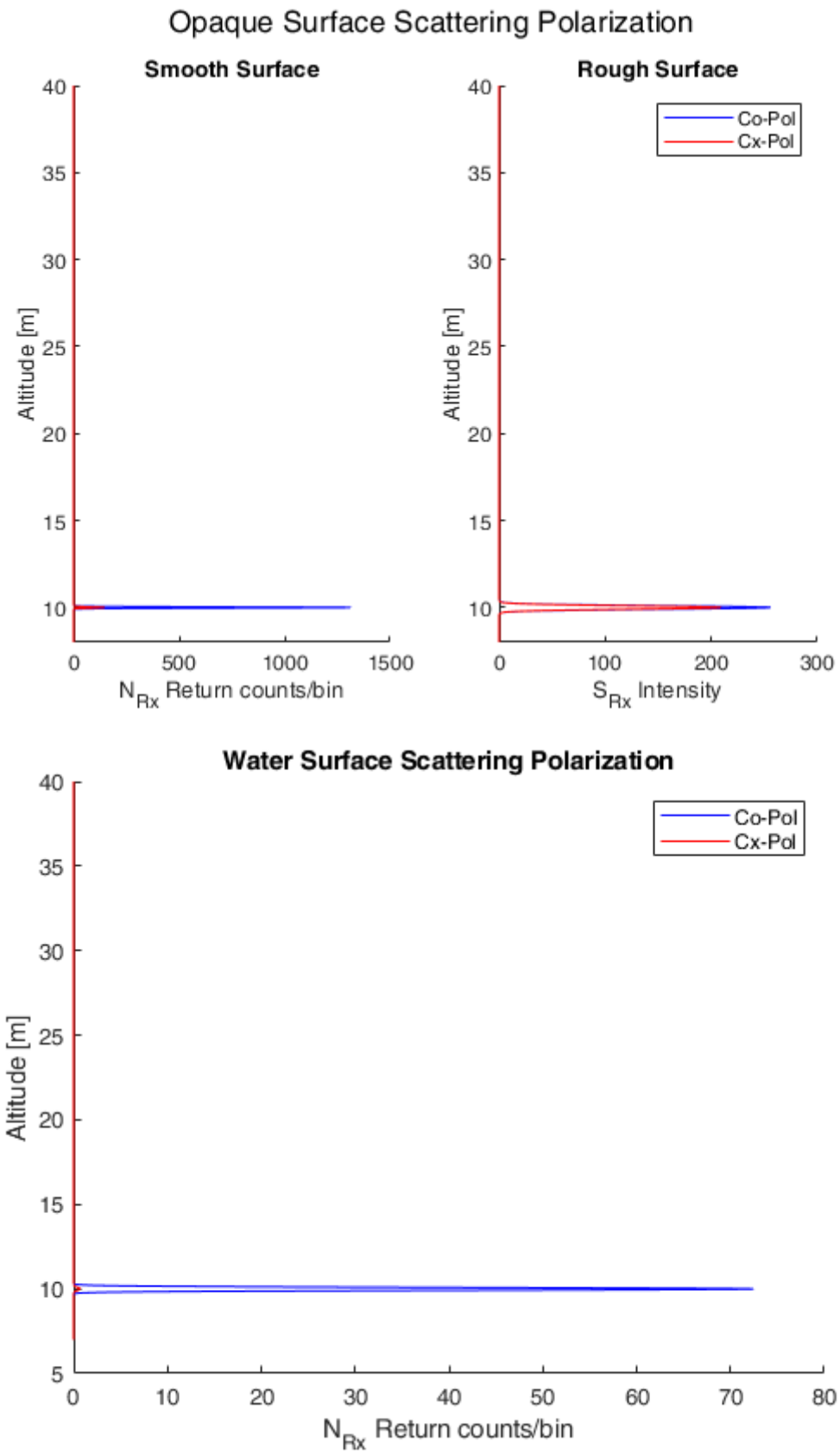
```
% Today
year = 2023;
time = 12*3600;
doy = day(datetime('now'),'dayofyear');

dR = lp.BinSize;
alts = lidarHeight:-dR:(targetHeight-3); %High to low
nBins = length(alts);

% MSIS datas
molCols = [1, 2, 3, 4, 5, 7, 8, 9];

%Rayleigh
[TR, rhoR] = atmosnrlmsise00(alts,lat,lon,year,doy,time, 'None');
TR = TR(:,2); %Just want atmospheric temp
nR = sum(rhoR(:,molCols), 2);
n_VR = nR./av; %n/V in ideal gas law [mols/m^3]
PR = (n_VR.*Ru.*TR)/100; %pressure [Pa->mbar]

betaRL = (2.938e-32)*(PR./TR).*(1./((lp.wl).^(4.0117))); %Rayleigh
 backscatter

P = 0.7629*(1+0.9324*cosd(180)^2); %Rayleigh scattering phase function
betaTotal = (betaRL.*4*pi)./P;
Trans = exp(-cumtrapz(-alts,betaTotal));


% NRl = NL*betaRL'*dR.*(A./((alts-targetHeight).^2)).*(lp.t^2)*lp.eta;
% NRl=flip(NRl); %The Rayleigh counts will happen right in front of
 the detector (why we need overlap function)
% This will look diff that for Na bc NL (from pulse energy),
% A (reciever area) and eta are ALL much lower

% Overlap
[G] = Overlap(alts, lp);
```

Stokes Stuff

```
t = targetHeight/c;

% ROUGH SURFACE

% Define ground
roughGroundPulseWidth = .1;
reflectionGround1 =  normpdf(alts, targetHeight,
 roughGroundPulseWidth);

refAng = pi; % assuming lambertian scattering
wGround = 0.2; %Ground albedo (reflectivity)
BRDF =  wGround/refAng;

% Add dimension to surface using normal distrobution
```

```
reflectionGround1 = BRDF*reflectionGround1./max(reflectionGround1);

% Depolarization
groundD1 = .9;

% Solve SVLE
[Sco1, Scx1] = SVLE(alts, groundD1, D, NL, dR, reflectionGround1,
 Trans, betaRL, G, Matrix, lp, []);

% figure()
% title('Rough Surface')
% hold on
% % plot(Sco1(1,1000:end), alts(1000:end), 'b');
% % plot(Scx1(1,1000:end), alts(1000:end), 'r');
% plot(Sco1(1,:), alts, 'b');
% plot(Scx1(1,:), alts, 'r');
% legend('Co-Pol', 'Cx-Pol');
% xlabel('S_{Rx} Intensity');
% ylabel('Altitude [m]');



% SMOOTH SURFACE (LAMBERTIAN)

% Define ground
smoothGroundPulseWidth = .03;
reflectionGround2 =  normpdf(alts, targetHeight,
 smoothGroundPulseWidth);

reflectionGround2 = wGround*reflectionGround2./max(reflectionGround2);


% Depolarization
groundD2 = .2;
% 80-20 aluminum alloy
[Sco2, Scx2] = SVLE(alts, groundD2, D, NL, dR, reflectionGround2,
 Trans, betaRL, G, Matrix, lp, []);
%
figure()
sgtitle('Opaque Surface Scattering Polarization')

subplot(1,2,1)
title('Smooth Surface')
hold on
plot(Sco2(1,:), alts, 'b');
plot(Scx2(1,:), alts, 'r');
xlabel('N_{Rx} Return counts/bin');
ylabel('Altitude [m]');
ylim([8 40])

subplot(1,2,2)
title('Rough Surface')
hold on
% plot(Sco1(1,1000:end), alts(1000:end), 'b');
```

```
% plot(Scx1(1,1000:end), alts(1000:end), 'r');
plot(Sco1(1,:), alts, 'b');
plot(Scx1(1,:), alts, 'r');
legend('Co-Pol', 'Cx-Pol');
xlabel('S_{Rx} Intensity');
ylabel('Altitude [m]');
ylim([8 40])



% WATER SURFACE (FRESNEL)

% Compute elements for water surface fresnel reflec mat
WAlpha = 0.5*(tan(lp.Pointing - lp.WTransAngle)./ ...
                  tan(lp.Pointing + lp.WTransAngle)).^2;

WEta = 0.5*(sin(lp.Pointing - lp.WTransAngle)./ ...
                  sin(lp.Pointing + lp.WTransAngle)).^2;

WGamma = - (tan( lp.Pointing - lp.WTransAngle).* ...
                           sin(lp.Pointing - lp.WTransAngle))./ ...
                 (tan( lp.Pointing + lp.WTransAngle).* ...
                             sin(lp.Pointing + lp.WTransAngle));

oceanPulseSize = .08;
reflectionWater = normpdf(alts, targetHeight, oceanPulseSize);
reflectionWater = reflectionWater./ max(reflectionWater);


groundD3 = .02; %delta =.02

[Sco3, Scx3] = SVLE(alts, groundD3, D, NL, dR, reflectionWater, Trans,
 betaRL, G, Matrix, lp, [WAlpha, WEta, WGamma]);

figure()
title('Water Surface Scattering Polarization')
hold on
% plot(Sco3(1,1000:end), alts(1000:end), 'b');
% plot(Scx3(1,1000:end), alts(1000:end), 'r');
plot(Sco3(1,:), alts, 'b');
plot(Scx3(1,:), alts, 'r');
legend('Co-Pol', 'Cx-Pol');
xlabel('N_{Rx} Return counts/bin');
ylabel('Altitude [m]');
```


Opaque Surface Scattering Polarization
Smooth Surface
Rough Surface
Co-Pol
Cx-Pol
Altitude [m]
40
35
30
25
20
15
10
0
500
1000
1500
100
200
300
N_Rx Return counts/bin
S_Rx Intensity
Water Surface Scattering Polarization
5
0
10
20
30
40
50
60
70
80
N_Rx Return counts/bin

Published with MATLAB® R2020b

Function to Compute the Stokes Vectors for a Dual-Polarized System

```
function [Srxco, Srxcx] = SVLE(alts, DSurf, Dair, NL, dR, surfReflec,
 Trans, betaRL, G, Matrix, lp, wps)
    % Inputs:
        % alts:        array of altitudes the signal is propogating
 through
        % DSurf:       depolarization value of the surface of
 reflection
        % DSurf:       depolarization value of the air
        % NL:          number of transmitted photons
        % dR:          integration range bin
        % surfReflec:  array of amount of surface reflectance at
 altitude*
        %              *0s except for at surface
        % Trans:       amount of transmission at altitudes
        % betaRL:      rayleigh backscatter coefficient at altitudes
        % Matrix:      mueller matrix descriptions
        % lp:          parameters of the lidar system being used
        % wps:         water surface parameters: empty if not water
 surface
        %              [alpha, eta, gamma]


    nBins = length(alts);

    % Transmitter Path                    (Mtx)
    Mtx   = Matrix.Trans(lp.txEff);

    % Receiver path                        (Mrx)
    MrxCo = Matrix.Trans(lp.rxEff)*Matrix.Pol(1,lp.CoOffset);
    MrxCx = Matrix.Trans(lp.rxEff)*Matrix.Pol(1,lp.CxOffset);

    % preallocate
    Srxco = zeros(4, nBins);
    Srxcx = zeros(4, nBins);

    for m = 1:nBins

        % Atmospheric Scattering Matrix: Depolarization/Rayleigh beta
        F = Matrix.Depol(Dair).*betaRL(m);

        % First atmospheric transmission       (Tatm1)
        Tatm1 = Matrix.Trans(Trans(m));

        % Second atmospheric transmission      (Tamt2)
        Tatm2 = Matrix.Trans(Trans(m));

        % Transmitted Vector                   (Stx)
        Stx   = Matrix.Vec(lp.CoOffset);
```

```
        ScaleAtmos = lp.A./((alts(m)-alts(1))^2)*dR*NL*G(m); % can add
 more shots & overlap function here
        ScaleGround = lp.A./((alts(m)-alts(1))^2)*NL*G(m);
%         ScaleAtmos = lp.A./((alts(m)-alts(1))^2)*NL*G(m); % can add
 more shots & overlap function here
%         ScaleGround = lp.A./((alts(m)-alts(1))^2)*NL*G(m);

        if isempty(wps)
            % Surface Depolarizing Lambertian Scattering
            Rg = Matrix.Depol(DSurf).*surfReflec(m);

            Srxco(:,m) = (ScaleAtmos*MrxCo*Tatm2*F*Tatm1*Mtx*Stx)+...
                (ScaleGround*MrxCo*Tatm2*Rg*Tatm1*Mtx*Stx);
        %     Srxco(:,m) =
 MrxCo*(ScaleAtmos*Tatm2*F*Tatm1*Mtx*Stx)+...
        %         MrxCo*(ScaleGround*Tatm2*Rg*Tatm1*Mtx*Stx);
            Srxcx(:,m) = (ScaleAtmos*MrxCx*Tatm2*F*Tatm1*Mtx*Stx)+...
                (ScaleGround*MrxCx*Tatm2*Rg*Tatm1*Mtx*Stx);
        else

            Rw = -Matrix.KatRefl(wps(1),wps(3),wps(3)).* ...
                Matrix.Depol(DSurf).*surfReflec(m);

            Srxco(:,m) = (ScaleAtmos*MrxCo*Tatm2*F*Tatm1*Mtx*Stx)+...
                (ScaleGround*MrxCo*Tatm2*Rw*Tatm1*Mtx*Stx);
        %     Srxco(:,m) =
 MrxCo*(ScaleAtmos*Tatm2*F*Tatm1*Mtx*Stx)+...
        %         MrxCo*(ScaleGround*Tatm2*Rg*Tatm1*Mtx*Stx);
            Srxcx(:,m) = (ScaleAtmos*MrxCx*Tatm2*F*Tatm1*Mtx*Stx)+...
                (ScaleGround*MrxCx*Tatm2*Rw*Tatm1*Mtx*Stx);

        end

    end

end
```

```
Not enough input arguments.

Error in SVLE (line 20)
    nBins = length(alts);
```

Published with MATLAB® R2020b

Table of Contents


```
% Written By:  Robert Stillwell & Rory Barton-Grimley
% Written For: ARSENL
% Updated: 2021-10-15 Kevin Sacca
%
% This functions defines a structure of function handles which are
 used to
% define the canonical Mueller matrices which are used in the SVLE.m
% function to calculate the complete polarization resolved
 measurement.
% Note here that hte definition of depolairzation d is different than
% depolarization delta per Matt Hayman's thesis.

function [Matrix] = FindMuellerMatrices

%
% Inputs: none
%
% Outputs: Matrix: A structure containing function handles defining
 the
%                  necessary Mueller matrix types
%
```

Depolarizer

```
Matrix.Depol = @(d) [1, 0 , 0 ,  0   ;
                     0,1-d, 0 ,  0   ;
                     0, 0 ,d-1,  0   ;
                     0, 0 , 0 ,2*(d-1)];
```

Uniform Depolarizer

```
Matrix.UniformDepol = @(d) [1, 0 , 0 ,  0   ;
                    0, d, 0 ,  0   ;
                    0, 0 , -d,  0   ;
                    0, 0 , 0 , -d];
```

Depolarizer

```
Matrix.NonUniformDepol = @(d1,d2,d3) [1, 0 , 0 ,  0   ;
                                     0,d1, 0 ,  0   ;
                                    0, 0 , -d2,  0   ;
                                    0, 0 , 0 , -d3];
```

Kattwar Reflection

```
Matrix.KatRefl = @(a,e,g) [a + e, a - e, 0,  0   ;
                       a - e, a + e, 0,  0   ;
                       0,       0,   g,  0   ;
                       0,       0,   0,  g];
```

Kattwar Transmission a -> w

```
Matrix.KatTransAW = @(a,e,g,k) k.*[a + e, a - e, 0,  0   ;
                                a - e, a + e, 0,  0   ;
                                0,       0,   g,  0   ;
                                0,       0,   0,  g]  ;
```

Kattwar Transmission w -> a

```
Matrix.KatTransWA = @(a,e,g,k) k.*[a + e, a - e, 0,  0   ;
                                  a - e, a + e, 0,  0   ;
                                    0,       0,   g,  0   ;
                                    0,       0,   0,  g]  ;
```

Half Wave Plate (vertical)

```
Matrix.HalfWave = [1,0, 0, 0;
                   0,1, 0, 0;
                   0,0,-1, 0;
                   0,0, 0,-1];
```

Linear Polarizer

```
Matrix.LinPol = .5*[1,1,0,0;
                    1,1,0,0;
                    0,0,0,0;
                    0,0,0,0];
```

Rotation matrix in radians

```
Matrix.Rot   = @(T) [1,   0    ,    0    ,0;
                     0,cos(2*T),-sin(2*T),0;
                     0,sin(2*T), cos(2*T),0;
                     0,   0    ,    0    ,1];
```

Rotation matrix in degrees

```
Matrix.RotD  = @(T) [1,    0    ,    0     ,0;
                     0,cosd(2*T),-sind(2*T),0;
                     0,sind(2*T), cosd(2*T),0;
                     0,    0    ,    0     ,1];
```

Transmission matrix

```
Matrix.Trans = @(T) [T,0,0,0;
                     0,T,0,0;
                     0,0,T,0;
                     0,0,0,T];
```

Rotating a matrix in radians

```
Matrix.MatRot  = @(M,T) Matrix.Rot(T)*M*Matrix.Rot(-T);
```

Rotating a matrix in degrees

```
Matrix.MatRotD = @(M,T) Matrix.RotD(T)*M*Matrix.RotD(-T);
```

Functional half wave plate

```
Matrix.HWP = @(Trans,Theta)
 Matrix.Trans(Trans)*Matrix.MatRotD(Matrix.HalfWave,Theta);
```

Functional polarizer

```
Matrix.Pol = @(Trans,Theta)
 Matrix.Trans(Trans)*Matrix.MatRotD(Matrix.LinPol,Theta);
```

Transmitted Vector

```
Matrix.Vec   = @(T) Matrix.RotD(T)*[1;1;0;0];
```

Background vector

```
Matrix.Back  = [1;0;0;0];

end
```

```
ans =

  struct with fields:

              Depol: @(d)[1,0,0,0;0,1-d,0,0;0,0,d-1,0;0,0,0,2*(d-1)]
      UniformDepol: @(d)[1,0,0,0;0,d,0,0;0,0,-d,0;0,0,0,-d]
   NonUniformDepol: @(d1,d2,d3)[1,0,0,0;0,d1,0,0;0,0,-d2,0;0,0,0,-d3]
           KatRefl: @(a,e,g)[a+e,a-e,0,0;a-e,a+e,0,0;0,0,g,0;0,0,0,g]
        KatTransAW: @(a,e,g,k)k.*[a+e,a-e,0,0;a-e,a
+e,0,0;0,0,g,0;0,0,0,g]
        KatTransWA: @(a,e,g,k)k.*[a+e,a-e,0,0;a-e,a
+e,0,0;0,0,g,0;0,0,0,g]
          HalfWave: [4×4 double]
            LinPol: [4×4 double]
               Rot: [function_handle]
              RotD: [function_handle]
             Trans: @(T)[T,0,0,0;0,T,0,0;0,0,T,0;0,0,0,T]
            MatRot: @(M,T)Matrix.Rot(T)*M*Matrix.Rot(-T)
           MatRotD: @(M,T)Matrix.RotD(T)*M*Matrix.RotD(-T)
               HWP: [function_handle]
               Pol: [function_handle]
               Vec: @(T)Matrix.RotD(T)*[1;1;0;0]
              Back: [4×1 double]
```

Published with MATLAB® R2020b

```
% function to call all of the _____ lidar parameters
function [lp] = lidar_params

    lp.pE = 5e-6;           % [J] pulse energy
    lp.rr = 19000;          % [Hz] repitition rate
    lp.wl = 532e-9;         % [m] laser wavelength in a vacuum
    lp.D = .05;             % [m] telescope diameter
    lp.A = (pi*(lp.D/2)^2); % [m] telescope area
    lp.BinSize = 0.005;    % [m] range bin size

%     lp.alt = 30;            % [m] altitude of lidar
%     lp.eta = 2.06e-4;       % full system efficiency
%     lp.eta = .05;       % full system efficiency
    lp.rxEff = .01;
    lp.txEff = .025;

    % transmitter
    lp.Lambda                   = 532.00e-9;    % laser wavelength
          [m]
    lp.LaserW0                  = 0.005;        % laser output beam
 radius       [m]
    lp.Divergence               = 10e-3;        % beam divergence
         [rad]

    % receiver
    lp.FNumTele                 = 13.9;         % F number of the
 telescope      [ ]
    lp.FieldStop                = 20e-3;         % Diameter of the
 field stop     [m]
    lp.Separation               = 75e-3;        % laser-telescope
 separation     [m]
    lp.BandPassFilt             = 0.0234;       % pass band of the
 filter used  [nm]
%     lp.TransPol                 = 2.5;          % Rx/Tx offset angle
          [deg]
    lp.FOV                      = (lp.FieldStop)/
(lp.FNumTele*lp.D); %8e-3;
%
    lp.Pointing = .001;
    lp.WTransAngle = asin(sin(lp.Pointing)/1.33);

%     lp.t = .9999;              % lower atm transmission at 589nm
    lp.t = 1;

    lp.CoOffset = 0;        % [deg] Rx/Tx offset angle for co-pol
    lp.CxOffset = 90;       % [deg] Rx/Tx offset angle for cx-pol
end


ans =

  struct with fields:
```

```
          pE: 5.0000e-06
          rr: 19000
          wl: 5.3200e-07
           D: 0.0500
           A: 0.0020
     BinSize: 0.0050
       rxEff: 0.0100
       txEff: 0.0250
      Lambda: 5.3200e-07
     LaserW0: 0.0050
  Divergence: 0.0100
    FNumTele: 13.9000
   FieldStop: 0.0200
  Separation: 0.0750
BandPassFilt: 0.0234
         FOV: 0.0288
    Pointing: 1.0000e-03
 WTransAngle: 7.5188e-04
           t: 1
    CoOffset: 0
    CxOffset: 90
```

Published with MATLAB® R2020b

```
% Written By:  Robert Stillwell
% Written For: ARSENL
% This function calculates the overlap function of the laser beam and
 the
% telescope assuming they are separated by some small distance. It
 assumes
% that the beam and telescope field of view expand as cones.

function [G] = Overlap(Distances, SP)

    % Inputs: Altitudes:  The altitudes of each lidar measurement
    %         SP:         The lidar system's operational parameters

    % Outputs: Overlap:   An array containing expected overlap
 percentage
    %                     given the system configuration of Supr

    Distances = -Distances + Distances(1,1);
```

```
Not enough input arguments.

Error in Overlap (line 15)
    Distances = -Distances + Distances(1,1);
```

Temp variables

```
    phi1   = SP.FOV/2;        % half of the telescope's FOV
    phi0   = SP.Divergence/2; % half of the beam divergence angle
    rsep   = SP.Separation;   % space between telescope and laser
```

Determining overlap proportion

This is from Halldorsson and Langerholc, App. Opt. 17 2 (1978)

```
    G=zeros(length(Distances),1);

    for m=1:length(Distances)
        rfov   = SP.D/2 + phi1*Distances(m);
        rlaser = sqrt(SP.LaserW0^2 + (phi0*Distances(m))^2);

        if rsep >= rfov + rlaser % before any overlap
            Ainter = 0;

        elseif rsep <= abs(rfov-rlaser) % complete overlap with laser
 < fov
            Ainter = (pi*rlaser^2);

        elseif rsep <= rfov + rlaser && rsep >= abs(rfov-
rlaser) %middle ground
            Ainter = rfov^2*acos((rsep^2+rfov^2-rlaser^2)/
(2*rsep*rfov)) + ...
```

```
                rlaser^2*acos((rsep^2+rlaser^2-rfov^2)/
(2*rsep*rlaser)) - ...
                1/2*sqrt(((rfov+rlaser)^2-rsep^2)*(rsep^2-(rfov-
rlaser)^2));
        end

        G(m)=Ainter/(pi*rlaser^2);
    end

end
```

Published with MATLAB® R2020b