# UAV 搭載デジタルカメラ画像を用いた
# 樹木の二方向性反射特性
# Bidirectional reflectance distribution factor(BRDF) of trees using UAV-mounted digital camera images

高知工科大学大学院
工学研究科基盤工学専攻
社会システム工学コース
国土情報処理工学研究室
1265059 若吉慧門
指導教員 高木方隆
副指導教員 佐藤愼司
論文副審査 赤塚慎

2024 年 1 月 17 日

# 論文要旨

衛星リモートセンシングでは，衛星に搭載したさまざまな観測周波数のセンサから表層や地表からの反射光の観測を行っている．多時期の観測データを使用して解析を行うことで植生の季節変化と温暖化の影響について評価することが可能であり，時には将来予想にも活用される．しかし，衛星データの観測値には，観測位置と光源位置 (太陽位置) の二方向性による地物の反射特性が大きく影響している．二方向性反射特性 (Bidirectional reflectance distribution factor: 以下「BRDF」と呼ぶ) とは，ある特定の角度で入射する光源方向とカメラの観測方向により異なる対象物体の反射量の変化の様子を表したものである．多時期の観測データを使用して解析を行う際，観測位置と光源位置が常に同じであることは多くない．光源位置の問題として，太陽高度は年間変動があるため，反射量が見かけ上変化してしまう．衛星解析の現状として，観測地表面を等方散乱体と仮定して解析を行うことは多く，BRDF の影響が考慮されないまま解析が行われている．観測地表面は，環境や物体の状態によって近似的に等方散乱を起こすことはあるが，地表面に完全な等方散乱体は存在しない．特に植生を対象とした場合，植生の分布密度や種類，形状によって散乱の性質が変化するため，近似的に等方散乱とはならず，非等方散乱体として扱わなければならない．したがって BRDF は，観測地表面の物体やその状態，環境の違いによって衛星データの観測値に複雑な影響をもたらしている．

植生リモートセンシングにおいて，BRDF は対象物体の三次元構造との関わりが深いことが示唆されており，植生の物理量と関わりが大きく期待されていることから植生の BRDF を捉えることを目的とした BRDF の実測的研究が行われている．これまでの実測的研究では，衛星やヘリ，航空機に搭載した分光放射計を用いた計測だけであり，樹木の BRDF に影響を与える要素を詳細に追求した研究はない．BRDF に影響を与える要素を明らかにするために分光放射計に比べて高い地上分解能での観測が必要である．そこで近年の UAV 技術の発展から，比較的安価な UAV 搭載デジタルカメラを使用した BRDF 観測手法に着目した．本研究は UAV に搭載したデジタルカメラを用いた BRDF 観測の高精度化に向けた画像校正手法を構築し，樹木の BRDF に影響を与えている要素を明らかにすることを目的とした．

本研究でバッドピクセルと周辺減光に対する校正手法が構築できた．このことからフルサイズセンサカメラの Raw 画像を用いた高精度な BRDF 解析が可能となった．校正後の画像を用いた BRDF 解析結果は，対象樹木の BRDF は，多くが後方散乱性 (BRDF の傾きが負) を示した．負の傾きを示す要因は主に「葉の色の変化」と「影割合の変化」であった．太陽の順光側で観測した時に，葉の反射は大きく，影割合が小さい傾向にあり，結果的に BRDF は負の傾きを示した．しかし，一部の BRDF が正の示すものもあった．その要因は，樹木の順光側で葉群密度が低く，逆光側で葉群密度が高いことによる影割合の減少から BRDF が正の傾きを示す傾向が見られた．BRDF に影響を与える他の大きな要素として，「葉身の大きさ」「解析対象面の向き」が考えられるが，この二つの要素の影響については，傾向を明らかにすることができなかった．これらのことから，BRDF は葉の色の変化と影割合の変化の両方の影響を大きく受けることから葉の特徴と樹木の物理的特徴を捉えられていることが言える．

BRDF には，本研究で対象とした要素に加えて「葉の向き」や「葉の表面材質」「葉の健康状態 (湿潤状態)」が複合的に影響し合っていることが考えられる．BRDF に影響を与える要素を明らかにするためには，少なくとも葉身よりも細かな地上分解能での高解像度な BRDF 観測が必要である．さらに今後，植生の物理的指標として BRDF を用いるためには，光源位置の影響を明らかにするために多時期での考察が必要である．

## Abstract

Satellite remote sensing involves the observation of reflected light from the surface layer and the earth's surface with various observation frequencies mounted on satellites. Using multi-temporal observation data for analysis, it is possible to evaluate seasonal changes in vegetation and the effects of global warming, and sometimes it is also used for future forecasting. However, satellite data is greatly affected by the reflectance characteristics of surface objects due to the angle nature of the observation position and the light source position (the position of the sun). Bidirectional reflectance distribution factor (BRDF) is defined as BRDF (Bidirectional Reflectance Distribution Factor) is a measure of how the reflectance of an object varies depending on the angle of the incident light source and the direction of the camera. When analyzing multi-temporal observation data, the observation position and the light source position are not always the same. The problem with the source position is that the solar altitude fluctuates throughout the year, causing apparent changes in reflectance. In the current situation of satellite analysis, the surface of the observation site is often assumed to be an isotropic scatterer, and analysis is performed without considering the effect of BRDF. Although the surface of the observed land can be approximately isotropically scattered depending on the environment and the state of the objects, there is no perfect isotropic scatterer on the ground surface. In particular, when vegetation is the target, the nature of the scattering changes depending on the distribution density, and shape of the vegetation, so the scattering is not approximately isotropic and must be treated as an anisotropic scatterer. Therefore, the influence of BRDF has a complex effect on the satellite data due to differences in the objects on the surface of the observation site, their conditions, and the environment.

In the case of vegetation remote sensing, it has been suggested that BRDF is closely related to the three-dimensional structure of the target object. Therefore, the BRDF has been studied to capture the BRDF of vegetation. Previous studies have only used spectroradiometers on board satellites, helicopters, and airplanes, and have not investigated in detail the factors that affect the BRDF of each tree. In order to clarify the influence on the BRDF, observations with a higher ground resolution than spectroradiometers are needed. Therefore, we focused on the BRDF observation method using a relatively inexpensive UAV-mounted digital camera based on the recent development of UAV technology. The objectives of this study are to develop an image calibration method to improve the accuracy of BRDF observation using a UAV-mounted digital camera, and to clarify the factors that affect the BRDF of trees.

A calibration method for bad pixels and vignetting has been established, enabling highly accurate analysis using full-lens raw images. The BRDF analysis results showed that the BRDF of the target trees were mostly backscattered (negative BRDF slope). The main reasons for the negative slope were "leaf color change" and "shadow ratio change. When observed in the sunlit side, the leaf reflection tended to be larger and the shadow fraction smaller, resulting in a negative slope of the BRDF. However, some BRDFs were positive. The reason for this is that the trees tended to have lower leaf density on the forward-lit side and higher leaf density on the back-lit side, which reduced the shadow fraction, resulting in a positive BRDF slope. The other major factors affecting the BRDF are the size of the leaf blade and the orientation of the plane under analysis. We were not able to determine trends in the effects of these two factors. From these results, it can be said that the BRDF is greatly affected by both leaf color change and shadow ratio change, thus capturing both the leaf characteristics and the physical characteristics of the tree.

In addition to the factors studied in this study, leaf orientation, leaf surface material, and leaf health (wetness) may have a combined effect on the BRDF. High-resolution BRDF observations with ground resolution at least finer than leaf blade resolution are needed to clarify the factors that affect BRDF. Furthermore, in order to use BRDF as a physical indicator of vegetation in the future, it is necessary to consider multiple time periods in order to clarify

the influence of light source location.

# 目次

# 図目次

# 表目次

# 1　序論

## 1.1　二方向性反射特性 (BRDF) とは

衛星リモートセンシングでは，衛星に搭載したさまざまな観測周波数のセンサから表層や地表からの反射光の観測を行っている．多時期の観測データを使用して解析を行うことで植生の季節変化と温暖化の影響について評価することが可能であり，時には将来予想にも活用される．しかし，衛星データの観測値には，観測位置と光源位置 (太陽位置) の二方向性による地物の反射特性が大きく影響している．二方向性反射特性 (Bidirectional reflectance distribution factor: 以下「BRDF」と呼ぶ) とは，ある特定の角度で入射する光源方向とカメラの観測方向により異なる対象物体の反射量の変化の様子を表したものである．

多時期の観測データを使用して解析を行う際，観測位置と光源位置が常に同じであることは多くない．光源位置の問題として，太陽高度は年間変動があるため，反射量が見かけ上変化してしまう．観測位置の問題としては，近年の観測衛星ではの直下視撮影だけでなく，前視撮影や後視撮影が可能な衛星が増えている．これは対象をより多く撮影できるメリットがある反面，観測位置の変化の影響による反射量が見かけ上変化してしまう問題が生じる．衛星解析の現状として，観測地表面を等方散乱体と仮定して解析を行うことは多く，BRDF の影響が考慮されないまま解析が行われている．観測地表面は，環境や物体の状態によって近似的に等方散乱を起こすことはあるが地表面に完全な等方散乱体は存在しない．特に植生を対象とした場合，植生の分布密度や種類，形状によって散乱の性質が変化するため，近似的に等方散乱とはならず，非等方散乱体 (図 1.1) として扱わなければならない．したがって BRDF による影響は，観測地表面の物体やその状態，環境の違いによって衛星データの観測値に複雑な影響をもたらしている．

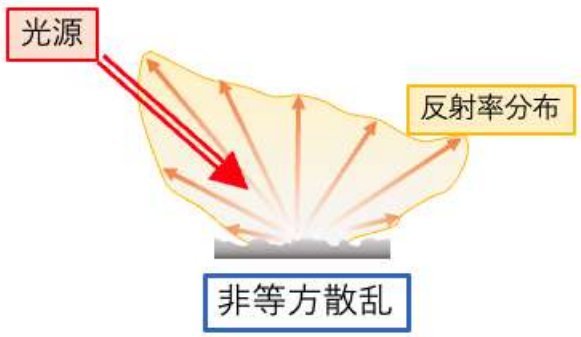



**図 1.1 非等方散乱体のイメージ**

図解説: 非等方散乱体とは，ある対象物体の特徴に影響して観測位置によって反射量が変化する物体.

植生の BRDF は衛星の誤差要因としてだけではなく, 植生の物理的指標の一つとして期待されている．例に同一の樹木 (スギ) を UAV で上空からほぼ同時刻に撮影し位相角 $\theta$ の違う画像を図 1.2 に示す．位相角 $\theta$ とは，太陽方向と観測方向の位相角のことである．図 1.2 の①は太陽の入射方向からスギを順光側で撮影し，④になるにつれて逆光側で撮影していることを表している．例に示すスギの場合，順光側で撮影した時の方が逆光側で撮影した時より反射が大きいことが確認できる．このように地表面に存在する多くの物体は非等方散乱体であるため，観測方向や光源方向が変化することで見え方が変化する．この特徴は物体によって異なるため，植生リモートセンシングでは，植生の物理的指標の一つとして期待されている．

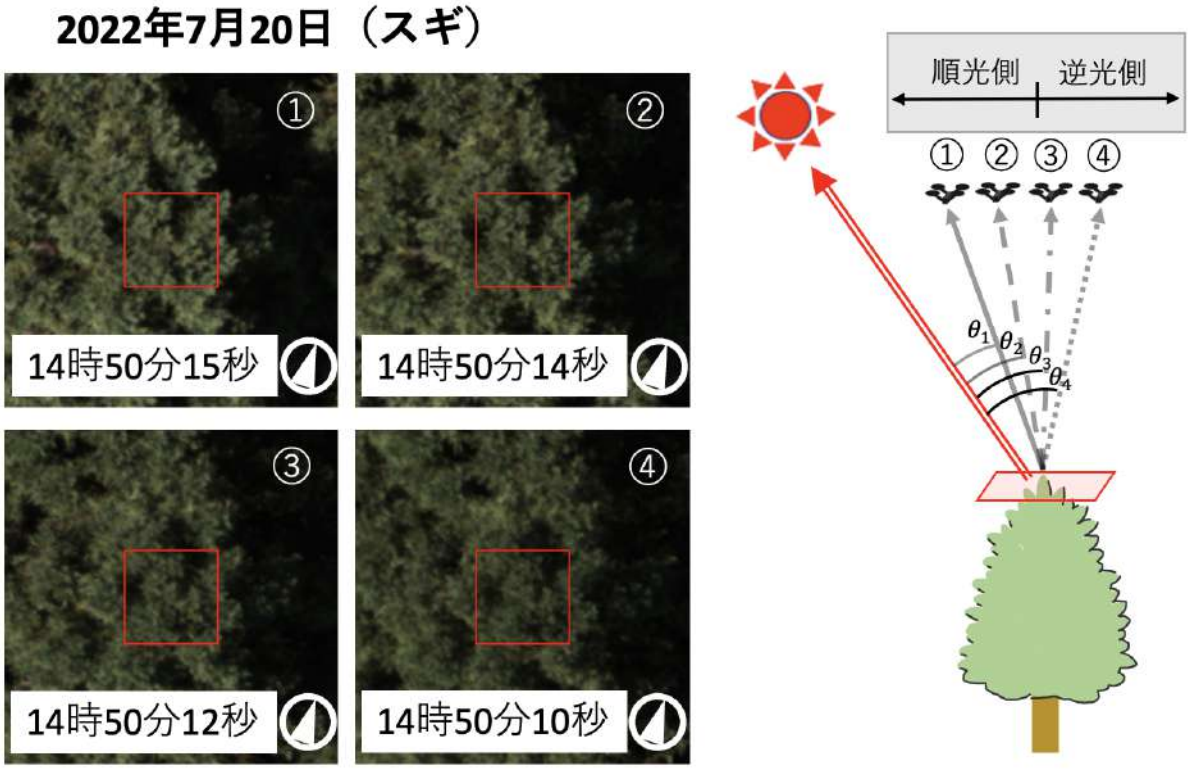



**図 1.2 観測方向の違いによる対象物の反射量の違い**

## 1.2 BRDF 観測の現状

植生リモートセンシングにおいて，BRDF は対象物体の三次元構造との関わりが深いことが示唆されており，植生の物理量と関わりが大きく期待されていることから植生の BRDF を捉えることを目的とした BRDF の実測的研究が行われている．BRDF 観測は太陽位置が変化しないよう短時間での観測が必要とされる．これまでの BRDF 観測では短時間での観測を実現するために，ヘリコプター [1] や観測タワー [2], ゴニオメーター [3] などの大規模な観測システムのもとで行われており，主に森林を対象としたものか地衣植物や低木植物といった下層植物の二つに大別される．対象が二つに大別される理由は，計測器として分光放射計が使用されていることが挙げられる．分光放射計を使用した場合，視野角内の情報を高波長分解能で取得できる反面，地上分解能は視野角に対して一つとなるため非常に低い．そのため，広範囲に広がる森林を対象とするか近距離での観測が容易な下層植物が BRDF 観測の対象となっており，樹木を単木単位で観測した例は少ない．これらの観測は，コストが非常に高く，観測データ蓄積のための高頻度な観測は現実的でない．さらに樹木の BRDF 観測は衛星やヘリコプター，航空機に搭載した分光放射計を用いた計測だけであり，樹木の BRDF に影響を与える要素を詳細に追求した研究はない．そこで近年の UAV 技術の発展から，比較的安価な UAV 搭載デジタルカメラを使用した BRDF 観測手法に着目した．

## 1.3 里山研究フィールドにおける BRDF 観測状況

本研究室では，計測器にデジタルカメラを使用した BRDF 観測を行っている．これまでにデジタルカメラによる BRDF 観測は行われておらず，本研究室の村井 [4] により提案されたデジタルカメラによる BRDF 観測手法を用いて観測を行っている．村井による研究では，APS-C センサのデジタルカメラを使用していた．APS-C センサのデジタルカメラは比較的レンズ歪みが小さいため，その影響については考慮されていなかった．また，JPG 形式による解析のため量子化 bit は 8bit と低く，1 バンドで色は 256 段階までしか記録できない．

本研究では，高解像度かつ高精度な解析を行うためフルサイズセンサカメラの Raw14bit を採用した．フルサイズセンサを採用することで APS-C センサより高解像度の撮影が可能であり，Raw14bit を採用することで 1 バンドで 16384 段階で色を記録することができる．しかし，フルサイズセンサの Raw 画像を扱う上でレンズ歪みは大きく，新たに解決しないといけない課題が生じる．

## 1.4 研究の目的と意義

そこで本研究は UAV に搭載したフルサイズセンサカメラの Raw 画像 (デジタルカメラ) を用いた BRDF 観測の高精度化に向けた画像校正手法を構築し，樹木の BRDF に影響を与えている要素を明らかにすることを目的とした．今回，里山研究フィールドを対象に落葉樹と常緑樹が共に着葉している夏場の 2022 年 7 月 20 日の観測データを使用して樹木ごとの BRDF の把握を行った．

本解析は，フルサイズセンサの Raw 画像を使用していることから特有の現象が生じる．このデジタルカメラ特有の現象は撮影画像に大きく影響を与えるため校正することが必要不可欠である．その一つに Raw 画像を扱う上でバッドピクセルの問題がある．バッドピクセルとは，常に明るい点として記録されるホットピクセルや光が当たっているのに黒い点として記録されるデッドピクセルのようなレンズに入る光の反射を無視して起こるピクセルエラーのことである．植生観測において，デジタルカメラ画像に記録される色値を正確に読み取るためにはこのバッドピクセルを無視することはできない．さらにフルサイズセンサを扱う上で周辺減光と呼ばれる現象が生じる．周辺減光とは画像の四隅が暗くなる現象のことである．これはレンズサイズや撮影設定によって度合いが変化するものであり，画像の見た目にも大きく影響する．特にフルサイズセンサを使用する場合，周辺減光の影響が大きい．しかしバッドピクセルや周辺減光の影響を考慮した画像解析は行われていない．これらのことから，本研究の目的と意義を以下のように設定した．

研究の目的

- フルサイズセンサの Raw 画像に対する校正手法を構築する..
- 単木単位かつ多樹木の BRDF 解析を行う．
- BRDF に影響を与えている要素を明らかにする．

研究の意義

- デジタルカメラを使用することで BRDF 観測を広範囲かつ高精度に扱うことができる．
- 様々な対象物の BRDF を明らかにできる．

# 2　観測対象フィールドと対象樹木

## 2.1　観測対象フィールド

本研究室では，高知県香美市土佐山田町佐岡地区に位置する里山研究フィールドを対象地域として 2021 年 7 月から月 1 回の頻度で BRDF 観測を行っている．観測条件として晴天であることが求められるため，天候に恵まれず欠測している月もある．図 2.1 に里山研究フィールドの地図，図 2.2 に里山研究フィールドのオルソ画像を示した．解析範囲は里山研究フィールドの 300m×300m とした．

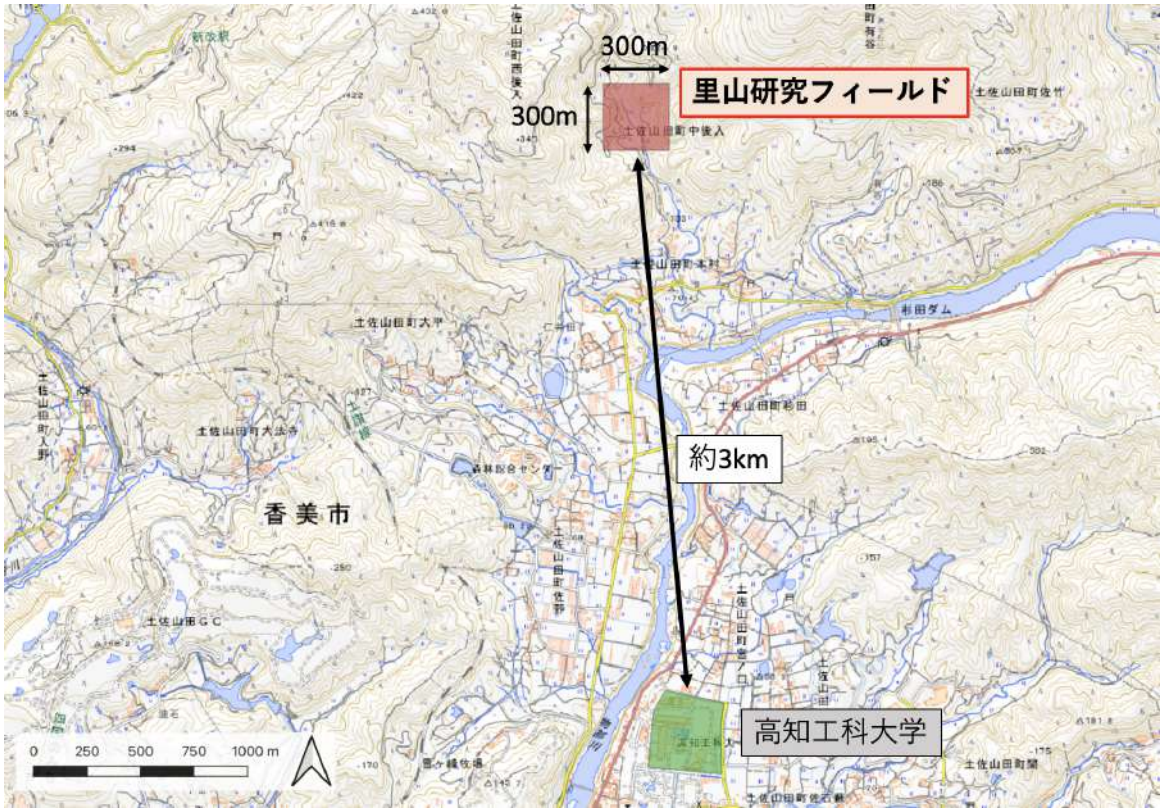



図 2.1 里山研究フィールドの地図

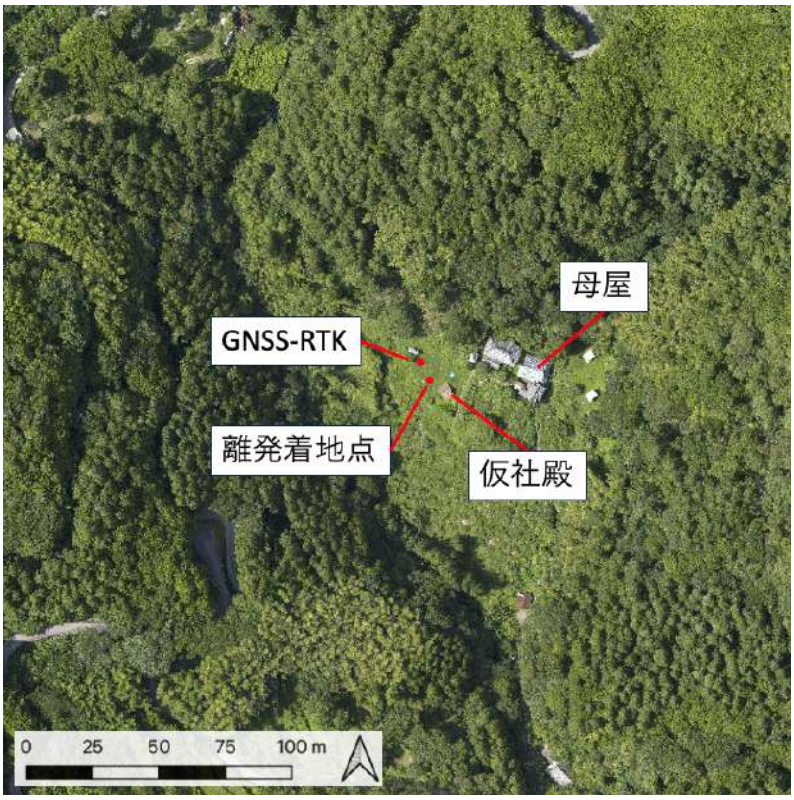



**図 2.2 里山研究フィールド (2022/07/20) のオルソ画像**

## 2.2 対象樹木

BRDF の対象とする樹木は，里山研究フィールド内の 49 本を対象とした．里山研究フィールド内の樹種については，2020 年 5 月に単木調査が行われたデータ [5] が shape ファイル形式のポイントデータで公開されているためこのデータをもとに表 2.1 に示す樹木を選択した．図 2.3 に対象樹木の位置関係を示す．

**表 2.1 対象樹木**

| 樹種 | 樹名 | 本数 | 樹種 | 樹名 | 本数 |
|---|---|---|---|---|---|
| 落葉広葉樹 | アキニレ | 2 | 常緑広葉樹 | アラカシ | 7 |
| | イロハカエデ | 3 | | バクチノキ | 1 |
| | カキ | 3 | | ビワ | 1 |
| | ケヤキ | 2 | | ヒサカキ | 1 |
| | コナラ | 1 | | クロガネモチ | 1 |
| | クリ | 3 | | ミカン | 2 |
| | クワ | 1 | | ツバキ | 1 |
| | ムクノキ | 1 | | ヤマモモ | 2 |
| | ヤマザクラ | 3 | | ユズ | 3 |
| 落葉針葉樹 | イチョウ | 1 | 常緑針葉樹 | スギ | 9 |
| 常緑竹 | ホウライチク | 1 | | ヒノキ | 9 |

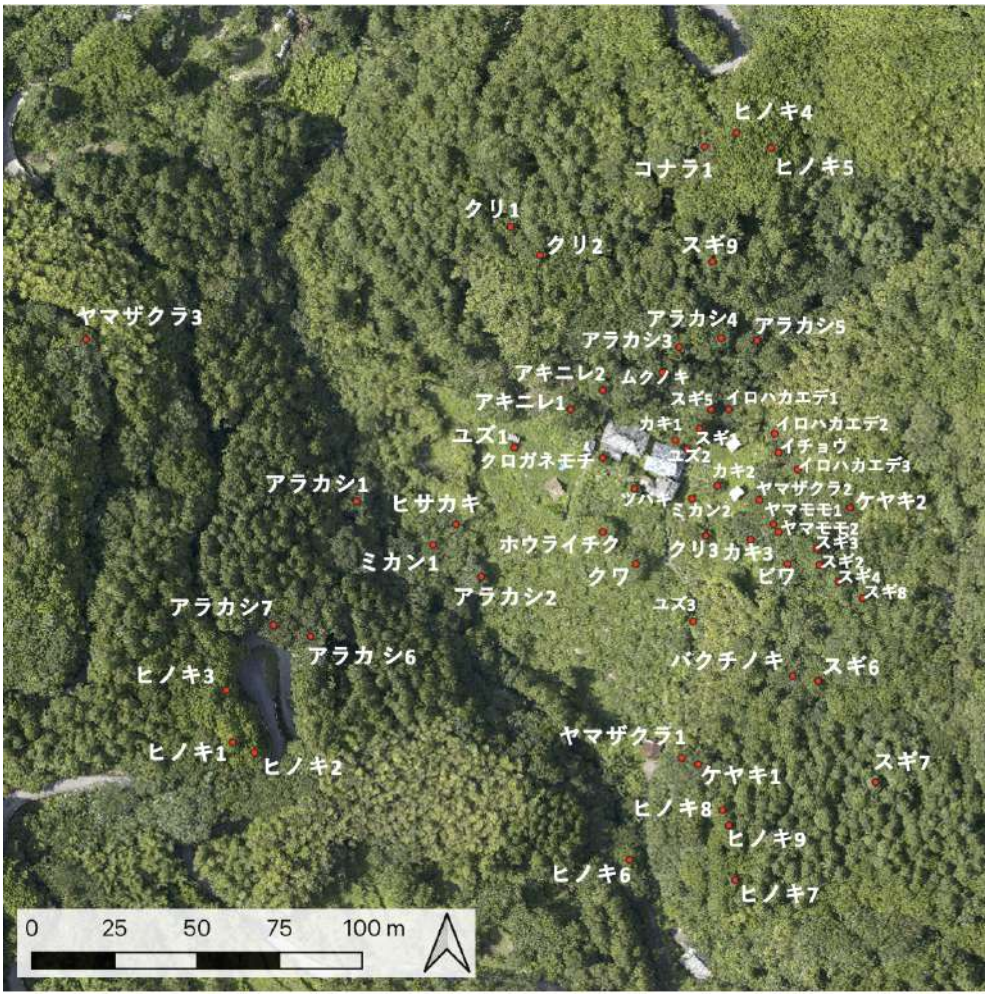

ヒノキ4
コナラ1
ヒノキ5
クリ1
クリ2
スギ9
ヤマザクラ3
アラカシ4
アラカシ5
アラカシ3
アキニレ2
ムクノキ
スギ5
イロハカエデ1
アキニレ1
カキ1
イロハカエデ2
スギ
ユズ1
イチョウ
ユズ2
クロガネモチ
カキ2
イロハカエデ3
アラカシ1
ヤマザクラ2
ヒサカキ
ツバキ
ミカン2
ヤマモモ1
ケヤキ2
ヤマモモ2
スギ3
ホウライチク
クリ3
カキ3
ミカン1
スギ2
クワ
ビワ
スギ4
アラカシ2
スギ8
ユズ3
アラカシ7
アラカシ6
バクチノキ
スギ6
ヒノキ3
ヤマザクラ1
ヒノキ1
ヒノキ2
スギ7
ケヤキ1
ヒノキ8
ヒノキ9
ヒノキ6
ヒノキ7
0
25
50
75
100 m

図 2.3 対象樹木

# 3　観測機材とその校正

## 3.1　観測機材

BRDF 観測を行うために，UAV(無人航空機) を用いて里山研究フィールドの直下視撮影を行った．本研究で使用した UAV は DJI 社の「MATRICE 300 RTK」，デジタルカメラは DJI 社の「Zenmuse P1」である．UAV とデジタルカメラの外観を図 3.1，図 3.2 に示す．

図 3.1 MATRICE 300 RTK

図 3.2 Zenmuse P1

対象地域内に，RTK-GNSS 測位のための RTK-GNSS 受信機を設置した．RTK-GNSS とは，基準点と観測点 (UAV の位置座標) の 2 点を同時に観測する測位方法のことである．2 点を同時に測位することで衛星と受信機の時計誤差を消去することができ，高精度な位置座標を取得することが可能となる．対象地域内のあらかじめ座標が分かっている基準点に RTK-GNSS 受信機を設置し，UAV の位置座標を高精度に取得する．本研究で使用した RTK-GNSS 受信機は DJI 社の「D-RTK 2」である．RTK-GNSS 受信機の外観を図 3.3 に示す．本研究で使用した機材の仕様を表 3.1 に示す．

図 3.3 D-RTK 2

表 3.1 使用機器の仕様

| メーカー | DJI |
|---|---|
| UAV 機体 | MATRICE 300 RTK |
| 飛行型 | 回転翼 |
| 重量 | 6.3kg(バッテリー 2 個搭載時) |
| 最長飛行時間 | 55 分 |
| 最高飛行速度 | 17m/s |
| 天候制限 | 12m/s 以下 |
| 搭載カメラ | Zenmuse P1 |
| センサ | CMOS |
| レンズ | 35mm |
| 量子化 bit 数 | 14bit |
| 有効画素 | 4,500 万ピクセル |
| 静止画像解像度 | 3:2, 8192×5460 |
| RTK-GNSS 受信機 | D-RTK 2 |
| 測位精度 (水平方向) | 1cm |
| 測位精度 (水平方向) | 2cm |

本研究における BRDF 観測はデジタルカメラを使用して対象範囲を面的に観測できることが最大の特徴である．そのため観測方法は，これまでは対象物体に対して半円状に観測していたのに対してデジタルカメラを使用した場合は対象範囲をなぞるような面的な観測方法となる．図 3.4 に 2022 年 7 月 20 日に撮影した UAV の飛行経路を示す．UAV の進行方向は太陽方位と重なるように設定しており，オーバーラップ率 90%, サイドラップ率を 80% に保ちながら U ターンを繰り返し撮影を行った．U ターンを区切りとして UAV の経路それぞれを pass1〜pass12 とする．撮影日時の太陽高度は 52 度，太陽方位は 260 度．飛行設定，カメラ設定を表 3.2 に示す．

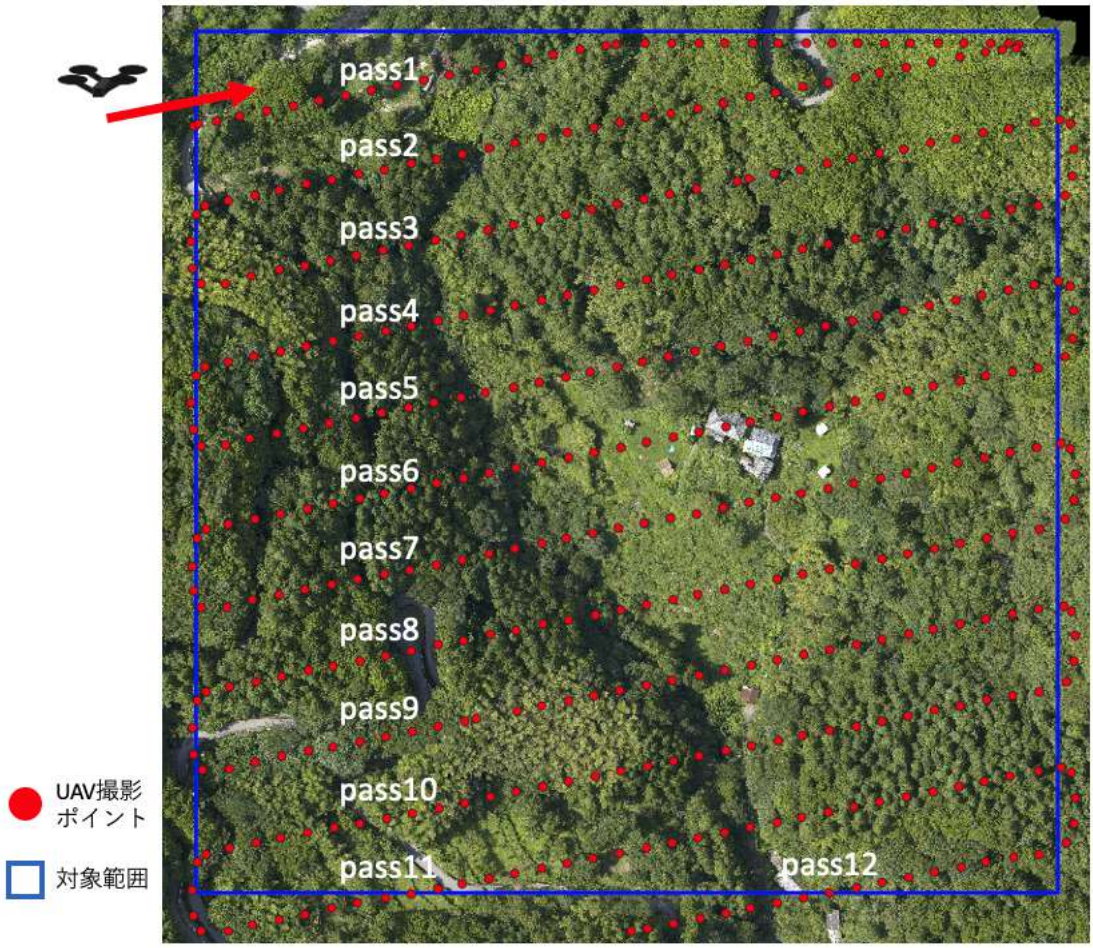



図 3.4 UAV の飛行経路

表 3.2 飛行設定とカメラ設定

| | |
|---|---|
| 観測日 | 2022 年 7 月 20 日 |
| 機体 | MATRICE 300 RTK |
| 搭載カメラ | DJI Zenmuse P1 |
| レンズ絞り | 5.6 |
| ISO | 200 |
| シャッタースピード | 1/350 秒 |
| 画像ファイル形式 | RAW |
| 撮影開始時間 | 14 時 46 分 37 秒 |
| 撮影終了時間 | 14 時 59 分 25 秒 |
| 撮影時間 | 12 分 48 秒 |
| オーバーラップ率 | 90% |
| サイドラップ率 | 80% |
| 飛行高度 | 離発着地点から 135m |
| 地上分解能 | 1.05cm/pixel |
| 撮影枚数 | 422 枚 |

## 3.2 観測データの Raw 現像

本研究の BRDF 観測では，デジタルカメラで撮影したデータを Raw ファイル形式で保存している．Raw ファイル形式とは，イメージセンサが記録した未加工の 2 次元配列データのことである．そのため Raw ファイル形式のデータを現像処理し，画像に変換する必要がある．Raw 現像には, プログラミング言語 python ライブラリの「rawpy.postprocess」関数を使用して行った．表 3.3 に「postprocess」関数の入力値 (Raw 現像パラメータ) を示す．rawpy による Raw 現像のための各パラメータの意味や設定方法については，キャンプ工学 [6] の「rawpy で Python 現像」を参考にした．Raw 現像パラメータの設定は，Raw データで記録された値にできるだけ補正が入らない設定とした．Raw 現像のプログラムは，付録 9.2[B] に記載する．

表 3.3 rawpy によるデベイヤ設定

| パラメータ | パラメータ名 | 設定値 |
|---|---|---|
| 色深度 | output_bps | 16 |
| 色空間 | outputcolor | rawpy.ColorSpace.raw |
| スケール調整の有無 | no_auto_scal | False |
| 自動露出の有無 | no_auto_bright | True |
| ホワイトバランス | use_camera_wb | True |
| | use_auto_wb | False |
| | user_wb | [2.6640625,1.0,1.8046875,0.0] |
| デモザイク方法 | demosaic_algorithm | rawpy.DemosaicAlgorithm.LINEAR |
| ガンマ補正 | gamma | [1.00,0.00] |

本研究では，Raw 現像時の量子化 bit(色深度) は 16bit に設定した．色深度とは，ピクセルが持つ色の情報のことであり，16bit のとき一つのピクセルは 0〜65535 のいずれかの数値を記録することができる．この記録している数値のことを DN 値と呼ぶ．本研究で使用している P1 カメラは，14bit であるため設定値が 16bit になっているということは 0〜16383(14bit) の値が 0〜65535(16bit) に引き伸ばされているということであるため DN 値で記録される値は 0〜65535 で表現されるが実質的に表現される色の段階は 0〜16383 である．

DN 値が最小値の 0 の時は，最も暗い状態を表現し，ピクセルは黒く表現される．DN 値が最大値の 65535 の時は，最も明るい状態を表現し，ピクセルは白く表現される．

## 3.3 デジタルカメラ画像のバッドピクセルの特定と除去

Raw 画像を扱う上でバッドピクセルの問題が生じる．バッドピクセル (Bad Pixel) とは，常に明るい点として記録されるホットピクセルやクールピクセル，光が当たっているのに黒い点として記録されるデッドピクセルのようなレンズに入る光の反射を無視して起こるピクセルエラーのことである．デジタルカメラ画像に記録される色値を正確に読み取るためにはこのバッドピクセルは異常値になるため除去しなければならない．

バッドピクセルの特定方法について説明する．まず，バッドピクセルの特定には複数枚のダーク画像が必要となる．ダーク画像とは，全く光を当てない状態で撮影してカメラ固有のノイズだけを記録した画像のことである．本研究におけるダーク画像の撮影はカメラのキャップを装着したまま撮影したものとし，植生観測の前後にダーク画像を 10〜30 枚程度撮影している．python ライブラリの「rawpy.enhance.find_bad_pixels」関数を使用し，複数枚のダーク画像を関数に与えるとバッドピクセル (ホットピクセルとデッドピクセル) の画像座標を返される．関数によるバッドピクセルの特定はダーク画像が多いほど精度が良くなる．与えるダーク画像が少ないほど本来バッドピクセルではないものが誤判定によりバッドピクセルと認識される場合がある．誤判定によるバッドピクセルの過剰除去を防ぐため，2022 年 6 月〜2023 年 5 月の 1 年間の植生観測 13 回分に撮影したダーク画像合計 297 枚を使用してバッドピクセルの特定を行った．各観測の複数枚のダーク画像を関数に与え，バッドピクセルの画像座標を特定した．各観測のバッドピクセルの特定プロセスイメージを図 3.5 に示す．P1 カメラにおける 13 回の観測のバッドピクセルの発現箇所と発現確率を図 3.6 に示す．本研究の画像解析には，図 3.6 の赤色で示した出現確率が 80% より大きいものを無視して解析を行うものとした．さらにバッドピクセルはその周辺にも影響を及ぼしている可能性を考慮し，1 つのバッドピクセルに対しそれを取り囲む 8 ピクセルも除去対象とした．その結果, バッドピクセル判定が 80% より大きいものは 6pix 存在し，本研究における画像解析から除去するものは 6pix×9pix≡54pix とした．

バッドピクセル特定のプログラムは，付録 9.1[A] に記載する．

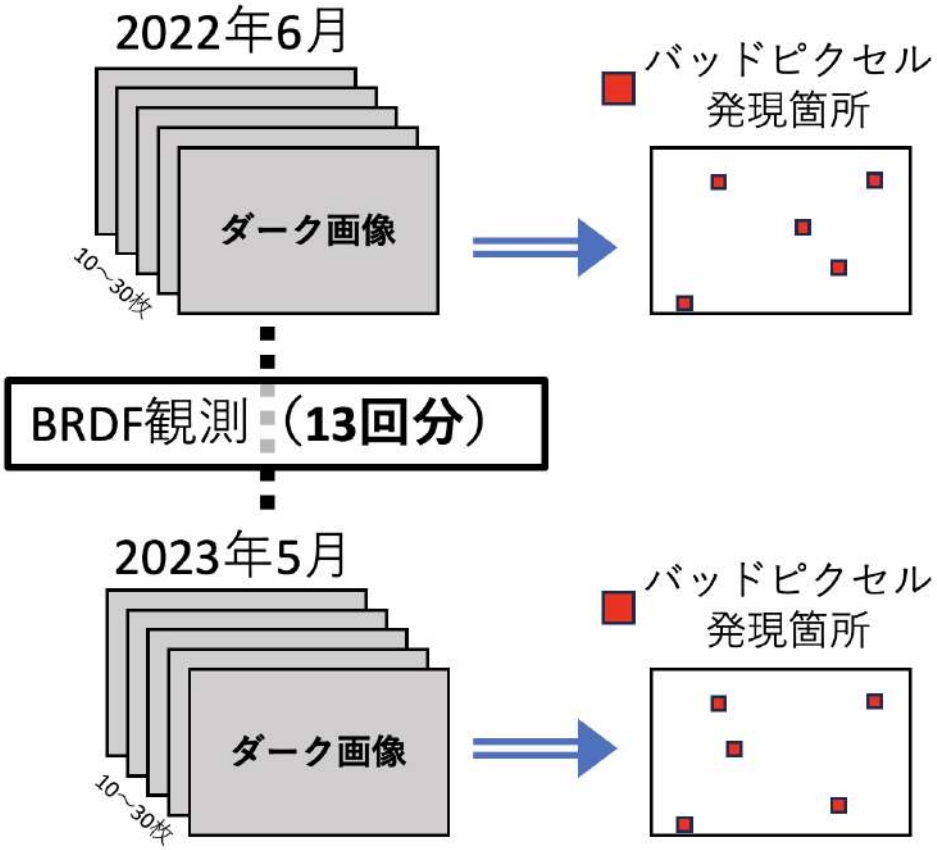



図 3.5 各観測のバッドピクセルの特定プロセスイメージ

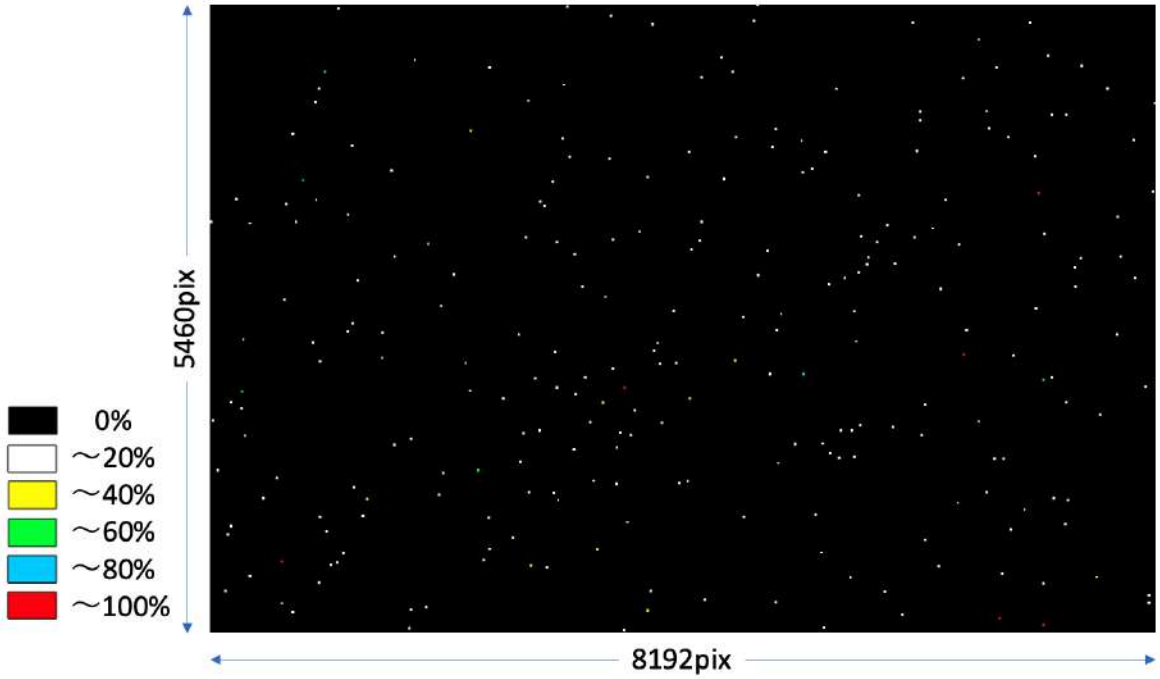



図 3.6 P1 カメラ画像におけるバッドピクセルの発現箇所と発現確率

## 3.4　デジタルカメラ画像の周辺減光補正

デジタルカメラ特有の現象として周辺減光と呼ばれるものがある．周辺減光とは画像の四隅が暗くなる現象のことである．これはレンズサイズや撮影設定によって度合いが変化するものであり，画像の見た目にも大きく影響する場合がある．特にフルサイズセンサを使用する場合は，周辺減光の影響が大きい．図 3.7 に周辺減光が生じている画像を示す．この画像は，均一な色を対象に撮影したものであるが画像の四隅が暗くなっていることが確認でき，この現象が周辺減光である．. デジタルカメラを使用した色の評価を行う際に周辺減光による影響は大きく，この現象を校正することは必要不可欠である．

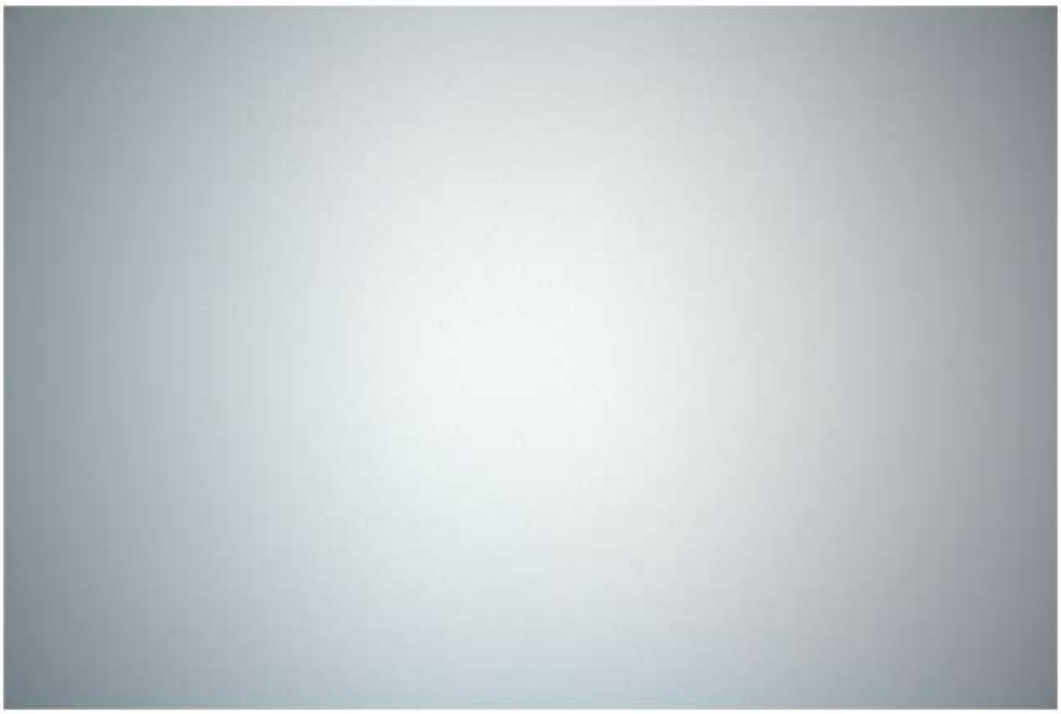


**図 3.7 周辺減光が生じている画像**

周辺減光補正方法について説明する．まず, 周辺減光の度合いを確認する必要がある．そのためには, 完全に均一な色を撮影し確認する必要がある．そこで本研究では，完全に均一な色を表現するものとして積分球を使用した．TRIOPTICS 社に協力を得て，積分球を使用した撮影を行った．使用した積分球は,Image Engineering 社の「LE7 VIS-IR」である．この積分球は，様々なスペクトルを生成でき，97% 以上の均一性を再現することが可能である．積分球の外観を図 3.8 に示す．図 3.8 では，発光部分にダイナミックレンジ計測用のフィルターを挿入しているが均一な面の撮影には拡散フィルター (ディフューザー) を用いた．

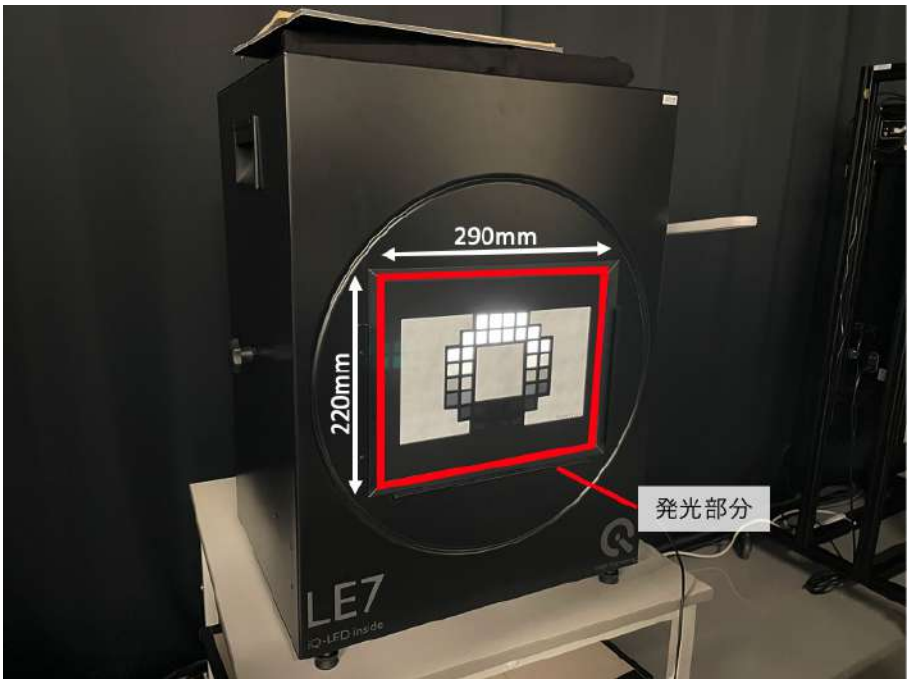



図 3.8 LE7 VIS-IR

撮影設定によって周辺減光の度合いは変化してしまうため，絞りは観測時と同じ設定にした．シャッタースピードに関しては，周辺減光に与える影響を限りなく低いと考え積分球を黒潰れや白飛びをしない標準露出設定で撮影を行った．撮影設定と撮影風景を表 3.4，図 3.9 に示す．P1 カメラで積分球を撮影した画像と周辺減光の度合いを図 3.10 に示す．周辺減光の度合いは，画像中の DN 値の最大値からの差を表している．

表 3.4 カメラ設定

| | |
|---|---|
| 撮影日 | 2022 年 11 月 17 日 |
| 撮影枚数 | 26 枚 |
| カメラ | DJI Zenmuse P1 |
| レンズ絞り | 5.6 |
| ISO | 200 |
| シャッタースピード | 1/640 秒 |
| 画像ファイル形式 | RAW |

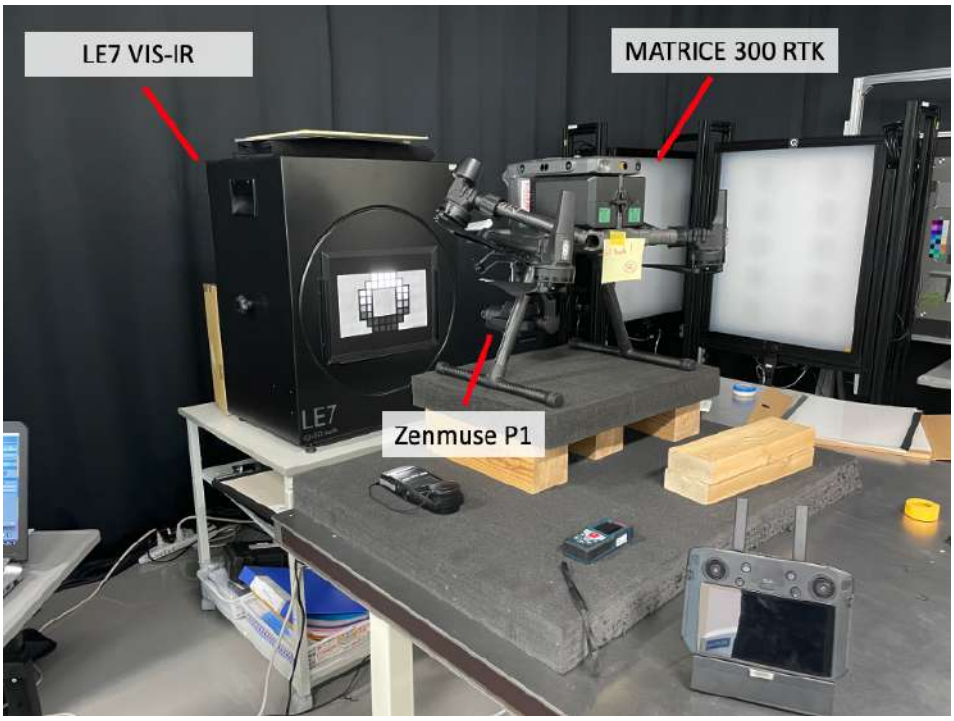



図 3.9 撮影風景

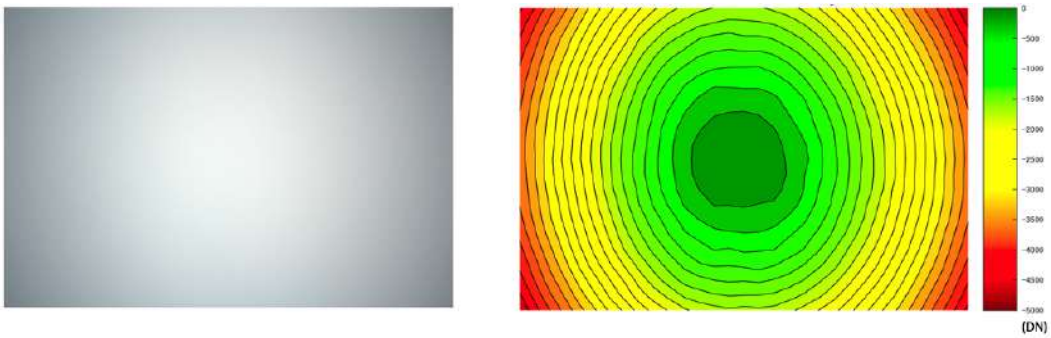



図 3.10 P1 カメラで積分球を撮影した画像 (左) とその周辺減光の度合い (右)

周辺減光補正には，積分球を複数枚撮影した画像の平均画像をフラット画像として用いる．式 (3.1) は，周辺減光補正係数 $\alpha$ の算出式である．フラット画像にフラット画像の最大値 (画像中心付近) の値を除することで $\alpha$ を算出する．式 (3.2) は，周辺減光補正後画像 Q の算出式である．Q は，原画像 P に $\alpha$ を除することで算出する．例に図 3.11 を P として，算出した $\alpha$ から周辺減光補正を施した結果を図 3.12 に示す．

周辺減光補正係数作成のプログラムは，付録 9.1[A] に記載する．

バッドピクセルと周辺減光に対する校正手法が構築できたことでフルサイズセンサの Raw 画像を用いた高精度な解析が可能となった．

$$\alpha = F \div F_{max} \tag{3.1}$$

$$Q = P \div \alpha \tag{3.2}$$

| | |
|---|---|
| $P$ | 補正をかける原画像 |
| $F$ | フラット画像 |
| $F_{max}$ | フラット画像の最大値 |
| $\alpha$ | 周辺減光補正係数 |
| $Q$ | 周辺減光補正後の画像 |

**図 3.11 周辺減光補正前の画像**

**図 3.12 周辺減光補正後の画像**

補正前に比べて補正後画像の四隅の色が明るくなっていることができる．観測で撮影した原画像には，周辺減光補正を施して解析処理を行った．

## 3.5　3 次元点群データの作成

空中写真測量技術の SfM を用いて 3 次元点群データの作成を行った．SfM(Structure from Motion) とは空中から地上の写真を複数枚撮影し，それをコンピュータ処理することで，計測対象物の 3 次元点群データを復元する技術である．点群はその名の通り，点の群れで構成されており，1 つ 1 つの点は直交座標 (XYZ) と色 (RGB) の情報を持つ．2 次元の写真から 3 次元の点群を作成するためには，人間の目が立体感を認識できる原理と同じ「視差」を利用している．そのため，写真測量には少しずつ位置の違う複数枚の写真が必要となる．本研究では，対象範囲を撮影した 422 枚の画像から 3 次元点群を作成した．3 次元点群作成のための SfM 処理には，Agisoft 社のソフトウェア「metashape」を使用した．作成された点群を図 3.13 に示す．


**図 3.13 SfM で作成した里山研究フィールドの 3 次元点群モデル**

3 次元点群データの幾何補正には，あらかじめ測量を行なって地上座標がわかっている基準点を使用した．あらかじめ基準点は GNSS 測量により計測した．基準点位置と基準点の地上座標を図 3.14，表 3.5 に示す．基準点を用いた幾何補正の結果による各基準点の幾何補正誤差を表 3.6 に示す．次節で説明する 3 次元点群データを使用して作成する DSM の地上分解能は 20cm であり，幾何補正誤差は大きいところでも 5cm 以内と非常に低いため点群の精度は高いと言える．このことから本研究における解析では，位置精度の問題はないこととする．

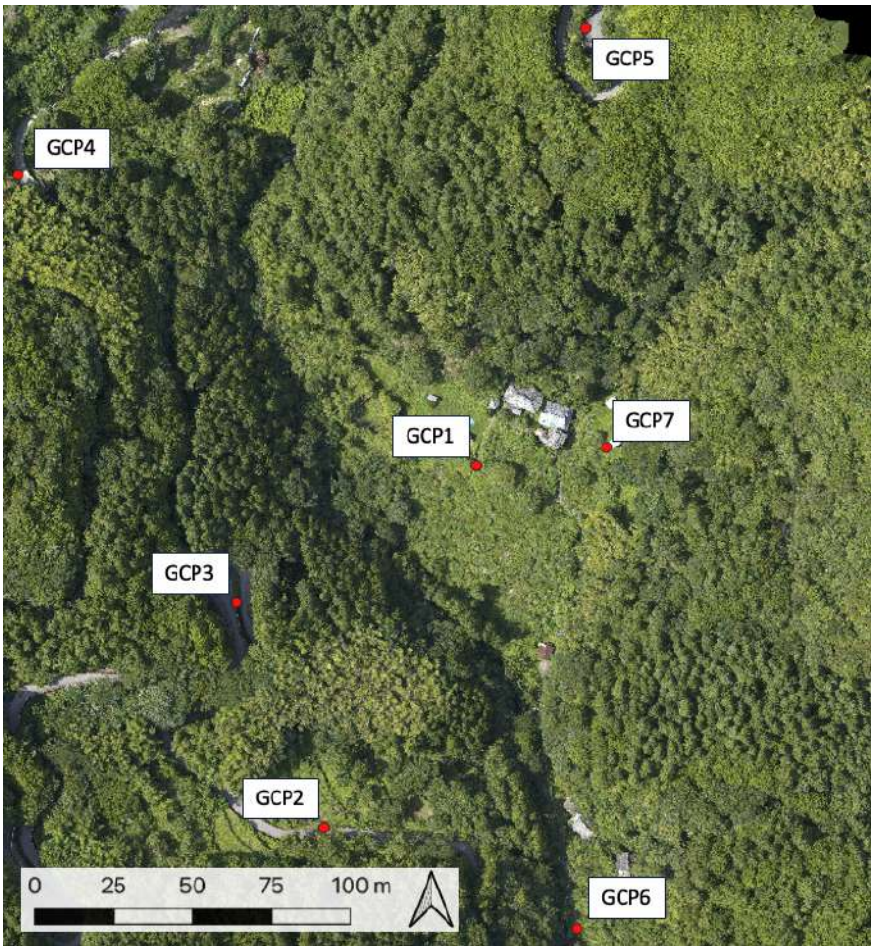



図 3.14 基準点位置

表 3.5 基準点地上座標

| 測点 | X(m) | Y(m) | Z(m) |
|---|---|---|---|
| GCP1 | 20222.277 | 71692.981 | 159.8887 |
| GCP2 | 20270.979 | 71699.849 | 168.3477 |
| GCP3 | 20260.091 | 71520.774 | 142.5047 |
| GCP4 | 20262.891 | 71855.743 | 231.2277 |
| GCP5 | 20051.349 | 71801.150 | 191.7367 |
| GCP6 | 20132.778 | 71641.974 | 180.5897 |
| GCP7 | 20165.272 | 71558.223 | 167.3047 |

表 3.6 幾何補正誤差

| 測点 | X(mm) | Y(mm) | Z(mm) |
|---|---|---|---|
| GCP1 | 3.135 | 5.973 | 44.850 |
| GCP2 | 10.242 | -10.792 | -33.313 |
| GCP3 | 9.615 | -1.359 | -1.489 |
| GCP4 | -5.781 | -3.466 | -0.505 |
| GCP5 | -0.805 | -1.938 | -0.137 |
| GCP6 | -3.544 | 15.310 | -15.352 |
| GCP7 | -6.593 | -3.728 | 5.945 |

## 3.6　DSM の作成

300m×300m の対象範囲を 20cm×20cm でメッシュ化し，3 次元点群データからグリッド内の最高点を抽出する方法で地上分解能 20cm の DSM を作成した．DSM とは，数値表層モデル (Digital Surface Model) と呼び，樹木高さと建物高さを含む地物表面の標高からなる 3 次元データのことである．DSM のプログラムは，付録 9.3[C] に記載する．図 3.15 に対象範囲における地上分解能 20cm の DSM を示す．

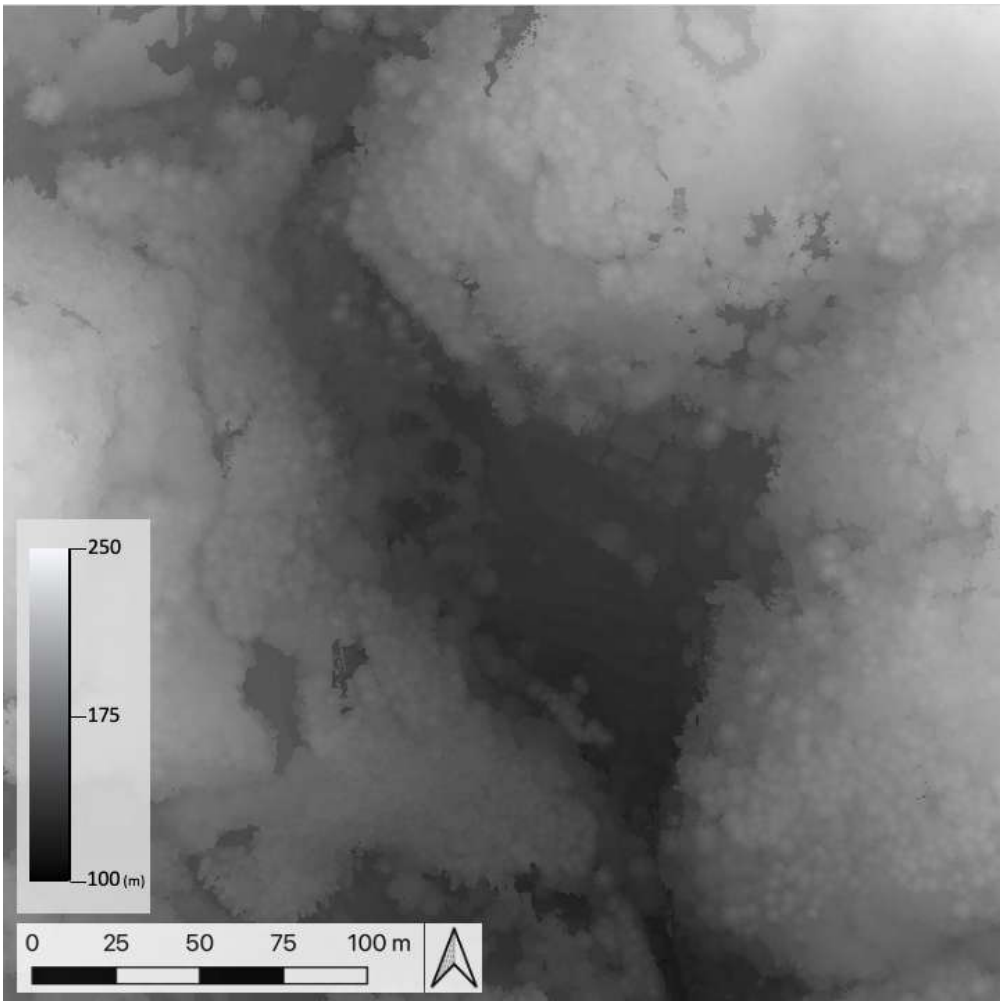



図 3.15 DSM

# 4　BRDF 解析手法

本章では，BRDF 解析手法について説明する．図 4.1 に BRDF 観測から BRDF の算出・可視化までの流れを示す．前章でカメラ校正と SfM 処理の手順ついて記した．BRDF 解析では，「周辺減光補正後の原画像」と「DSM」を使用して，「オルソフォト変換」と「オクルージョン処理」，「対象樹木に対するカメラ方向ベクトルと太陽方向ベクトルの位相角 $\theta$」を算出した．BRDF の算出と可視化には，「オルソフォト」と「位相角 $\theta$」を用いた．各項目の詳細な手順と手法は次節より説明する．

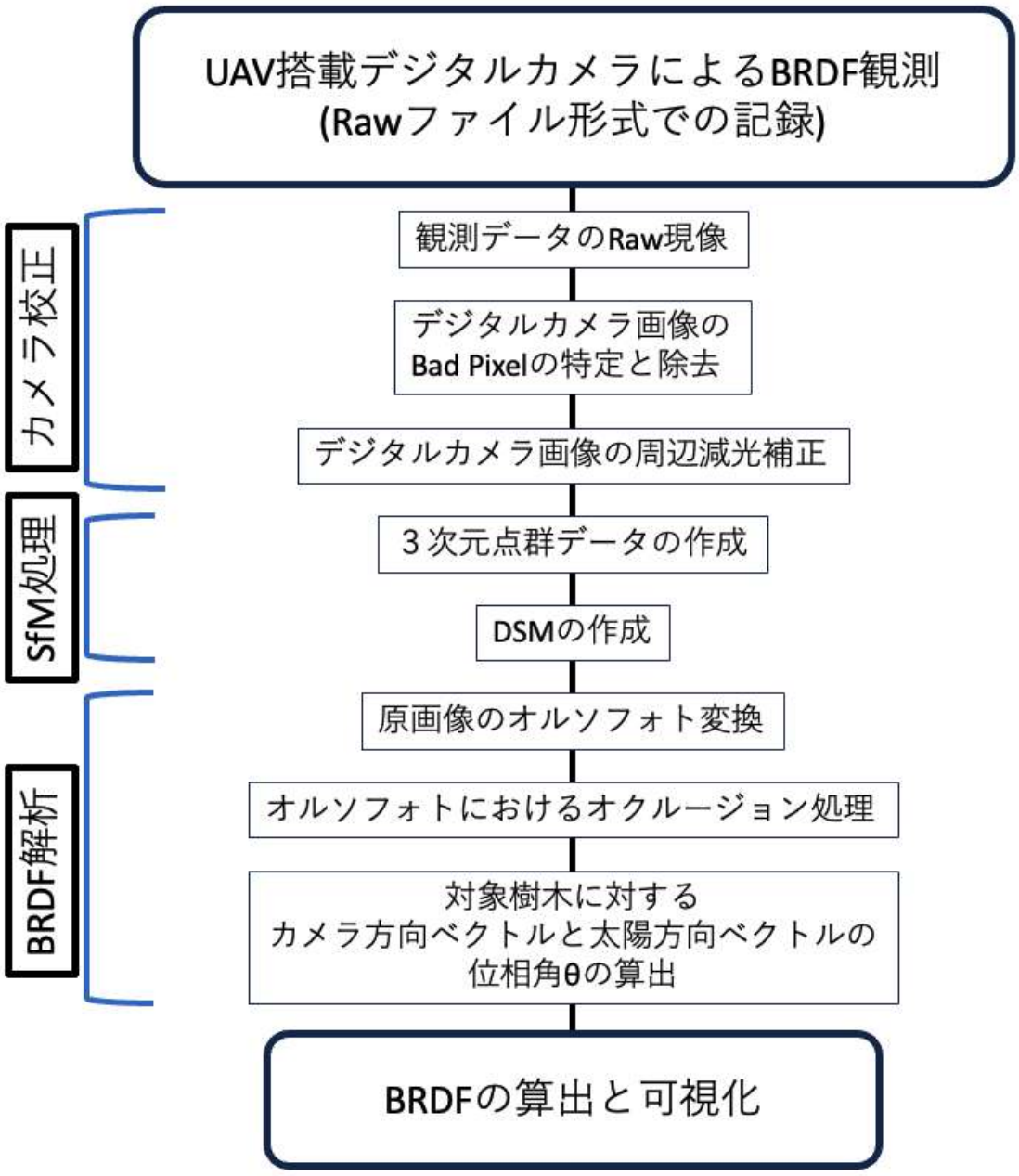



図 4.1 BRDF 解析フロー

## 4.1　原画像のオルソフォト変換

オルソフォト変換とは，中心投影された空中写真の原画像を地図と同じ真上から見たような画像に正射変換したものである．オルソフォトの変換アルゴリズムを図 4.2 に示す．

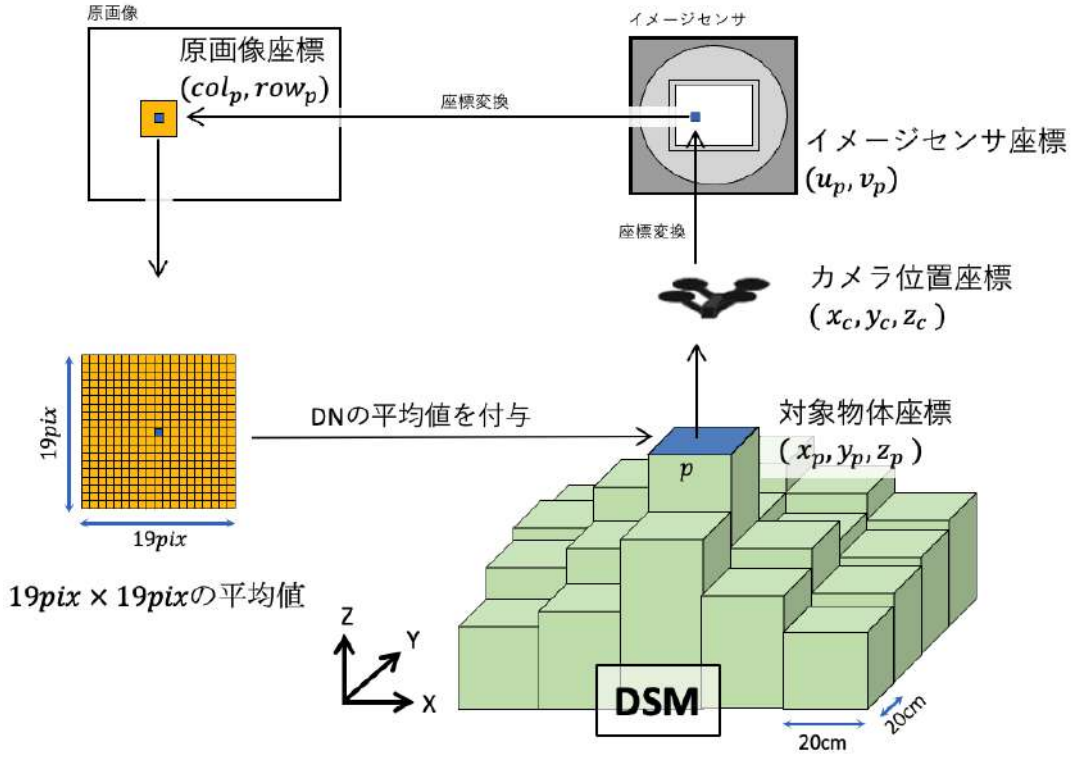



**図 4.2 オルソフォトの変換アルゴリズム**

DSM には地上座標 (対象物体座標) が付与されており，DSM 上の対象物体座標 ($x_p$,$y_p$,$z_p$) とカメラ位置座標 ($x_c$,$y_c$,$z_c$) からイメージセンサ座標 ($u_p$,$v_p$)，原画像座標へと変換を繰り返し，DSM 上のある地点 ($x_p$,$y_p$,$z_p$) が原画像のどこ ($col_p$,$row_p$) に投影されているのかを算出することで原画像の DN 値を付与している．原画像はおよそ 1.05cm/1pix の地上分解能で撮影されているため，撮影された原画像座標 (col,row) を中心に，20cm に相当する 19pix×19pix の DN 値の平均値を付与した．オルソフォト変換のプログラムは，付録 9.3[C] に記載する．図 4.3 に図 3.12 をオルソフォト変換した結果を示す．

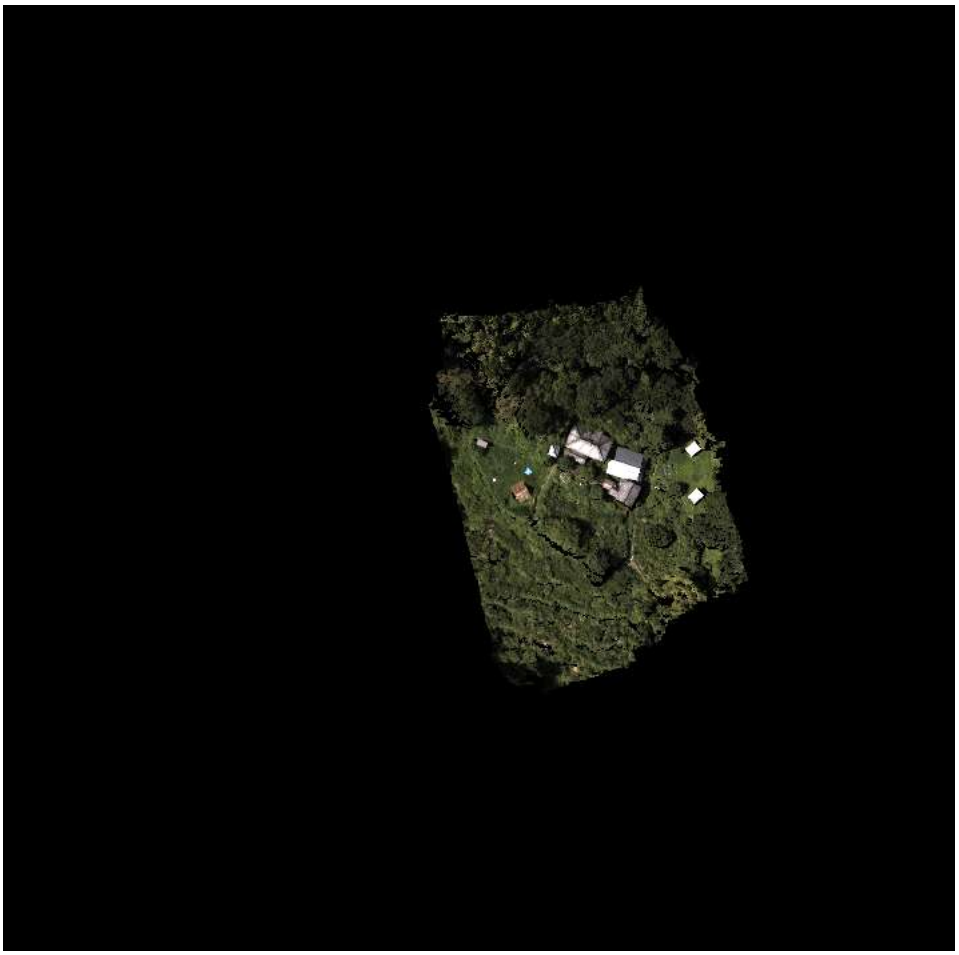


図 4.3 オルソフォト

## 4.2　オルソフォトにおけるオクルージョン処理

第 4.1 節で示したオルソフォトには観測位置からは障害物によって本来遮られて見えるはずのない不可視エリアに不自然な箇所が存在する．オルソフォトの生成過程で誤った DN 値を付与していることが原因で生じている．例に図 4.3 の母屋を拡大した画像を図 4.4 に示す．図 4.4 に写っている家屋の屋根が不自然に伸びている．これらは本来，見えるはずのない不可視エリアである．このような，手前にある障害物に隠れて後ろの物体が隠れている状態をオクルージョンという．オクルージョンを考慮しなければ誤った DN 値を含まれたまま解析することになってしまうため変換したオルソフォトにオクルージョン処理 (可視・不可視判定) を施す必要がある．


**図 4.4 オルソフォトにおけるオクルージョン**

オクルージョン処理のイメージを図 4.5 に示す．点 p からカメラに向かって伸びる直線が途中に障害物に当たると不可視判定，途中で当たることなく UAV まで到達すれば可視判定とした．図 4.5 では,p1,p2,p3 が可視判定，p4,p5 が不可視判定となる．オクルージョン処理のプログラムは，付録 9.3[C] に記載する．このとき不可視判定エリアは，DN 値を 0 とした．オクルージョン処理結果を図 4.6 に示す．

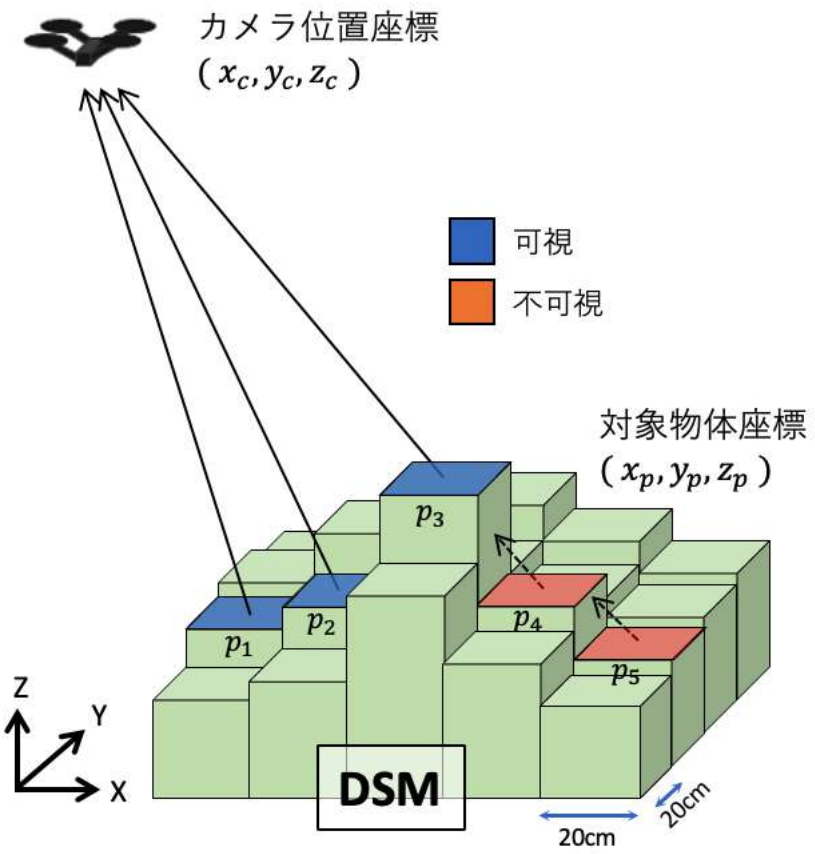



図 4.5 オクルージョン処理イメージ

図 4.6 オクルージョン処理後のオルソフォト

## 4.3　対象樹木に対するカメラ方向ベクトルと太陽方向ベクトルの位相角 $\theta$ の算出

次に対象樹木に対するカメラ方向ベクトルと太陽方向ベクトルの成す角である位相角 $\theta$ を求める．カメラ方向ベクトルは，SfM 処理を行った時に取得できるカメラ位置姿勢情報と対象樹木の位置情報 $(x_p,y_p,z_p)$ から算出した．太陽方向ベクトルは，python ライブラリの「PyEphem」を使用して対象樹木の位置から見た太陽高度と太陽方位を取得し，対象樹木の位置情報 $(x_p,y_p,z_p)$ からの太陽方向ベクトルを算出した．算出した二つの方向ベクトルから成す角 $\theta$ を求める．$\theta$ のイメージを図 4.7 に示す．カメラ方向ベクトルと太陽方向ベクトルの算出プログラムは，付録 9.4[D]，位相角 $\theta$ の算出プログラムは，付録 9.5[E] に記載する．

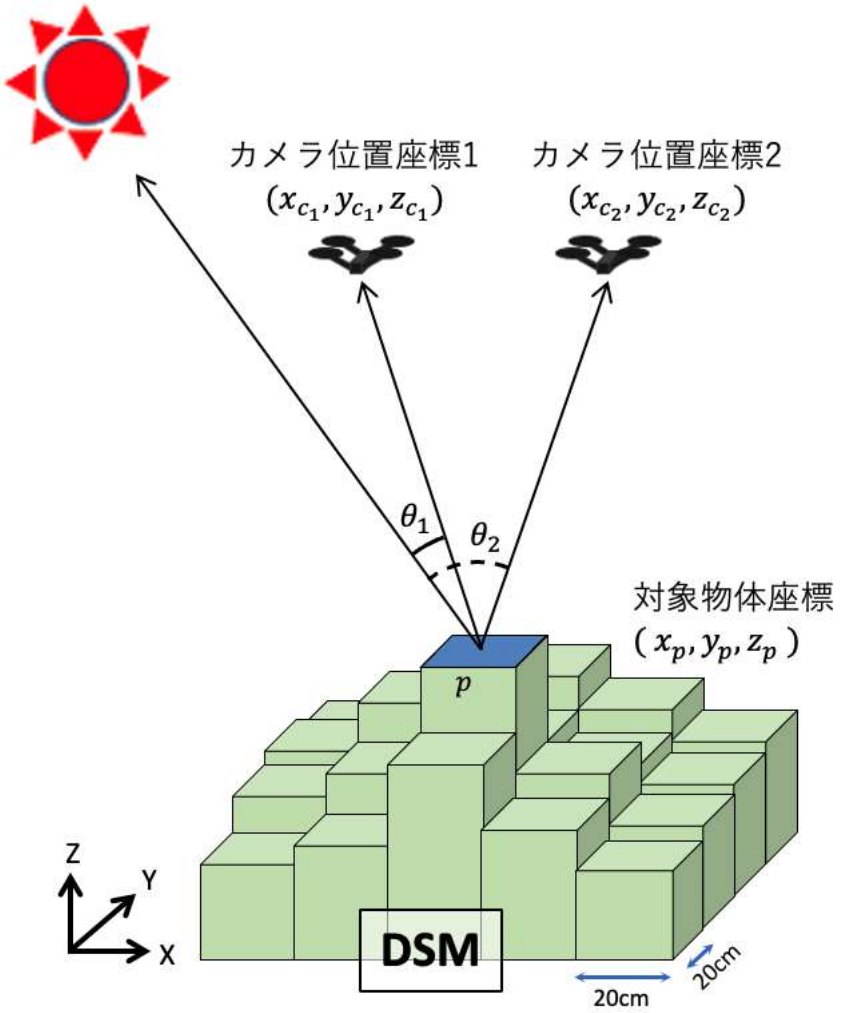



図 4.7 $\theta$ のイメージ

## 4.4 BRDF の可視化

対象樹木は，3 つまたは 4 つの pass で撮影された画像に写っている．例に図 4.8 にヒノキとスギの撮影ポイントを示す．黄丸で示した位置がヒノキとスギがそれぞれ撮影された UAV のポイントとなる．ヒノキは 21 枚，スギは 25 枚撮影されている．

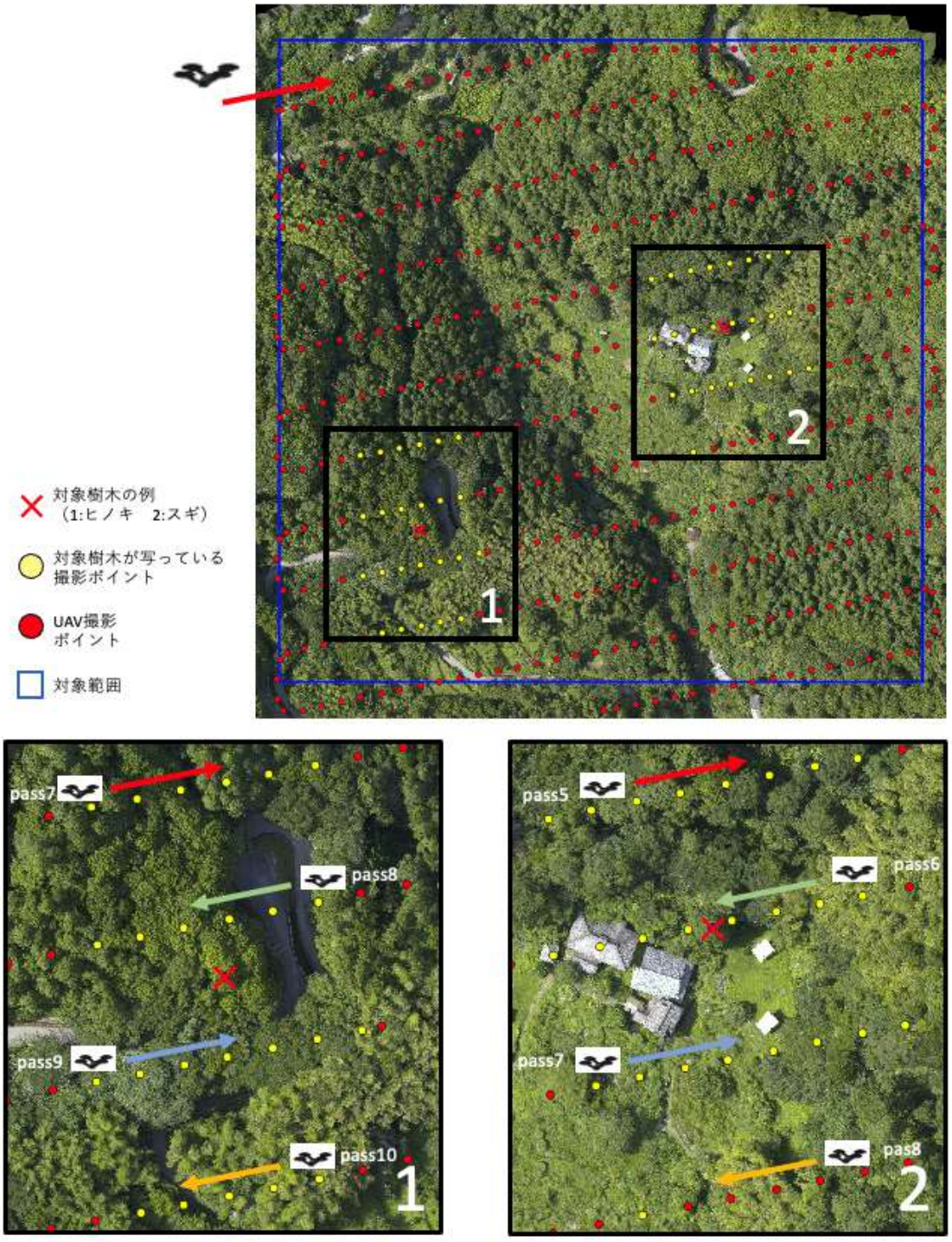



図 4.8 対象樹木と撮影ポイントの例 (1: ヒノキ,2: スギ)

対象樹木が写った各原画像のオルソフォトと $\theta$ を用いて BRDF を可視化した．結果を図 4.9 に示す．グラフの横軸は $\theta$，縦軸はオルソフォトの緑バンドの DN 平均値 (0.6m × 0.6m の範囲の平均値) である．縦軸の DN 平均値は，16bit の 0〜65535 で表現されるが本研究で対象とする樹木の DN 値の平均は 15000 以内に収まることから縦軸は DN 値の 0〜15000 を 0〜1 に正規化して表現している．グラフ中の赤の縦線は，太陽の天頂角を表しており，赤線よりも左側は太陽の入射方向に対して順光側，右側は逆光側となっている．BRDF の可視化プログラムは，付録 9.5[E] に記載する．

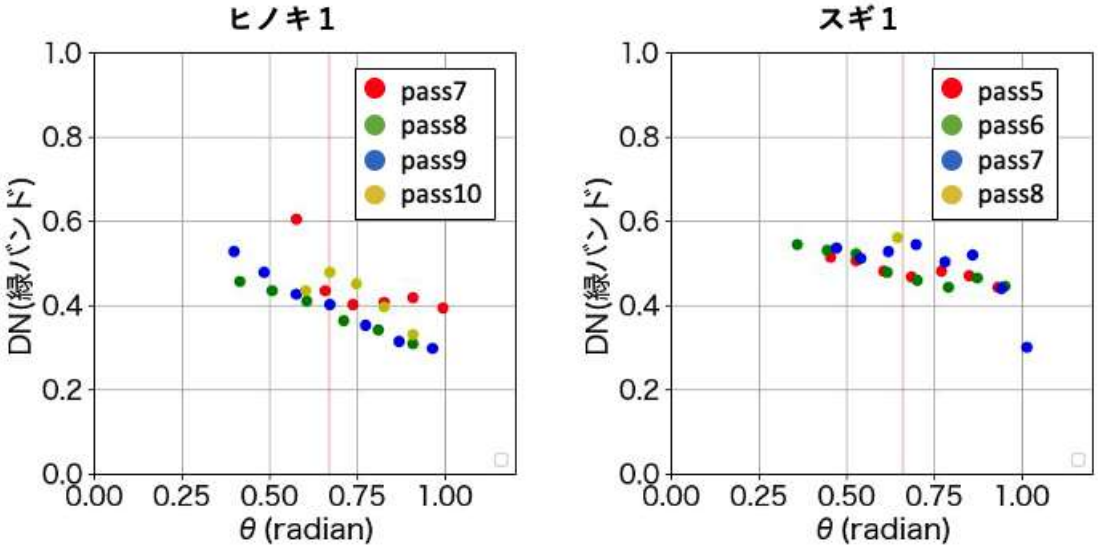



図 4.9 可視化した BRDF の例

ヒノキとスギの BRDF を可視化した結果から負の傾きが見て取れる．しかし，度合いは違うもののばらつきが生じている．このばらつきの原因は，樹木の principal plane と UAV の軌道のズレにあるものと考えられる．樹木の principale plane とは，図 4.10 に示すように樹木と樹木天頂，太陽が同一平面上に存在する主平面のことである．そこで，本研究では対象樹木に対する複数の pass から樹木の principale plane と UAV の軌道が最も近い pass のみで BRDF の可視化を行う．

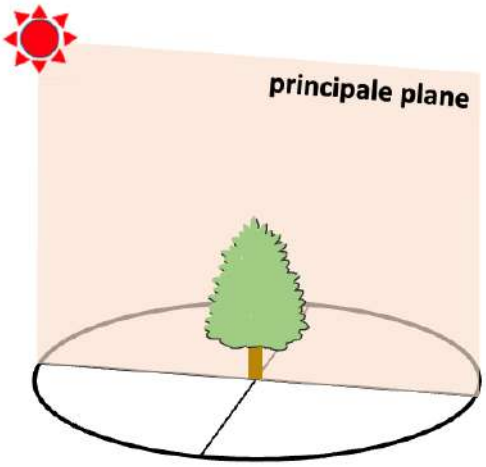



図 4.10 principale plane のイメージ

図 4.9 で可視化した BRDF から樹木の principale plane と UAV の軌道が最も近い pass のみで再び散布図を作成し，BRDF を可視化した．可視化した BRDF から回帰直線を求め，相関係数，傾き，切片を算出した．回帰直線の結果から，相関係数が 0.9 以上のばらつきの少ない散布図となった．本研究では，算出した回帰直線の傾きを樹木の BRDF とする．

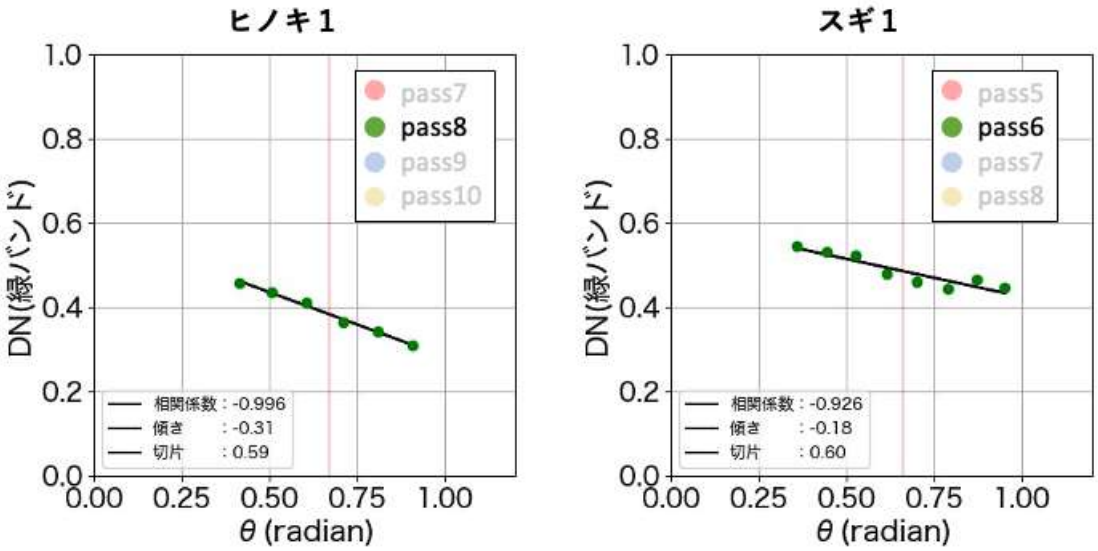
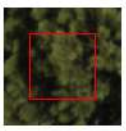
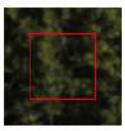



**図 4.11 樹木の principale plane と UAV の軌道が最も近い pass で可視化した BRDF**

BRDF 解析結果が原画像と比較した際にどのような傾向を捉えているのかを確認した．ヒノキ 1 とスギ 1 の原画像を図 4.12 に示す．それぞれの BRDF の負の傾きを示す結果は，各原画像と比較すると葉の色の変化が BRDF と対応していることが確認できる．このことから BRDF は，樹木の物理的特徴を捉えていると言える．

**ヒノキ１ pass8の各原画像**　解析対象面(約60cm×約60cm)

| | | | | | | |
|---|---|---|---|---|---|---|
| 原画像 |  |  |  |  |  |  |
| θ (radian) | 0.42 | 0.51 | 0.61 | 0.71 | 0.81 | 0.91 |

**スギ１ pass6の各原画像**　解析対象面(約60cm×約60cm)

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| 原画像 |  |  |  |  |  |  |  |  |
| θ (radian) | 0.36 | 0.44 | 0.52 | 0.61 | 0.70 | 0.79 | 0.87 | 0.95 |

**図 4.12 ヒノキ 1 とスギ 1 の各原画像**

# 5　BRDF 解析結果

本章では，前章までに解説した手法から樹種別に BRDF の結果を示す．

## 5.1　落葉広葉樹の BRDF

落葉広葉樹の BRDF 解析結果から程度に違いはあるものの多くの樹木が負の傾きを示している．全体的に傾きの度合いは小さく，イロハカエデのように正の傾きを示すものがある．さらに，対象樹木の多くは直線的な傾向であるが，落葉広葉樹は他樹種に比べてコナラやクリ，ヤマザクラのように曲線的な傾向が一部見られる．しかし樹種判別が可能なほど樹種別に BRDF の特徴は現れない結果となった．

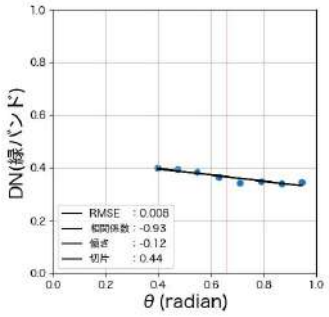



図 5.1 アキニレ 1

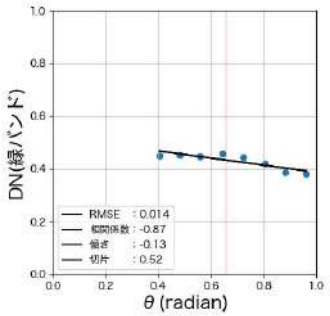



図 5.2 アキニレ 2

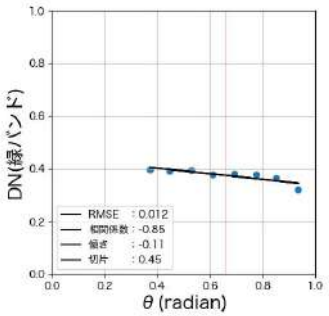



図 5.3 イロハカエデ 1

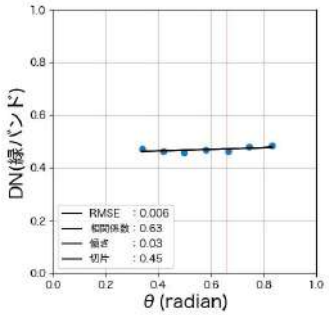



図 5.4 イロハカエデ 2

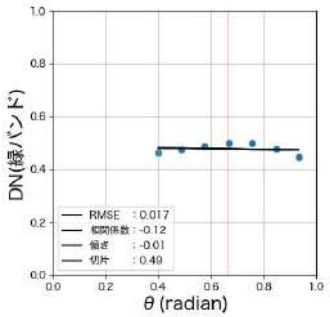



図 5.5 イロハカエデ 3

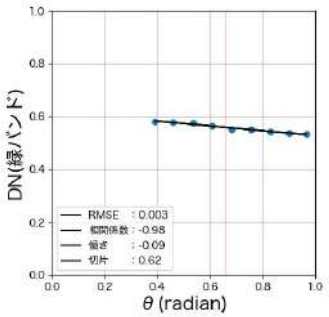



図 5.6 カキ 1

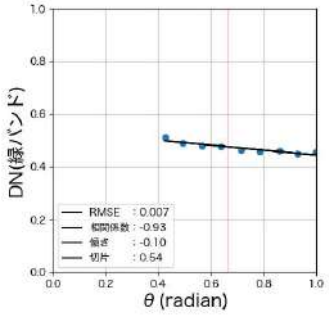



図 5.7 カキ 2

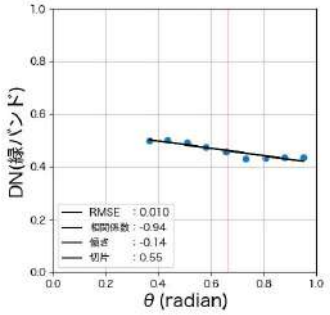



図 5.8 カキ 3

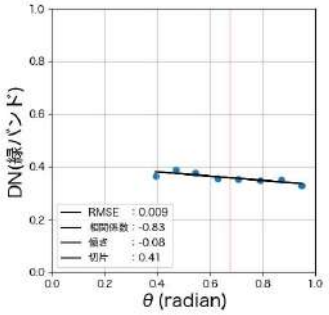



図 5.9 ケヤキ 1

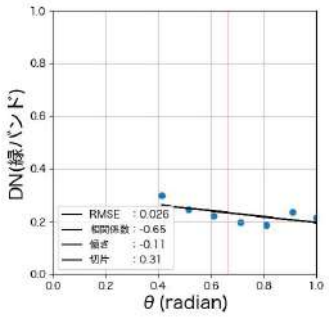



図 5.10 ケヤキ 2

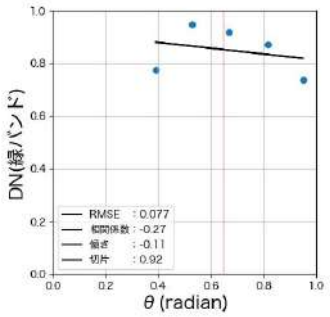



図 5.11 コナラ

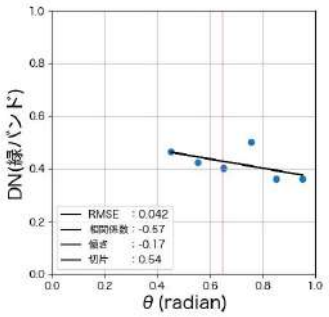



図 5.12 クリ 1

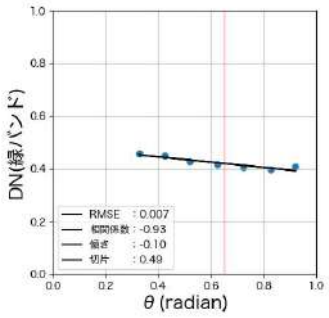



図 5.13 クリ 2

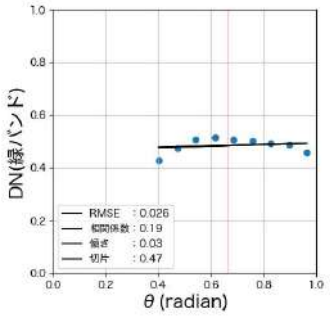



図 5.14 クリ 3

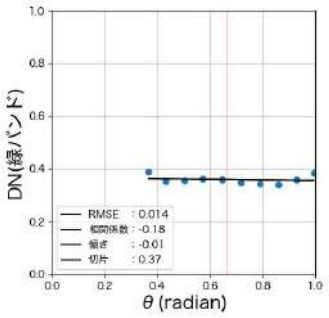



図 5.15 クワ

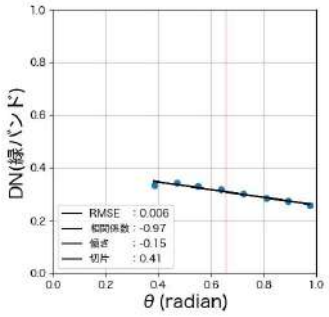



図 5.16 ムクノキ

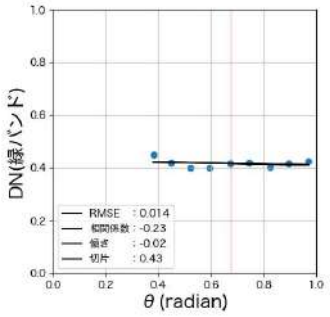



図 5.17 ヤマザクラ 1

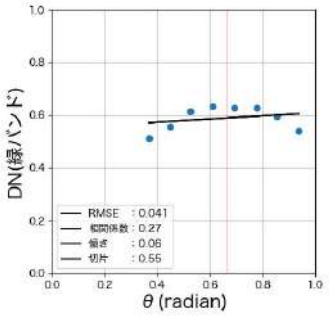



図 5.18 ヤマザクラ 2

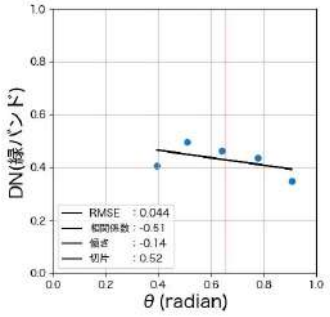



図 5.19 ヤマザクラ 3

## 5.2 常緑広葉樹の BRDF

常緑広葉樹の BRDF 解析結果は，全体的に負の傾きを示しているが落葉広葉樹に比べて正の傾きを示す樹木がやや多い．常緑針葉樹は，落葉広葉樹に比べて直線的な傾向が多く見られる．

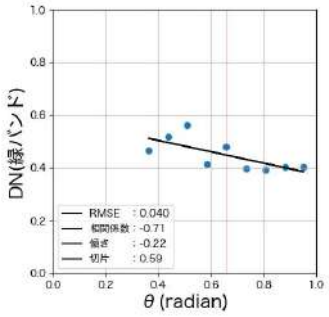



図 5.20 アラカシ 1

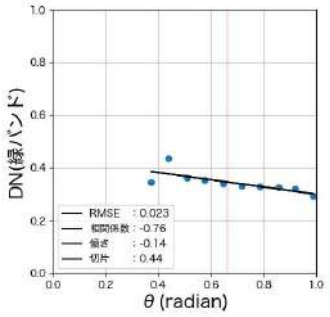



図 5.21 アラカシ 2

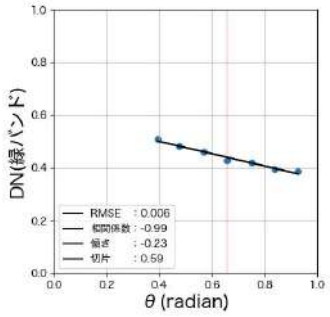



図 5.22 アラカシ 3

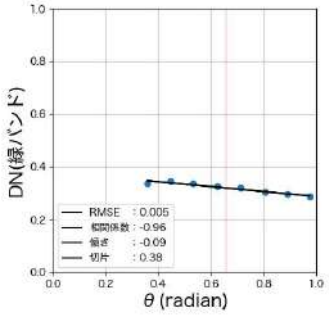



図 5.23 アラカシ 4

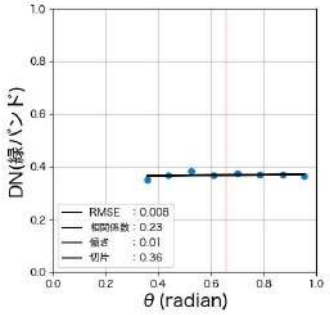



図 5.24 アラカシ 5

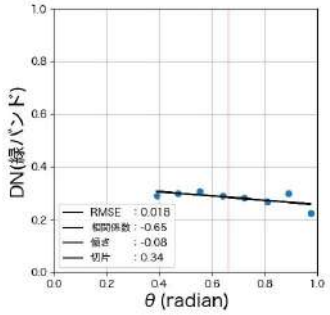



図 5.25 アラカシ 6

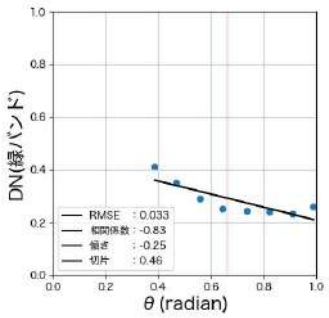



図 5.26 アラカシ 7

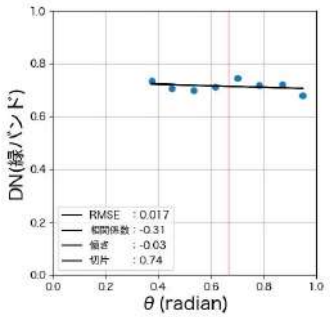



図 5.27 ビワ

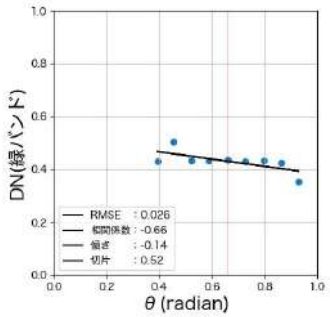



図 5.28 ヒサカキ

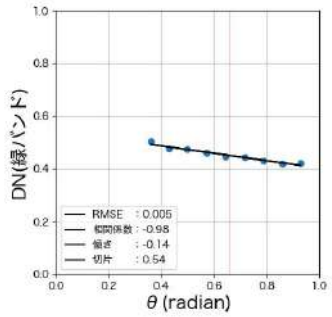



図 5.29 クロガネモチ

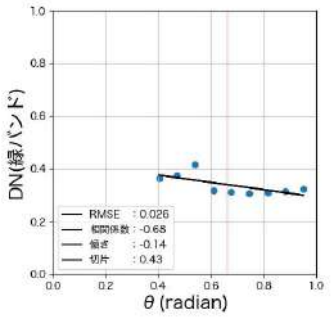



図 5.30 ミカン 1

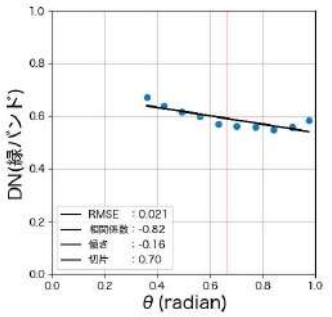



図 5.31 ミカン 2

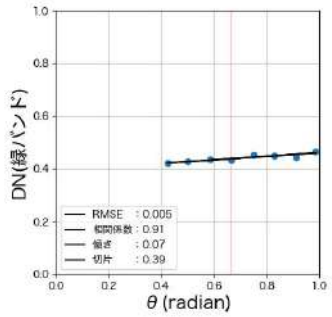



図 5.32 ヤマモモ 1

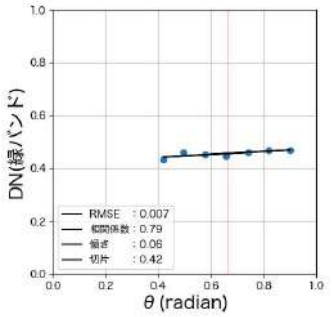



図 5.33 ヤマモモ 2

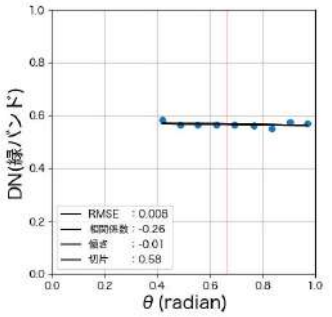



図 5.34 ツバキ

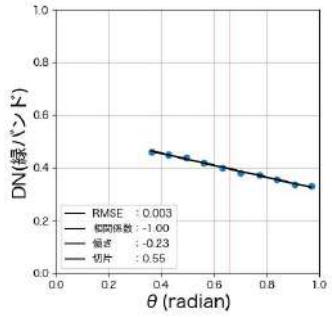



図 5.35 ユズ 1

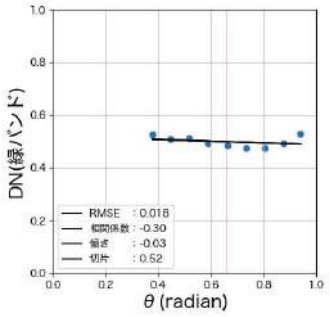



図 5.36 ユズ 2

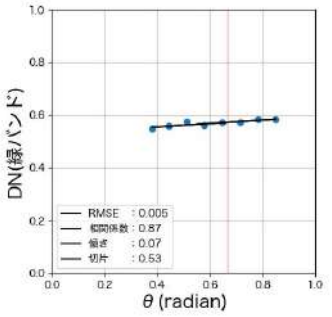



図 5.37 ユズ 3

## 5.3 常緑針葉樹の BRDF

常緑針葉樹の BRDF 解析結果は，全て負の傾きを示している．落葉広葉樹や常緑広葉樹に比べて BRDF の傾きが大きいものが見られる．常緑針葉樹の中でもスギに比べてヒノキの方が BRDF の傾きが大きい傾向にある．

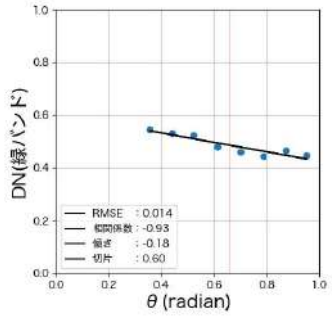



図 5.38 スギ 1

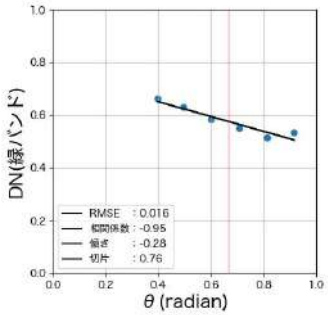



図 5.39 スギ 2

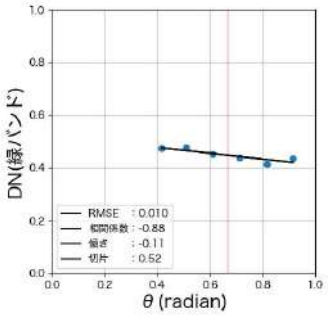



図 5.40 スギ 3

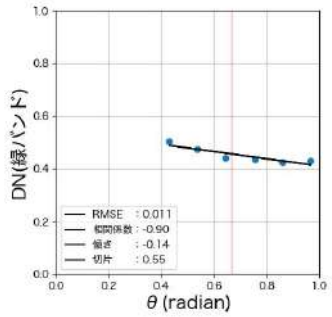



図 5.41 スギ 4

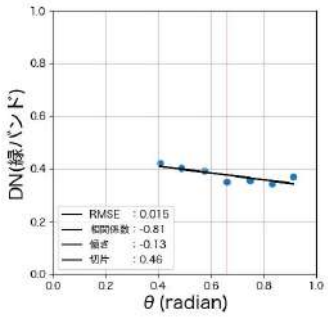



図 5.42 スギ 5

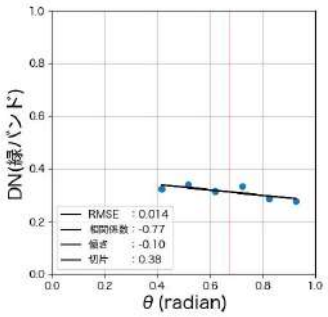



図 5.43 スギ 6

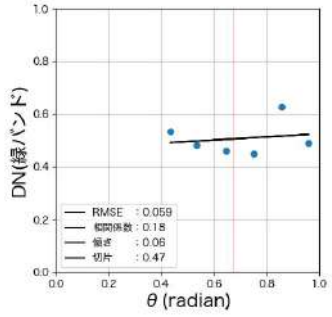



図 5.44 スギ 7

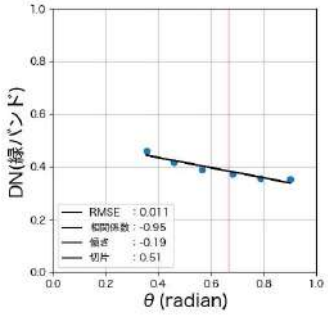



図 5.45 スギ 8

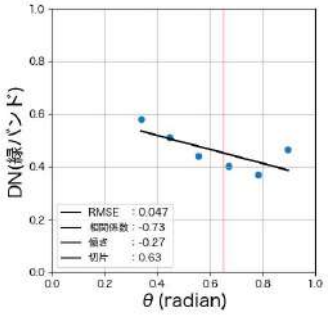



図 5.46 スギ 9

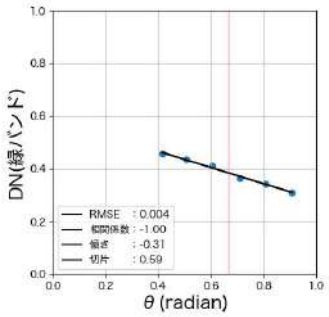



図 5.47 ヒノキ 1

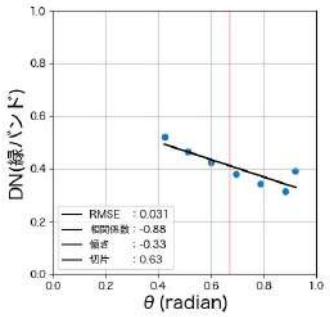



図 5.48 ヒノキ 2

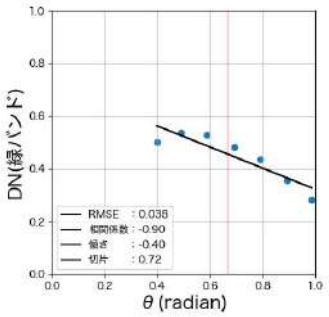



図 5.49 ヒノキ 3

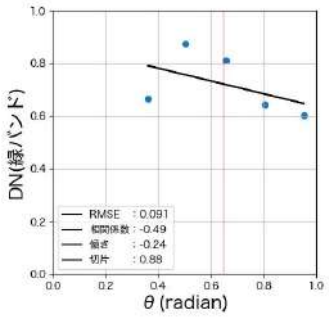



図 5.50 ヒノキ 4

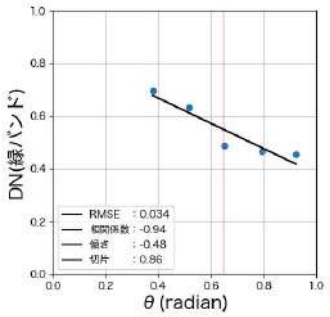



図 5.51 ヒノキ 5

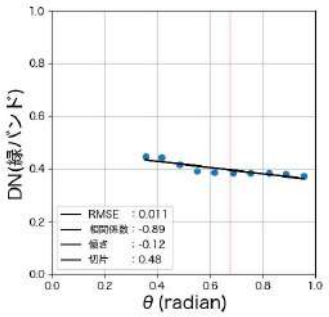



図 5.52 ヒノキ 6

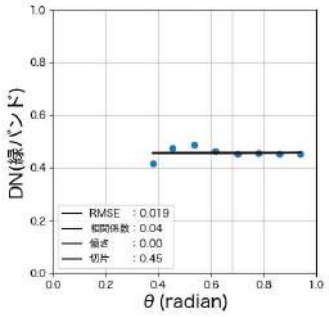



図 5.53 ヒノキ 7

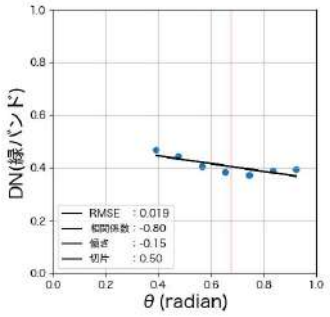



図 5.54 ヒノキ 8

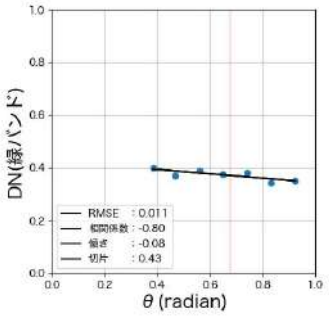



図 5.55 ヒノキ 9

## 5.4　その他の BRDF

落葉針葉樹と常緑竹は，対象とする樹木が少なく樹種による傾向は見られないが他の対象樹木と同様に負の傾きが見られた．

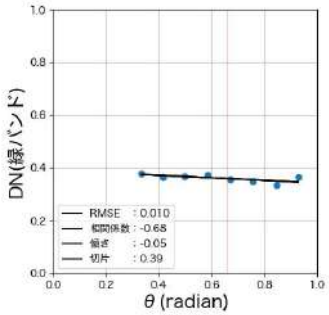



図 5.56 イチョウ (落葉針葉樹)

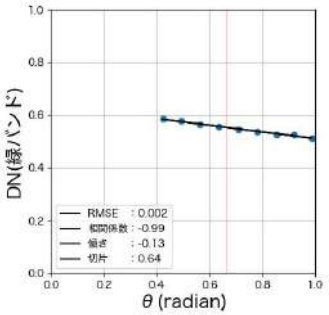



図 5.57 ホウライチク (常緑竹)

## 5.5　BRDF に影響を与えている要素の推察

BRDF は，葉の色の変化の他にも樹木の物理的な特徴を捉えているのではないかと考える．BRDF 解析結果に影響を与える要素として「影割合」「葉身」「解析対象面の向き」の 3 要素が大きいと考える．「影割合」は，は密度や樹木形状，「葉身」は，葉によってできる影の大きさや反射面が 1pix に与える影響，「解析対象面」の向きは，葉の色の変化度合いや反射の強さ，などの影響を BRDF に与えていると考えるそこで，BRDF の数値的な特徴量として回帰直線の傾きを使用し，各要素が BRDF に与える影響を推察する．DN 値のばらつきが大きい不自然なデータは RMSE による閾値を設定し，あらかじめ推察の対象とはしない．この時，RMSE の閾値は 0.05 としそれより大きいものは推察には用いない．閾値により対象としない樹木は，コナラ，スギ 7，ヒノキ 4 である．

### 5.5.1　影割合の影響

解析対象面の影割合の変化が BRDF に与える影響を推察した．本研究において影割合は，正規化後の緑バンドの DN が 0.33(1/3) 以下となるピクセルを影とし，影のピクセル数を解析対象面のピクセル総数で徐した値とする．DN の正規化については 4.4 節で説明している．スギ 1 を例に説明する．スギ 1 の 8 枚の原画像と影になる箇所を図 5.58 に示す．赤マスクしたピクセルが緑バンドの DN が 0.33 以下のピクセルである．原画像の解析対象面内の緑バンドを平均値を縦軸に置いた図が BRDF である．BRDF と同様に縦軸を影割合にした図を影割合変化図とした．この二つの図の傾きを比較することで BRDF と影割合の関係を推察した．その結果を図 5.59 に示す．

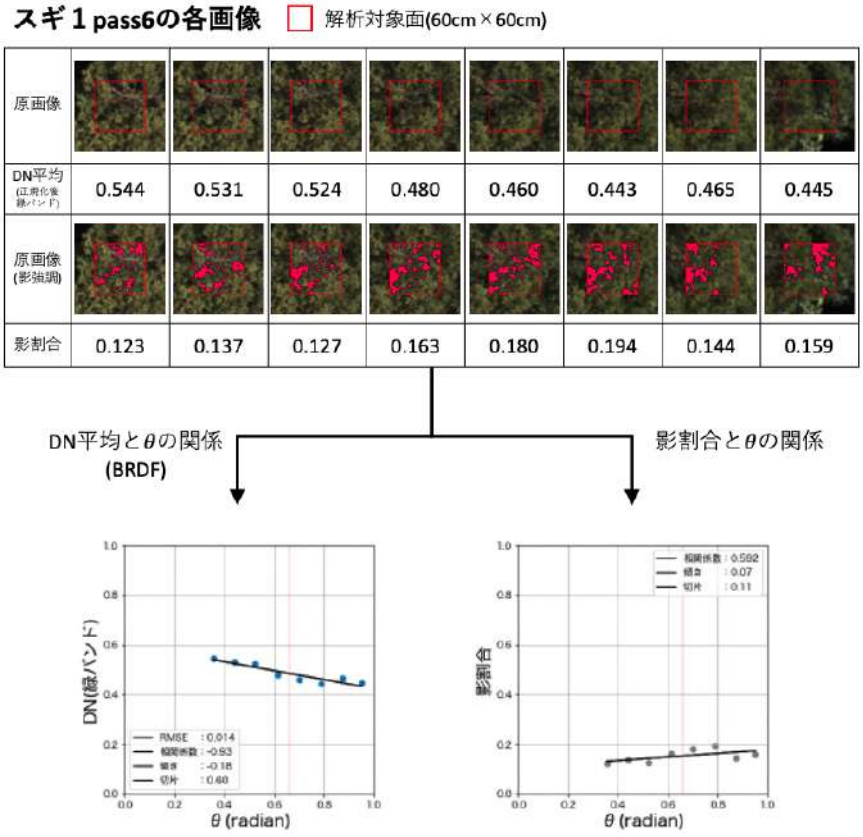



図 5.58 BRDF と影割変化合図

図 5.59 の横軸は，影割合変化図の傾き，縦軸は BRDF の傾きである．散布図の結果からばらつきはあるものの負の相関が見られる．これは影割合の変化が BRDF に与えている影響が大きいと言える．さらに BRDF が正の傾きを示す条件は，位相角 $\theta_{rad}$ が大きくなるにつれて影割合が減少している時に限られることが分かった．$\theta_{rad}$ が大きくなる (観測位置が順光側から逆光側に移動する) につれて影割合が低下するため結果的に平均した DN が増加し，BRDF の傾きが正になっている (図 5.60)．影割合の変化は，順光側 ($\theta_{rad}$ 小) から見た時と逆光側 ($\theta_{rad}$ 大) から見た時の葉群密度の差に影響することが考えられるため樹木の物理的な特徴を捉えていると言える．

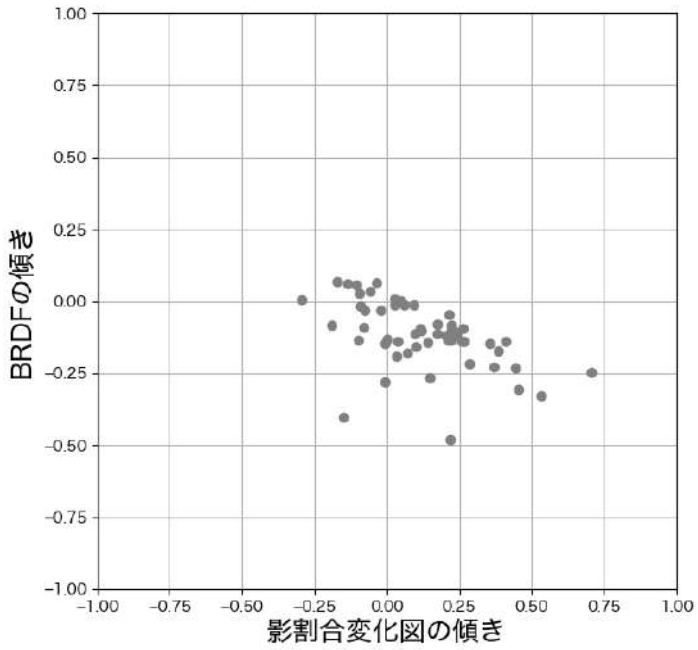



図 5.59 BRDF と影割合変化図の関係

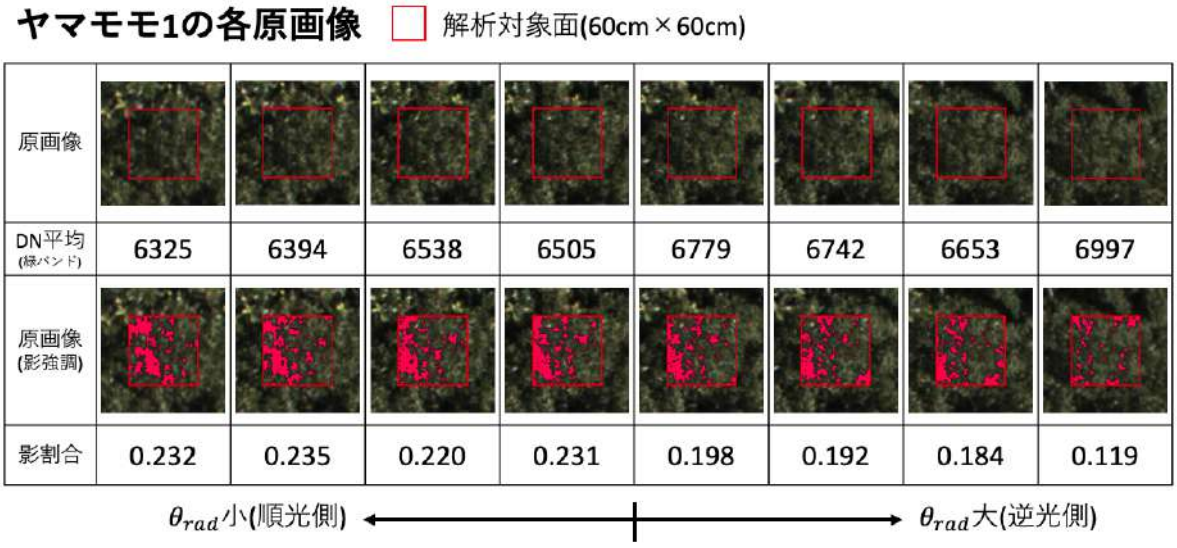



図 5.60 ヤマモモ 1 の各原画像と影割合

### 5.5.2 葉身の影響

葉身とは，図 5.61 に示すような葉の長さのことを指す．BRDF には，樹木ごとの葉身の影響が大きいのではないかと考える．葉身の大きさに応じて樹木に形成される影の量が変化するのではないか考える．本研究では，樹種ごとの一般的な葉身の大きさが BRDF に与える影響を推察した．対象樹種の一般的な葉身を表 5.1 に示す．一般的な葉身は，主に千葉県立中央博物館が公開している「樹木検索図鑑 [7]」の樹木データから参照した．また，樹木検索図鑑にない樹種データは，「三河の植物観察 [8]」「葉と枝による樹木検索図鑑 [9]」を参照した．


図 5.61 葉身

表 5.1 一般的な葉身 (cm)

| 樹種 | 樹名 | 葉身 | 葉身 (中央値) | 樹種 | 樹名 | 葉身 | 葉身 (中央値) |
|---|---|---|---|---|---|---|---|
| 落葉広葉樹 | アキニレ | 3〜5 | 4 | 常緑広葉樹 | アラカシ | 7〜12 | 9.5 |
| | イロハカエデ | 3〜7 | 5 | | バクチノキ | 10〜20 | 15 |
| | カキ | 7〜16 | 11.5 | | ビワ | 15〜30 | 22.5 |
| | ケヤキ | 3〜14 | 7.5 | | ヒサカキ | 3〜8 | 5.5 |
| | コナラ | 7〜14 | 7.5 | | クロガネモチ | 5〜8 | 6.5 |
| | クリ | 7〜20 | 13.5 | | ミカン | 9〜11 | 10 |
| | クワ | 6〜14 | 10 | | ツバキ | 5〜12 | 8.5 |
| | ムクノキ | 4〜10 | 7 | | ヤマモモ | 5〜12 | 8.5 |
| | ヤマザクラ | 8〜12 | 10 | | ユズ | 6〜9 | 7.5 |
| 落葉針葉樹 | イチョウ | 4〜8 | 6 | 常緑針葉樹 | スギ | 1 | 1 |
| 常緑竹 | ホウライチク | 6〜12 | 9 | | ヒノキ | 0.2〜0.3 | 0.25 |

参照: ミカン, ユズ (葉と枝による樹木検索図鑑 [9])
ホウライチク (三河の植物観察 [8])
その他 (樹木検索図鑑 [7])

図 5.62 に BRDF の傾きと一般的な葉身の関係を示す．図 5.62 の横軸は一般的な葉身の中央値，縦軸は BRDF の傾きである．この結果からは，葉身の大きさが BRDF の傾きに影響を与えていることは明らかにはならなかった．この要因としては，葉身を一般的な大きさとして評価したことにあると考えられる．樹木の葉身は，「樹木の成長度合いの違い」や「樹木の上部と下部の違い」などによって大きさが違うことが考えられる．そのため，葉身の影響を推察するためには正しい葉身の大きさを取得する必要がある．

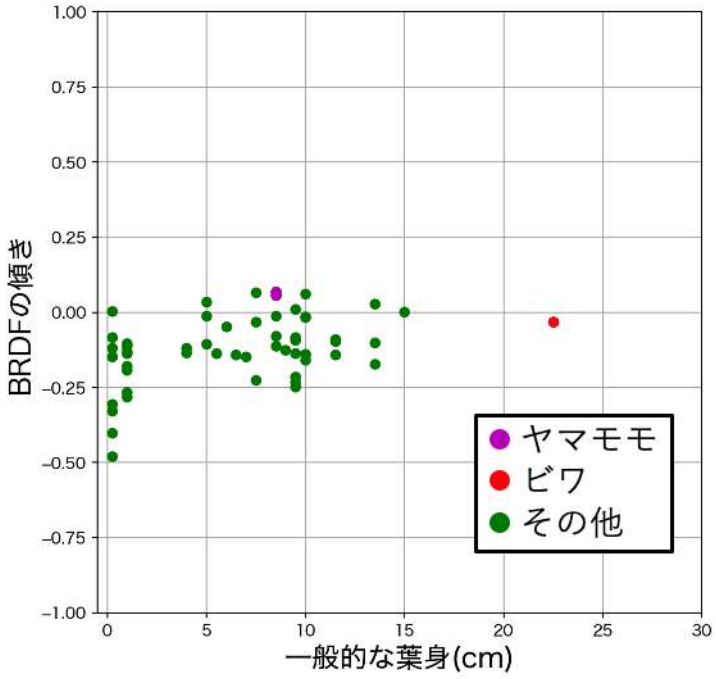



図 5.62 BRDF と葉身の関係

適切に葉身の影響を推察するための改善方法として，撮影した画像から解析対象面の葉の葉身を直接計測す

る方法である．例にビワとヤマモモを撮影した画像を図 5.63 に示す．一般的にビワは葉身が約 22.5cm と非常に大きいため画像上で 1 枚の葉が確認できる．しかし，ヤマモモのように葉身が約 8.5cm になると本観測で撮影した画像では葉身が確認できるほどの解像度がない．葉身の影響を適切に推察するためには，ほとんどの樹木において画像上で葉身が読み取れるほどの高解像度の観測が必要である．

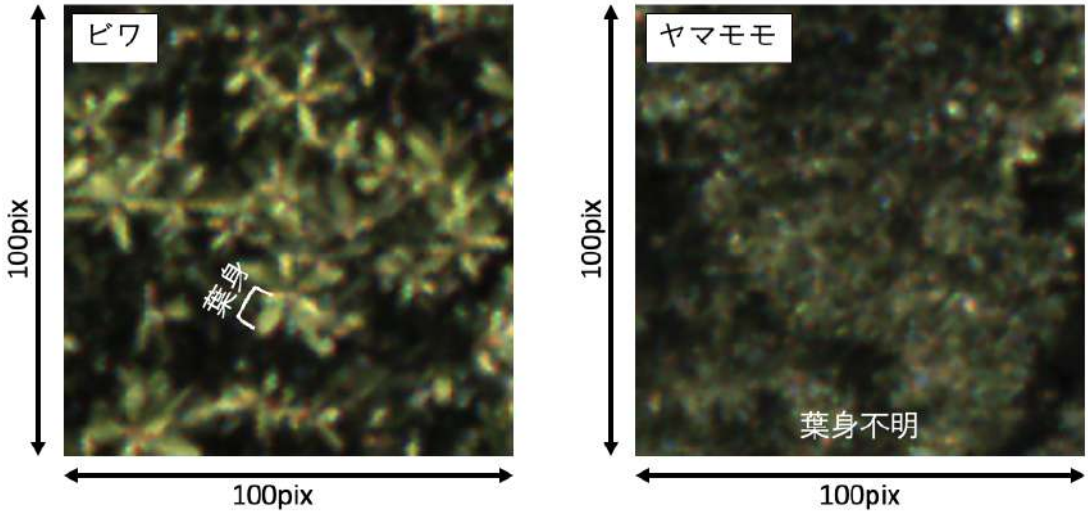



**図 5.63 原画像で見た葉身**

### 5.5.3　解析対象面の向きの影響

解析対象面の向きが BRDF に与える影響を推察した．解析対象面の向きとは，図 5.64 のように 3 次元点群で表現された対象樹木 (ヒノキ 1) の最小二乗平面を算出し，平面が向いている方位とその傾斜角度のことである．この時，最小二乗平面は北北東を向き，傾斜角度は 14.4° である．

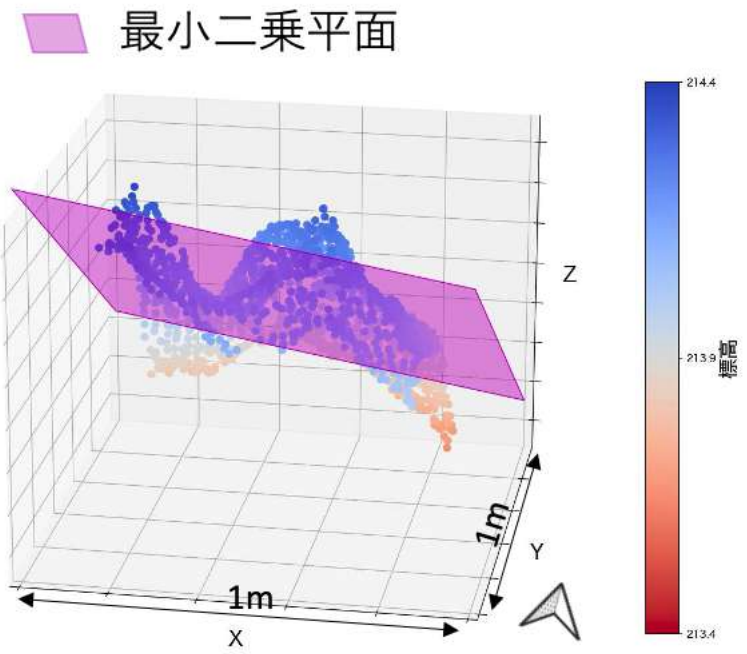



図 5.64 解析対象面の点群形状と最小二乗平面

BRDF と解析対象面の向きとの比較結果を図 5.65 に示す，結果は，BRDF の傾きと解析対象面の向きとの関係は明らかにならなかった．BRDF は反射強さを見ているため，面の向いている向きは BRDF の傾向に影響を与える大きな要因の一つであると考える．そのため，本推察で解析対象面の向きの影響が確認できなかった原因は，解析対象面の向きの他に葉の向いている方向や樹頂点の凹凸度合いなどの要素が複雑に影響し合っていることが考えられる．

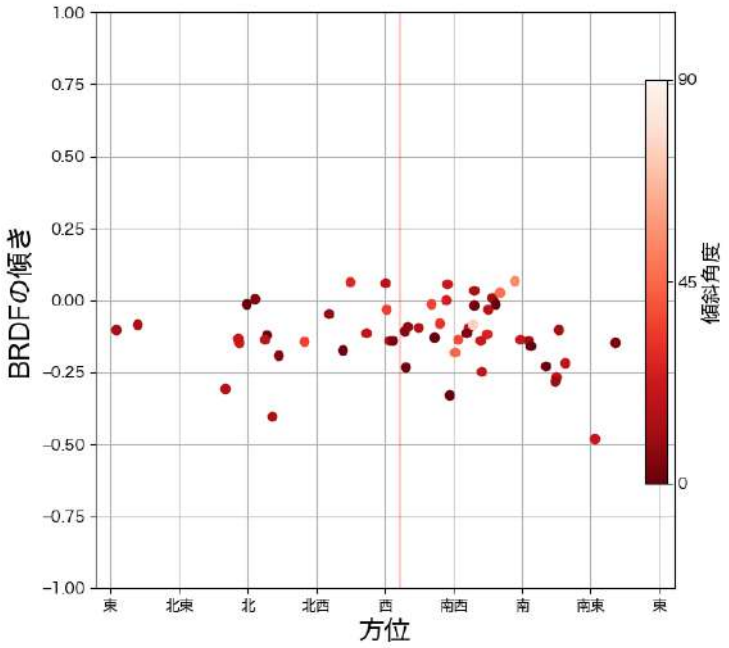



図 5.65 BRDF と解析対象面の向きとの関係

# 6 結論

## 6.1 結論

本研究では，UAV 搭載デジタルカメラによる樹木の二方向性反射特性 (BRDF) 解析を行った．まず，バッドピクセルと周辺減光に対する校正手法を構築した．手法の構築からフルサイズセンサカメラの Raw 画像を使用した高精度な解析が可能となった．

対象樹木の BRDF は，多くが回帰直線で表現することができた．また，傾向として後方散乱性 (BRDF の傾きが負) を示す結果となった．BRDF に影響を与える要素として，「影割合の変化」「一般的な葉身の大きさ」「解析対象面の向き」が考えられる．まず，影割合の変化との推察では，BRDF の傾きの度合に影響を与えていることが分かった．影割合の変化が大きいほど BRDF の負の傾きが大きくなることから影割合が BRDF に影響を与えいていると言える．さらに BRDF が正の傾きを示す条件は，位相角 $\theta_{rad}$ が大きくなるにつれて影割合が減少している時に限られることが分かった．樹木の葉は基本的に後方散乱性であり，観測位置が順光側から逆光側になるにつれて DN 値が減少していく傾向にある．しかし，BRDF には影が含まれているため樹木の観測位置の違いによる影の量の変化から BRDF は正の傾きになると考えられる．

葉身の大きさと解析対象面の向きの影響については，本研究では明らかにならなかった．しかし，葉身の大きさは葉群の影の形成に影響すると考えられ，解析対象面は，光源となる太陽の反射方向に大きく影響することが考えられるため，BRDF に与える影響は大きいと考える．他にも BRDF に影響を与える要素として，「葉の向き」や「葉の表面材質」「葉の健康状態 (湿潤状態)」が影響しているのではないかと考えられる．これらの結果から，BRDF は葉の色の変化と影割合の変化の両方の影響を大きく受けることから葉の特徴と樹木の物理的特徴を捉えられていることが言える．しかし，その他の BRDF に影響を与える要素は複合的に影響し合っていることが考えられるため，BRDF に影響を与えている要素を明らかにするためには，少なくとも葉身より細かな地上分解能での高解像度な BRDF 観測が必要である．

## 6.2 課題

本研究は，分光放射計を使用した実測研究に比べて多くの樹種を対象として BRDF 観測を行った．観測方法の課題として，BRDF に影響を与える要素を明らかにするためには，葉身より細かな地上分解能での高解像度な BRDF 観測を行った上で対象樹木の物理的特徴と葉の状態や大きさなどを細かく把握した上で解析が必要である．

解析方法の大きな課題として，本研究はデジタルカメラを使用していることから反射率ではなく DN 値で評価しており，放射量的な値である点にある．反射率による評価ができないため，現状，他の時期の解析結果との比較ができず，BRDF の時系列変化 (季節変化) を評価することができない．デジタルカメラを使用した解析では今後，DN 値を反射率へ変換する手法が必要不可欠である．また，本研究における樹木の BRDF とは影を含んだままの状態での観測位置の違いによる反射量の変化のこととしたが樹種ごとの特徴を捉えるためには影の影響を含まない状態が適切である可能性がある．さらに今後，植生の物理的指標として BRDF を用いるためには，葉の色の影響を明らかにするために多時期での考察が必要である．

# 7　引用文献と参考文献

## 引用文献


[1] S.Sandmeier,D.W.Deering:Structure Analysis and Classification of Boreal Forests Using Airborne Hyperspectral BRDF Data from ASAS,Remote Sens. Environ.,69, pp.281-295,1999

[2] 長谷川宏一, 泉岳樹, 松山洋, 梶原康司, 本多嘉明: カラマツ林における方向別分光反射特性と植物フェノロジーの関係, 第 120 回日本森林学会大会予稿集,Pa1-21,2009

[3] Jouni I.Peltoniemi,Sanna Kaasalainen,Jyri Näränen, Miina Rautiainen,Pauline Stenberg,Heikki Smolander,Sampo Smolander, Pekka Voipio:BRDF measurement of understory vegetation in pine forests: dwarf shrubs, lichen, and moss,Remote Sens. Environ.,94 pp.343-354,2005

[4] 村井亮介:UAV に搭載したデジタルカメラによる植生の生物季節解析, 高知工科大学,2022 年博士論文

[5] 高知工科大学, ”里山単木調査データ”, 里山工学,https://satoyama.kochi-tech.ac.jp/data/里山単木調査,(参照 2024-1-17)

[6] ”キャンプ工学”,https://campkougaku.com/category/python/,(参照 2024-1-17)

[7] 千葉県立中央博物館, ”樹木検索図鑑”,http://www.chiba-museum.jp/jyumoku2014/kensaku/,(参照 2024-1-17)

[8] ”三河の植物観察”,https://mikawanoyasou.org/,(参照 2024-1-17)

[9] ”葉と枝による樹木検索図鑑”,https://elm3.web.fc2.com/,(参照 2024-1-17)

# 参考文献


[10] 陳路, 古海忍, 村松加奈子, 本多嘉明, 梶原康司, 醍醐元生:ADEOS-Ⅱ/GLI データを用いた全球植生純一次生産量の推定における二方向性反射率の影響評価, 同志社大学ワールドワイドビジネスレビュー,9 巻,1号,pp.90-102,2007

[11] 長谷川宏一, 酒井健吾, 泉岳樹, 松山洋: 森林を対象とした BRDF の実測研究および数値シミュレーション研究の現状と発展の方向性, 日本リモートセンシング学会誌,Vol.36,No.3,pp.225-235,2016

[12] 長谷川宏一, 松山洋, 都築勇人, 末田達彦: 植生指標を用いた葉面積指数の把握に二方向性が及ぼす影響-カナダ北西部における山火事後の遷移段階にある植生を対象に-, 日本リモートセンシング学会誌,Vol.26,No.3,pp.186-201,2006

[13] 長峰克己, 村上仁士: 二方向性反射シミュレーション-特に森林地帯を事例にして-, 写真測量とリモートセンシング,Vol.20,No.2,1981

[14] 小野裕作, 梶原康司, 本多嘉明: 樹冠形状を反映した多角の分光反射率の推定に関する研究, 写真測量とリモートセンシング,Vol.49,No.2,pp.58-6,2010

[15] Jing M. Chen,Jane Liu,Sylvain G. Leblanc,Roselyne Lacaze,Jean-Louis Roujean:Multi-angular optical remote sensing for assessing vegetation structure and carbon absorption,Remote Sens. Environ.,84,pp.516-525,2003

[16] Janne Heiskanen:Tree cover and height estimation in the Fennoscandian tundra-taiga transition zone using multiangular MISR data,Remote Sens. Environ.,103,pp.97-114,2006

[17] Stefan R. Sandmeier,Elizabeth M. Middleton,Donald W. Deering,Wenhan Qin:The potential of hyperspectral bidirectional reflectance distribution function data for grass canopy characterization,Journal of Geophysical Research: Atmospheres,Vol.104,Isuue D8,p.9547-9560,1999

[18] Hankui K. Zhang,Daivid P. Roy,Lin Yan,Zhongbin Li,Haiyan Huang,Eric Vermote,Sergii Skakun,Jean-Claude Roger:Characterization of Sentinel-2A and Landsat-8 top of atmosphere, surface, and nadir BRDF adjusted reflectance and NDVI differences,Remote Sens. Environ.,215 pp.482-494,2018

[19] D.P.Roy,H.K.Zhang,J.Ju,J.L.Gomez-Dans,P.E.Lewis,C.B.Schaaf,Q.Sun,J.Li,H.Huang,V.Kovalskyy:A general method to normalize Landsat reflectance data to nadir BRDF adjusted reflectance,Remote Sens. Environ.,176 pp.255-271,2016

[20] 友常溝利, 小島崇弘, 永井信, 小林祥子, 関川清広: ドローン空撮画像における生物季節を利用した里山樹林の樹種判別-都市域に残された里山生態系の構造解析に向けて-, 玉川大学農学部研究紀要, 第 6号,25-35,2021

[21] 蚊野浩: デジタルカメラのしくみと画像処理, 画像電子学会誌,Vol.41,No.3,p.288-295,2012

[22] 高木方隆: 国土を測る技術の基礎, 日本測量協会,2012

[23] 米富源一郎: 光源とカメラの位置関係が UAV 植生画像に及ぼす影響, 高知工科大学, 学士論文,2022

# 8　謝辞

本研究を進めるにあたり，多くのご指導を頂いた高知工科大学の高木方隆教授に心から感謝いたします．研究活動を通して，私自身には持っていなかった新しい考え方や知見，知識を学ぶことができ，様々な測量技術に触れる貴重な体験をいただきました．普段の私生活では体験できないような里山暮らしならではの活動は，非常に楽しく研究室に配属されてからの 3 年半は驚くほど早く感じました．先生から教えを頭に置き，今後も頑張ります．

また，副指導教員である佐藤愼司教授には，修士の研究を進めるにあたり，3 者会議で大変お世話になりました．的確なアドバイスのもとご指導をいただいたことを心から感謝いたします．

本研究室の村井亮介助教には，これまで研究に使用するデータを撮影し続けていただきました．本研究を進めるにあたり，多大なご協力をいただいたことを心から感謝いたします．様々な道具やカメラ，植物，革細工など多くの知識や経験があり，私自身大変影響を受けました．

研究室にてこれまでお世話になった同期，後輩の皆さんには，私と仲良くしてくださり大変感謝しています．本研究室で研究活動ができた時間は，とても大切な思い出です，本研究に関わって下さった方々に厚く御礼申し上げます．

最後に，多大なサポートしてくれた家族に心より感謝します．

2024 年 1 月

# 9 付録

本研究で作成したプログラムを実行順に以下に示す．プログラム内容の詳細は次節より説明する．

[A] A_BadPixel_ShadingCorrection.py (3.3 節，3.4 節)
(バッドピクセルの特定, 周辺減光補正係数作成のプログラム)

[B] B_P1_DNGtoTIF-Shading_Correction_Image.py (3.2 節)
(Raw 現像のプログラム)

[C] 1_V_XY-imageCreate-P1.py (3.6 節，4.1 節，4.2 節)
(DSM の作成，オルソフォト変換, オクルージョン処理のプログラム)

[D] 2_Calculate_angle_and_color-P1.py (4.3 節)
(カメラ方向ベクトルと観測方向ベクトル，DN 値の平均を算出するプログラム)

[E] 3_Make_graph_BRDF-P1.py (4.3 節，4.4 節)
(位相角 $\theta$ の算出，BRDF 可視化のプログラム)

## 9.1　バッドピクセルの特定, 周辺減光補正係数作成のプログラム

プログラム名 : A_BadPixel_ShadingCorrection.py (3.3 節，3.4 節)

本プログラムでは，バッドピクセルの特定と周辺減光補正係数を作成した．入力データは，複数の BRDF 観測で撮影したダーク画像，Image Engineering 社の積分球「LE7 VIS-IR」を撮影した画像である．出力データは，バッドピクセルの画像座標データ，バッドピクセルの出現率を可視化した画像，周辺減光補正係数と可視化した画像である．周辺減光補正係数は，バッドピクセル箇所を除去した係数データである．以下にプログラムのソースコードを記す．

Listing 1 A_BadPixel_ShadingCorrection.py

```
import cv2
import glob
import os
import rawpy
import rawpy.enhance
import numpy as np
from tqdm import tqdm
import tifffile as tif

cwd = os.getcwd()
path  = os.path.splitext(cwd[:len(cwd)-15])[0]
#使用HDD path = /Volumes/Capotasto2

################################################################################################
# A.1 入出力データ------------------------------------------------------------------------------
################################################################################################

#A.3    BRDF観測日のリスト-
InputData_1  = path+"/2023EW/Data/Saoka-20*-P1"
#A.3 各観測日のダーク画像
InputData_2  = "/Dark/*.DNG"
#A.4 積分球を撮影した画像
InputData_3  = path+"/TRIOPTICS_For_P1/PhotoImage_P1/Flat_LightSourceFromLE7/"+\
                +"FNumber5.6/*.DNG"

#A.3 各観測日のバッドピクセルの画像座標データ
OutputData_1 = "-BadPixelLocation.npz"
#A.3 バッドピクセルの出現率別の位置画像
OutputData_2 = path+"/2023EW/Data/Result_BadPixel-P1/"
#A.4 周辺減光補正係数
OutputData_3 = path+"/TRIOPTICS_For_P1/PhotoImage_P1/Result_LimbDarkening/"+\
                +"P1-FlatCofficient.npz"
#A.4 周辺減光補正係数画像
OutputData_4 = path+"/TRIOPTICS_For_P1/PhotoImage_P1/Result_LimbDarkening/"+\
                +"P1-FlatCofficient.tif"
```

```
#############################################################################################
# A.2 Raw現像関数-------------------------------------------------------------------------
#############################################################################################

#デベイヤの設定
RawPyParameters1 = rawpy.Params(
                        #色深度
                        output_bps              = 16,
                        #色空間
                        output_color            = rawpy.ColorSpace.raw,  # raw=0
                        #スケール調整の有無
                        no_auto_scale           = False,
                        #露出
                        no_auto_bright          = True,
                        #ホワイトバランス
                        use_camera_wb           = True,
                        use_auto_wb             = False,
                        user_wb                 = [2.6640625, 1.0, 1.8046875, 0.0],
                        #デモザイク
                        demosaic_algorithm      = rawpy.DemosaicAlgorithm.LINEAR,
                        #階調の応答特性
                        gamma                   = [1.00,0.00] #ガンマ無し
                    )

# Raw Image PostProcess Function
def RawPostprocess(raw_original):
    #ディベイヤ処理する
    RGR_original = raw_original.postprocess(params=RawPyParameters1)
    hight, width, channel = RGR_original.shape
    raw_original.close()
    return RGR_original, hight, width, channel

#############################################################################################
# A.3 バットピクセルの特定----------------------------------------------------------------
#############################################################################################

ObservationData = sorted(glob.glob(os.path.join(InputData_1)))
ObservationData = ObservationData[14::]

for i in ObservationData:
    print(i)
    List_RawD =glob.glob(os.path.join(i+InputData_2))
    print("ダーク画像枚数：",len(List_RawD))
    if  List_RawD == []:
        print("No_DarkFolder")
        pass
    else:
        bad_pixels    = rawpy.enhance.find_bad_pixels(List_RawD)
        hot_pixels    = rawpy.enhance.find_bad_pixels(List_RawD,find_hot=True,
                                                      find_dead=False,
```

```
                                                    confirm_ratio=0.9)
        deadpixels    = rawpy.enhance.find_bad_pixels(List_RawD,find_hot=False,
                                                      find_dead=True,
                                                      confirm_ratio=0.9)
        np.savez(i+"/"+i[-22:]+OutputData_1,bad_pixels)
        print("バッドピクセル個数",len(bad_pixels),",ホット個数",len(hot_pixels),
              ",デッド個数",len(deadpixels))

ihight, iwidth = 5460,8192 #P1カメラの画像サイズ-
BadPixelLocationImage = np.zeros((ihight,iwidth))
DarkFoldaNumber = 0
for i in ObservationData:
    BadPixelLocation_path = glob.glob(os.path.join(i+"/"+i[-22:]+OutputData_1))
    if  BadPixelLocation_path == []:
        print("No_BadPixelLocationData")
        pass
    else:
        DarkFoldaNumber += 1
        BadPixelLocation_npz = np.load(BadPixelLocation_path[0])
        BadPixelLocation     = BadPixelLocation_npz["arr_0"]

        for badpixel in BadPixelLocation:
            BadPixelLocationImage[badpixel[0],badpixel[1]] = \
                BadPixelLocationImage[badpixel[0],badpixel[1]] + 1

BadPixelLocationImage2 = BadPixelLocationImage / DarkFoldaNumber

def BadPixelLocation_MoreThen(Num):
    BadPixelLocation_0 = []
    for i in tqdm(range(ihight)):
        for j in range(iwidth):
            if BadPixelLocationImage2[i,j] > Num:
                BadPixelLocation_1 = []
                BadPixelLocation_1.insert(0,i)
                BadPixelLocation_1.insert(1,j)
                BadPixelLocation_0.append(BadPixelLocation_1)
    return BadPixelLocation_0

BadPixelLocation_MoreThen0   = BadPixelLocation_MoreThen(0)
BadPixelLocation_MoreThen10  = BadPixelLocation_MoreThen(0.1)
BadPixelLocation_MoreThen20  = BadPixelLocation_MoreThen(0.2)
BadPixelLocation_MoreThen30  = BadPixelLocation_MoreThen(0.3)
BadPixelLocation_MoreThen40  = BadPixelLocation_MoreThen(0.4)
BadPixelLocation_MoreThen50  = BadPixelLocation_MoreThen(0.5)
BadPixelLocation_MoreThen60  = BadPixelLocation_MoreThen(0.6)
BadPixelLocation_MoreThen70  = BadPixelLocation_MoreThen(0.7)
BadPixelLocation_MoreThen80  = BadPixelLocation_MoreThen(0.8)
BadPixelLocation_MoreThen90  = BadPixelLocation_MoreThen(0.9)
BadPixelLocation_MoreThen100 = []
for i in tqdm(range(ihight)):
```

```
    for j in range(iwidth):
        if BadPixelLocationImage2[i,j] >= 1:
            BadPixelLocation_1 = []
            BadPixelLocation_1.insert(0,i)
            BadPixelLocation_1.insert(1,j)
            BadPixelLocation_MoreThen100.append(BadPixelLocation_1)

def BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen,Num):
    print("バッドピクセル数:",len(BadPixelLocation_MoreThen),"個")
    Image = np.zeros((ihight,iwidth))
    for i in BadPixelLocation_MoreThen:
        Image[i[0]-10:i[0]+11,i[1]-10:i[1]+11] = 1
    tif.imsave(OutputData_2+"BadPixels_MoreThen_"+str(Num)+"%.tif",Image)

    return Image

BadPixelLocation_Image0   = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen0,0)
BadPixelLocation_Image10  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen10,10)
BadPixelLocation_Image20  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen20,20)
BadPixelLocation_Image30  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen30,30)
BadPixelLocation_Image40  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen40,40)
BadPixelLocation_Image50  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen50,50)
BadPixelLocation_Image60  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen60,60)
BadPixelLocation_Image70  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen70,70)
BadPixelLocation_Image80  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen80,80)
BadPixelLocation_Image90  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen90,90)
BadPixelLocation_Image100  = BadPixelLocation_MoreThenImage(BadPixelLocation_MoreThen100,100)

##############################################################################
# A.4 周辺減光補正係数の作成----------------------------------------------------
##############################################################################

List_Raw =glob.glob(os.path.join(InputData_3))
#加算明合成下準備
iTotalNum = len(List_Raw)

with rawpy.imread(List_Raw[0]) as raw_original:
    RGR_original, ihight, iwidth, ichannel = RawPostprocess(raw_original)
    imAdd                                  = np.zeros((ihight, iwidth, ichannel,
                                                       iTotalNum), np.uint16)

#レンズシェーディング係数算出の下処理　: 標準となる白板撮影画像を複数枚画像から作成する
for inum in tqdm(range(0,iTotalNum)):
    #ディベイヤ処理画像情報を取得する
    with rawpy.imread(List_Raw[inum]) as raw_original: #▼箱用下処理　1
        RGR_original, hight, width, channel = RawPostprocess(raw_original)
        print(RGR_original)

    #合成
    imAdd[:,:,:,inum] = RGR_original
```

```
#標準白板撮影画像データ
iMean = np.mean(imAdd, axis = 3) #バンドごとに平均を入れる

iMean2 = iMean.copy()

#バッドピクセルを除去
for bad in BadPixelLocation_MoreThen80:
    iMean2[bad[0]-1:bad[0]+2,bad[1]-1:bad[1]+2] = 0
    iMean2[5272-1:5272+2,4706-1:4706+2] = 0

#一括で計算して保存
iFlatCoefficient_R = iMean2[:,:,0] / np.max(iMean2[:,:,0])
iFlatCoefficient_G = iMean2[:,:,1] / np.max(iMean2[:,:,1])
iFlatCoefficient_B = iMean2[:,:,2] / np.max(iMean2[:,:,2])
iFlatCoefficient = cv2.merge((iFlatCoefficient_R, iFlatCoefficient_G, iFlatCoefficient_B))

np.savez(OutputData_3, iFlatCoefficient)
tif.imsave(OutputData_4,iFlatCoefficient)
```

## 9.2 Raw 現像のプログラム

プログラム名 : B_P1_DNGtoTIF-Shading_Correction_Image.py (3.2 節)

本プログラムは，Raw データを現像するプログラムである．9.1 で作成したバッドピクセル箇所を除去した周辺減光補正係数を使用し，本プログラムの Raw 現像は，バッドピクセルの除去と周辺減光補正を同時に行なった，入力データは，カメラの位置姿勢情報データ，周辺減光補正係数，Raw データである．出力データは，周辺減光補正後の原画像である．以下にプログラムのソースコードを記す．

Listing 2 B_P1_DNGtoTIF-Shading_Correction_Image.py

```
import os
import cv2
import sys
import glob
import rawpy
import rawpy.enhance
import numpy as np
import pandas as pd
from tqdm import tqdm
import tifffile as tif
import matplotlib.pyplot as plt

cwd = os.getcwd()
path  = os.path.splitext(cwd[:len(cwd)-15])[0]
#使用HDD path = /Volumes/Capotasto2

################################################################################
# B.1 入出力データ------------------------------------------------------------
################################################################################

Date    = "20220720"
Time    = "1453"
Camera  = "P1"

#カメラ位置姿勢情報データ
InputData_1  = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
               +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-CameraOmegaPhiKappa.txt"
#周辺減光補正係数
InputData_2  = path+"/TRIOPTICS_For_P1/PhotoImage_P1/Result_LimbDarkening/"+\
               +"P1-FlatCofficient.npz"
#Rawデータ-
InputData_3  = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+"/RAW/"

#周辺減光補正後の原画像
OutputData_1 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+"/tif/"+\
               +"Shading_Correction_Image/"
```

```
cameradata=pd.read_csv(InputData_1,delim_whitespace=True,
                       names=("photoID","x","y","z","Omega","Phi","Kappa",
                              "r11","r12","r13","r21","r22","r23","r31","r32","r33"),
                              skiprows=2)
CAMERADATA=cameradata
photolist=np.array(cameradata["photoID"])
potolist=photolist.tolist()

###############################################################################
# B.2 Raw現像関数-------------------------------------------------------------
###############################################################################

#デベイヤの設定
RawPyParameters1 = rawpy.Params(
                        #色深度
                        output_bps          = 16,
                        #色空間
                        output_color        = rawpy.ColorSpace.raw,  # raw=0
                        #スケール調整の有無
                        no_auto_scale       = False,
                        #露出
                        no_auto_bright      = True,
                        #ホワイトバランス
                        use_camera_wb       = True,
                        use_auto_wb         = False,
                        user_wb             = [2.6640625, 1.0, 1.8046875, 0.0],
                        #デモザイク
                        demosaic_algorithm  = rawpy.DemosaicAlgorithm.LINEAR,
                        #階調の応答特性
                        gamma               = [1.00,0.00] #ガンマ無し
                    )

def RawPostprocess(raw_original):
    #ディベイヤ処理する
    RGR_original = raw_original.postprocess(params=RawPyParameters1)
    hight, width, channel = RGR_original.shape
    raw_original.close()
    return RGR_original, hight, width, channel

###############################################################################
# B.3 Raw現像と周辺減光補正---------------------------------------------------
###############################################################################

#周辺減光補正係数の読み込み
FlatCofficient = np.load(InputData_2)
FlatCofficient = FlatCofficient["arr_0"]

for ID in tqdm(photolist):
    Path=sorted(glob.glob(os.path.join(InputData_3+ID+"*.DNG"))) #画像の読み込み
    for f in Path:
```

```
    #RAW
    img = rawpy.imread(f)
    RGR_original, hight, width, channel = RawPostprocess(img)
img_R = np.round(np.array(RGR_original[:,:,0]) / FlatCofficient[:,:,0])
img_G = np.round(np.array(RGR_original[:,:,1]) / FlatCofficient[:,:,1])
img_B = np.round(np.array(RGR_original[:,:,2]) / FlatCofficient[:,:,2])
img_bgr = cv2.merge((img_R, img_G, img_B))
img_bgr = np.array(img_bgr,dtype='uint32')

tif.imsave(OutputData_1+ID+".tif", img_bgr)
```

## 9.3 DSM の作成，オルソフォト変換, オクルージョン処理のプログラム

プログラム名 : 1_V_XY-imageCreate-P1.py (3.6 節，4.1 節，4.2 節)

本プログラムは，DSM の作成，オルソフォト変換，オクルージョン処理を実行するプログラムである．入力データは，3 次元点群データ，カメラ位置姿勢除法データ，カメラパラメータデータ，周辺減光補正後の原画像である．出力データは，DSM，オクルージョン処理後のオルソフォトである．以下にプログラムのソースコードを記す.

Listing 3 1_V_XY-imageCreate-P1.py

```
import os
import cv2
import sys
import math
import glob
import xmltodict
import numpy as np
import pandas as pd
from tqdm import tqdm
from skimage import io
from osgeo import osr, gdal

cwd = os.getcwd()
path  = os.path.splitext(cwd[:len(cwd)-15])[0]
#使用HDD path = /Volumes/Capotasto2


#############################################################################################
# 1.1 解析対象範囲----------------------------------------------------------------------------
#############################################################################################

Date    = "20220720"
Time    = "1453"
Camera  = "P1"
ImageType      = sys.argv[3]

#ボクセルの対象範囲
GCPx,GCPy, GCPz= 20205,71707,180      #対象範囲の中心座標
xmin_0 = GCPx-150
ymin_0 = GCPy-150
zmin_0 = GCPz-70
xmax_0 = GCPx+150
ymax_0 = GCPy+150
zmax_0 = GCPz+70
Interval = 0.2
#ボクセルの範囲
xmin_1,xmax_1 = xmin_0-1 , xmax_0+1
ymin_1,ymax_1 = ymin_0-1 , ymax_0+1
zmin_1,zmax_1 = zmin_0-1 , zmax_0+1
```

```
#1.3 次元点群データ　 3
InputData_1 = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-RandomPointCloud.txt"
#1.5 カメラ位置姿勢情報データ
InputData_2 = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-CameraOmegaPhiKapp1.txt"
#1.5 カメラパラメータデータ
InputData_3 = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-CameraXML.xml"
#1.5 原画像
InputData_4 =path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/tif/"+ImageType

#1.2 投影面テキスト　 ()
OutputData_1 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/check/XYmesh.txt"
#1.2 投影面　 (numpy)
OutputData_2 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/XYmesh_XYZ.npz"
#1.3 ボクセルデータ
OutputData_3 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/check/cutvoxel"+Date+".txt"
#1.3 ボクセルデータ
OutputData_4 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/"+Date+"voxel_XYZD.npz"
#1.4　 DSM
OutputData_5 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/check/XY_intersection"+Date+".txt"
#1.4　 DSM
OutputData_6 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/"+Date+"intersection_XYZ"
#1.5 オルソフォト
OutputData_7 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/tif/"+ImageType

#############################################################################################
# 1.2 投影面の作成(XYmesh)----------------------------------------------------------------
#############################################################################################

#格子点の間隔絶対座標()
s=0.2

#対角線の長さ
a=xmax_0-xmin_0
b=ymax_0-ymin_0
c=zmax_0-zmin_0
xsize1 = int(round(a/Interval))+1
ysize1 = int(round(b/Interval))+1
zsize1 = int(round(c/Interval))+1
```

```
xmesh = np.zeros((ysize1,xsize1))
ymesh = np.zeros((ysize1,xsize1))
zmesh = np.zeros((ysize1,xsize1))

tmp0 = np.array([])
tmp_list0 = tmp0.tolist()
k=0
with tqdm(total=xsize1) as pbar:
    while k < xsize1:
        pbar.update(1)
        X=xmin_0+(k*Interval)
        j=0
        Y=ymax_0
        while j < ysize1:
            #print(i,k)
            xmesh[j][k] = X
            ymesh[j][k] = Y
            zmesh[j][k] = zmax_0
            arr=[]
            arr.insert(0,X)
            arr.insert(1,Y)
            arr.insert(2,zmax_0)
            tmp_list0.append(arr)
            #print(i,Y)
            j=j+1
            Y=ymax_0-(j*Interval)
        k=k+1
tmp0=np.array(tmp_list0)

np.savetxt(OutputData_1,tmp0,fmt=["%.7f", "%.7f", "%.7f"])
np.savez(OutputData_2, xmesh,ymesh,zmesh)

#############################################################################################
#  ボクセルに点群データの属性値を付与 1.3----------------------------------------------
#############################################################################################

#観測対象エリアの空ボクセルを作成
xsize2 = int(round((xmax_1-xmin_1)/Interval))
ysize2 = int(round((ymax_1-ymin_1)/Interval))
zsize2 = int(round((zmax_1-zmin_1)/Interval))

#ボクセルの中心の地図座標間隔なのでボクセルの中心は刻み(0.20.1)
xcenter = xmin_1 + 0.5 * Interval
ycenter = ymax_1 - 0.5 * Interval
zcenter = zmin_1 + 0.5 * Interval

#ボクセルRGBN
Xvoxel = np.zeros((zsize2,ysize2,xsize2))
Yvoxel = np.zeros((zsize2,ysize2,xsize2))
Zvoxel = np.zeros((zsize2,ysize2,xsize2))
```

```
D = np.zeros((zsize2,ysize2,xsize2))

tmp1=np.array([])
tmp_list1=tmp1.tolist()

#点群データを１行ずつ読み込んで,で作ったボクセル配列に属性値として追加させるnumpy
with open(InputData_1,'r') as f:
    #header = next(f)
    for line in tqdm(f):
        xnum = int(round((float(line.split()[0])-xmin_1)/Interval,4))
        ynum = int(round((ymax_1-float(line.split()[1]))/Interval,4))
        znum = int(round((float(line.split()[2])-zmin_1)/Interval,4))
        if 0 <= xnum < xsize2 and 0 <= ynum < ysize2 and 0 <= znum < zsize2:
            D[znum][ynum][xnum] = 1 #作成するボクセルないの時１
            Xvoxel[znum][ynum][xnum] = xcenter+xnum*Interval
            Yvoxel[znum][ynum][xnum] = ycenter-ynum*Interval
            Zvoxel[znum][ynum][xnum] = zcenter+znum*Interval
            arr=[]
            arr.insert(0,xcenter+xnum*Interval)
            arr.insert(1,ycenter-ynum*Interval)
            arr.insert(2,zcenter+znum*Interval)
            tmp_list1.append(arr)
tmp1=np.array(tmp_list1)
np.savetxt(OutputData_3,tmp1,fmt=["%.7f","%.7f","%.7f"])
np.savez(OutputData_4, xcenter,ycenter,zcenter,D)

#############################################################################################
# 1.4 DSMの作成-------------------------------------------------------------------------------
#############################################################################################

mesh =np.load(OutputData_2)
X=mesh["arr_0"]
Y=mesh["arr_1"]
Z=mesh["arr_2"]
rowsize,colsize=np.shape(X)
x0=X[0][0]
y0=Y[0][0]
z0=Z[0][0]
x1=X[0][colsize-1]
y1=Y[0][colsize-1]
z1=Z[0][colsize-1]
x2=X[rowsize-1][colsize-1]
y2=Y[rowsize-1][colsize-1]
z2=Z[rowsize-1][colsize-1]
v1x=x1-x0
v1y=y1-y0
v1z=z1-z0
v2x=x2-x0
v2y=y2-y0
v2z=z2-z0
```

```
A=v2y*v1z-v1y*v2z
B=v2z*v1x-v1z*v2x
C=v2x*v1y-v1x*v2y
#法線ベクトル
r=A**2+B**2+C**2
R=math.sqrt(r)
tx =  (A/R) * Interval
ty = -(B/R) * Interval
tz = -(C/R) * Interval
voxelrange=[]
irange,jrange,krange=D.shape

n=1
ysize3,xsize3=np.shape(X)
N=ysize3*xsize3
xintersection = np.zeros((ysize3,xsize3))
yintersection = np.zeros((ysize3,xsize3))
zintersection = np.zeros((ysize3,xsize3))
tmp2=np.array([])
tmp_list2=tmp2.tolist()
VOXEL= np.load(InputData_2)
Xvoxel=VOXEL["arr_0"]
Yvoxel=VOXEL["arr_1"]
Zvoxel=VOXEL["arr_2"]
D=VOXEL["arr_3"]

k=0
kc=0
j=0
with tqdm(total=N) as pbar:
    while n <= N:
        pbar.update(1)
        if j == ysize3:
            kc=0
            k=k+1
            j=0

        xv=X[j][k]
        yv=Y[j][k]
        zv=Z[j][k]

        count=[]
        if tx != 0:
            x1d=(xmin_1-xv)/tx
            x2d=(xmax_1-xv)/tx
            count.append(x1d)
            count.append(x2d)

        if ty != 0:
```

```
            y1d=(ymin_1-yv)/ty
            y2d=(ymax_1-yv)/ty
            count.append(y1d)
            count.append(y2d)

        if tz != 0:
            z1d=(zmin_1-zv)/tz
            z2d=(zmax_1-zv)/tz
            count.append(z1d)
            count.append(z2d)

        count=np.max(count)

        C=0
        while True:
            if  C > count:
                kc=kc+1
                j=j+1
                break
            xv = xv + tx
            yv = yv + ty
            zv = zv + tz
            C=C+1

            if xmin_1 <= xv <= xmax_1 and ymin_1 <= yv <= ymax_1 and zmin_1 <= zv <= zmax_1:
                K = int(round((xv - xmin_1) / Interval))
                J = int(round((ymax_1 - yv) / Interval))
                I = int(round((zv - zmin_1) / Interval))
                if 0 <= I < irange and 0<= J < jrange and 0<= K < krange:
                    if D[I][J][K] == 1:
                        xintersection[j][k]=xv
                        yintersection[j][k]=yv
                        zintersection[j][k]=zv+Interval
                        print(J,K,j,k)
                        arr=[]
                        arr.insert(0,xv)
                        arr.insert(1,yv)
                        arr.insert(2,zv+Interval)
                        tmp_list2.append(arr)
                        kc=kc+1
                        j=j+1
                        break
        n=n+1

tmp2=np.array(tmp_list2)
np.savetxt(OutputData_5,tmp2)
np.savez(OutputData_6, xintersection,yintersection,zintersection)
```

```
#############################################################################################
#  オルソフォト変換 1.5----------------------------------------------------------------------
#############################################################################################

cameradata=pd.read_csv(InputData_2,delim_whitespace=True,
                       names=("photoID","x","y","z","Omega","Phi","Kappa",
                              "r11","r12","r13","r21","r22","r23","r31","r32","r33"),
                              skiprows=2)
CAMERADATA=cameradata
photolist=np.array(cameradata["photoID"])
intersection= np.load(OutputData_5)
XI=intersection["arr_0"]
YI=intersection["arr_1"]
ZI=intersection["arr_2"]

with open(InputData_3, encoding='utf-8') as fp: #読み込み xml
    xml_data = fp.read()
    # xml → dict
    dict_data = xmltodict.parse(xml_data)

def changeXYZtoIJK (a,b,c):
    xnum =int(round((a-xmin_1)/Interval))
    ynum =int(round((ymax_1-b)/Interval))
    znum =int(round((c-zmin_1)/Interval))
    return xnum,ynum,znum

def InvisibleJudgment(X1,Y1,Z1,X2,Y2,Z2): #可視・不可視判定オクルージョン処理()
    Judgment = "Visible"
    GEO_P1      = X1,Y1,Z1                  #の地上座標UAV
    GEO_P2      = X2,Y2,Z2                  #判定箇所の地上座標
    num = 1
    t   = 1
    #判定箇所からまでの２次元距離UAV
    num2 = np.sqrt(((GEO_P2[0]-GEO_P1[0])**2+(GEO_P2[1]-GEO_P1[1])**2))
    num3 = (1/num2)/5 #0.2の精度にするためm(/5)

    while num < int(num2):

        t =  t - num3
        #直線の方程式
        X_cross  = GEO_P1[0] + t*(GEO_P2[0]-GEO_P1[0])
        Y_cross  = GEO_P1[1] + t*(GEO_P2[1]-GEO_P1[1])
        Z_cross  = GEO_P1[2] + t*(GEO_P2[2]-GEO_P1[2])
        if (X_cross - xmin_0)/0.2 > 1500 or (ymax_0 - Y_cross)/0.2 > 1500:
            pass
        elif Z_cross <= ZI[int((ymax_0-Y_cross)/0.2),int((X_cross-xmin_0)/0.2)]:
            Judgment = "Invisible"
            break
        num += 1
    return Judgment
```

```
for ID in photolist:
    print(ID)
    data = CAMERADATA
    data=data[data["photoID"]==ID]
    idx=list(data.index)
    Index=sum(idx)
    params=dict_data["document"]["chunk"]["sensors"]["sensor"]["calibration"]
    try:
        f=float(params["f"])
    except TypeError:
        params=dict_data["document"]["chunk"]["sensors"]["sensor"]["calibration"][1]
    f=float(params["f"])
    cx=float(params["cx"])
    cy=float(params["cy"])
    width=float(params["resolution"]["@width"])
    height=float(params["resolution"]["@height"])

    C = f * 0.0043948613649283609/1000
    W = width * 0.0043948613649283609/2000
    H = height * 0.0043948613649283609/2000

    x = float(data["x"])
    y = float(data["y"])
    z = float(data["z"])
    O = data["Omega"]
    P = data["Phi"]
    K = data["Kappa"]

    #度数から弧度
    rad_O = math.radians(float(O))
    rad_P = math.radians(float(P))
    rad_K = math.radians(float(K))

    #三角関数の計算
    sin_O = math.sin(-float(rad_O))
    cos_O = math.cos(-float(rad_O))
    sin_P = math.sin(-float(rad_P))
    cos_P = math.cos(-float(rad_P))
    sin_K = math.sin(-float(rad_K))
    cos_K = math.cos(-float(rad_K))
    irange,jrange,krange=D.shape

    n=1
    tmp = np.array([])
    tmp_list = tmp.tolist()
    arr2 = np.array([[cos_K,-sin_K,0], [sin_K,cos_K,0], [0,0,1]]) #kappa
    arr3 = np.array([[cos_P,0,sin_P], [0,1,0], [-sin_P,0,cos_P]]) #phi
    arr4 = np.array([[1,0,0], [0,cos_O,-sin_O], [0,sin_O,cos_O]]) #omega
    arr5 = np.dot(arr2, arr3)
```

```
arr6 = np.dot(arr5, arr4)
N=ysize1*xsize1
ki=0
kc=0
ji=0
u = np.zeros((ysize1,xsize1))
v = np.zeros((ysize1,xsize1))
with tqdm(total=N) as pbar:
    while n<=N:
        n=n+1
        pbar.update(1)
        if ji == ysize1:
            kc=0
            ki=ki+1
            ji=0
        point_mesh_x = XI[ji][ki]
        point_mesh_y = YI[ji][ki]
        point_mesh_z = ZI[ji][ki]

        if point_mesh_x == 0 and point_mesh_y == 0 and point_mesh_z == 0:
            kc=kc+1
            ji=ji+1
        else:
            arr1 = np.array([[point_mesh_x-float(x)], [point_mesh_y-float(y)],
                             [point_mesh_z-float(z)]]) #終点始点-
            arr7 = np.dot(arr6, arr1)
            U = -C * (arr7[0,0] / arr7[2,0]) #共線条件式
            V = -C * (arr7[1,0] / arr7[2,0])
            u[ji][ki] = float(U)
            v[ji][ki] = float(V)
            kc=kc+1
            ji=ji+1

Path=sorted(glob.glob(os.path.join(InputData_4+"/"+ID+"*.tif"))) #画像の読み込み
for f in Path:
    #JPEG
    img = io.imread(f)

img_R = np.array(img[:,:,0])
img_G = np.array(img[:,:,1])
img_B = np.array(img[:,:,2])

h, w = img_G.shape
pixel_width =  0.0043948613649283609 / 1000
pixel_height = 0.0043948613649283609 / 1000
#仮想投影面の範囲点の個数()
kasou_row , kasou_col = np.shape(u) #1501,1501

#仮想画像それぞれの２次元配列RGB
virtual_r_image = np.zeros((kasou_row,kasou_col), np.uint32) ##1501,1501
```

```
virtual_g_image = np.zeros((kasou_row,kasou_col), np.uint32)
virtual_b_image = np.zeros((kasou_row,kasou_col), np.uint32)

C=0
n=1
k=0
kc=0
j=0

GEO_P1      = x,y,z                                          #の地上座標UAV
IMG_P1      = (x - xmin_0)/0.2 , (ymax_0 - y)/0.2 , z/0.2    #のボクセル番地UAV

#投影面の中心番地
centerj = int(kasou_col/2)
centerk = int(kasou_row/2)
N = kasou_col * kasou_row
#print(N)
for j in tqdm(range(kasou_row)):
    for k in range(kasou_col):
        if j == rowsize:
            kc=0
            k=k+1
            j=0
        U = u[j][k]
        V = v[j][k]
        col_1 = U / pixel_width
        row_1 = V / pixel_height

        col_2 = col_1 + w/2
        row_2 = h/2 - row_1

        col_3 = col_2 + cx #±
        row_3 = row_2 + cy #±

        Col = int(col_3)
        Row = int(row_3)

        if 0 < Col < w and 0 < Row < h :
            Col_min,Col_max = np.array(max(0, Col-9)),np.array(min(w,Col+10))
            Row_min,Row_max = np.array(max(0, Row-9)),np.array(min(h,Row+10))

            r_sum = np.sum(img_R[Row_min:Row_max,Col_min:Col_max])
            r_n   = np.count_nonzero(img_R[Row_min:Row_max,Col_min:Col_max])
            g_sum = np.sum(img_G[Row_min:Row_max,Col_min:Col_max])
            g_n   = np.count_nonzero(img_G[Row_min:Row_max,Col_min:Col_max])
            b_sum = np.sum(img_B[Row_min:Row_max,Col_min:Col_max])
            b_    = np.count_nonzero(img_B[Row_min:Row_max,Col_min:Col_max])


            r = np.array(round(r_sum / r_n,0))
```

```
                g = np.array(round(g_sum / g_n,0))
                b = np.array(round(b_sum / b_n,0))

                virtual_r_image[j][k] = r
                virtual_g_image[j][k] = g
                virtual_b_image[j][k] = b

                if U == 0 and V == 0:
                        if j == centerj and k == centerk:
                            pass
                        else:
                            virtual_r_image[j][k] = 0
                            virtual_g_image[j][k] = 0
                            virtual_b_image[j][k] = 0

                #オクルージョン処理
                if virtual_r_image[j][k] == 0 and \
                    virtual_g_image[j][k] == 0 and \
                        virtual_b_image[j][k] == 0 :
                    pass

                else:
                    GEO_P2      = XI[j,k],YI[j,k],ZI[j,k]              #判定する地上座標
                    Judgment = InvisibleJudgment(GEO_P1[0],
                                                 GEO_P1[1],GEO_P1[2],
                                                 GEO_P2[0],GEO_P2[1],

                                                 GEO_P2[2])

                    if Judgment == "Invisible": #不可視判定
                        virtual_r_image[j][k] = 0
                        virtual_g_image[j][k] = 0
                        virtual_b_image[j][k] = 0


            else:
                pass

dtype = gdal.GDT_UInt32
band = 3 #バンド数
#空の出力ファイル
output = gdal.GetDriverByName('GTiff').Create(OutputData_7+"/XY_"+ID+".tif",
                                              xsize1, ysize1, band, dtype,
                                                options = ['PHOTOMETRIC=RGB'])

output.SetGeoTransform((xmin_0,Interval,0,ymax_0,0,-Interval)) #座標系指定
srs = osr.SpatialReference() #空間参照情報
srs.ImportFromEPSG(2446)
output.SetProjection(srs.ExportToWkt()) #空間情報を結合
output.GetRasterBand(1).WriteArray(virtual_r_image)
```

```
output.GetRasterBand(2).WriteArray(virtual_g_image)
output.GetRasterBand(3).WriteArray(virtual_b_image)
output.FlushCache()          #ディスクに書き出し
output = None
```

## 9.4 カメラ方向ベクトルと太陽方向ベクトル，DN 値の平均を算出するプログラム

プログラム名 : 2_Calculate_angle_and_color-P1.py (4.3 節)

本プログラムは，カメラ方向ベクトルと太陽方向ベクトル，緑バンドの DN 値の平均を算出するプログラムである．入力データは，DSM，カメラ位置姿勢情報データ，オルソフォトである．出力データは，カメラ方向ベクトル，太陽方向ベクトル，緑バンドの DN 値の平均を記録した csv ファイルである．以下にプログラムのソースコードを記す.

Listing 4 2_Calculate_angle_and_color-P1.py

```
import os
import csv
import sys
import math
import ephem
import glob
import numpy as np
import pandas as pd
from tqdm import tqdm
from skimage import io
from pyproj import Transformer

cwd = os.getcwd()
path  = os.path.splitext(cwd[:len(cwd)-15])[0]
#使用HDD path = /Volumes/Capotasto2

#############################################################################################
# 2.1 解析対象範囲----------------------------------------------------------------------------
#############################################################################################

#ボクセルの対象範囲
GCPx,GCPy, GCPz= 20205,71707,180
x0 = GCPx-150
y0 = GCPy-150
z0 = GCPz-70
x1 = x0 + 300
y1 = y0 + 300
z1 = z0 + 140

#ボクセルの間隔
voxel_interval = 0.2
Interval = voxel_interval

#ボクセルの範囲
xmin,xmax = x0 , x1
ymin,ymax = y0 , y1
zmin,zmax = z0 , z1
```

```
#############################################################################################
# 2.2 対象地点の入力　画像座標の計算----------------------------------------------------------
#############################################################################################
Name = str(sys.argv[1])

#対象の地上座標を記録したcsv
TreesList = pd.read_csv(path+"/2023EW/BRDF/対象樹木BRDFver0105.csv", index_col=0)
print(TreesList.loc[Name])
input_data_X,input_data_Y = float(TreesList.loc[Name]['X']), float(TreesList.loc[Name]['Y'])

Xnum = int(round((input_data_X-xmin)/Interval,4))
Ynum = int(round((ymax-input_data_Y)/Interval,4))
col,row = Xnum,Ynum
pix = col,row
pixel=str(pix)

photo_num = 422

Ave_range = 60
ImageType      = "Shading_Correction_Image"
Date           = "20220720"
Time           = "1453"
Camera         = "P1"

#2.3, 2.5   DSM
InputData_1  = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
               +"/result/"+Date+"intersection_XYZ.npz"
#2.3, 2.4 カメラ位置姿勢情報データ
InputData_2  = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
               +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-CameraOmegaPhiKappa.txt"
#2.5 オルソフォト
InputData_3  = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
               +"/result/tif/"

#2.6 カメラ方向ベクトル, 太陽方向ベクトル, 値の平均   DN
OutputData_1 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+"/result/csv/"\
               +ImageType+"/Angle_and_color/Angle_and_color-"\
               +Name+"(Row="+str(row)+",Col="+str(col)+").csv"

#############################################################################################
# 2.3 解析地点日時→太陽高度方位角,,----------------------------------------------------------
#############################################################################################

def Sun_Location(Lon,Lat,Day,Hour,Minutes,Second):
    kochi = ephem.Observer()
    kochi.lat, kochi.lon = Lat,Lon
    kochi.date = Day+ '/' + Hour+':'+Minutes+':'+Second
    kochi.date -= ephem.hour * 9

    lt = ephem.localtime(kochi.date)
```

```
    sun = ephem.Sun(kochi)
    sun.compute(kochi)

    rhl = sun.hlon - ephem.degrees('180:00:00')
    dhl = ephem.degrees(rhl).norm
    sun_alt=float(str(sun.alt).split(':')[0])+ \
        float(str(sun.alt).split(':')[1])/60+ \
            float(str(sun.alt).split(':')[2])/3600
    sun_az=float(str(sun.az).split(':')[0])+ \
        float(str(sun.az).split(':')[1])/60+ \
            float(str(sun.az).split(':')[2])/3600
    return sun_alt,sun_az

#の読み込み DATA
POINT = np.load(InputData_1)
z=POINT["arr_2"]
x=POINT["arr_0"]
y=POINT["arr_1"]
input_data_X = x[row][col]
input_data_Y = y[row][col]
input_data_Z = z[row][col]

cameradata=pd.read_csv(InputData_2,delim_whitespace=True,
                       names=("photoID","x","y","z","Omega","Phi","Kappa",
                              "r11","r12","r13","r21","r22","r23","r31","r32","r33"),
                              skiprows=2)
photolist=np.array(cameradata["photoID"])
photolist=photolist.tolist()
photolist = photolist[0:photo_num]
cameradata = cameradata[0:photo_num]

#原画像の撮影時刻を調べる
tmp_time = np.array([])
tmplist_time = tmp_time.tolist()
Scene_line = sum(1 for line in photolist)

for n in photolist:
    img = n
    Hour    = n[12:14]
    Minutes = n[14:16]
    Seconds = n[16:18]
    day     = [n[4:8],"/",n[8:10],"/",n[10:12]]
    Day = ""
    for val in day:
        if val != n[10:12]:
            Day += val
        else:
            Day += val
    arr=[]
    arr.insert(0,n)
```

```
        arr.insert(1,Day)
        arr.insert(2,Hour)
        arr.insert(3,Minutes)
        arr.insert(4,Seconds)
        tmplist_time.append(arr)

tmp_time=np.array(tmplist_time)

#############################################################################################
# 2.4 カメラ方向ベクトルと太陽方向ベクトルの算出----------------------------------------
#############################################################################################

tmp_theta = np.array([])
tmplist_theta = tmp_theta.tolist()
transformer = Transformer.from_crs("epsg:2446", "epsg:4326")
Lat_data,Lon_data= transformer.transform(input_data_Y,input_data_X)

#各原画像の太陽高度、太陽方位角を求める（太陽高度:、太陽方位角sun_alt:）sun_az
tmp_time_line = sum(1 for line in tmplist_time)
for n in list(range(0,tmp_time_line)):
    sun_alt,sun_az = Sun_Location(str(Lon_data),str(Lat_data),tmp_time[n][1],
                                  tmp_time[n][2],tmp_time[n][3],tmp_time[n][4])
    rad_sun_alt = math.radians(sun_alt)
    rad_sun_az = math.radians(sun_az)

    #から太陽pixel入射光への単位ベクトル（()sun_x,sun_y,）sun_z
    sun_x =math.sin(rad_sun_az)*math.cos(rad_sun_alt)
    sun_y =math.cos(rad_sun_az)*math.cos(rad_sun_alt)
    sun_z =math.sin(rad_sun_alt)

    cameradata=pd.read_csv(InputData_2,delim_whitespace=True,
                           names=("photoID","x","y","z","Omega","Phi","Kappa",
                                  "r11","r12","r13","r21","r22","r23","r31","r32","r33"),
                                  skiprows=2)

    CAMERADATA = cameradata["photoID"] == tmp_time[n][0]
    CAMERADATA = cameradata[CAMERADATA]

    uAV_X = CAMERADATA["x"]
    uAV_Y = CAMERADATA["y"]
    uAV_Z = CAMERADATA["z"]

    #型のデータから値だけを抽出Series
    UAV_X = uAV_X.iloc[-1]
    UAV_Y = uAV_Y.iloc[-1]
    UAV_Z = uAV_Z.iloc[-1]

    #からへのベクトルpixelUAV(uav_x_unit,uav_y_unit,uav_z_unit)
    uav_x =UAV_X-input_data_X
    uav_y =UAV_Y-input_data_Y
```

```
    uav_z =UAV_Z-input_data_Z
    uav_vector = math.sqrt((uav_x**2)+(uav_y**2)+(uav_z**2))#ベクトルの長さ

    uav_x_unit =uav_x/uav_vector
    uav_y_unit =uav_y/uav_vector
    uav_z_unit =uav_z/uav_vector
    arr=[]
    arr.insert(0,sun_x)
    arr.insert(1,sun_y)
    arr.insert(2,sun_z)
    arr.insert(3,uav_x_unit)
    arr.insert(4,uav_y_unit)
    arr.insert(5,uav_z_unit)
    tmplist_theta.append(arr)

tmp_theta=np.array(tmplist_theta)

#############################################################################################
# 2.5 解析地点×33(60cm x 60cm 相当の)値の平均値の導出RGB----------------------------------
#############################################################################################

#解析地点周辺×の 33値の平均値の導出RGB
intersection= np.load(InputData_1)
XI=intersection["arr_0"]
YI=intersection["arr_1"]
ZI=intersection["arr_2"]
tmp_color = np.array([])
tmplist_color = tmp_color.tolist()

tmp_time_line = sum(1 for line in tmplist_time)
for n in tqdm(list(range(0,photo_num))):

    img = io.imread(InputData_3+"/"+ImageType+"/XY_"+photolist[n]+".tif")

    r_band = img[:,:,0]
    g_band = img[:,:,1]
    b_band = img[:,:,2]
    #それぞれRGB (Ave_range/20) x (Ave_range/20) の平均を求める------------------------
    r_sum = np.sum(img[row-1:row+2,col-1:col+2,0])
    r_n   = np.count_nonzero(img[row-1:row+2,col-1:col+2,0])
    g_sum = np.sum(img[row-1:row+2,col-1:col+2,1])
    g_n   = np.count_nonzero(img[row-1:row+2,col-1:col+2,1])
    b_sum = np.sum(img[row-1:row+2,col-1:col+2,2])
    b_n   = np.count_nonzero(img[row-1:row+2,col-1:col+2,2])

    if b_n == 0 or g_n == 0 or r_n == 0 :
        b_mean     = float("nan")
        g_mean     = float("nan")
        r_mean     = float("nan")
    else:
```

```
            b_mean      = np.array(round(b_sum / b_n,0))
            g_mean      = np.array(round(g_sum / g_n,0))
            r_mean      = np.array(round(r_sum / r_n,0))

        g_ravel = np.ravel(img[row,col,1])

        arr=[]
        arr.insert(0,r_mean)
        arr.insert(1,g_mean)
        arr.insert(2,b_mean)
        tmplist_color.append(arr)

tmp_color=np.array(tmplist_color)
tmp_photolist = (np.array(photolist)).reshape((photo_num,1))
print(len(tmp_photolist))
TMP = np.hstack([tmp_photolist,tmp_theta,tmp_color])
TMP=np.array(TMP)
df=pd.DataFrame(TMP)

#色値のない行番号を取得
delindex = df[df[8] == 'nan'].index
df.drop(delindex,inplace=True)

#############################################################################################
# 2.6 csvファイルの書き出し------------------------------------------------------------
#############################################################################################

csv_file = glob.glob(os.path.join(OutputData_1))
print(csv_file)
if csv_file==[]:
    header = ["ID", "sun_x","sun_y", "sun_z","uav_x", "uav_y","uav_z", "r","g","b"]

    with open(OutputData_1, "w") as f:
        writer = csv.writer(f)
        writer.writerow(header)
    f.close()

    df = np.array(df)
    for i in tqdm(range(len(df))):
        df_ID     = str(df[i][0])
        df_sun_x  = float(df[i][1])
        df_sun_y  = float(df[i][2])
        df_sun_z  = float(df[i][3])
        df_uav_x  = float(df[i][4])
        df_uav_y  = float(df[i][5])
        df_uav_z  = float(df[i][6])
        df_r      = float(df[i][7])
        df_g      = float(df[i][8])
        df_b      = float(df[i][9])
```

```
with open(OutputData_1,"a") as f:
    print(OutputData_1)
    writer = csv.writer(f)
    writer.writerow([df_ID,df_sun_x,df_sun_y,df_sun_z,
                     df_uav_x,df_uav_y,df_uav_z,df_r,df_g,df_b])
```

## 9.5 位相角 $\theta$ の算出，BRDF 可視化のプログラム

プログラム名 : 3_Make_graph_BRDF-P1.py (4.3 節，4.4 節)

本プログラムは，位相角 $\theta$ の算出，BRDF の可視化を実行するプログラムである．入力データは，カメラ方向ベクトルと太陽方向ベクトル，緑バンドの DN 平均値を記録した csv ファイル，カメラ位置姿勢情報データ，DSM, カメラパラメータデータ，周辺減光補正後の原画像である．出力データは，対象樹木の原画像座標，解析対象面の平均値，最小値，最大値，標準偏差，影のピクセ数を記録した csv ファイル，BRDF を可視化した図，BRDF の相関係数，傾き，切片を記録した csv ファイル，影割合変化図である．以下にプログラムのソースコードを記す．

Listing 5 3_Make_graph_BRDF-P1.py

```
import os
import csv
import sys
import math
import glob
import xmltodict
import numpy as np
import pandas as pd
from tqdm import tqdm
from skimage import io
import matplotlib.cm as cm
import matplotlib.pyplot as plt
import matplotlib.patches as pat
import matplotlib.ticker as ticker
import mpl_toolkits.mplot3d.art3d as art3d
from matplotlib.ticker import MultipleLocator, FormatStrFormatter

plt.rcParams['font.family'] = 'Hiragino Sans'
np.set_printoptions(precision=15, floatmode='maxprec')

cwd = os.getcwd()
path  = os.path.splitext(cwd[:len(cwd)-15])[0]
#使用HDD path = /Volumes/Capotasto2

##############################################################################
# 3.1 対象地点の入力　画像座標の検索----------------------------------------------
##############################################################################

Name = str(sys.argv[1])
#対象の地上座標を記録したcsv
TreesList = pd.read_csv(path+"/2023EW/BRDF/対象樹木BRDFver0105.csv", index_col=0)
input_data_X,input_data_Y = float(TreesList.loc[Name]['X']), float(TreesList.loc[Name]['Y'])

#ボクセルの対象範囲
GCPx,GCPy, GCPz= 20205,71707,180
x0 = GCPx-150
```

```
y0 = GCPy-150
z0 = GCPz-70
x1 = x0 + 300
y1 = y0 + 300
z1 = z0 + 140

#ボクセルの間隔
voxel_interval = 0.2
Interval = voxel_interval

#ボクセルの範囲
xmin,xmax = x0 , x1
ymin,ymax = y0 , y1
zmin,zmax = z0 , z1

#対角線の長さ
a=xmax-xmin
b=ymax-ymin
c=zmax-zmin

xsize1 = int(round(a/Interval))+1
ysize1 = int(round(b/Interval))+1
zsize1 = int(round(c/Interval))+1

#ボクセルの範囲
xmin_1,xmax_1 = xmin-1 , xmax+1
ymin_1,ymax_1 = ymin-1 , ymax+1
zmin_1,zmax_1 = zmin-1 , zmax+1

#観測対象エリアの空ボクセルを作成
xsize2 = int(round((xmax_1-xmin_1)/Interval))
ysize2 = int(round((ymax_1-ymin_1)/Interval))
zsize2 = int(round((zmax_1-zmin_1)/Interval))
D = np.zeros((zsize2,ysize2,xsize2))

Xnum = int(round((input_data_X-xmin)/Interval,4))
Ynum = int(round((ymax-input_data_Y)/Interval,4))
col,row = Xnum,Ynum
pix = col,row
pixel=str(pix)

Ave_range = 60
ImageType      = "Shading_Correction_Image"
Date     = "20220720"
Time     = "1453"
Camera   = "P1"

#############################################################################################
# 3.2 対象地点の入力　画像座標の検索----------------------------------------------------
#############################################################################################
```

```
#3.1 太陽方向ベクトル，カメラ方向ベクトル，緑バンド平均値　 DN
InputData_1 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+"/result/csv/"+\
                +ImageType+"/Angle_and_color/Angle_and_color-"+\
                +Name+"(Row="+str(row)+",Col="+str(col)+").csv"
#3.1 カメラ位置姿勢情報データ
InputData_2 = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-CameraOmegaPhiKappa.txt"
#3.4　 DSM
InputData_3 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/"+Date+"intersection_XYZ.npz"
#3.4 カメラパラメータデータ
InputData_4 = path+"/2023EW/Data/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/Saoka-"+Date+"-"+Time+"-"+Camera+"-CameraXML.xml"
#3.4 原画像
InputData_5 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/tif/"+ImageType+"/"

#3.4 対象樹木の原画像座標
OutputData_1 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/csv/ImageCoordinates/ImageCoordinate_"+\
                +Name+"(Row="+str(row)+",Col="+str(col)+").csv"
#3.4 解析対象面の平均値，最小値，最大値，標準偏差，影のピクセル数
OutputData_2 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/csv/RGB/RGB_"+Name+"(Row="+str(row)+",Col="+str(col)+").csv"
#3.6 の可視化　 BRDF全(pass)
OutputData_3 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/graph/"+ImageType+"/BRDF/Old-BRDF-"+Name+".jpg"
#3.6 の可視化　 BRDF principale plane ののみpass緑バンド(値平均DN)
OutputData_4 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/graph/"+ImageType+"/BRDF/Old2-BRDF-"+Name+".jpg"
#3.6 の可視化　 BRDF principale plane ののみpass（(G/R+G+）値平均BDN)
OutputData_5 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/graph/"+ImageType+"/BRDF/Normalize_Old2-BRDF-"+Name+".jpg"
#3.6 の相関，傾き，切片を記録するファイル　 BRDFcsv
OutputData_6 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/csv/Comparison/BRDF_infomation_ver0105.csv"
#3.6 影割合変化図
OutputData_7 = path+"/2023EW/BRDF/Saoka-"+Date+"-"+Time+"-"+Camera+\
                +"/result/graph/"+ImageType+"/BRDF/Old2-Shadow-"+Name+".jpg"
A_C_Vevtor = pd.read_csv(InputData_1)

photolist = np.array(A_C_Vevtor["ID"])
sun_x = A_C_Vevtor["sun_x"]
sun_y = A_C_Vevtor["sun_y"]
sun_z = A_C_Vevtor["sun_z"]
uav_x = A_C_Vevtor["uav_x"]
uav_y = A_C_Vevtor["uav_y"]
uav_z = A_C_Vevtor["uav_z"]
ave_r = A_C_Vevtor["r"]
```

```
ave_g = A_C_Vevtor["g"]
ave_b = A_C_Vevtor["b"]

Cameradata=pd.read_csv(InputData_2,delim_whitespace=True,
                       names=("x","y","z","Omega","Phi","Kappa",
                              "r11","r12","r13","r21","r22","r23","r31","r32","r33")
                              ,skiprows=2, index_col=0)
UAV_X = Cameradata["x"]
UAV_Y = Cameradata["y"]
UAV_Z = Cameradata["z"]

#############################################################################################
# 3.3 位相角θ, 高度角の計算UAV-------------------------------------------------------------
#############################################################################################

Theta_List = []       #位相角θ
UAV_alt_List = []     #高度角UAV
Sun_alt_List = []

for i in range(len(A_C_Vevtor)):
    #観測方向ベクトルと太陽方向ベクトルの位相角  θ
    inner_product  = (sun_x[i]*uav_x[i])+(sun_y[i]*uav_y[i])+(sun_z[i]*uav_z[i])
    theta = math.acos(inner_product / \
                      math.sqrt((sun_x[i]**2+sun_y[i]**2+sun_z[i]**2) * \
                                (uav_x[i]**2+uav_y[i]**2+uav_z[i]**2)))
    theta2 = math.degrees(theta)

    inner_product2  = (-sun_x[i]*uav_x[i])+(-sun_y[i]*uav_y[i])+(0*uav_z[i])
    uav_alt = math.acos(inner_product2 / \
                        math.sqrt(((-sun_x[i])**2+(-sun_y[i])**2+0**2) * \
                                  (uav_x[i]**2+uav_y[i]**2+uav_z[i]**2))) #俯角
    uav_alt2 = math.degrees(uav_alt)

    inner_product3  = (sun_x[i]*sun_x[i])+(sun_y[i]*sun_y[i])+(0*sun_z[i])
    sun_alt = math.acos(inner_product3 / \
                        math.sqrt(((sun_x[i])**2+(sun_y[i])**2+0**2) * \
                                  (sun_x[i]**2+sun_y[i]**2+sun_z[i]**2))) #俯角
    sun_alt2 = math.degrees(sun_alt)

    arr1=[]
    arr1.insert(0,photolist[i])
    arr1.insert(1,theta2)
    Theta_List.append(arr1)

    arr2=[]
    arr2.insert(0,photolist[i])
    arr2.insert(1,uav_alt2)
    UAV_alt_List.append(arr2)

    arr3=[]
```

```
    arr3.insert(0,photolist[i])
    arr3.insert(1,sun_alt2)
    Sun_alt_List.append(arr3)

#############################################################################################
# 3.4 解析対象面の平均値，最小値，最大値，標準偏差，影のピクセル数の計算-----------------------
#############################################################################################

#画像座標を記録するファイルの作成 csv
cameradata=pd.read_csv(InputData_2,delim_whitespace=True,
                       names=("photoID","x","y","z","Omega","Phi","Kappa",
                              "r11","r12","r13","r21","r22","r23","r31","r32","r33"),
                              skiprows=2)
CAMERADATA=cameradata
intersection= np.load(InputData_3)
XI=intersection["arr_0"]
YI=intersection["arr_1"]
ZI=intersection["arr_2"]
with open(InputData_4, encoding='utf-8') as fp: #読み込み xml
    xml_data = fp.read()
    dict_data = xmltodict.parse(xml_data)

csv_file = glob.glob(os.path.join(OutputData_1))
print(csv_file)
if csv_file==[]:
    header = ["ID", "Row","Col"]

    with open(OutputData_1, "w") as f:

        writer = csv.writer(f)
        writer.writerow(header)

    f.close()
###ファイルに画像座標を書き込み csv
    for ID in tqdm(photolist):
        data = CAMERADATA
        data=data[data["photoID"]==ID]
        idx=list(data.index)
        Index=sum(idx)
        params=dict_data["document"]["chunk"]["sensors"]["sensor"]["calibration"]

        try:
            f=float(params["f"])
        except TypeError:
            params=dict_data["document"]["chunk"]["sensors"]["sensor"]["calibration"][1]

        f=float(params["f"])
        cx=float(params["cx"])
        cy=float(params["cy"])
        width=float(params["resolution"]["@width"])
```

```
height=float(params["resolution"]["@height"])

C = f * 0.004394861364928360 9/1000
W = width * 0.004394861364928360 9/2000
H = height * 0.004394861364928360 9/2000

x = float(data["x"])
y = float(data["y"])
z = float(data["z"])
O = data["Omega"]
P = data["Phi"]
K = data["Kappa"]

#度数から弧度
rad_O = math.radians(float(O))
rad_P = math.radians(float(P))
rad_K = math.radians(float(K))

#三角関数の計算
sin_O = math.sin(-float(rad_O))
cos_O = math.cos(-float(rad_O))
sin_P = math.sin(-float(rad_P))
cos_P = math.cos(-float(rad_P))
sin_K = math.sin(-float(rad_K))
cos_K = math.cos(-float(rad_K))
irange,jrange,krange=D.shape

n=1
tmp = np.array([])
tmp_list = tmp.tolist()
arr2 = np.array([[cos_K,-sin_K,0], [sin_K,cos_K,0], [0,0,1]]) #kappa
arr3 = np.array([[cos_P,0,sin_P], [0,1,0], [-sin_P,0,cos_P]]) #phi
arr4 = np.array([[1,0,0], [0,cos_O,-sin_O], [0,sin_O,cos_O]]) #omega
arr5 = np.dot(arr2, arr3)
arr6 = np.dot(arr5, arr4)
N=ysize1*xsize1
ki=0
kc=0
ji=0
u = np.zeros((ysize1,xsize1))
v = np.zeros((ysize1,xsize1))
with tqdm(total=N) as pbar:
    while n<=N:
        n=n+1
        pbar.update(1)
        if ji == ysize1:
            kc=0
            ki=ki+1
            ji=0
        point_mesh_x = XI[ji][ki]
```

```
            point_mesh_y = YI[ji][ki]
            point_mesh_z = ZI[ji][ki]

            if point_mesh_x == 0 and point_mesh_y == 0 and point_mesh_z == 0:
                kc=kc+1
                ji=ji+1
            else:
                arr1 = np.array([[point_mesh_x-float(x)], [point_mesh_y-float(y)],
                                 [point_mesh_z-float(z)]]) #終点始点-
                arr7 = np.dot(arr6, arr1)
                U = -C * (arr7[0,0] / arr7[2,0]) #共線条件式
                V = -C * (arr7[1,0] / arr7[2,0])
                u[ji][ki] = float(U)
                v[ji][ki] = float(V)
                kc=kc+1
                ji=ji+1


Path=sorted(glob.glob(os.path.join(InputData_5+ID+"*.tif"))) #画像の読み込み
img = io.imread(Path[0])
img_R = np.array(img[:,:,0])
img_G = np.array(img[:,:,1])
img_B = np.array(img[:,:,2])

h, w = img_G.shape
pixel_width =  0.0043948613649283609 / 1000
pixel_height = 0.0043948613649283609 / 1000
#仮想投影面の範囲点の個数()
kasou_row , kasou_col = np.shape(u) #1501,1501

#仮想画像それぞれの２次元配列RGB
virtual_r_image = np.zeros((kasou_row,kasou_col), np.uint32) ##1501,1501
virtual_g_image = np.zeros((kasou_row,kasou_col), np.uint32)
virtual_b_image = np.zeros((kasou_row,kasou_col), np.uint32)

C=0
n=1
k=0
kc=0
j=0

GEO_P1      = x,y,z                                          #の地上座標UAV
IMG_P1      = (x - xmin)/0.2 , (ymax - y)/0.2 , z/0.2    #のボクセル番地UAV

#投影面の中心番地
centerj = int(kasou_col/2)
centerk = int(kasou_row/2)
N = kasou_col * kasou_row
#print(N)
for j in range(kasou_row):
```

```
            for k in range(kasou_col):
                if j == ysize1:
                    kc=0
                    k=k+1
                    j=0
                U = u[j][k]
                V = v[j][k]
                col_1 = U / pixel_width
                row_1 = V / pixel_height
                #print(col_1,row_1)

                col_2 = col_1 + w/2
                row_2 = h/2 - row_1

                col_3 = col_2 + cx #±
                row_3 = row_2 + cy #±

                Col = int(col_3)
                Row = int(row_3)

                if Row>0 and Row<5460 and Col>0 and Col<8192:
                    if j==int(row) and k==int(col) :
                        with open(OutputData_1,"a") as f:
                            #print(OutputData_1)
                            writer = csv.writer(f)
                            writer.writerow([ID,Row,Col])

else:
    print("作成済みcsv:"+OutputData_1)

Coordinates = pd.read_csv(OutputData_1)
ImageRow = Coordinates["Row"]
ImageCol = Coordinates["Col"]

#ファイルの作成 csv
csv_file2 = glob.glob(os.path.join(OutputData_2))
if csv_file2==[]:
    print("情報を記録するファイルの作成Gcsv")
    header = ["ID",
              "Ori_R_Mean","Ori_G_Mean","Ori_B_Mean","Ori_G_Normalize",
              "Ori_R_Min","Ori_G_Min","Ori_B_Min",
              "Ori_R_Max","Ori_G_Max","Ori_B_Max",
              "Ori_R_Std","Ori_G_Std","Ori_B_Std",

              "Ortho_R_Mean","Ortho_G_Mean","Ortho_B_Mean","Ortho_G_Normalize",
              "Shadow_pixel"
              ]

    with open(OutputData_2, "w") as f:
```

```
    writer = csv.writer(f)
    writer.writerow(header)

f.close()

print("情報を計算G")
Coordinate = pd.read_csv(OutputData_1)

Original_Image_Data = []

print("情報を計算DN")
for i in tqdm(range(len(Coordinate))):
    ID = Coordinate["ID"][i]
    print(ID)
    Row = Coordinate["Row"][i]
    Col = Coordinate["Col"][i]
    Path=sorted(glob.glob(os.path.join(InputData_5+ID+"*.tif"))) #画像の読み込み
    img = io.imread(Path[0])
    Row_min,Row_max = max(0,Row-28),min(int(img.shape[0])-1,Row+29)
    Col_min,Col_max = max(0,Col-28),min(int(img.shape[1])-1,Col+29)

    img2 = img[Row_min:Row_max,Col_min:Col_max,:]

    Ori_R_Mean              = np.nanmean(img2[:,:,0])
    Ori_G_Mean              = np.nanmean(img2[:,:,1])
    Ori_B_Mean              = np.nanmean(img2[:,:,2])
    Ori_G_Normalize         = Ori_G_Mean/(Ori_R_Mean+Ori_G_Mean+Ori_B_Mean)
    Ori_R_Min               = np.nanmin(img2[:,:,0])
    Ori_G_Min               = np.nanmin(img2[:,:,1])
    Ori_B_Min               = np.nanmin(img2[:,:,2])
    Ori_R_Max               = np.nanmax(img2[:,:,0])
    Ori_G_Max               = np.nanmax(img2[:,:,1])
    Ori_B_Max               = np.nanmax(img2[:,:,2])
    Ori_R_Std               = np.nanstd(img2[:,:,0])
    Ori_G_Std               = np.nanstd(img2[:,:,1])
    Ori_B_Std               = np.nanstd(img2[:,:,2])

    #影のピクセル数をカウント
    Shadow_img = img2.copy()
    count_pix = 0
    for m in range(img2.shape[0]):
        for n in range(img2.shape[1]):
            if img2[m,n,1]<4500:
                count_pix += 1

    #原画像情報
    arr_Original=[]
    arr_Original.insert(0,ID)
    arr_Original.insert(1,Ori_R_Mean)          #平均値
    arr_Original.insert(2,Ori_G_Mean)
```

```
            arr_Original.insert(3,Ori_B_Mean)
            arr_Original.insert(4,Ori_G_Normalize)  #正規化

            arr_Original.insert(5,Ori_R_Min)        #最小値
            arr_Original.insert(6,Ori_G_Min)
            arr_Original.insert(7,Ori_B_Min)

            arr_Original.insert(8,Ori_R_Max)        #最大値
            arr_Original.insert(9,Ori_G_Max)
            arr_Original.insert(10,Ori_B_Max)

            arr_Original.insert(11,Ori_R_Std)       #標準偏差
            arr_Original.insert(12,Ori_G_Std)
            arr_Original.insert(13,Ori_B_Std)

            arr_Original.insert(14,count_pix)       #影のピクセル数
            Original_Image_Data.append(arr_Original)

else:
    print("作成済みcsv:"+OutputData_2)


#############################################################################################
# 3.5 princpal の抽出plane-------------------------------------------------------------
#############################################################################################

# plncipale 以外のルートを削除palene

RootSplit = []

InitialValue = 0
for i in tqdm(range(len(photolist))):
    try:

        if int(photolist[i][-4::]) + 1 != int(photolist[i+1][-4::]):
            RootSplit.apend(InitialValue+1)

    except:
        RootSplit.append(InitialValue+1)

    InitialValue += 1

#何目がpassprincipale なのかplane
if str(sys.argv[2]) == "a":
    principal_root = RootSplit[int((len(RootSplit)/2) - 1)],RootSplit[int(len(RootSplit)/2)]
elif str(sys.argv[2]) == "b":
    principal_root = RootSplit[int((len(RootSplit)/2) - 2)],RootSplit[int((len(RootSplit)/2)-1)]
elif str(sys.argv[2]) == "c":
    principal_root = RootSplit[-2],RootSplit[-1]
elif str(sys.argv[2]) == "d":
```

```
    principal_root = RootSplit[-3],RootSplit[-2]


if csv_file2==[]:
    #画像ID
    pri_photolist           = np.array(photolist[principal_root[0]:principal_root[1]])
    pri_photolist2          = pri_photolist[:]

    #位相角θ
    pri_Theta_List          = np.array(Theta_List[principal_root[0]:principal_root[1]])
    pri_Theta_List2         = np.array(pri_Theta_List[:,1],dtype='float32')

    #高度角UAV
    pri_UAV_alt_List        = np.array(UAV_alt_List[principal_root[0]:principal_root[1]])
    pri_UAV_alt_List2       = np.array(pri_UAV_alt_List[:,1],dtype='float32')

    #太陽高度角
    pri_Sun_alt_List        = np.array(Sun_alt_List[principal_root[0]:principal_root[1]])
    pri_Sun_alt_List2       = np.array(pri_Sun_alt_List[:,1],dtype='float32')

    #原画像Row
    pri_ImageRow            = np.array(ImageRow[principal_root[0]:principal_root[1]])
    pri_ImageRow2           = pri_ImageRow[:]

    #原画像Col
    pri_ImageCol            = np.array(ImageCol[principal_root[0]:principal_root[1]])
    pri_ImageCol2           = pri_ImageCol[:]

    #地上座標UAV
    pri_UAV_X2              = np.array([])
    pri_UAV_Y2              = np.array([])
    pri_UAV_Z2              = np.array([])
    for i in range(len(pri_photolist2)):
        pri_UAV_X = float(Cameradata.loc[pri_photolist2[i]]['x'])
        pri_UAV_Y = float(Cameradata.loc[pri_photolist2[i]]['y'])
        pri_UAV_Z = float(Cameradata.loc[pri_photolist2[i]]['z'])

        pri_UAV_X2 = np.append(pri_UAV_X2,pri_UAV_X)
        pri_UAV_Y2 = np.append(pri_UAV_Y2,pri_UAV_Y)
        pri_UAV_Z2 = np.append(pri_UAV_Z2,pri_UAV_Z)

    #20分解能オルソ画像から計算した平均cm
    pri_ave_r               = np.array(ave_r[principal_root[0]:principal_root[1]])
    pri_ave_r2              = pri_ave_r[:]
    pri_ave_g               = np.array(ave_g[principal_root[0]:principal_root[1]])
    pri_ave_g2              = pri_ave_g[:]
    pri_ave_b               = np.array(ave_b[principal_root[0]:principal_root[1]])
    pri_ave_b2              = pri_ave_b[:]

    #20分解能オルソ画像のを正規化cmDN
```

```
    pri_ave_n               = pri_ave_g2/(pri_ave_r2+pri_ave_g2+pri_ave_b2)

    pri_Original            = np.array(Original_Image_Data[principal_root[0]:principal_root[1]])
    pri_Original2           = np.array(pri_Original[:])

    #ファイルに書き込みcsv
    for Num in range(len(pri_photolist2)):
        ID                              = pri_photolist2[Num]
        Ori_R_Mean              = pri_Original2[Num,1]
        Ori_G_Mean              = pri_Original2[Num,2]
        Ori_B_Mean              = pri_Original2[Num,3]
        Ori_G_Normalize         = pri_Original2[Num,4]
        Ori_R_Min               = pri_Original2[Num,5]
        Ori_G_Min               = pri_Original2[Num,6]
        Ori_B_Min               = pri_Original2[Num,7]
        Ori_R_Max               = pri_Original2[Num,8]
        Ori_G_Max               = pri_Original2[Num,9]
        Ori_B_Max               = pri_Original2[Num,10]
        Ori_R_Std               = pri_Original2[Num,11]
        Ori_G_Std               = pri_Original2[Num,12]
        Ori_B_Std               = pri_Original2[Num,13]
        Ori_Shadow_pixel        = pri_Original2[Num,14]
        Ortho_R_Mean            = pri_ave_r2[Num]
        Ortho_G_Mean            = pri_ave_g2[Num]
        Ortho_B_Mean            = pri_ave_b2[Num]
        Ortho_G_Normalize       = pri_ave_n[Num]

        with open(OutputData_2,"a") as f:
            writer = csv.writer(f)
            writer.writerow([ID,Ori_R_Mean,Ori_G_Mean,Ori_B_Mean,
                             Ori_G_Normalize,
                             Ori_R_Min,Ori_G_Min,Ori_B_Min,
                             Ori_R_Max,Ori_G_Max,Ori_B_Max,
                             Ori_R_Std,Ori_G_Std,Ori_B_Std,
                             Ori_Shadow_pixel,
                             Ortho_R_Mean,Ortho_G_Mean,Ortho_B_Mean,
                             Ortho_G_Normalize
                             ])

else:


    pri_photolist           = np.array(photolist[principal_root[0]:principal_root[1]])
    pri_photolist2          = pri_photolist[:]

    pri_Theta_List          = np.array(Theta_List[principal_root[0]:principal_root[1]])
    pri_Theta_List2         = np.array(pri_Theta_List[:,1],dtype='float32')

    pri_UAV_alt_List        = np.array(UAV_alt_List[principal_root[0]:principal_root[1]])
    pri_UAV_alt_List2       = np.array(pri_UAV_alt_List[:,1],dtype='float32')
```

```
    pri_Sun_alt_List        = np.array(Sun_alt_List[principal_root[0]:principal_root[1]])
    pri_Sun_alt_List2       = np.array(pri_Sun_alt_List[:,1],dtype='float32')

    pri_ImageRow            = np.array(ImageRow[principal_root[0]:principal_root[1]])
    pri_ImageRow2           = pri_ImageRow[:]

    pri_ImageCol            = np.array(ImageCol[principal_root[0]:principal_root[1]])
    pri_ImageCol2           = pri_ImageCol[:]


    pri_UAV_X2              = np.array([])
    pri_UAV_Y2              = np.array([])
    pri_UAV_Z2              = np.array([])
    for i in range(len(pri_photolist2)):
        pri_UAV_X = float(Cameradata.loc[pri_photolist2[i]]['x'])
        pri_UAV_Y = float(Cameradata.loc[pri_photolist2[i]]['y'])
        pri_UAV_Z = float(Cameradata.loc[pri_photolist2[i]]['z'])

        pri_UAV_X2 = np.append(pri_UAV_X2,pri_UAV_X)
        pri_UAV_Y2 = np.append(pri_UAV_Y2,pri_UAV_Y)
        pri_UAV_Z2 = np.append(pri_UAV_Z2,pri_UAV_Z)

RGBInfo = pd.read_csv(OutputData_2)

Ori_R_Mean          = np.array(RGBInfo["Ori_R_Mean"],dtype='float32')
Ori_G_Mean          = np.array(RGBInfo["Ori_G_Mean"],dtype='float32')
Ori_R_Mean          = np.array(RGBInfo["Ori_B_Mean"],dtype='float32')
Ori_G_Normalize     = np.array(RGBInfo["Ori_G_Normalize"],dtype='float32')
Ori_R_Min           = np.array(RGBInfo["Ori_R_Min"],dtype='float32')
Ori_G_Min           = np.array(RGBInfo["Ori_G_Min"],dtype='float32')
Ori_B_Min           = np.array(RGBInfo["Ori_B_Min"],dtype='float32')
Ori_R_Max           = np.array(RGBInfo["Ori_R_Max"],dtype='float32')
Ori_G_Max           = np.array(RGBInfo["Ori_G_Max"],dtype='float32')
Ori_B_Max           = np.array(RGBInfo["Ori_B_Max"],dtype='float32')
Ori_R_Std           = np.array(RGBInfo["Ori_R_Std"],dtype='float32')
Ori_G_Std           = np.array(RGBInfo["Ori_G_Std"],dtype='float32')
Ori_B_Std           = np.array(RGBInfo["Ori_B_Std"],dtype='float32')
Ori_Shadow_pixel    = np.array(RGBInfo["Shadow_pixel"],dtype='float32')
Ortho_R_Mean        = np.array(RGBInfo["Ortho_R_Mean"],dtype='float32')
Ortho_G_Mean        = np.array(RGBInfo["Ortho_G_Mean"],dtype='float32')
Ortho_B_Mean        = np.array(RGBInfo["Ortho_B_Mean"],dtype='float32')
Ortho_G_Normalize   = np.array(RGBInfo["Ortho_G_Normalize"],dtype='float32')

#太陽高度の平均
Sun_alt =  np.average(pri_Sun_alt_List2)

#樹木の真上にが通ると仮定した時の傾きと切片UAV
UAVPath_Slope = (pri_UAV_Y2[0]-pri_UAV_Y2[-1])/(pri_UAV_X2[0]-pri_UAV_X2[-1])
TreePath_Slice = input_data_Y - UAVPath_Slope * input_data_X
```

```
#樹木と軌道との水平距離UAV
Hdist = round(abs(UAVPath_Slope*pri_UAV_X2[0] + (-1)*pri_UAV_Y2[0] + TreePath_Slice) / \
                (np.sqrt(UAVPath_Slope**2 + (-1)**2)),2)
#軌道位相角度
OrbitalPhaseAngle = round(
    np.degrees(
        np.arcsin(
            (np.sin(np.radians(90)) / \
             (np.sqrt((pri_UAV_Z2[0] - ZI[row,col])**2 + Hdist**2)))*Hdist)),2)

#############################################################################################
# 3.6 グラフの描画----------------------------------------------------------------------------
#############################################################################################
#の可視化　全BRDFpass
def LinearStrtch(P,Pmin,Pmax,Qmin,Qmax):
    Qmin,Qmax = Qmin,Qmax
    Pmin,Pmax = Pmin,Pmax
    a    = (Qmax - Qmin) / (Pmax - Pmin)
    b    = Qmin - a*Pmin
    Q    = a*P + b
    return Q

def degrees_to_radians(degrees):
    return math.radians(degrees)

X_data01 = list(map(float,np.array(Theta_List)[:,1]))
Y_data01 = list(map(float,np.array(ave_g)))

#相関係数cor傾き,p切片[0],p[1]
p1 = np.polyfit(X_data01,Y_data01,1)
f1 = np.poly1d(p1)
cor1 = np.corrcoef(X_data01,Y_data01)[0,1]

plt.rcParams['figure.subplot.bottom'] = 0.15
plt.rcParams['figure.subplot.left'] = 0.17

#の設定 Figure
fig01 = plt.figure(figsize=(5,5))
ax01 = fig01.add_subplot(1,1,1)

ax01.set_title(Name)
ax01.set_xlabel("θ(radian)")
ax01.set_ylabel("DN_G")
ax01.set_xlim(0,90)
ax01.set_ylim(0,20000)
ax01.grid(True)

#相関係数 ()
ax01.plot(X_data01, f1(X_data01), color = "black", label="相関係数:{:.3f}".format(cor1))
```

```
#傾き ()
ax01.plot(X_data01, f1(X_data01), color = "black", label="傾き    :{:.2f}".format(p1[0]))
#切片 ()
ax01.plot(X_data01, f1(X_data01), color = "black", label="切片    :{:.1f}".format(p1[1]))

ax01.legend(loc="upper left")
ax01.scatter(X_data01,Y_data01)
fig01.show()
fig01.savefig(OutputData_3)

#############################################################################################
#  の可視化BRDF principale plane ののみpass緑バンド(値平均DN)

X_data02 = list(map(float,np.array(pri_Theta_List2)))
Y_data02 = list(map(float,np.array(Ori_G_Mean)))
X_data03 = list(map(degrees_to_radians, X_data02))
Y_data03 = LinearStrtch(Ori_G_Mean,0,15000,0,1)

#相関係数cor傾き,p切片[0],p[1]
p2 = np.polyfit(X_data03,Y_data03,1)
f2 = np.poly1d(p2)
cor2 = np.corrcoef(X_data03,Y_data03)[0,1]

#の設定 Figure
fig02 = plt.figure(figsize=(5,5))
ax02 = fig02.add_subplot(1,1,1)

ax02.set_xlabel("θ(radian)",fontsize=18)
ax02.set_ylabel("DN緑バンド()",fontsize=18)
ax02.set_xlim(0,1)
ax02.set_ylim(0,1)
ax02.yaxis.set_major_formatter(FormatStrFormatter('%.1f'))
ax02.grid(True)

squared = 0
for (i,j,k) in zip(X_data03,Y_data03,np.array(np.array(pri_photolist2))):
    squared = squared + (j - (p2[0]*i + p2[1]))**2

RMSE = '{:4f}'.format(np.sqrt(squared / len(X_data03))) #指数表現をしないように設定
# (RMSE)
ax02.plot(X_data03, f2(X_data03), color = "black", label="  :RMSE"+RMSE[0:5])
#相関係数 ()
ax02.plot(X_data03, f2(X_data03), color = "black", label="相関係数:{:.2f}".format(cor2))
#傾き ()
ax02.plot(X_data03, f2(X_data03), color = "black", label="傾き    :{:.2f}".format(p2[0]))
#切片 ()
ax02.plot(X_data03, f2(X_data03), color = "black", label="切片    :{:.2f}".format(p2[1]))

#天頂を線で表現
Zenith_theta = math.radians((90 - Sun_alt))
```

```
ax02.plot((Zenith_theta,Zenith_theta), (0,1),color = "r",alpha=0.2)

ax02.legend(loc="lower left")
ax02.scatter(X_data03,Y_data03)

fig02.savefig(OutputData_4)

#################################################################################################
#  の可視化BRDF principale plane ののみpass（(G/R+G+）値平均BDN)

X_data04 = list(map(float,np.array(pri_Theta_List2)))
Y_data04 = list(map(float,np.array(Ori_G_Normalize)))#Ori_G_Normalize

X_data05 = list(map(degrees_to_radians, X_data04))
Y_data05 = Y_data04

#相関係数cor傾き,p切片[0],p[1]
p3 = np.polyfit(X_data05,Y_data05,1)
f3 = np.poly1d(p3)
cor3 = np.corrcoef(X_data05,Y_data05)[0,1]

#の設定 Figure
fig03 = plt.figure(figsize=(5,5))
ax03 = fig03.add_subplot(1,1,1)

ax03.set_title(Name)
ax03.set_xlabel("θ(radian)")
ax03.set_ylabel("G/(R+G+B)")
ax03.set_xlim(0,1)
ax03.set_ylim(0.3,0.5)
ax03.yaxis.set_major_formatter(FormatStrFormatter('%.2f'))
ax03.grid(True)

#相関係数 ()
ax03.plot(X_data05, f3(X_data05), color = "black", label="相関係数:{:.3f}".format(cor3))
#傾き ()
ax03.plot(X_data05, f3(X_data05), color = "black", label="傾き    :{:.2f}".format(p3[0]))
#切片 ()
ax03.plot(X_data05, f3(X_data05), color = "black", label="切片    :{:.2f}".format(p3[1]))

ax03.legend(loc="upper right")
ax03.scatter(X_data05,Y_data05,s=15)
ax03.plot((Zenith_theta,Zenith_theta), (0,1),color = "r",alpha=0.2)

squared2 = 0
for (i,j,k) in zip(X_data05,Y_data05,np.array(np.array(pri_photolist2))):
    ax03.plot(i,j,'o')
    ax03.annotate(k[-4::], xy=(i, j),size=5)
    squared2 = squared2 + (j - (p3[0]*i + p3[1]))**2
```

```
RMSE2 = '{:f}'.format(np.sqrt(squared2 / len(X_data05)))
fig03.savefig(OutputData_5)

#############################################################################################
#  影割合変化図

X_data06 = list(map(float,np.array(pri_Theta_List2)))
Y_data06 = list(map(float,np.array(Ori_Shadow_pixel)))

X_data07 = list(map(degrees_to_radians, X_data06))
Y_data07 = np.array(Y_data06) / 3249 #57 x 57pix


#相関係数cor傾き,p切片[0],p[1]
p4 = np.polyfit(X_data07,Y_data07,1)
f4 = np.poly1d(p4)
cor4 = np.corrcoef(X_data07,Y_data07)[0,1]

#の設定 Figure
fig04 = plt.figure(figsize=(5,5))
ax04 = fig04.add_subplot(1,1,1)

#ax04.set_title(Name)
ax04.set_xlabel("θ(radian)",fontsize=18)
ax04.set_ylabel("影割合",fontsize=18)
ax04.set_xlim(0,1)
ax04.set_ylim(0,1)
ax04.yaxis.set_major_formatter(FormatStrFormatter('%.1f'))
ax04.grid(True)

#相関係数 ()
ax04.plot(X_data07, f4(X_data07), color = "black", label="相関係数:{:.3f}".format(cor4))
#傾き ()
ax04.plot(X_data07, f4(X_data07), color = "black", label="傾き    :{:.2f}".format(p4[0]))
#切片 ()
ax04.plot(X_data07, f4(X_data07), color = "black", label="切片    :{:.2f}".format(p4[1]))

ax04.legend(loc="upper right")
ax04.scatter(X_data07,Y_data07,color='gray')
ax04.plot((Zenith_theta,Zenith_theta), (0,1),color = "r",alpha=0.2)
fig04.savefig(OutputData_7)
Shadow_mean = np.mean(Y_data07)
BRDFinfo_csv = glob.glob(os.path.join(OutputData_6))
if BRDFinfo_csv==[]:
    header = ["Name", "correlation","slope","intercept","RMSE",
              "correlation2","slope2","intercept2","RMSE2",
              "OrbitalPhaseAngle",
              "Shadow_correlation2","Shadow_slope","Shadow_intercept",
              "Shadow_mean"
              ]
```

```
    with open(OutputData_6, "w") as f:
        writer = csv.writer(f)
        writer.writerow(header)
    f.close()
with open(OutputData_6,"a") as f:
    writer = csv.writer(f)
    writer.writerow([Name,cor2,p2[0],p2[1],RMSE,cor3,p3[0],p3[1],
                     RMSE2,OrbitalPhaseAngle,cor4,p4[0],p4[1],Shadow_mean])
```