令和 7 年度

修士学位論文

複数 UAV による同期画像を用いた越波構造の四次元計測

4-D reconstruction of wave overtopping by synchronized UAV's

指導教員　佐藤 慎司 教授

副指導教員　鈴木 卓 准教授

副審査員　高木 方隆 教授

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

池本 千馬

2026 年 1 月 21 日

要旨

複数 UAV による同期画像を用いた越波構造の四次元計測

池本 千馬

台風や高潮に起因する高波の越波によって、港湾や漁港の泊地周辺において、船舶の転覆や浸水などの被害が発生している。

現在、これらの越波被害に対する対策としては、地形情報等に基づく越波高の推定や、実際の最高越波高の観測によって越波規模を把握し、それに基づいて離岸堤や消波ブロックを設置して越波を防ぐ手法がとられている。しかし、対策が必要なすべての海岸において、適切な措置が講じられているわけではない。

離岸堤の大型化や消波ブロックの増設・大型化などにより、越波による被害を軽減させることは可能である。しかし、むやみに対策規模を拡大することは、被害の防止に繋がる一方で、工事費用の増大や景観の悪化を招く。したがって、想定される越波に対して適切な規模の対策を講じることが求められる。

そこで本研究では、適切な越波対策を設計するためには越波の流量や流体構造を正確に把握する必要があると考え、その基本情報となる越波水塊の挙動を詳細に解明することを目的とする。

越波構造を把握する手段として、UAV を複数台飛行させ、ステレオ視を用いて水塊の挙動を解析する手法をとった。

最終的なターゲットとする現地スケールの越波としては、高知県安芸漁港沖防波堤で報告されている30m前後の高さまで打ちあがるものを想定しているが、本研究では、高さ 10m程度の小規模な越波に対して計測手法を開発する。

調査地点は高知港東の離岸堤周辺を選定した。3 台の UAV で越波の 4K 動画撮影を約 15 分間行った。飛行直前に時計の映像を映しておくことで、3 台分の動画をほぼ同時刻の 3 枚 1 組の同期画像でフレーム数約 3 万コマに分割した。

用いた UAV は GNSS(Global Navigation Satellite System)搭載であるため、撮影される静止画には、撮影時点の UAV の位置や姿勢が記録される。しかしながら、本研究では、動画をフレーム画像にしたものを分析対象にするため、強風の影響を受けて小刻みに変動するカメラの位置・姿勢情報が分割した個々の画像には記録されていない。ステレオ視にはカメラの位置・姿勢情報が必須であるため、何らかの方法でこれらを推定する必要がある。

ステレオ視分析では、カメラの位置・姿勢から越波水塊などの動的対象点の三次元位置情報を推定することになるが、本研究では、まず、複数画像で位置情報の判明している固定対象点を元に、強風の影響を受けてフレームごとに変動するカメラ位置・姿勢を推定した。固定対象点は、事前に対象領域の写真測量を行っておくことで、詳細 DEM(Digital Elevation

Model)を作成して設定した。このようにして、フレーム画像ごとに判明したカメラ位置・姿勢を元に、改めて動的対象物である波浪や越波の形状をステレオ視を用いて推定した。

このようにして、推定したカメラ位置・姿勢 UAV のフライトログとカメラ位置の推定結果を比較したところ、十分な精度であることを確認した。

越波構造の形状推定を行うなかで、精度や点群密度に影響する因子として、特徴点探索・マッチングにおける空間解像度、点群の品質制御や用いる UAV の台数などが、重要であることを見出した。SfM(Structure from Motion)/MVS(Multi-View Stereo)分析に用いたソフトウェア Agisoft Metashape pro では特徴点探索に係るパラメータは、アライメント精度のパラメータとして設定できる。また、点群の品質制御は、ステレオ視分析で生成される深度マップで急激な変化を許容する空間スケールとして設定できる。なお、Metashape を用いた分析では、python スクリプトを用いて、複数時刻の分析を連続的に自動分析させた。

UAV 台数については、2 台に減らすと、かなり精度が低くなり、形状取得が行えない領域が増えたため、3 台以上に台数を増やすことの優位性が示された。

特徴点処理における空間解像度や点群の品質制御に用いる空間スケールは、地上解像度が 2cm/pix 程度であった本研究においては 1x1 ピクセルで最も細かく設定するより、2x2 や 4x4 ピクセル程度で設定した方が、復元される点群の数が多く、越波の内部構造をより詳細にとらえることが確認できた。これらの処理スケールは地上解像度で換算すると 4~8cm のスケールとなり、高さ 10m 程度の越波における SfM/MVS 分析の最適空間スケールを推定することができた。

以上より、本手法を用いることで、越波の 3 地点以上からの UAV で撮影した動画で越波の内部構造を可視化することが可能になった。

キーワード：越波、UAV、ステレオ視、四次元計測、カメラ位置推定、空間解像度

Abstract

4-D reconstruction of wave overtopping by synchronized UAV's

Ikemoto Kazuma

Wave overtopping caused by typhoons and storm surges leads to severe damage around ports and fishing harbors, including vessel capsizing and coastal flooding. Current countermeasures involve estimating overtopping heights based on topography or historical observations and installing offshore breakwaters or wave-dissipating blocks. However, appropriate measures are not yet implemented on all vulnerable coasts. While increasing the size of breakwaters or the volume of armor units can mitigate damage, excessive scaling leads to rising construction costs and degradation of the coastal landscape. Therefore, it is essential to design countermeasures proportional to the anticipated scale of overtopping.

To understand the overtopping structure, a method was employed using multiple UAVs to analyze water mass behavior via stereo vision. While the ultimate target is the large-scale overtopping (approximately 30 meters high) reported at the Aki Fishing Harbor offshore breakwater in Kochi Prefecture, this study develops the measurement method targeting 10-meter-class overtopping around the detached breakwater east of Kochi Port.

Three UAVs captured 4K video of the overtopping for approximately 15 minutes. Temporal synchronization was achieved by filming a clock prior to flight, allowing the footage to be split into approximately 30,000 sets of simultaneous frame images. Although the UAVs record GNSS data for still images, the extracted video frames lack the metadata for camera position and attitude—which fluctuate minutely due to strong winds—required for stereo vision. Therefore, these parameters had to be estimated.

In the stereo vision analysis, while the goal is to estimate the 3D positions of dynamic targets (water masses), this study first estimated the frame-by-frame camera positions and attitudes using fixed control points. These fixed points were established based on a detailed DEM created via prior photogrammetry of the target area. Based on these determined camera parameters, the shapes of the dynamic waves and overtopping were reconstructed. A comparison of the estimated camera positions with the UAV flight logs confirmed sufficient accuracy.

In estimating the overtopping shape, it was found that factors such as spatial resolution in feature matching, point cloud quality control, and the number of UAVs are critical. The analysis used Agisoft Metashape Professional, where feature search parameters can be set as alignment accuracy, and point cloud quality control can be managed via the spatial scale allowing changes in depth maps. The process was automated using Python scripts for continuous multi-step analysis.

Regarding the number of UAVs, reducing the unit count to two significantly lowered accuracy and increased blind spots, demonstrating the superiority of using three or more UAVs. Furthermore, under a ground resolution of approximately 2 cm/pix, it was confirmed that setting the feature processing scale to 2x2 or 4x4 pixels (equivalent to 4–8 cm on the ground) yielded a larger number of reconstructed points and captured the internal structure in greater detail than the finest 1x1 pixel setting. Thus, the optimal spatial scale for SfM/MVS analysis of 10-meter-class overtopping was estimated.

In conclusion, this method enables the visualization of the internal structure of wave overtopping using video captured by UAVs from three or more viewpoints.

Key words Wave overtopping, UAV, Stereo vision, 4-D measurement, Camera pose estimation, Spatial resolution

目次

図目次

表目次

第 1 章

緒論

台風による越波は海に面する多くのエリアで発生し、船の転覆といった海上での被害だけでなく、浸水や構造物の破壊などの原因となる。実際に、2018 年台風 24 号接近時の高知県安芸市安芸漁港では高さ 5m 程度の防波堤が設置されている中、推定高さ 30m の越波が発生している。

そして、越波は動的でその形状は時々刻々と変化するため、三次元構造を把握することは困難である。そのため、本研究ではステレオ視と呼ばれる手法をとることとする。ステレオ視とは、**図 1** のような対象物とカメラの配置関係から、2 視点からの視差を用いて物体の遠近感を把握する手法である。機械的に行うためのメカニズムとしては、複数視点から対象物の写真を撮影する。そして、カメラ間の距離と物体までの距離、カメラの内部パラメータを用い、画像間で同一点を見つけ、マッチングさせることで、対象点までの距離を計測することができる。この操作をすべての同一点が見つかる間繰り返すことで対象物の形状を復元する。

本研究では、このステレオ視の手法を用いて越波構造を把握する。さらに、動画撮影によって時系列データを取得することで、三次元構造に時間軸を加えた「四次元構造」の把握を行うことを目的とする。

本論文は全 6 章で構成される。第 2 章では、ステレオ視や UAV を用いて越波や海面の変動を調査した既往研究についてレビューを行い、本研究の位置づけについて述べる。第 3 章では、現地調査と調査方法について述べる。第 4 章ではステレオ視を用いた三次元復元までの手順を述べる。第 5 章では、作成した三次元データについての精度や質について調査を行い、その結果を述べる。第 6 章では結論と今後の課題について述べる。

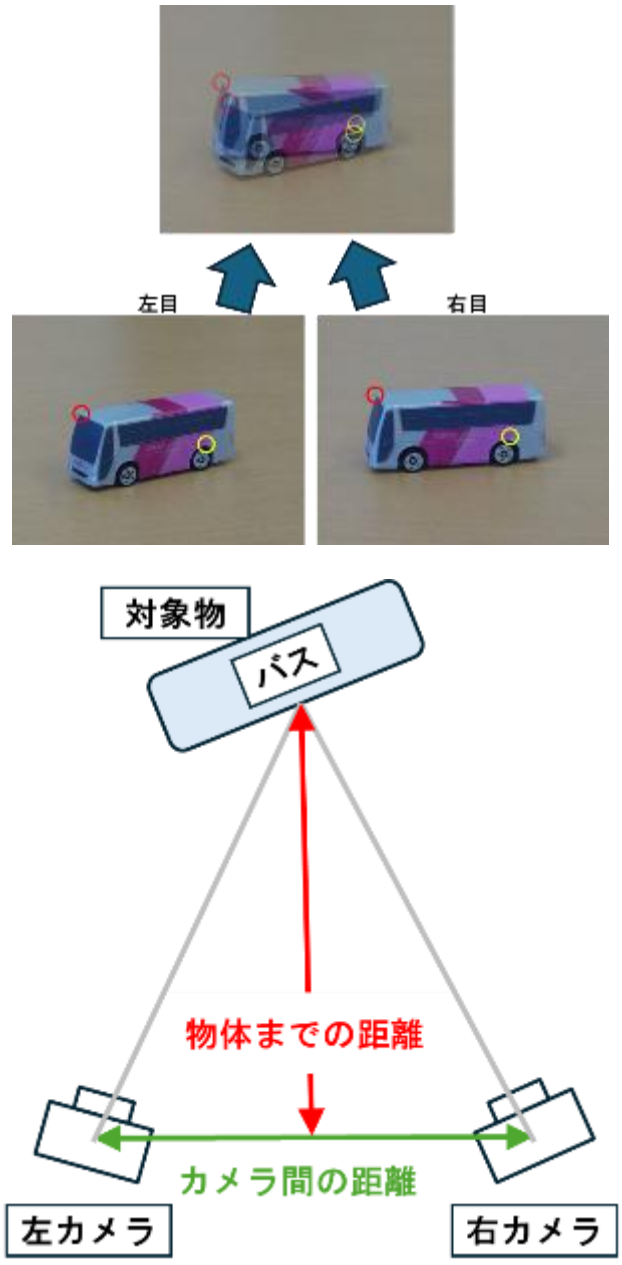

図１　ステレオ視のイメージ図

第 2 章

既往研究

本研究で調査対象とする「越波」とは、台風や暴風、高潮などの影響で発生した波浪が、海岸に設置された堤防などを乗り越えて港湾や陸地側へ流入する現象である。

この現象は、陸上構造物の破壊や塩害を引き起こすだけでなく、港湾内の急激な水位変動による船舶の転覆といった甚大な被害をもたらす。

2.1 越波の観測手法について

波浪や越波の観測や推定には、従来いくつかの手法が存在する。

1 つは、合田の越波流量算定図表などを元に越波量を推定する方法である(合田, 1975)。これは、モデルを用いて越波流量を算定しているものであり、実際の越波の観測に用いることはできない。

また、海面にブイを設置し、その変動の計測手法もある。狭い範囲でブイを細かく設置し、変動を計測する手法がある(Y Higuchi, 2023)が、本研究で対象とする数十 m 単位で打ちあがるような越波に対しては、ブイが一瞬で沈没したり流されたりするリスクがあり、越波の検出に活用することは困難である。

さらに、地上の 2 地点に設置したカメラ画像を利用した、ステレオ視という手法も存在する(有田ら, 2006)。ステレオ視は、 2 視点以上から撮影を行い、カメラの位置情報や焦点距離などのパラメータ、これらと画像内で特徴量抽出アルゴリズムなどを用いることで、同一点を探索し、解析するというものである。位置情報を正確に決めるために、地上に設置し撮影を行うものが主流だが、この手法だと、打ちあがる越波に対して撮影アングルが限定されてしまい、取得できる越波構造に死角が生じるという問題が発生する。

2.2 ステレオ視を用いた越波の観測手法について

Vieira ら(2025)は海面の変動を安価なカメラを 2 台設置し、ステレオ視によって海面の変動について、調査を行っている。ステレオ視を海面に用いることが可能であり、波の周期などの計測が可能であるということを調査したものであるが、本研究で想定している数十 m 規模の越波を対象としており、ステレオ視という手法によって海面の形状は取得可能であるが、陸上にカメラを設置すると越波に対して見上げる形となり、越波の構造を検出可能な範囲が限られる。そして、海岸に設置すると台風接近時などは越波の影響を受けて撮影が困難になる可能性もあるので越波を撮影するうえでこの手法と環境では難しい。越波の構造把握をするうえで、海岸にカメラ設置が可能な環境であるかといった周辺環境に影響を受

けずに撮影を行うためにも、UAV を用いて撮影を行うことが推奨される。

UAV を用いた波浪分析としては、三戸部ら(2019)があり、UAV 2 台を飛行させ、砕波後の波面に対して特徴量抽出アルゴリズムを用いて特徴点の検出とマッチングを行った。しかし、位置情報については撮影時の EXIF 情報を用いているということや、カメラの角度は考慮せずに計算を行っていることもあり、本研究のような強風条件での適用には工夫が必要である。

越波の構造を把握する手段として、ステレオ視を行うことは、有望な手法ではあるが、カメラの設置個所によって取得するデータが大きく左右されてしまうため、環境に依存しない UAV による撮影を用いたステレオ視を行う。

しかし、本来ステレオ視を行う上では、カメラの位置情報など、カメラパラメータ、画像のマッチングを元に、被写体を三次元化するという技術である。(**図 2**)しかし、本手法では、写真ではなく動画で撮影をしているため、カメラの位置情報が不足している。そのため、本研究では、不足しているカメラの位置情報をステレオ視の技術と位置情報の判明している海岸の静止物を用いることで逆推定することとした。実際には周辺の固定物に基準点を配置して行った。

そして、判明したカメラの位置情報と越波の写真を利用し、ステレオ視を行い、越波構造の把握を行うこととした。

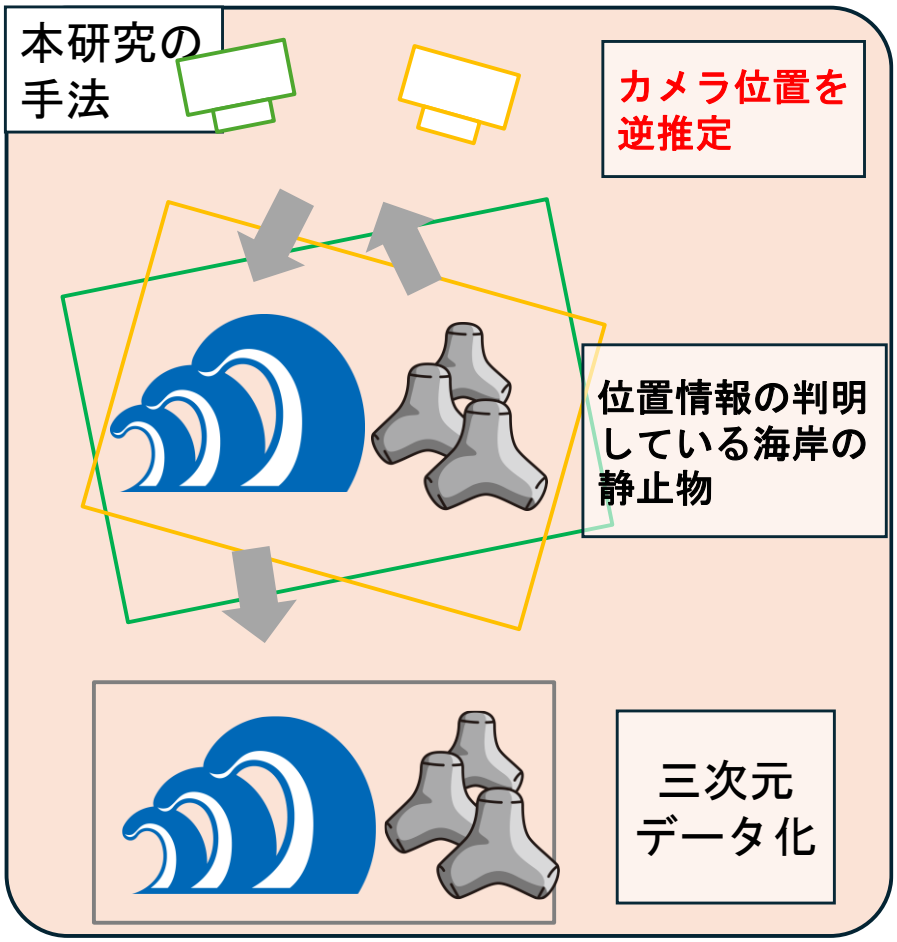

図３　本研究でのステレオ視の手法

第 3 章

現地調査

3.1 調査地の概要

現地調査は**図 4** に示した、高知県高知市高知港東における離岸堤周辺で実施した。同地点では、右の写真からわかるように、もともと潜堤のみが施工されていたが、潜堤の上に波消しブロックを積み上げて、離岸堤化する工事が行われている。本研究では、部分的に離岸堤化された開口部において間欠的に発生した越波を対象とした。

図 4　調査対象地点

3.2 現地調査 1　対象地域の詳細 DEM の作成

越波観測を行う周辺地形の形状を取得するための現地調査 2024 年 9 月 14 日に実施した。RTK-GNSS を搭載した UAV(DJI Phantom4 RTK+RTK2)によって越波の発生する離岸堤と周辺を撮影した。1198 枚の位置情報を含んだ写真を撮影した。撮影した箇所においては**図 5** に表示されている点のある箇所で撮影を行った。そして、SfM/MVS ソフトウェアの Agisoft Metashape Professional を用いて海面が静穏なときの離岸堤の形状を復元し、空間解像度 5.18cm/pix の詳細 DEM を作成した。1198 枚の写真を使用したが、海面も含め撮影されているため、DEM を作るうえですべてが使用できたわけではない。図の中で青と白の点があるが、青については DEM を作成時に使用されたもので、647 枚の写真を用いて**図 6** にある DEM を作成した。

作成された詳細 DEM を確認したところ、次に述べる台風通過時においても、離岸堤の波消しブロックは移動が見られず、離岸堤領域は固定物領域であると考えて良いことを確認した。

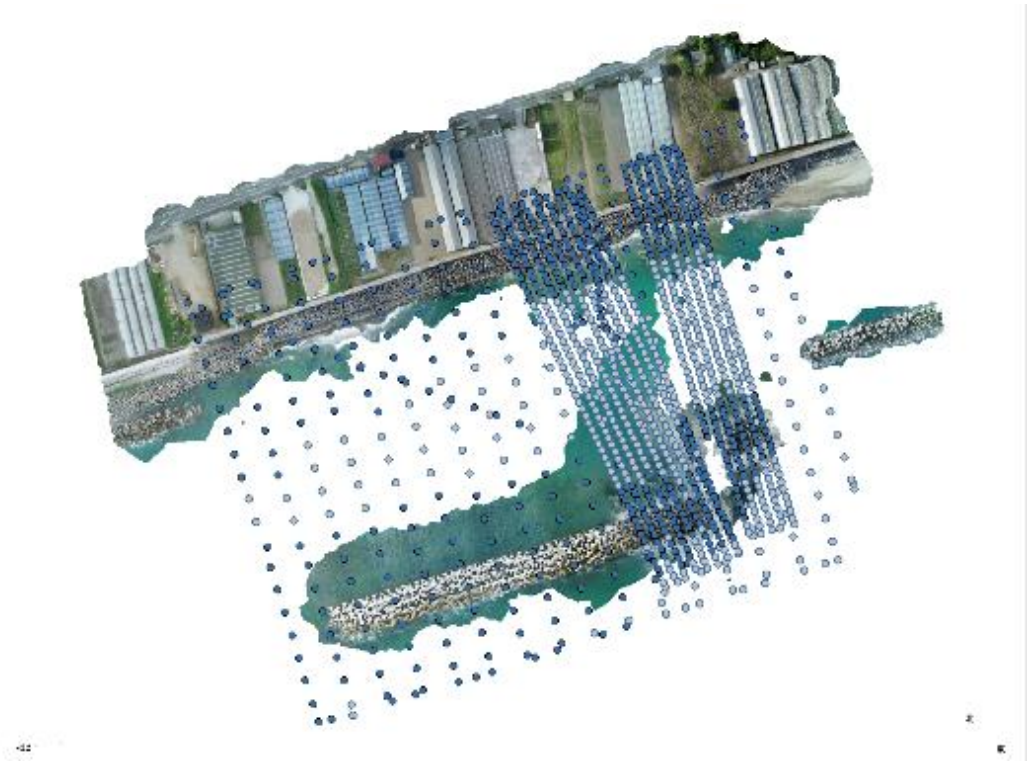

図 5　写真測量の実行エリア

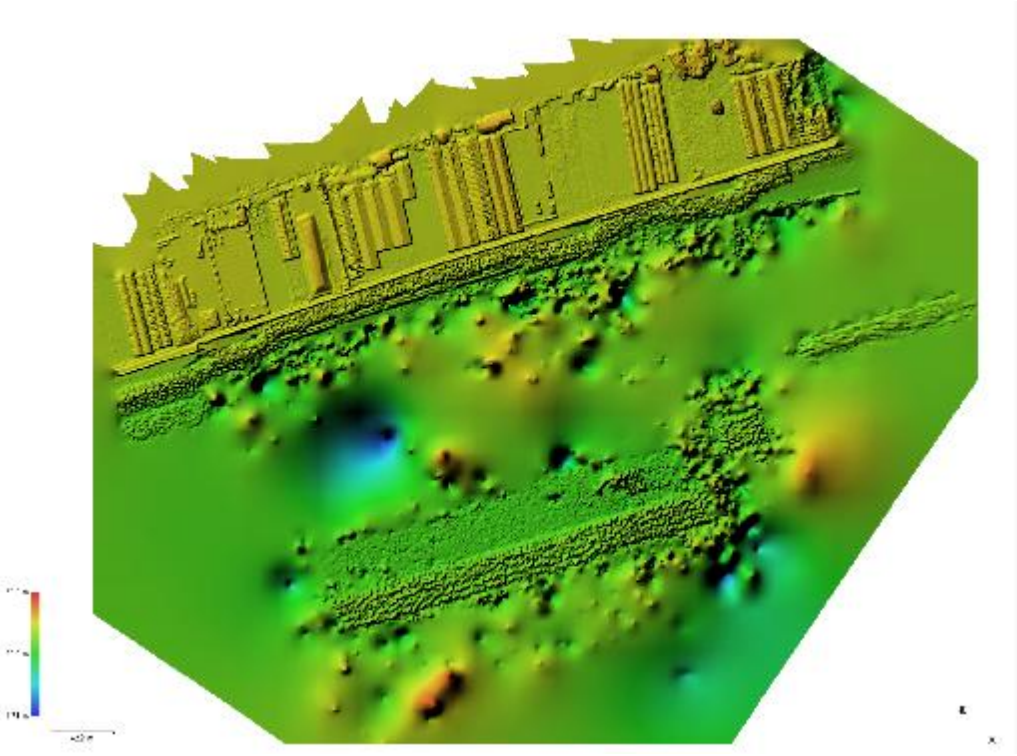

図 6　対象エリア DEM

3.3 現地調査２　越波の撮影

2024 年の台風１３号接近時において実際に発生した越波を 4K 動画撮影する現地調査を 9 月 15 日に実施した。RTK-GNSS(Real Time Kinematic-Global Navigation Satellite System)を搭載した UAV(DJI Phantom4 RTK 2 台、DJI Matrice 300 RTK 1 台) 3 台を使用し、３方向から越波の撮影を 4K 動画で撮影した。Phantom4 の 2 台は、高度約 40m から越波を見下ろす視点より 30fps のフレームレートで撮影した。Matrice の 1 台は、高度約 70m から海面に対し垂直に近い視点より越波を見下ろし、60fps のフレームレートで撮影した。それぞれの機体カメラは連続で 15 分間動画撮影を行った。**図 7** ,**図 8** が Phantom4 RTK に該当し、**図 9** が Matrice 300 RTK に該当する。

撮影は台風 13 号接近時に行い、撮影地点で発生した風速については、9:50~10:10 での最大瞬間風速を調べた。付近の気象庁の観測地点は南国日章と高知があったため、それらの最大瞬間風速を調べると 4.1m/s であった。ただし、実際の撮影環境は海上且つ上空であるため、UAV 機体と同行度の環境における風速はさらに速いと推測される。約 15 分間の動画撮影時間中において数回、UAV から強風に対する警告メッセージが発せられ、期待が強風にさらされていることが確認されたが、３台とも無事に着陸させることができた。

同時刻の有義波高については、高知港に設置されているナウファスのデータを参照した。[11]同日の 9:40~10:20 の有義波高は最大で 1.98m を示し、これは日常的にみられる穏やかな波ではないが、台風時の波高と比較すると中規模の波である。

図 7　越波撮影 UAV 1

図８　越波撮影 UAV２

図９　越波撮影 UAV３

第 4 章

ステレオ視による三次元復元

4.1 UAV 動画の時間同期

ステレオ視による 3 次元復元を行うには、3 台の UAV カメラによる撮影動画から時刻同期したフレーム画像をそろえる必要がある。そこで UAV を飛行させる直前と着陸直後に、同じ時計の映像をそれぞれ撮影しておき、秒数が切り替わるタイミングを基準に同期させた。Phantom4RTK は 30fps の動画であったため、撮影時刻差は 0.033 秒以内に収まっている。そして、それぞれの動画をフレーム画像に変換し、3 枚 1 組のほぼ同時刻にトリオ画像を分析対象とする。

4.2 UAV カメラ位置の逆推定

1. **基準点の設置**：作成した詳細 DEM 上の離岸堤領域において、位置情報の基準となる任意の点を選定し、その三次元位置情報を取得した。（**図１０**参照）
2. **画像上へのマーカー設置**：
   ・視認性：どのカメラアングルや画像においても観測可能であること。
   ・安定性：越波による隠ぺいや水没の可能性が低い場所であること。
   ・配置形状：すべての点が一直線上に並ぶと、点群作成時にその直線を軸とした回転方向への誤差が発生する可能性があるため、一直線上に並べることを避け、円や矩形を描くように点を配置した。
   手順 1 で定めた点に対応する位置を、フレーム画像上においても特定しプロットした。実際にプロットした基準点は 10 か所であり、上記の選定基準に基づき設置した。具体的な設置地点については**図１１,１２,１３**に示す。選定基準については、
   そして、この画像の各点に対して、位置情報のひもづけを行い、基準フレームとなる、トリオ画像を作成した。このトリオ画像を基準トリオと呼ぶ。
3. **マーカーの自動設置とカメラ位置の推定**：2.で作成した基準トリオ画像を用意し、カメラ位置を知りたい基準トリオ以外の画像を用意する。用意したトリオ画像を対照トリオと呼ぶ。そして、Metashape のソフトを用い、基準トリオの 3 枚と対照トリオの 3 枚で、アライメント処理を行った。この操作を行うことで、基準トリオを元に対照トリオの 3 枚に対して画像マッチングを行い、自動で位置情報の入ったマーカーが入力される。これにより、カメラの位置情報の推定が行える。**（図１４）**
4. **ループ処理**：3.の操作を越波の構造把握を行いたい全ての対照フレーム、対照トリオに置き換えて 3 の操作を繰り返すことで、すべての対照フレームで、カメラ位置の推定

を行う。(図１４参照)

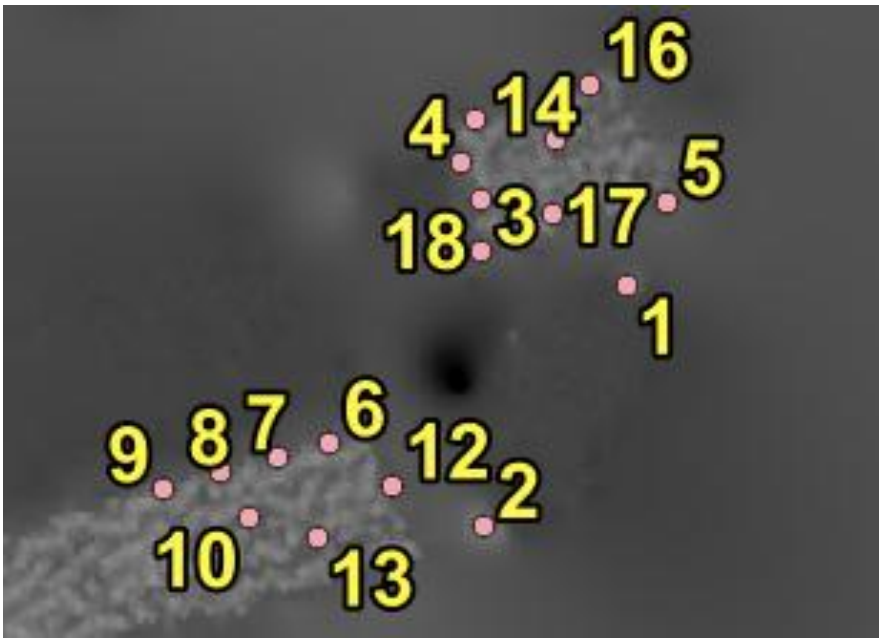

図１０ 基準点の設置個所

図１１ UAV１でのマーカー設置個所

図１２　UAV２でのマーカー設置個所

図１３　UAV３でのマーカー設置個所

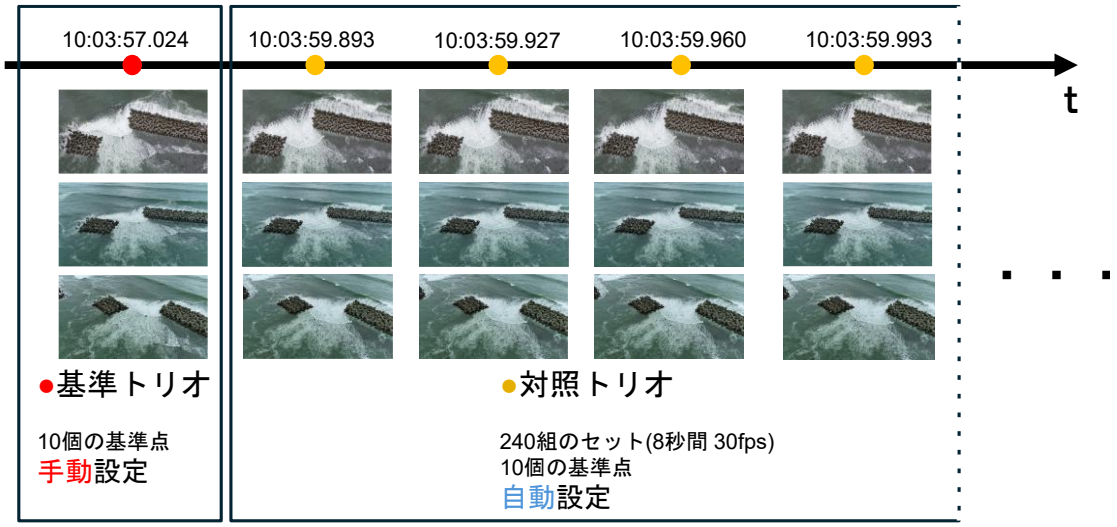

図１４　カメラ位置の推定手法

4.3 点群作成

4.2 でカメラ位置の判明したトリオ画像で、Metashape を用いて点群データを作成する。図のようにカメラの位置情報が判明し、特徴量アルゴリズムで同一特徴点を探索する。そして、トリオ画像内で判明した、1 つの同一特徴点を**図１５**のようにそれぞれ点を探索する。この操作をすべての特徴点へ繰り返すことで、越波の 3 次元を復元し、越波の点群を作成する。

次に、越波点群を時系列順に表示することで、越波の形状とその変化を動画として取得した。点群表示の際には、点の色定義を次の 2 つのパターンを用いて実行した。1 つ目が、**図１７**のように実物の色（RGB）をそのまま色の要素として取り込んだものであり、動画としても実際に越波を撮影したと同様の見た目で作成される。2 つ目が、点の標高値に応じて色付けしたものである。**図１８**のように海面や越波の高低差がわかるように作成した。これにより、打ちあがった越波がより強調されるため、取得したいデータである越波が視覚的に判別しやすい。

作成された点群は**図１７**から**図２２**までに添付する**図１７**と**図１８**は作成した点群の全体像である。そして、**図１６**には点群動画作成時の視点を示す。黄色の矢印方向からの視点で**図１９**と**図２０**を作成した。赤矢印の方向からの視点では**図２１**と**図２２**に作成した点群を示す。それぞれ越波発生前と越波発生時で、比較をすると越波発生時には明確に点群で赤い点が増えており、越波の打ちあがった様子が把握できた。

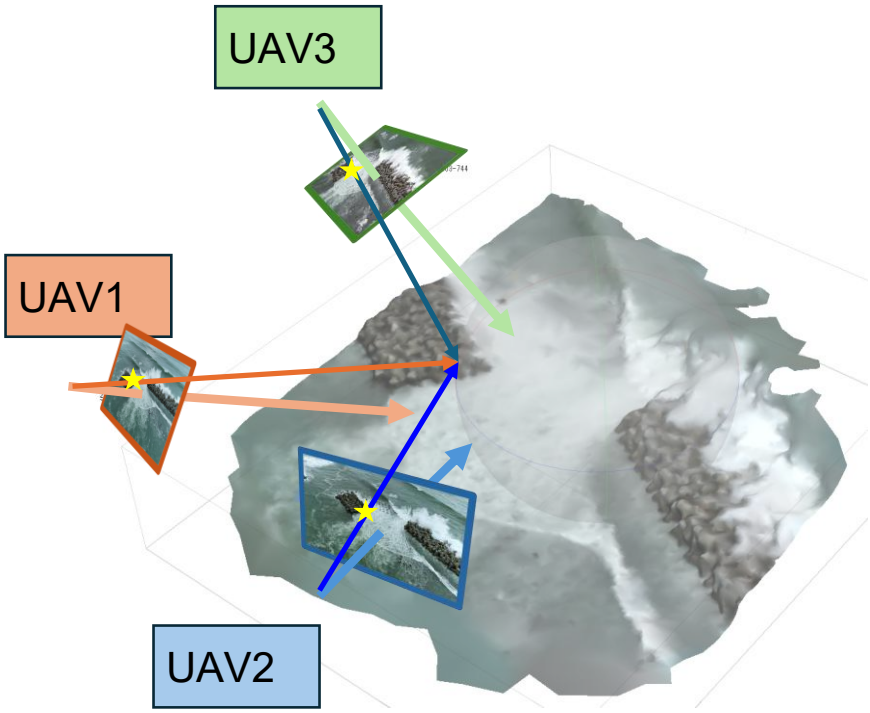

図１５　ステレオ視点群探索

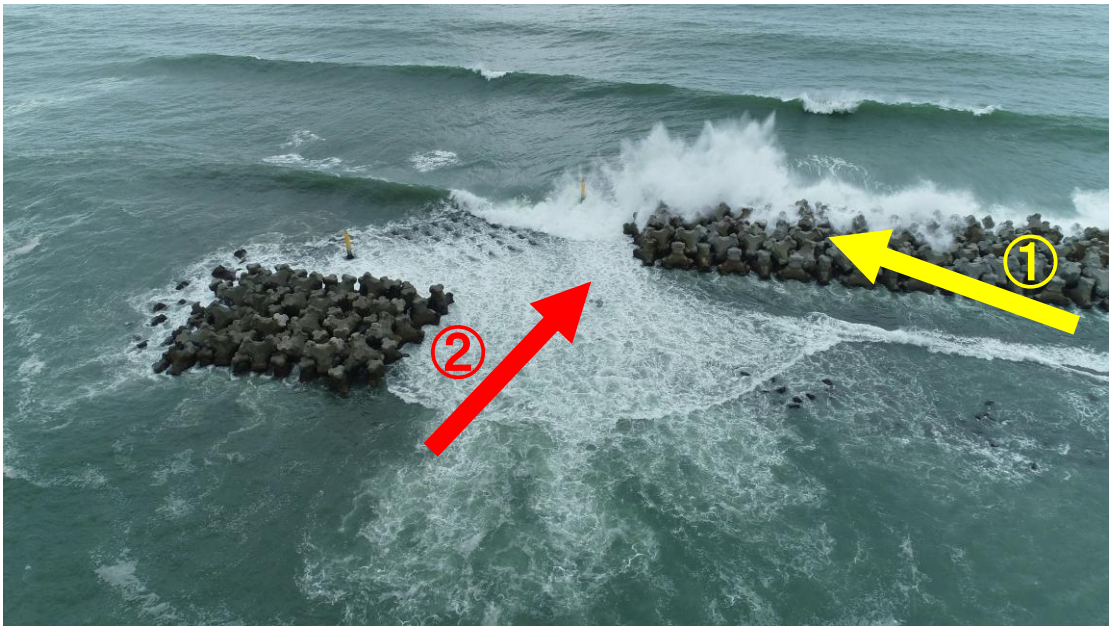

図１６　点群撮影時の撮影方向

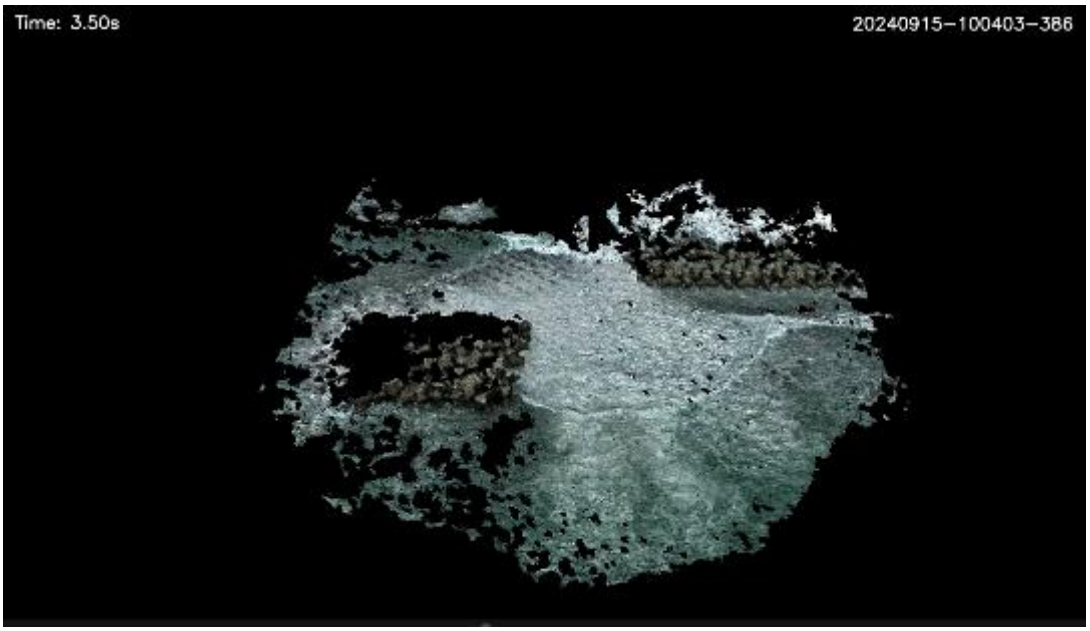

図１７ 全体の点群

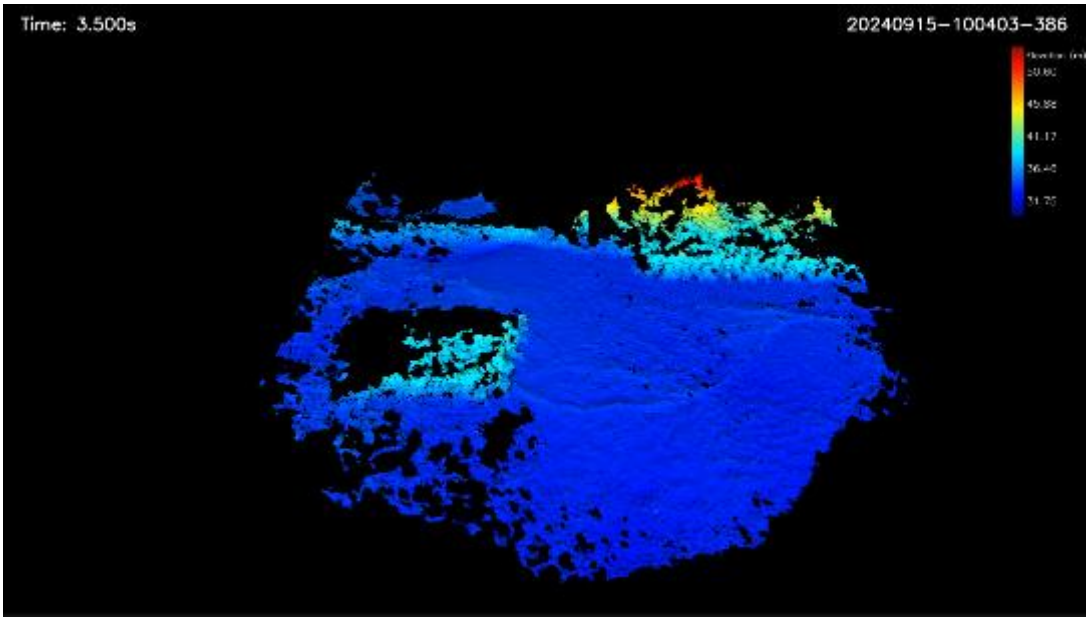

図１８ 全体の点群 高低差による色付け

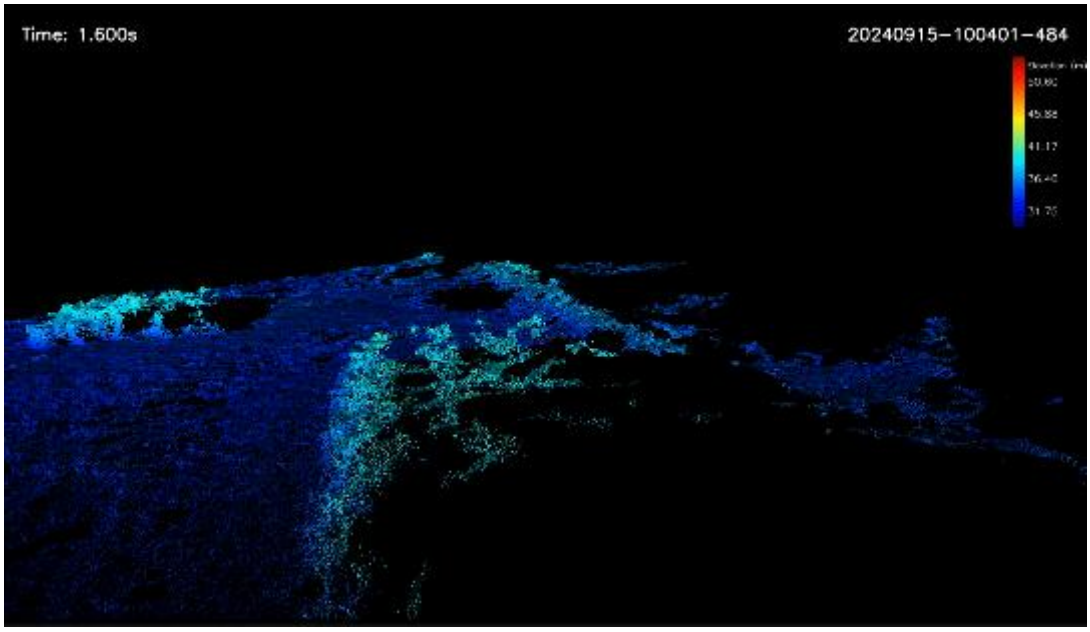

図１９　点群 視点① 平穏時

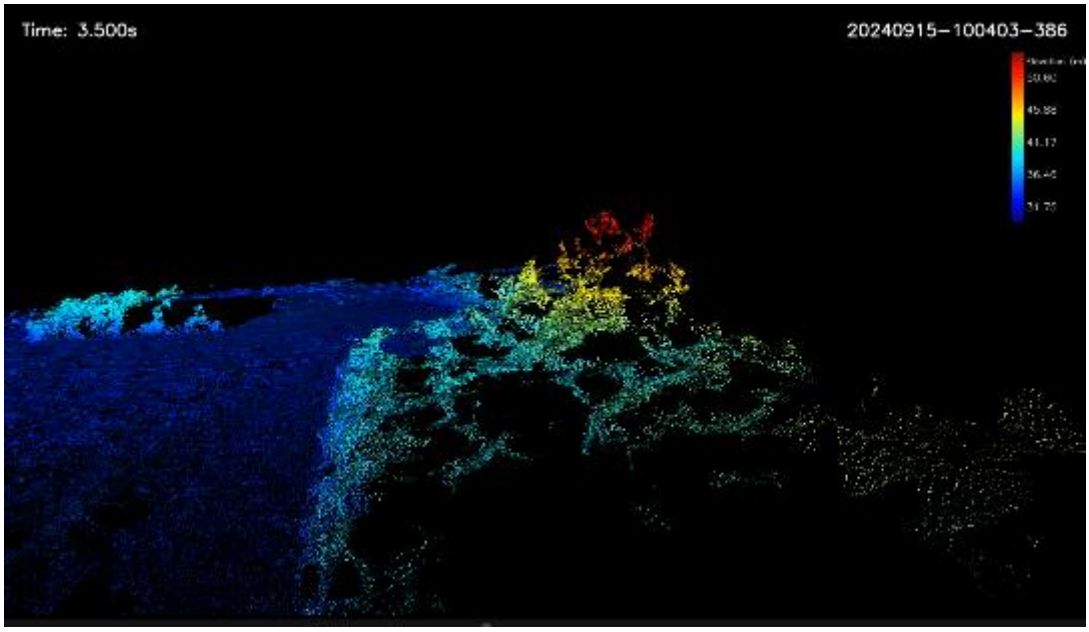

図２０　点群 視点① 越波発生時

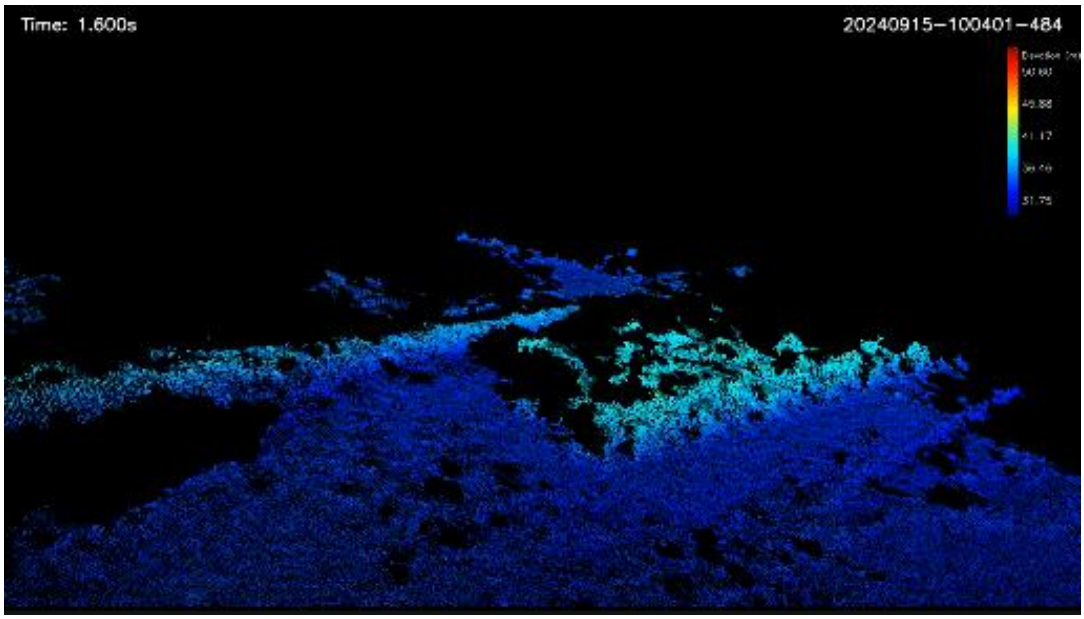

図２１　点群 視点② 平穏時

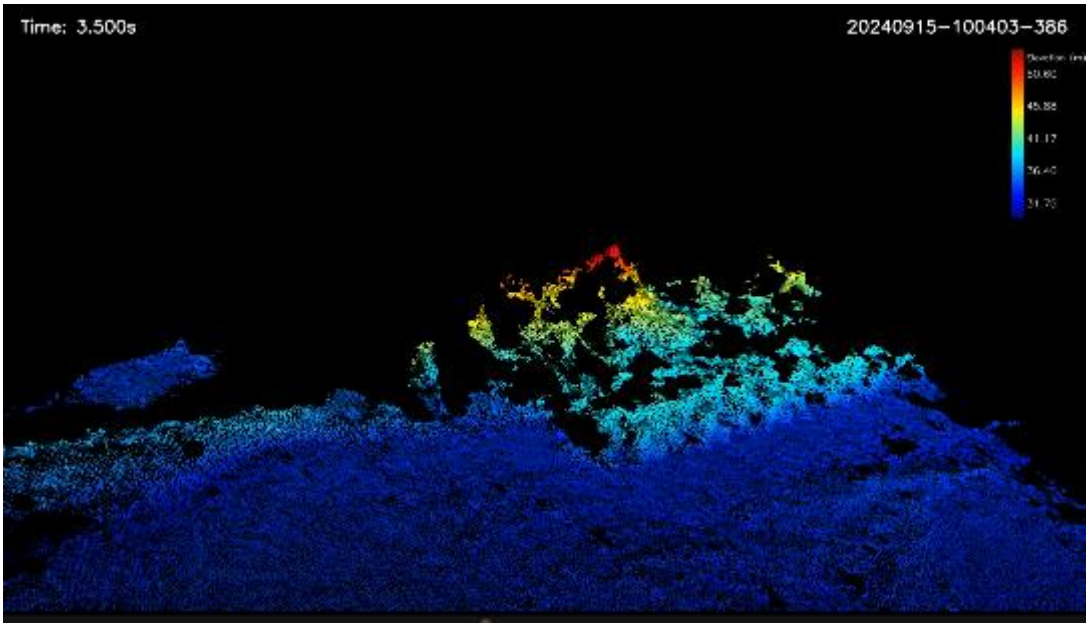

図２２　点群 視点② 越波発生時

第 5 章

結果と考察

5.1 UAV の位置推定に関する精度確認

フライトログと UAV の逆推定を 3 段階のアライメント精度「最高・高・中」のパラメータで推定を行い、位置情報を比較した。

図２３，図２４，図２５に示したグラフを比較すると、フライトログの結果はほとんど変動がなく、対象時間内で基本的には直線的な変化だった。つぎに、アライメントの精度を「最高・高・中」の三段階で比較を行った。

5.1.1 結果

パラメータごとの結果を表示する

・**「最高」**　　：Phantom4 で 1 m 程度、Matrice で 2 m 程度の範囲で移動が確認された。

・**「高・中」**　　：２つの精度の場合、ほとんど同じ UAV の位置情報を推定した。
Phantom4 で 1 m 程度、Matrice で 2 m 程度の範囲で移動が確認された。

・**「フライトログ」**：ほとんど移動が見られなかった。

UAV ごとの結果を表示する

・**「Phantom4」（2 台）**　　：「最高」「高・中」「フライトログ」の 3 グループに分かれる形で違う結果を示した。

・**「Matrice」（1 台）**　　：「最高・フライトログ」「高・中」の 2 つのグループに分けられる結果を示した。

フライトログを基準にすると、いずれの位置推定も Phantom4 で 1 m程度、Matrice で 2 m程度のばらつきに収まっている。それぞれの飛行高度が Phantom4 で 40m、Matrice で 70mであることを考慮すると、位置推定の誤差は小さい値に収まっていることが分かる。

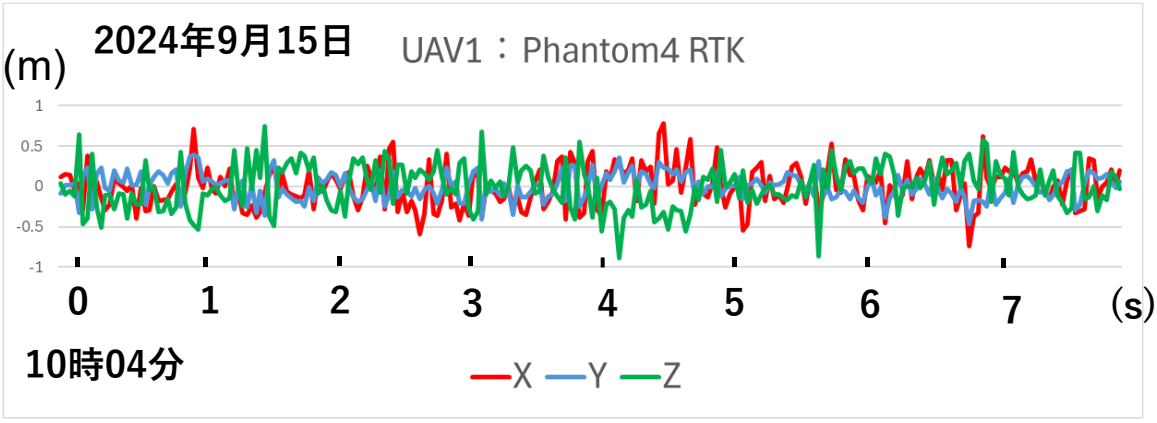

図２３　Phantom4 ① 位置推定結果

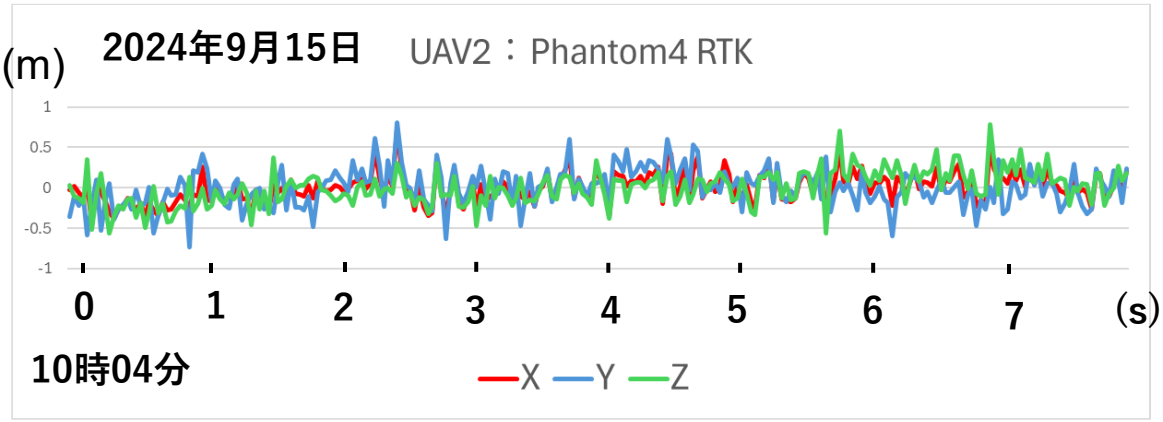

図２４　Phantom4 ② 位置推定結果

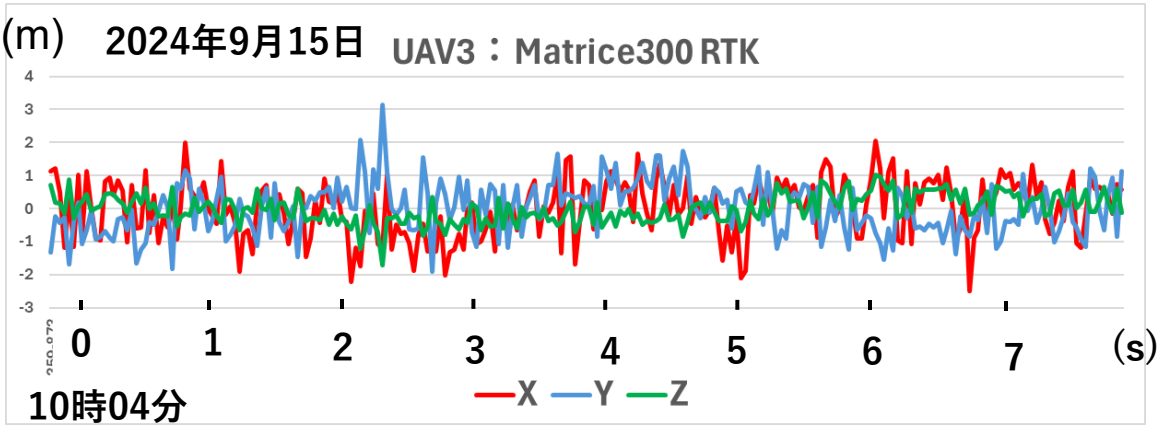

図２５　Matrice 位置推定結果

5.1.2 考察

アライメント精度「高・中」の際は、一貫して近い位置推定の値を示していたが、「最高」の際には、Phantom4 ではいずれの値とも異なる数値を出していたが、Matrice においては、フライトログと近い値を出していた。これは、アライメント精度を上げることで、かえって微細なノイズを拾うなどして推定結果がばらつき他の設定のときとは異なる値を示した可能性が示唆された。

5.2 点群作成パラメータの概略的な比較

本研究では 3 台の UAV を利用し、撮影を行ったが 3 視点から撮影を行う必要性と、越波を検出するうえで、適切な空間解像度を考察する。まず、Metashape による 3 次元復元過程で求められるアライメント精度と深度マップ品質パラメータについて検討する。アライメント精度は、同一点を探し画像マッチングを行う際に、画像解像度をアップまたはダウンスケーリングすることで、計算処理や適切な解像度に調整するものである。また、深度マップ品質は、画像マッチング後に点群などの三次元モデルを作成するうえでの解像度をダウンスケーリングする手法である。アップまたはダウンスケーリングするうえでの選択項目と解像度の関係については**表 1** に記載したとおりである。

まず、アライメント精度を「中」に設定し、深度マップ品質をすべてのパターンで実行し、目視による簡易比較を行った。そこで作成した点群を**図２ 6** に示す。左側の品質が高く、右側は低くなっている。

品質を「低」以下に下げすぎると点群の密度が粗くなり、海面や離岸堤の形状復元の精度が大きく下がり、パラメータとして適切ではないことがわかる。一方で、「中」以上については、検出できた範囲に違いがあるが、明確な違いは見られなかった。この時点では、越波構造を取得するうえでは深度マップ品質が中以上であることが適切であると判断される。

5.3 ステレオ視の視点数比較

アライメント精度を「高」に固定し、深度マップ品質を（1:最高, 2:高, 4:中）の 3 段階、２視点と 3 視点の計 6 通りで点群を作成した。そして、120 組の画像で点群を作成し、エラーを出さずに作成できた割合を比較する。

5.3.1 結果

UAV の台数が 2 台のときと 3 台のときを比較した、120 組のトリオ画像で点群データを作成し、エラーを出さずに作成できた割合と、処理の過程で作成されるタイポイントの数を

比較する。**図１６**に結果を示す。**表２**に示すように、台数が 2 台の時には、120 組の組のペア画像から点群を作成した際には、10 組前後が失敗し、同一点が見つけられず、画像のマッチングに失敗している。これに対し、3 台の UAV を使用し、120 組のトリオ画像から点群を作成すると、すべての組で問題なく、点群を作成することに成功した。

以上より、本研究の環境においてステレオ視に求められる視点の数は 3 つ以上、すなわち、3 台の UAV の画像すべてを用いた方が良いことが確認された。

5.4 アライメント精度・深度マップ品質比較

5.2 の比較手法と同様の方法で比較を行う。アライメント精度（0:最高, 1:高, 2:中）と深度マップ品質（1:最高, 2:高, 4:中）のそれぞれ 3 通りの組み合わせで 9 パターンでの比較を行った。120 組のトリオ画像で点群を作成し、エラーを出さずに作成できた割合を比較する。

5.4.1 結果

アライメント精度と深度マップの品質を調整することで、越波を検出するうえで適切なパラメータ精度を調査した。アライメントと深度マップを行う上で変更できるパラメータはそれぞれ 5 段階である。アライメント精度の最高品質の手法については、1 ピクセルを 4 ピクセルに拡張する技術を用いているが、それ以外の操作については、解像度を等倍または下げることを示している。

まず、エラーを出さずに作成できた割合を比較した。（**表２**）深度マップ品質を変更しても出力が成功した割合はあまり変わらなかったが、アライメント精度を変更すると、出力に成功した割合が大きく変わった。そのため、出力ができるかどうかについては、アライメント精度に依存することが判明した。

出力に成功した 6 パターンのすべてで越波が打ちあがったタイミングの点群を作成し、比較した。海面と越波のそれぞれが検出できているかを比較すると、図の中で左 2 つにおいては海面に隙間がなく抽出できている。加えて越波に関しても打ちあがっている越波が図の中で左 2 つでは越波についてもより広範囲で取得できている。

ただし、パラメータを下げることは解像度を下げることと同義であり、全体的にデータを取得するうえではパラメータを上げすぎない設定が推奨されるが、越波の細かな形状を取得するうえでは、パラメータを上げる必要がある。

そして、最高でも最低でもないパラメータで点群データが精度よく取得可能な理由としては、初めに作成した DEM 地上解像度に依存していると推定される。DEM の地上解像度は 5.18cm/pix であり、上記で判明した最適なパラメータの際には、地上解像度は 4-8cm/pix の地上解像度となり、DEM の値とおおむね一致していることがわかる。

表１　処理解像度一覧

アライメント精度	処理解像度		深度マップ品質	処理解像度
0：最高	４倍解像度		1：最高	等倍解像度
1：高	等倍解像度		2：高	1/4 解像度
2：中	1/4 解像度		4：中	1/16 解像度
4：低	1/16 解像度		8：低	1/64 解像度
8：最低	1/64 解像度		16：最低	1/256 解像度

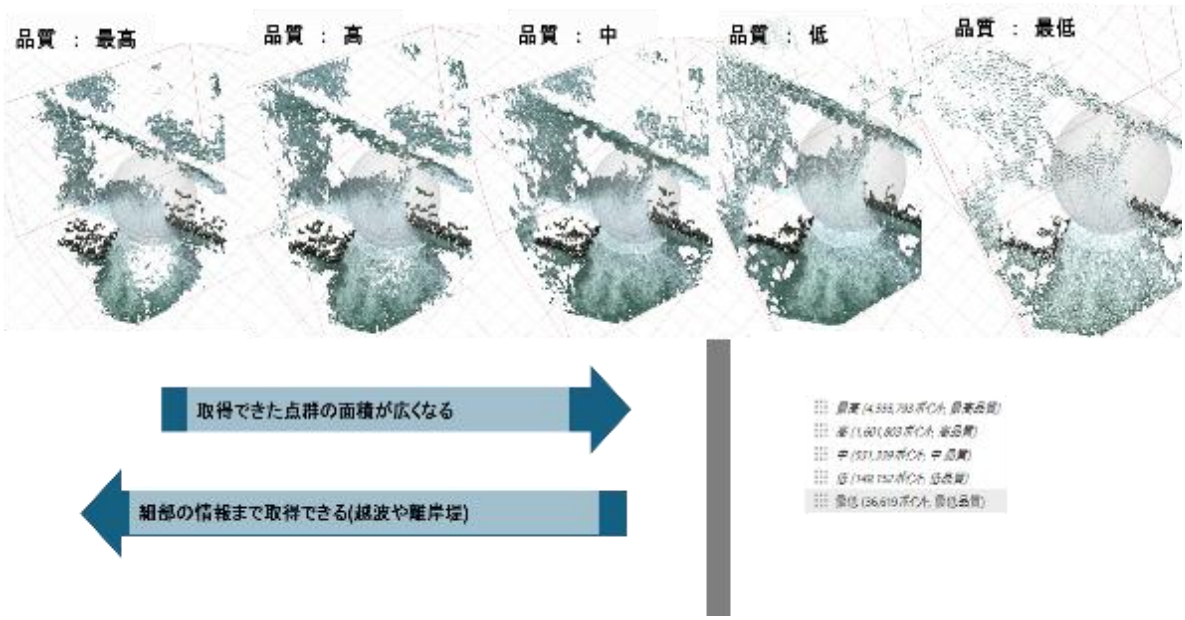

図２６　簡易パラメータ比較

5.5 ボクセル化による越波の詳細構造

点群をボクセル化することで越波の空間分布を定量的に分析した。ボクセル化を行うことで取得したいデータは海面の変動ではなく、越波構造の把握のために行っているため、海面部分のデータは検討の対象としない。

ボクセル化では、空間を指定した立方体に区分けする。今回は、数十 cm から数 m 単位の越波水塊を特に対象としていたため、1m x 1m x 1m のブロックに区分けした。そして、海面変動の影響を取り除くため、海面高より少し高い基準を定め、それ以上の空間をブロッ

クに区分けする。そして、区分けした空間内に点が存在するか否かで、評価する。

ボクセルの色付けに関しては、点を検出したボクセルの高度に応じて色付けを行うパターン（**図２７**）と、ボクセル内の点群の総数によって色付けを行うパターン（**図２８**）の２通りで比較を行った。

表２ パラメータ・視点数の比較結果

ファイル名	アライメント精度	タイポイント(平均個数 ※)	深度マップ品質	枚数	出力数(/120)
20251110_3	高	3327.4	中	3	120
20251110_4	高	3734.8	高	3	120
20251110_5	高	3573.2	最高	3	120
20251110_6	高	819.2	最高	2	111
20251110_7	高	790.2	高	2	109
20251110_8	高	892.8	中	2	106
20251111_1	最高	2073.2	最高	3	81
20251111_2	最高	2302.6	高	3	78
20251111_3	最高	1999.2	中	3	81
20251111_4	中	3518.8	中	3	120
20251111_5	中	3189	高	3	120
20251111_6	中	3321.8	最高	3	120

5.5.1 結果

図１６で示した２方向の視点から越波を観察する。**図１６**内の①の黄色矢印方向の視点を**図２９**と**図３０**に、②の赤矢印方向の視点を**図３１**と**図３２**に表示する。そして、表示している画像は**図２７**から**図３２**まで同じ瞬間を表示しており、越波が発生し最高点に到達した瞬間を切り抜き表示している。

点群のみでは体積などの数値データに変換できないため、ボクセルデータにすることで、1 ㎥のブロックがいくつあるかを調べる。それにより、越波の体積密度を推定することが可能になる。そして、各ボクセル内に含まれている点の総数から、越波流量を推定することの実現可能性を示す。

図２９、図３１を詳しく見ると、越波水塊は海面から最高点まで Z 方向に連続して分布するのではなく、密度の高い領域が不連続なパッチ状に網目構造を形成していることがわかる。その領域は、XY 方向に連続した領域を形成していることもわかる。このような越波

水塊の構造は、既往研究で指摘された例はなく、世界で初めて本研究で発見された構造である。

5.5.2 考察

Z 方向の連続した分布ではなく、パッチ状の越波が検出できた理由を考察する。越波が発生し、消滅する中での段階による違いだと推測される。越波が発生した直後は、ほとんどが水だけで構成された越波であり、途中からは打ちあがった波が砕けて空中で空気を含みながら離散する。

今回ボクセル化したデータでは、越波の初期段階ではなく離散する越波を検出したためパッチ状の越波が確認できたと推測する。また、今回の越波高さが１０m 程度と小規模であったため、離散するまでの時間が短く、低密度の空気を多く含んだ状態が検出できたと推測される。以上のことから、検出された成分は以下のように分類する。

・高密度成分：主に水の流れ, Z 方向に連続し繋がりが見える
・低密度成分：空気も多く含んだ水しぶき, パッチ状、網目状または不定形

Z 方向の連続した分布ではなく、パッチ状の越波が検出できた理由を考察する。まず、越波を 2 つに分類する。1 つ目は越波発生直後の空気をあまり含まない、高密度な越波であり、Z 方向に連続した繋がりがある。2 つ目は越波が打ちあがり、空気を含みながら空中で拡散する段階にある、低密度な越波であり、パッチ状または網目状にも見え不定形である。これらのうち、越波流量のシミュレーションや既存のステレオ視を用いて計測できていたものは高密度な越波であったが、本研究で観測に成功したものは 2 つ目の低密度な越波であり、パッチ状の越波として観測された。

これにより、いままでの手法では観測できていなかった、空中で拡散するような不連続な越波形状に対して計測が可能になった。

Z 方向の連続した分布ではなく、網目状の越波が検出できた理由を考察する。まず、離岸堤について、積み方が整積みであり、規則的に積まれている。そのため、離岸堤にあたり打ちあがる越波に関しても規則的に打ちあがる。その結果、各離岸堤にあたった越波が、左右に分かれることにより、周期的に越波に網目状の形状が形成されると推測される。

加えて、本研究で検証された越波は、低密度な越波のみである。発生した越波が低密度な越波のみであった理由については、越波高さが１０m程度と離岸堤に対して小規模であったため、離散するまでの時間が短く、低密度な越波のみが観測されたと推測される。

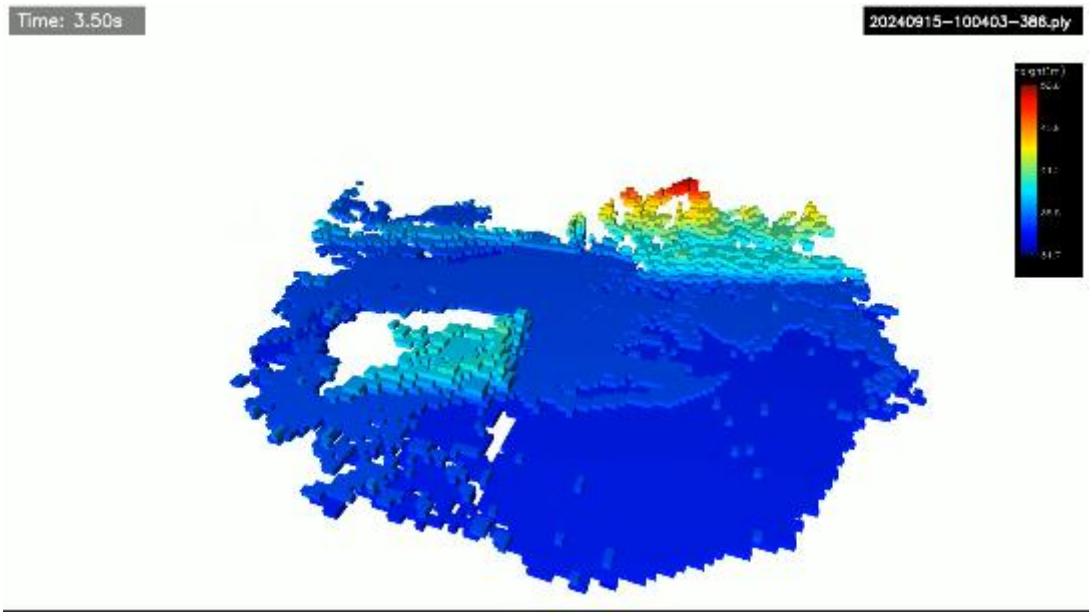

図２７　ボクセル 全体 高低差による色付け

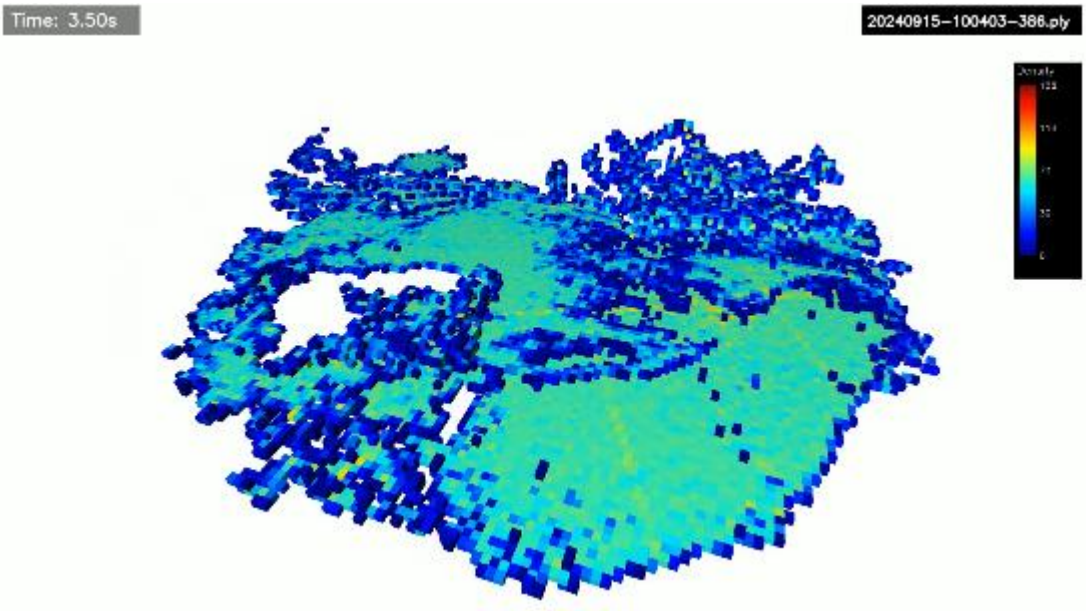

図２８　ボクセル 全体 点群密度による色付け

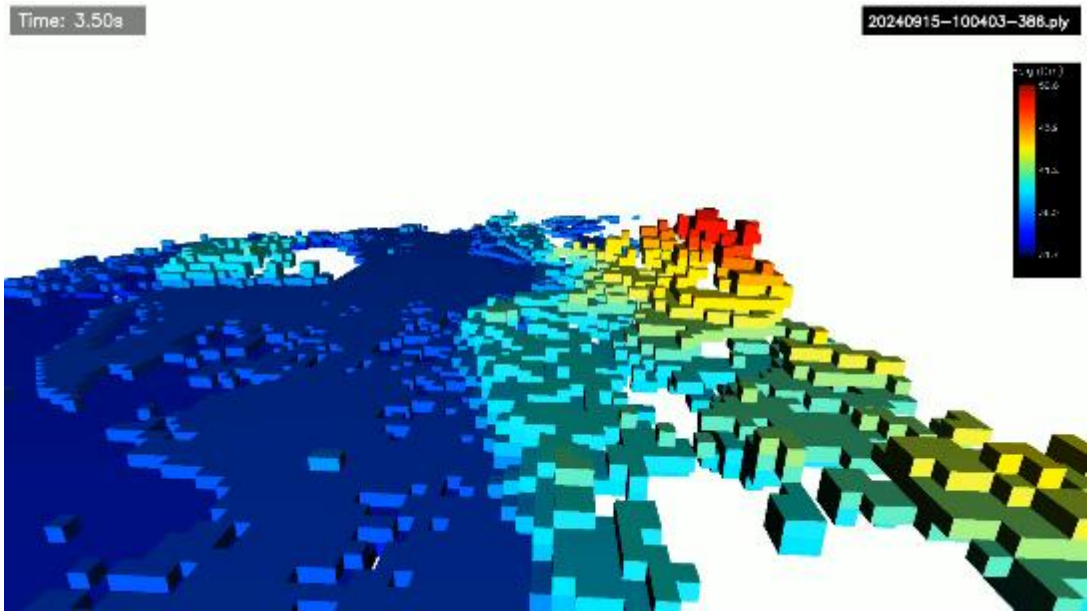

図２９　ボクセル 視点① 高低差による色付け

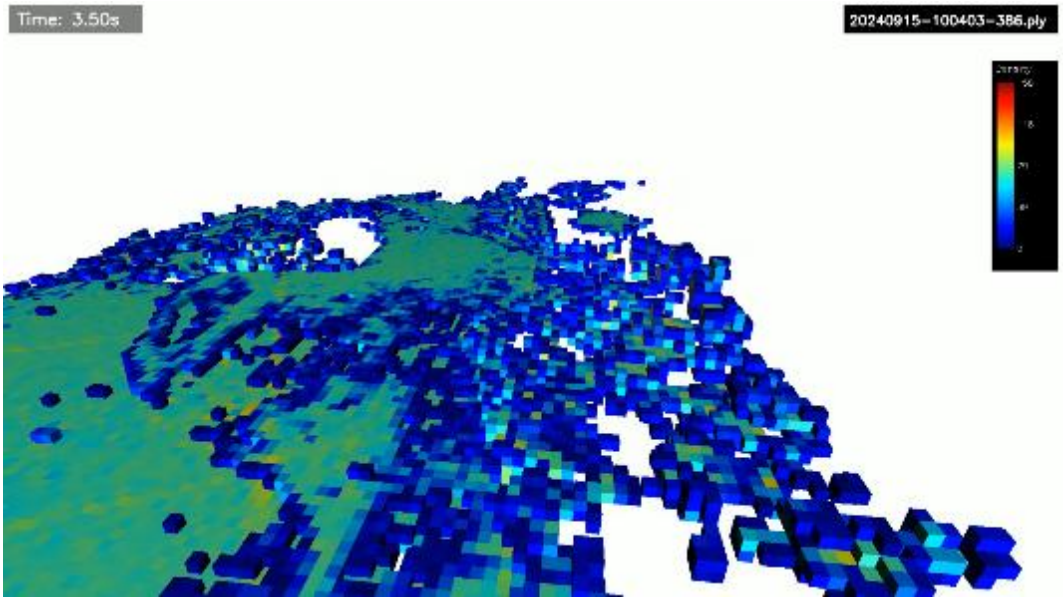

図３０　ボクセル 視点① 点群密度による色付け

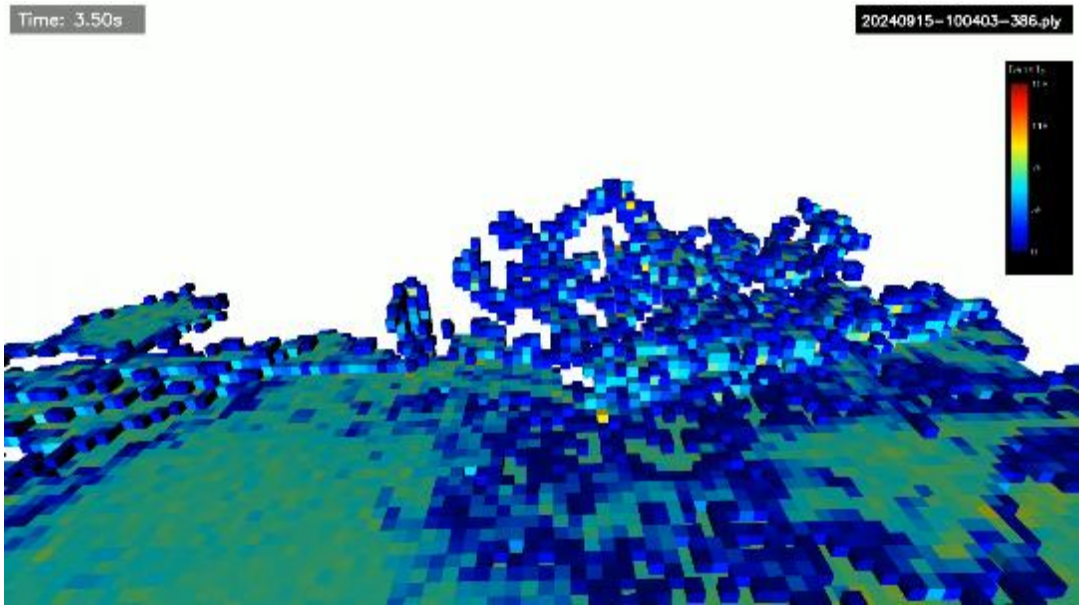

図３２　ボクセル 視点② 点群密度による色付け

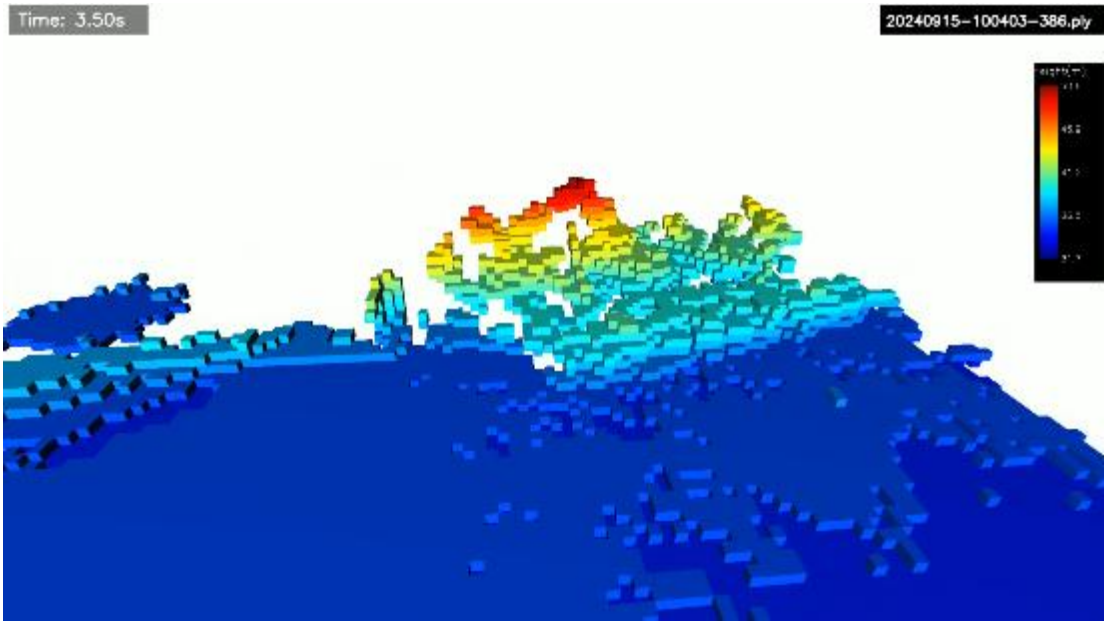

図３１　ボクセル 視点② 高低差による色付け

第 6 章

結論

6.1 結論

本研究では、不定形で変動の大きい越波水塊に対し、複数台の UAV を同期したステレオ視によって越波の四次元（三次元空間+時間）構造を計測する手法を開発した。海岸の強風条件で同期撮影した動画に対して、フレームごとのカメラ位置と姿勢を精度良く推定する手法を提案し、SfM/MVS 分析での点群作成に最適な空間解像度を決定した。そして、四次元による観測によって、時間変化による、越波の構造の変化について、現時点で把握は行えていないが今後推定する手法としては有用であると考えられる。

主要な結論は次のとおりである。

(1) 海岸の強風条件で撮影した動画では、UAV の動揺によりフレームごとのカメラ位置やカメラ姿勢は小刻みに変動するが、これらの諸情報はフレーム単位では記録されていない。高精度の四次元計測に不可欠となるフレームごとのカメラ位置・姿勢は、基準フレームの静止構造物上に手動設置し、対照フレームには自動設置されるマーカーから逆推定できることを示した。

(2) SfM/MVS 分析に用いる UAV の台数は、ステレオ視分析に最低必要な 2 台では、点群作成率が 90%程度になるのに対し、3 台の UAV を用いることですべての時刻で点群が 100%作成できることが確認された。これにより、越波構造の計測には、３視点以上からのステレオ計測が必要であることを示した。

(3) SfM/MVS 分析においては、特徴点探索・マッチングに用いる空間解像度と深度マップの品質処理に関して適切なパラメータを選定することが重要である。本研究で対象とした、高さ１０m規模の越波および海面の三次元復元では、空間解像度４～８cm 程度とするのが最適であることを見出した。

(4) この手法によって、従来では困難であった動的かつ不定形な越波水塊の挙動を、非接触で詳細に可視化・計測が可能であることを示した。

6.2 今後の課題

本研究では、高さ 10m程度の越波に対して、3 台の UAV で機動的に動画撮影することで、その詳細構造を分析できることが示された。また、越波領域のボクセル分析により、越波水塊がパッチ上に分散して分布していることが判明した。これらの情報を時間軸上で追跡することにより、越波量など工学的に重要な情報を定量的に評価することが今後の課題

である。また、越波の発生状況は、波浪、構造物の条件によって多様に変化するため、観測事例を増やして、本手法の汎用性を確認することも今後の課題である。

参考文献

[1]合田良実，岸良安治，神山豊：不規則波による防波護岸に関する実証研究，港湾技術研究所報告　vol.14 No.4 DEC 1975

[2]Yuya Higuchi, Hidetaka Houtani, Rodolfo T. Goncalves, Yasuo Yoshimura, Shinichiro Hirabayashi, Hideyuki Suzuki, Hideo Orihara：Stereo Reconstruction Method for 3D Surface Wave Fields around a Floating Body Using a Marker Net in a Wave Tank , JMSE/Volume11/ Issue9/10.3390/jmse11091683

[3]有田守，出口一郎：ステレオ画像を用いた波面計測　可視化情報/Vol.26/Suppl./No.2

[4]Matheus Vieira, C.Guedes Soares, Pedro V.Guimaraes, Flippo Bergamasco, Ricardo M.Campos：Nearshore space-time ocean wave observation using low-cost video cameras Coastal Engineering /Volume 197/15 April 2025/104694

[5]三戸部佑太，新道健人，鈴木彰容，田中仁：二台の UAV によるステレオ画像を用いた波浪観測手法の基礎的検討　土木学会論文集 B2(海岸工学),Vol.75, No.2, I_1273-I_1278, 2019

[6]上谷大陽，鈴木直弥，池田篤俊：海洋での波情報計測に向けた IMU センサ搭載小型ブイの開発　土木学会論文集 B2(海岸工学), Vol. 78, No.2, I_121-I_126, 2022

[7]中世古蓮汰：紀南管内における越波防止による安全対策について　一般部門(安全・安心)I:No.04

[8]田中聡，仲座栄三，福森匡秦，宮里信寿，Carolyn SCHAAB：規則波を用いた直立護岸上の越波流量に関する研究　土木学会論文集 B2(海岸工学), Vol.77, No.1, 40-54, 2021

[9]岡安徹也：高知県菜生海岸における被災事例調査　JICE REPORT vol.8/05.11 17

[10]榊山勉，今井澄雄：消波護岸の越波に関する数値シミュレーション　海岸工学論文集第 43 巻（1996）

謝辞

本論文を執筆するにあたり、多くの方々に多大なるご支援とご協力を賜りました。ここに心より感謝の意を表します。

まず、指導教員である佐藤愼司先生（高知工科大学教授）には、研究の方向性から論文の執筆に至るまで、終始懇切丁寧なご指導とご鞭撻を賜りました。深く感謝申し上げます。

共同研究者である村井亮介様（高知大学特任研究員）には、UAV やカメラの運用技術に関する詳細なご助言や、現地観測における多大なるご協力をいただきました。厚く御礼申し上げます。

また、本論文の作成にあたり、副指導教員として有益なご助言をいただきました鈴木卓先生（高知工科大学准教授)、ならびに副審査員として貴重なご意見を賜りました高木方隆先生（高知工科大学教授）に、心より感謝申し上げます。

研究室の同窓生の皆様、ならびに大学の友人の皆様には、研究に関する議論のみならず、日々の会話を通じて多くの刺激と励ましをいただきました。精神的な支えとなってくださったことに深く感謝いたします。

最後に、大学院生活を温かく見守り、精神的にも経済的にも支えてくれた家族、そして本研究に関わってくださった全ての皆様に、心からの感謝を申し上げます。

解析コード

コード１

4.2 から 4.3 を実行するための python コード
コード作成には、Google Gemini を活用した。

```
# -*- coding: utf-8 -*-

import Metashape
import os
import re

# ---------------------------------------------------------------------------
# ---  ユーザー設定項目  ---
# ---------------------------------------------------------------------------

# Metashape プロジェクトファイルのパス
project_path = "D:¥¥IkemotoKazuma¥¥metashape¥¥20260108_1.psx"

# 基準となるチャンクの名前
BASE_CHUNK_NAME = "original"

# 各ドローンの最初の画像ファイルのパス
phantom1_initial_path = r"D:¥UAV¥Phantom4RTK_1¥FrameImage-DJI_0169¥Phantom4_20240915-100359-849.jpg"
phantom2_initial_path = r"D:¥UAV¥Phantom4RTK_2¥FrameImage-DJI_0032¥Phantom4_20240915-100359-860.jpg"
matrice_initial_path = r"D:¥UAV¥Matrice300RTK¥Frame-DJI_20240915095232_0004¥Matrice300_20240915-100359-840.jpg"

# 認識する画像ファイルの拡張子
IMAGE_EXTENSIONS = ('.jpg', '.jpeg', '.png', '.gif', '.bmp', '.tiff')

# 処理する画像の最大数（初期画像を除く）
```

```
MAX_IMAGES_TO_PROCESS = 240

# 【変更】1セットのアライメントに追加する「各ドローンごとの」画像枚数
# ここを 2 に設定すると、3台 × 2枚 = 合計6枚 追加されます。
# 以前は 5 でした。
IMAGES_PER_SET_PER_DRONE = 1

# スキップ間隔の設定
# Phantom 1, 2 はこの間隔でスキップ
BASE_SKIP_INTERVAL = 1
# Matrice はこの間隔の2倍でスキップ
MATRICE_SKIP_MULTIPLIER = 2

# Metashape 処理の品質設定
# アライメント精度（downscale): Highest=0, High=1, Medium=2, Low=4, Lowest=8
ALIGNMENT_DOWNSCALE = 2        # アライメント精度（1 = High）

# デプスマップ品質（downscale): UltraHigh=1, High=2, Medium=4, Low=8, Lowest=16
DEPTH_MAPS_DOWNSCALE = 4       # デプスマップ品質（1 = Ultra High, 2 = High）

# ベースとなる出力先ディレクトリ
BASE_OUTPUT_DIRECTORY = "D:¥¥IkemotoKazuma¥¥output¥¥20260108-1"

# 点群データ(.ply)の出力先ディレクトリ（ベースディレクトリを基準にする）
OUTPUT_POINTCLOUD_DIRECTORY = os.path.join(BASE_OUTPUT_DIRECTORY,
"Pointcloud")

# ----------------------------------------------------------------------
# --- ヘルパー関数 ---
# ----------------------------------------------------------------------

def collect_drone_images(initial_path, max_limit, extensions, skip_interval=1):
    """
    指定されたフォルダから画像ファイルを時系列順に収集し、リスト化します。
    """
    base_dir = os.path.dirname(initial_path)
```

```
    images_to_return = []
    all_files_with_datetime_part = []

    print(f"'{base_dir}' から画像を収集しています...")

    try:
        if not os.path.isdir(base_dir):
            print(f"警告: ベースディレクトリ '{base_dir}' が見つかりません。")
            if os.path.exists(initial_path):
                images_to_return.append(initial_path)
            return images_to_return

        for item_name in os.listdir(base_dir):
            full_item_path = os.path.join(base_dir, item_name)
            if os.path.isfile(full_item_path) and item_name.lower().endswith(extensions):
                # ファイル名からタイムスタンプ部分を抽出（例: 20240915-100359）
                match = re.search(r'_(¥d{8}-¥d{6})', item_name)
                if match:
                    datetime_part = match.group(1)
                    all_files_with_datetime_part.append((datetime_part, full_item_path))

        all_files_with_datetime_part.sort(key=lambda x: x[0])

        start_index = -1
        for i, (_, path) in enumerate(all_files_with_datetime_part):
            if path == initial_path:
                start_index = i
                break

        if start_index != -1:
            images_to_return.append(all_files_with_datetime_part[start_index][1])
            # 開始ファイル以降をスキップ間隔で収集
            current_idx_to_add = start_index + skip_interval
                while len(images_to_return) < max_limit and current_idx_to_add <
len(all_files_with_datetime_part):
                 images_to_return.append(all_files_with_datetime_part[current_idx_to_ad
```

```
d][1])
                    current_idx_to_add += skip_interval

        elif os.path.exists(initial_path):
            images_to_return.append(initial_path)
            print(f"警告: 指定された開始ファイル '{initial_path}' は時系列ソートリストに含まれませんでした。")

    except Exception as e:
        print(f"エラー: '{base_dir}' から画像を収集中にエラーが発生しました: {e}")

    print(f"収集完了: {len(images_to_return)} 個の画像")
    return images_to_return

# ---------------------------------------------------------------------
# --- メイン処理 ---
# ---------------------------------------------------------------------

def main():
    """
    Metashape 自動処理のメイン関数
    """
    print("スクリプトを開始します。")

    # スクリプトが使用している Metashape のバージョンを確認して表示
    try:
        metashape_version = Metashape.app.version
        print(f"¥nMetashape ライブラリのバージョン: {metashape_version}")
        version_tuple = tuple(map(int, metashape_version.split('.')))
    except Exception as e:
        print(f"¥nMetashape のバージョンを取得できませんでした: {e}")
        return

    # --- 1. 画像リストの準備 ---
    print("¥n--- ステップ 1: 画像リストの生成 ---")
    total_limit = MAX_IMAGES_TO_PROCESS + 1
```

```
        phantom1_images = collect_drone_images(phantom1_initial_path, total_limit, IMAGE_EXTENSIONS, skip_interval=BASE_SKIP_INTERVAL)
        phantom2_images = collect_drone_images(phantom2_initial_path, total_limit, IMAGE_EXTENSIONS, skip_interval=BASE_SKIP_INTERVAL)
        matrice_images = collect_drone_images(matrice_initial_path, total_limit, IMAGE_EXTENSIONS, skip_interval=BASE_SKIP_INTERVAL * MATRICE_SKIP_MULTIPLIER)

    proc_p1 = phantom1_images[1:]
    proc_p2 = phantom2_images[1:]
    proc_m = matrice_images[1:]

    # 【修正】セット数の計算を、定数 IMAGES_PER_SET_PER_DRONE (2) に基づくように変更
        num_sets = min(len(proc_p1), len(proc_p2), len(proc_m)) // IMAGES_PER_SET_PER_DRONE

    if num_sets == 0:
        print(f"¥n 処理対象となる画像セットがありません。各ドローンの画像が{IMAGES_PER_SET_PER_DRONE}枚以上あるか確認してください。")
        return
    print(f"¥n 合計 {num_sets} セットの処理を実行します。")
    print(f"各セット追加枚数: {IMAGES_PER_SET_PER_DRONE * 3}枚（各ドローン{IMAGES_PER_SET_PER_DRONE}枚)")

    # --- 2. Metashape プロジェクトの準備 ---
    print("¥n--- ステップ 2: Metashape プロジェクトの準備 ---")
    doc = Metashape.Document()
    try:
        doc.open(project_path)
        print(f"プロジェクト '{project_path}' を開きました。")
    except RuntimeError as e:
        print(f"エラー: プロジェクトを開けませんでした: {e}")
        return

    chunk_original = None
```

```
    for ch in doc.chunks:
        if ch.label == BASE_CHUNK_NAME:
            chunk_original = ch
            break

    if not chunk_original:
        print(f"エラー: 基準となるチャンク '{BASE_CHUNK_NAME}' が見つかりませんでした。")
        return
    print(f"基準チャンクとして '{chunk_original.label}' を使用します。")

    # --- 2.5. 出力ディレクトリの準備 ---
    print("¥n--- ステップ 2.5: 出力ディレクトリの準備 ---")
    try:
        # 点群フォルダを作成
        os.makedirs(OUTPUT_POINTCLOUD_DIRECTORY, exist_ok=True)
        print(f"  - 出力先ディレクトリを確認・作成しました。")
        print(f"      - 点群: {OUTPUT_POINTCLOUD_DIRECTORY}")
    except Exception as e:
        print(f"  - エラー: 出力ディレクトリの作成に失敗しました: {e}")
        return

    # --- 3. メインループ（セットごとの処理) ---
    for i in range(num_sets):
        set_number = i + 1
        print(f"¥n{'='*50}")
        print(f"--- セット {set_number}/{num_sets} の処理を開始 ---")
        print(f"{'='*50}")

        # 【修正】インデックス計算に IMAGES_PER_SET_PER_DRONE を使用
        start_index = i * IMAGES_PER_SET_PER_DRONE
        end_index = start_index + IMAGES_PER_SET_PER_DRONE

        current_p1 = proc_p1[start_index:end_index]
        current_p2 = proc_p2[start_index:end_index]
        current_m = proc_m[start_index:end_index]
```

```
        # 【修正】枚数チェックも定数と比較
        if not (len(current_p1) == IMAGES_PER_SET_PER_DRONE and
                len(current_p2) == IMAGES_PER_SET_PER_DRONE and
                len(current_m) == IMAGES_PER_SET_PER_DRONE):
            print(f"[セット {set_number}] 画像枚数が不足しているためスキップします。
")
            continue

        current_set_photos = current_p1 + current_p2 + current_m

        # --- ② チャンク複製 ---
        print(f"¥n[セット {set_number}] ステップ②: チャンクを複製しています...")
        new_chunk = chunk_original.copy()
        new_chunk.label = f"Set_{set_number:02d}_Alignment"
        print(f"  - 新しいチャンクを作成しました: '{new_chunk.label}'")

        # --- ③ 画像追加 ---
        print(f"¥n[セット {set_number}] ステップ③: 画像を追加しています...")
        new_chunk.addPhotos(current_set_photos)
          print(f"  - {len(current_set_photos)} 枚の画像を追加しました。 (合計:
{len(new_chunk.cameras)}枚)")

        # --- ④ アライメント ---
        print(f"¥n[セット {set_number}] ステップ④: 写真アライメントを実行していま
す...")
        try:
            new_chunk.matchPhotos(downscale=ALIGNMENT_DOWNSCALE,
                                  generic_preselection=True,     # 元は True
                                  reference_preselection=False,  # 元は False
                                  reset_matches=True,
                                  filter_stationary_points=False)
            new_chunk.alignCameras()
            doc.save()
            print("  - アライメントが完了し、プロジェクトを保存しました。")
        except RuntimeError as e:
```

```
            print(f"  - エラー: アライメントに失敗しました: {e}")
            continue

        # --- ④-2 領域の自動調整 ---
        print(f"¥n[セット {set_number}] ステップ④-2: 処理領域をタイポイントから手動で計算・調整しています...")
        try:
            if new_chunk.tie_points and len(new_chunk.tie_points.points) > 0:
                points = new_chunk.tie_points.points

                min_coord = Metashape.Vector([float('inf'), float('inf'), float('inf')])
                max_coord = Metashape.Vector([float('-inf'), float('-inf'), float('-inf')])

                for point in points:
                    if not point.valid:
                        continue

                    coord = point.coord

                    min_coord.x = min(min_coord.x, coord.x)
                    min_coord.y = min(min_coord.y, coord.y)
                    min_coord.z = min(min_coord.z, coord.z)
                    max_coord.x = max(max_coord.x, coord.x)
                    max_coord.y = max(max_coord.y, coord.y)
                    max_coord.z = max(max_coord.z, coord.z)

                if min_coord.x != float('inf'):
                    center = (max_coord + min_coord) / 2
                    size = max_coord - min_coord

                    new_chunk.region.center = center
                    new_chunk.region.size = size
                    new_chunk.region.rot = Metashape.Matrix.Diag([1, 1, 1])

                    doc.save()
                    print("  - 領域の調整が完了し、プロジェクトを保存しました。")
```

```
                else:
                    print("   - 警告: 有効なタイポイントが見つからなかったため、領域の調整をスキップします。")
            else:
                print("   - 警告: チャンクにタイポイントが存在しないため、領域の調整をスキップします。")
        except Exception as e:
            print(f"   - エラー: 領域の調整中に予期せぬエラーが発生しました: {e}")
            pass

        # --- ⑤ 点群作成（定数 IMAGES_PER_SET_PER_DRONE 回ループ） ---
        print(f"¥n[セット {set_number}] ステップ⑤: 時間ごとの点群を作成しています...")

        # 【修正】5 回固定ではなく、追加した枚数分（今回は 2 回）ループする
        for j in range(IMAGES_PER_SET_PER_DRONE):
            group_number = j + 1
                print(f"¥n   --- [セット {set_number}] 時間グループ {group_number}/{IMAGES_PER_SET_PER_DRONE} の処理 ---")

                time_group_photos = [os.path.normpath(p) for p in [current_p1[j], current_p2[j], current_m[j]]]

            dc_chunk = new_chunk.copy()
            dc_chunk.label = f"Set_{set_number:02d}_Group_{group_number:02d}"
            print(f"      - 処理用チャンクを作成: '{dc_chunk.label}'")

            enabled_cameras_count = 0
            # 使用するカメラ（3 枚）のみを有効化
            for camera in dc_chunk.cameras:
                normalized_camera_path = os.path.normpath(camera.photo.path)

                if normalized_camera_path in time_group_photos:
                    camera.enabled = True
                    enabled_cameras_count += 1
                else:
```

```
                camera.enabled = False

        print(f"      - {enabled_cameras_count} 台のカメラを有効化しました。")

        if enabled_cameras_count == 0:
            print(f"      - 警告: 有効なカメラが見つからなかったため、グループ
{group_number} の処理をスキップします。")
            doc.remove(dc_chunk)
            continue

        try:
            # 深度マップを作成
            print("      - 深度マップを作成しています...")
             dc_chunk.buildDepthMaps(downscale=DEPTH_MAPS_DOWNSCALE,
filter_mode=Metashape.AggressiveFiltering)
            print("      - 深度マップの作成が完了しました。")

            # 点群を作成
            if version_tuple[0] >= 2:
                print("      - Metashape 2.0+ API を使用して点群を作成します...")
                dc_chunk.buildPointCloud(source_data=Metashape.DepthMapsData
)
            else:
                print("      - Metashape 1.x API を使用して密な点群を作成しま
す...")
                dc_chunk.buildDenseCloud(source_data=Metashape.DepthMapsDat
a)

            print(f"      - グループ {group_number} の点群作成が完了しました。")

            doc.save()
            print(f"      - プロジェクトを保存しました。")

            # --- ファイル名生成ロジック ---
            phantom1_image_for_group = current_p1[j]
            base_filename = os.path.basename(phantom1_image_for_group)
```

```
                match = re.search(r'(¥d{8}-¥d{6}-¥d{3})', base_filename)

                if match:
                    timestamp_part = match.group(1)
                else:
                    print(f"  - 警告: Phantom1 のファイル名 '{base_filename}' からタイムスタンプが抽出できませんでした。チャンクラベルを使用します。")
                    timestamp_part = dc_chunk.label

                # --- ⑥ 点群データのエクスポート (PLY 形式) ---
                print(f"  - ステップ⑥: 点群を PLY 形式でエクスポートしています...")
                try:
                    output_filename = f"{timestamp_part}.ply"
                    output_path = os.path.join(OUTPUT_POINTCLOUD_DIRECTORY, output_filename)

                    if version_tuple[0] >= 2 and dc_chunk.point_cloud:
                        dc_chunk.exportPointCloud(path=output_path, format=Metashape.PointCloudFormat.PointCloudFormatPLY, crs=dc_chunk.crs)
                        print(f"  - エクスポート成功: '{output_path}'")
                    elif dc_chunk.dense_cloud:
                        dc_chunk.exportPoints(path=output_path, source_data=Metashape.DenseCloudData, format=Metashape.PointsFormatPLY, crs=dc_chunk.crs)
                        print(f"  - エクスポート成功: '{output_path}'")
                    else:
                        print("  - 警告: エクスポート対象の点群データが見つかりません。")

                except Exception as e:
                    print(f"  - エラー: 点群のエクスポートに失敗しました: {e}")

                doc.remove(dc_chunk)

            except Exception as e:
```

```
                    doc.remove(dc_chunk)
                    if "Zero resolution" in str(e):
                        print(f"      - 警告: 'Zero resolution' エラーのため、グループ {group_number} の処理をスキップします。")
                    else:
                        print(f"      - エラー: グループ {group_number} の処理中に失敗しました: {e}")
                    continue

            doc.save()

    print(f"¥n{'='*50}")
    print("すべての処理が完了しました。")
    print(f"{'='*50}")

if __name__ == "__main__":
    main()
```

コード 2

5.3 で行った、点群のボクセル化をおこなった python コードを下記に示す。
コード作成には、Google Gemini を活用した。

```
# -*- coding: utf-8 -*-
# 高低差による色付け

import open3d as o3d
import numpy as np
import cv2
import os
import glob

# --- 共通設定 ---
# 入力・出力の基準となるフォルダ
BASE_FOLDER = r"D:¥IkemotoKazuma¥output¥20260112-4"
```

```
PLY_FOLDER_PATH = os.path.join(BASE_FOLDER, "Pointcloud")

# 動画の設定
VIDEO_FPS = 30
IMAGE_WIDTH = 1920
IMAGE_HEIGHT = 1080
# 出力ファイル名を変更（密度版と区別するため）
VIDEO_SUFFIX = "side_height_colored"
OUTPUT_VIDEO_NAME = f"voxel_video_{VIDEO_SUFFIX}.mp4"

# ボクセル設定
VOXEL_SIZE = 1.0          # 1m 角
Z_THRESHOLD = 20.0       # 高度閾値（35m 以上）

def calculate_global_z_range(ply_files, z_filter):
    """
    全ファイルを走査して、フィルタリング後の点群の Z 座標（高さ）の最小値と最大値を特定する
    """
    print("¥n---  全体の高さ範囲を計算中... ---")
    global_min_z = float('inf')
    global_max_z = float('-inf')
    points_found = False

    for i, ply_path in enumerate(ply_files):
        print(f"¥r    スキャン中: {i+1}/{len(ply_files)}", end="")
        pcd = o3d.io.read_point_cloud(ply_path)
        if not pcd.has_points(): continue

        points = np.asarray(pcd.points)
        # 指定高度以上の点を抽出
        mask = points[:, 2] >= z_filter
        filtered_points = points[mask]

        if len(filtered_points) > 0:
            points_found = True
```

```
            # 現在のファイルの最小・最大 Z 座標を取得
            current_min_z = np.min(filtered_points[:, 2])
            current_max_z = np.max(filtered_points[:, 2])

            # 全体の最小・最大を更新
            if current_min_z < global_min_z: global_min_z = current_min_z
            if current_max_z > global_max_z: global_max_z = current_max_z

    if not points_found:
        print("¥n 警告: 指定された高度閾値を超える点群が見つかりませんでした。デフォルト範囲を使用します。")
        return 0.0, 1.0 # デフォルト値

    print(f"¥n 高さ範囲を確定: {global_min_z:.2f}m - {global_max_z:.2f}m")
    return global_min_z, global_max_z

def create_voxel_mesh(ply_path, voxel_size, z_filter, min_z, max_z):
    """
    指定されたファイルから高度フィルタリング済みのボクセルメッシュを生成する。
    色はボクセルの高さ（Z 座標）に基づいて決定される。
    """
    pcd = o3d.io.read_point_cloud(ply_path)
    if not pcd.has_points(): return None

    points = np.asarray(pcd.points)
    mask = points[:, 2] >= z_filter
    filtered_points = points[mask]

    if len(filtered_points) == 0: return None

    # ボクセルインデックスを計算
    voxel_indices = np.floor(filtered_points / voxel_size).astype(int)
    # ユニークなボクセルインデックスのみ取得（点数は不要なので return_counts=False）
    unique_indices = np.unique(voxel_indices, axis=0)

    # 各ボクセルの中心 Z 座標を計算
```

```
    #（インデックス * サイズ）でボクセルの底面の角の座標になり、それにサイズ/2 を足して中心とする
    voxel_centers_z = unique_indices[:, 2] * voxel_size + voxel_size / 2.0

    # 全体スケール(高さ)に基づいて色を決定
    if max_z > min_z:
        # 最小値から最大値の範囲で 0.0〜1.0 に正規化
        normalized_z = (voxel_centers_z - min_z) / (max_z - min_z)
    else:
        normalized_z = np.zeros_like(voxel_centers_z, dtype=float)

    #念のため 0-1 の範囲にクリップ
    normalized_z = np.clip(normalized_z, 0.0, 1.0)

    # カラーマップを適用
        colors_bgr = cv2.applyColorMap((normalized_z * 255).astype(np.uint8), cv2.COLORMAP_JET)
    colors_rgb = colors_bgr.reshape(-1, 3)[:, ::-1] / 255.0

    combined_mesh = o3d.geometry.TriangleMesh()
    for coord, color in zip(unique_indices, colors_rgb):
                cube = o3d.geometry.TriangleMesh.create_box(width=voxel_size, height=voxel_size, depth=voxel_size)
        cube.translate(coord * voxel_size)
        cube.paint_uniform_color(color)
        combined_mesh += cube

    combined_mesh.compute_vertex_normals()
    return combined_mesh

def draw_ui(frame, time_val, filename, min_val, max_val):
    """
    フレーム上に UI 要素（時間、ファイル名、カラーバー）を描画する
    カラーバーは高さを表示する。
    """
    font = cv2.FONT_HERSHEY_SIMPLEX
```

```
    alpha = 0.5
    overlay = frame.copy()

    # 1. 時間（左上）
    time_text = f"Time: {time_val:.2f}s"
    cv2.rectangle(overlay, (10, 10), (250, 60), (0, 0, 0), -1)
    cv2.addWeighted(overlay, alpha, frame, 1-alpha, 0, frame)
    cv2.putText(frame, time_text, (25, 45), font, 1.0, (255, 255, 255), 2, cv2.LINE_AA)

    # 2. ファイル名（右上）
    fn_text = filename
    (tw, th), _ = cv2.getTextSize(fn_text, font, 0.8, 2)
    cv2.rectangle(frame, (IMAGE_WIDTH - tw - 40, 10), (IMAGE_WIDTH - 10, 60), (0, 0,
0), -1)
    cv2.putText(frame, fn_text, (IMAGE_WIDTH - tw - 30, 45), font, 0.8, (255, 255, 255),
2, cv2.LINE_AA)

    # 3. カラーバー（右側）- 高さ表示に変更
    bar_w, bar_h = 30, 300
    bx, by = IMAGE_WIDTH - 120, 150
    # 背景
    cv2.rectangle(frame, (bx - 10, by - 40), (IMAGE_WIDTH - 10, by + bar_h + 40), (0, 0,
0), -1)
    # グラデーション
    gradient = np.linspace(255, 0, bar_h).astype(np.uint8).reshape(-1, 1)
    bar_img = cv2.applyColorMap(gradient, cv2.COLORMAP_JET)
    frame[by : by + bar_h, bx : bx + bar_w] = bar_img

    # ラベルを "Height(m)" に変更
    cv2.putText(frame, "Height(m)", (bx - 15, by - 20), font, 0.6, (255, 255, 255), 1)
    labels = 5
    for i in range(labels):
        # 値を計算
        val = max_val - (i * (max_val - min_val) / (labels - 1))
        py = by + int(i * bar_h / (labels - 1))
        # 小数点第 1 位まで表示
```

```
            cv2.putText(frame, f"{val:.1f}", (bx + bar_w + 5, py + 5), font, 0.5, (255, 255, 255),
1)

def main():
    ply_files = sorted(glob.glob(os.path.join(PLY_FOLDER_PATH, "*.ply")))
    if not ply_files:
        print("PLY ファイルが見つかりません。")
        return

    # 1. 全体スキャン（高さの範囲を計算）
    # 変更: 密度の代わりに Z 座標の範囲を計算する関数を呼び出す
    min_z, max_z = calculate_global_z_range(ply_files, Z_THRESHOLD)

    # 2. カメラ視点の決定（最初のファイルを使用）
    print("¥n--- カメラ視点を設定してください ---")
    # 変更: 引数を高さの範囲に変更
    sample_mesh = create_voxel_mesh(ply_files[0], VOXEL_SIZE, Z_THRESHOLD, min_z,
max_z)

    camera_params = []
    def store_params(vis):
        ctr = vis.get_view_control()
        camera_params.append(ctr.convert_to_pinhole_camera_parameters())
        vis.destroy_window()
        return True

    vis_setup = o3d.visualization.VisualizerWithKeyCallback()
    vis_setup.create_window("Set Camera Angle and Press 'Q'", width=IMAGE_WIDTH,
height=IMAGE_HEIGHT)
    if sample_mesh:
        vis_setup.add_geometry(sample_mesh)
    else:
        print("最初のフレームに表示するデータがありません。")

    vis_setup.register_key_callback(ord("Q"), store_params)
    vis_setup.run()
```

```
    if not camera_params: return
    params = camera_params[0]

    # 3. 動画の生成
    output_path = os.path.join(BASE_FOLDER, OUTPUT_VIDEO_NAME)
    fourcc = cv2.VideoWriter_fourcc(*'mp4v')
    writer = cv2.VideoWriter(output_path, fourcc, VIDEO_FPS, (IMAGE_WIDTH, IMAGE_HEIGHT))

    vis = o3d.visualization.Visualizer()
    vis.create_window(width=IMAGE_WIDTH, height=IMAGE_HEIGHT, visible=False)

    print(f"¥n--- 動画生成を開始します: {OUTPUT_VIDEO_NAME} ---")

    try:
        for i, ply_path in enumerate(ply_files):
            print(f"¥r  フレーム処理中: {i+1}/{len(ply_files)}", end="")

            # 変更: 引数を高さの範囲に変更
            mesh = create_voxel_mesh(ply_path, VOXEL_SIZE, Z_THRESHOLD, min_z, max_z)
            if mesh is None: continue

            vis.add_geometry(mesh)
            ctr = vis.get_view_control()
            ctr.convert_from_pinhole_camera_parameters(params, allow_arbitrary=True)

            vis.poll_events()
            vis.update_renderer()

            # キャプチャ
            img_f = vis.capture_screen_float_buffer(do_render=True)
            frame = (np.asarray(img_f) * 255).astype(np.uint8)
            frame_bgr = cv2.cvtColor(frame, cv2.COLOR_RGB2BGR)
```

```
            # UI 描画
            # 変更: 引数に高さの範囲を渡す
              draw_ui(frame_bgr, i / VIDEO_FPS, os.path.basename(ply_path), min_z,
max_z)

            writer.write(frame_bgr)
            vis.clear_geometries()

    finally:
        vis.destroy_window()
        writer.release()
        print(f"¥n🎉 すべての処理が完了しました: {output_path}")

if __name__ == "__main__":
    main()
```