エネルギー科学研究科

エネルギー社会・環境科学専攻修士論文

題目：

高速な類似画像検索のための多段階絞り込み処理フレームワークの提案

指導教員： 下田 宏 教授

氏名： 大橋 由暉

提出年月日： 平成30年2月13日(火)

論文要旨

題目：高速な類似画像検索のための多段階絞り込み処理フレームワークの提案

下田研究室,　大橋由暉

要旨：

近年、原子力発電プラントの保修･解体作業の支援に、拡張現実感（Augmented Reality : AR）を利用する研究が行われている。AR を用いることで、通常では目に見えない情報をユーザーに提示できるほか、3 次元の位置や方向に紐づいた情報を直感的に提示できるため、作業の効率や安全性を向上できると期待されている。

AR では、カメラで現実世界を撮影した映像に Computer Graphics（CG）を重畳表示するが、CG で表示された情報があたかも現実の空間内に存在するかのように見せるためには、カメラの移動に合わせて CG の表示を適切に変化させる必要がある。そのため AR の利用時には、トラッキングと呼ばれる、現実世界を撮影しているカメラの位置と方向（カメラ姿勢）を計測する技術が必要である。特に、原子力発電プラント内で AR を利用する場合に用いられることが多い、カメラ画像中の特徴を利用したマーカレストラッキングでは、多くの場合、高速かつ安定なトラッキングを実現するために、直前のトラッキング結果を利用するトラッキング手法（逐次トラッキング）が用いられる。しかし、カメラを急峻に動かすなどした場合に、一度逐次トラッキングに失敗すると、直前のトラッキング結果が得られないため、それ以降逐次トラッキングが行えなくなる。そこで，カメラを利用したマーカレストラッキングには、逐次トラッキングに失敗した際に、逐次トラッキングとは別の方法で現在のカメラ姿勢を推定し、逐次トラッキング可能な状態に復帰させる処理（リローカリゼーション）が組み込まれることが多い。リローカリゼーションでは、現在のカメラ姿勢を推定するために、過去に撮影された、撮影時のカメラ姿勢が既知の画像の中から、現在のカメラ画像と似た画像を見つけ出す、類似画像検索が行われる。

本研究では，リローカリゼーションの処理中の類似画像検索の性能を向上させるために、複数の類似画像検索手法を組み合わせて利用する、多段階絞り込み処理フレームワークを開発した。複数の類似画像検索手法を用いて段階的な絞り込みを行うことで、処理速度に特化した手法や、計算コストが高い代わりに精度良く類似画像を選出できる手法など、それぞれの手法の利点を併せ持った、より高性能な類似画像検索が実現可能である。また、開発したフレームワークには、AR を利用する環境や、作業支

援システムに要求されるリローカリゼーションの性能に応じて、段階的な絞り込みに利用する類似画像検索手法の組合せを自動的に最適化するアルゴリズムも実装した。

開発したフレームワークの自動最適化アルゴリズムが正しく機能することと、複数の類似画像検索手法を用いた段階的な絞り込み処理により、類似画像検索の性能が向上することを確認するために、原子炉廃止措置研究開発センター（ふげん）の純水装置室内で撮影した画像セットや 7-Scenes データセットを用いて評価実験を行った。評価の結果、開発したフレームワークの有効性が確認できた。開発したフレームワークを利用することで、リローカリゼーションの性能が向上し、AR を用いた作業支援システムの動作の安定性を向上できると期待される。

今後の課題として、組み合わせる類似画像検索手法のパラメータも含めた最適化アルゴリズムを実装することや、より効率的に最適な類似画像検索手法の組み合わせを探索するアルゴリズムを開発することが挙げられる。

目 次

図 目 次

表 目 次

第 1 章 序論

2011 年 3 月 11 日の東日本大震災により引き起こされた東京電力福島第一原子力発電所の事故を受けて、日本国内の全ての原子力発電プラントで原子炉の運転停止を余儀なくされた。事故から約 7 年が経過した今、関西電力高浜発電所の 3・4 号機など、一部は運転を再開しているが、未だに多くが運転を停止したままである。また、運転を停止している原子炉の内、17 基は廃炉が決定した [1]（平成 30 年 1 月 22 日時点）。

運転を再開した原子力発電プラントの保修・点検作業や、廃炉が決定した原子力発電プラントの解体作業は、放射線に汚染された物を扱うこともあるため、一般的なプラント内での作業に比べて、より一層安全確実に遂行する必要がある。また、放射能汚染区域内で作業を行う場合には、被曝のリスクを低減するために、作業時間を短くすることが望ましく、作業を効率的に遂行する必要がある。そこで、拡張現実感（Augumented Reality: AR）を用いて、原子力発電プラント内での保修・点検・解体作業を支援する研究が行われている [2]。AR を用いることで、通常では目に見えない情報をユーザに提示できるほか、3 次元の位置や方向に紐づいた情報を直感的に提示できるため、作業の効率や安全性を向上できると期待されている。

AR を利用するためには、現実世界の映像を撮影するカメラの位置と方向（カメラ姿勢）をリアルタイムに計測（トラッキング）する技術が必要である。トラッキングでは、カメラ姿勢計測の高精度化や処理の高速化のために、直前のカメラ姿勢を利用して現在のカメラ姿勢を計測する逐次トラッキングを主に行うが、AR システムの起動時や、逐次トラッキングに失敗した場合など、直前のカメラ姿勢が不明となった際には、逐次トラッキングとは別の方法で現在のカメラ姿勢を求め、逐次トラッキング可能な状態に復帰させる処理（リローカリゼーション）が必要である。

リローカリゼーションでは、類似する 2 つの画像は近いカメラ姿勢で撮影されているという仮定の下、現在のカメラで撮影された画像と類似する画像を、事前に作成されたデータベース（AR を利用する環境の様々な箇所で撮影された画像と撮影時のカメラ姿勢が記録されている）中から見つけ出し、その類似画像撮影時のカメラ姿勢を現在のカメラ姿勢の推定値とすることで、逐次トラッキングへの復帰処理を行う。そのため、類似画像検索の性能を向上させることで、リローカリゼーションの性能を向上できる。類似画像検索の性能は、速度と精度で表されるが、一般に両者の間にはトレー

ドオフの関係がある。そこで、本研究では、高速な類似画像検索手法と高精度な類似画像検索手法を組み合わせることで、それぞれの利点を併せ持った類似画像検索を実現するためのフレームワークを開発し、評価することを目的とする。フレームワークを利用してリローカリゼーションの性能を向上させることで、AR を用いた作業支援システムの動作を安定化できるため、作業支援の効果を高められると期待される。

本論文は、第 1 章の序論を含め、全 5 章で構成される。第 2 章では、本研究の背景と目的を述べる。第 3 章では、提案するフレームワークの詳細を説明し、第 4 章では、フレームワークの性能を評価するために行った実験について述べる。第 5 章では、本研究のまとめと今後の課題を述べる。

第 2 章 研究の背景と目的

2.1 研究の背景

近年、原子力発電プラント内での作業を AR を用いて支援する研究が行われている [2]。AR は、カメラで現実世界を撮影して取得した映像上に Computer Graphics（CG）を重畳表示することで、ユーザの感覚を拡張する技術である。カメラの移動に合わせて CG の表示を適切に変化させることで、CG で表示された情報があたかも現実の空間内に存在するかのように見せることができる。図 2.1 は AR を用いて原子力発電プラント内の放射線の強度分布を可視化した例、図 2.2 は解体対象の機器の上に切断箇所を重畳表示した例である。このように原子力発電プラント内で AR を用いることで、通常では目に見えない情報をユーザに提示できるほか、3 次元の位置や方向に紐づいた情報を直感的に提示できるため、作業の効率や安全性を向上できると期待されている [2]。

AR で重畳表示する CG を、現実世界の映像に応じた自然な見え方にするためは、現実世界を撮影しているカメラの位置と方向（カメラ姿勢）をリアルタイムに計測し、その結果に応じて適切に CG の描画を更新する必要がある。そのため AR の利用時には、トラッキングと呼ばれる、カメラ姿勢を計測する技術が必要である。しかし、構造が複雑な原子力発電プラント内で常に安定して使用できる手法は未だ報告されていない。以下では、既存のトラッキング手法の概要とその問題点を述べた後、トラッキングの安定化のために必要不可欠な、リローカリゼーションと呼ばれる技術について説明する。

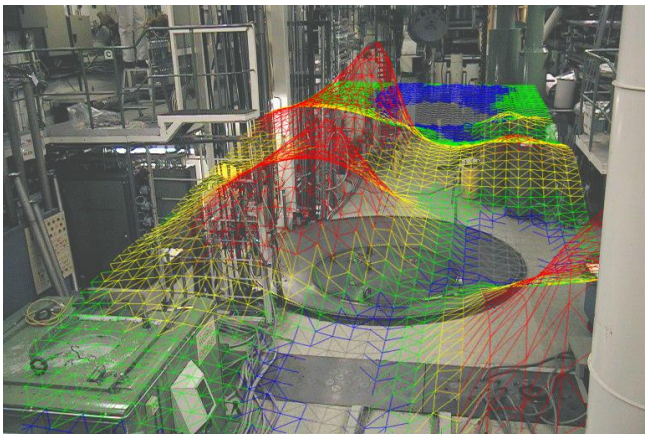


図 2.1: AR を用いて放射線の強度分布を可視化した例

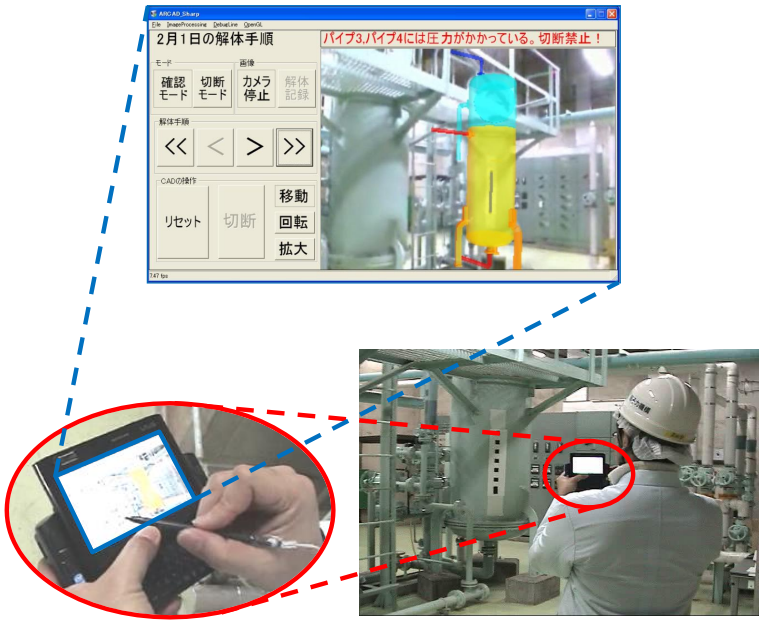


図 2.2: AR を用いて解体対象物に切断箇所を重畳表示した例

2.1.1 既存のトラッキング手法とその問題点

トラッキングには、利用するセンサの種類やその利用方法によって様々な手法が存在する [3]。ジャイロセンサや加速度センサなどの慣性センサを用いる手法では、センサの計測値を基にトラッキング開始時からのカメラ姿勢の相対的な変化量を計算することで現在のカメラ姿勢を求めるため、長時間利用する場合には誤差の蓄積により精度が低下するという欠点がある。磁気センサを用いる手法は高い精度でカメラ姿勢を計測できるが、機材が高価であり、また、機材 1 台当たりの計測可能範囲が狭いという問題がある。Global Positioning System（GPS）を用いる手法は、原子力発電プラント内のように衛星からの電波が届きにくい環境では利用できない。

一方で、カメラ画像を基にカメラ姿勢をトラッキングするビジョンベースの手法は、準備の手間やコストが小さく、導入が容易であるため、原子力発電プラント内での利用にも適している。ビジョンベースのトラッキング手法には、図 2.3 に示すような人工のマーカを環境内に設置して利用する手法や、図 2.4 に示すような、環境内に元々存在する特徴（自然特徴）を利用するマーカレスの手法がある [4]。前者の手法は、マーカの現実空間内での 3 次元位置と、カメラに写ったマーカの画像上での 2 次元座標の対応関係を基に、Perspective-n-Points（PnP）問題 [5] を解くことでカメラ姿勢を推定す

る。この手法は比較的容易に高い精度でトラッキングを実現できるが、マーカの設置および位置計測に手間がかかる、カメラにマーカが写る範囲でしか利用できない、マーカ自体が作業の邪魔になる可能性があるなどの問題がある。後者の手法は多くの場合、現在のフレームのカメラ画像に写った自然特徴と直前フレームのカメラ画像に写った自然特徴から、似た外見を持つ自然特徴同士を対応付け、対応付けた自然特徴のそれぞれの画像上での 2 次元座標の関係から 2 フレーム間のカメラの基礎行列を求めることで、直前のカメラ姿勢に対する現在の相対的なカメラ姿勢を求める。自然特徴同士を対応付ける際には、連続するフレーム間ではカメラ姿勢の変化は小さく、同一の自然特徴はそれぞれのフレームの画像上で近い位置に写っていると仮定し、対応付けるべき自然特徴の探索範囲を限定することで、処理を高速化しつつ誤った対応付けをする可能性を減らしている。このように、直前のフレームの情報と現在のフレームの情報のみを利用することで、現在のカメラ姿勢を高速に推定する処理を、本論文では逐次トラッキングと呼ぶ。これまでに、自然特徴を用いた逐次トラッキング手法としては、使用する自然特徴の種類やその表現方法が異なる様々な手法が研究開発されている [6–8]。しかし、逐次トラッキングでは、直前のカメラ姿勢が既知でなければ現在のカメラ姿勢を求めることができない。そのため、AR システムの起動時や、カメラを急峻に動かした場合、カメラの前を人が横切った場合など、直前のカメラ姿勢が不明となった際には、逐次トラッキングとは別の処理で現在のカメラ姿勢を求め、逐次トラッキング可能な状態に復帰させる必要がある。この復帰処理はリローカリゼーションと呼ばれ、ビジョンベースのマーカレストラッキングには、多くの場合、逐次トラッキングと併せて実装されている。

図 2.3: 人工マーカの例

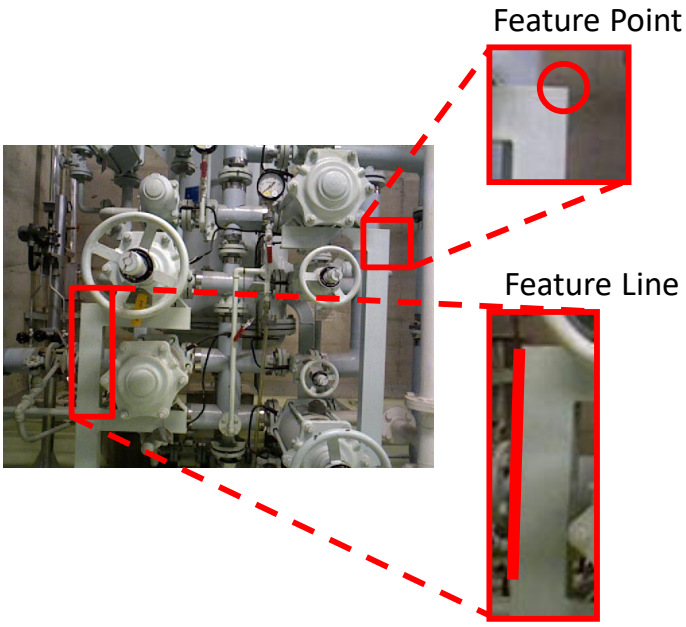


図 2.4: 原子力発電プラント内の自然特徴の例

2.1.2 既存のリローカリゼーション手法とその問題点

リローカリゼーションでは、撮影時のカメラ姿勢が既知である複数の画像を用いて、カメラ姿勢の推定に必要な情報を含むデータベース（DB）を事前に構築し、それを利用して現在のカメラ姿勢を推定する。リローカリゼーションには大きく二つの手法があり、画像中の自然特徴とその 3 次元位置を DB に保存し、利用する手法をキーポイントベースの手法、画像と画像取得時のカメラ姿勢を DB に保存し、利用する手法をキーフレームベースの手法と呼ぶ。以下ではそれぞれの手法の概要とその特徴を述べる。

2.1.2.1 キーポイントベースのリローカリゼーション手法

キーポイントベースの手法では、図 2.5 に示すように、事前にカメラ姿勢が既知の画像から自然特徴を抽出、その 3 次元位置を計算して DB に登録しておく。リローカリゼーション実行時には、図 2.6 に示すように、現在のカメラ画像（以下、クエリ画像）から自然特徴を抽出し、DB の自然特徴の中から最も類似するものを探し出して対応付ける。DB には各自然特徴の 3 次元位置も登録されているため、クエリ画像中の自然特徴の 2 次元座標と現実空間内での 3 次元座標が対応付けられたことになる。この対応関係から PnP 問題を解くことでカメラ姿勢を推定する。キーポイントベースのリロー

カリゼーションでは、クエリ画像から抽出された自然特徴と同一の自然特徴を、DB に保存された全ての自然特徴の中から見つけ出し、対応付けなければならない。そのため、逐次トラッキングで 2 フレーム間の自然特徴を対応付ける場合よりも、正しく高速に対応付けることが難しくなる。そこで、この問題を緩和するために、木構造の DB を利用する手法 [9] やあらかじめ自然特徴をグルーピングしておく手法 [10] などが開発されている。また、誤った対応付けを含む場合にも安定してカメラ姿勢を推定するために、RANSAC[11] などのロバスト推定手法が用いられることも多い。

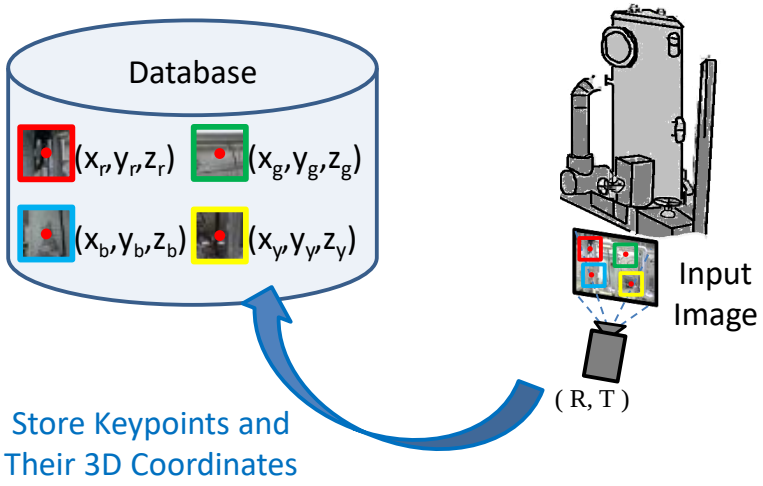


図 2.5: キーポイント DB の作成

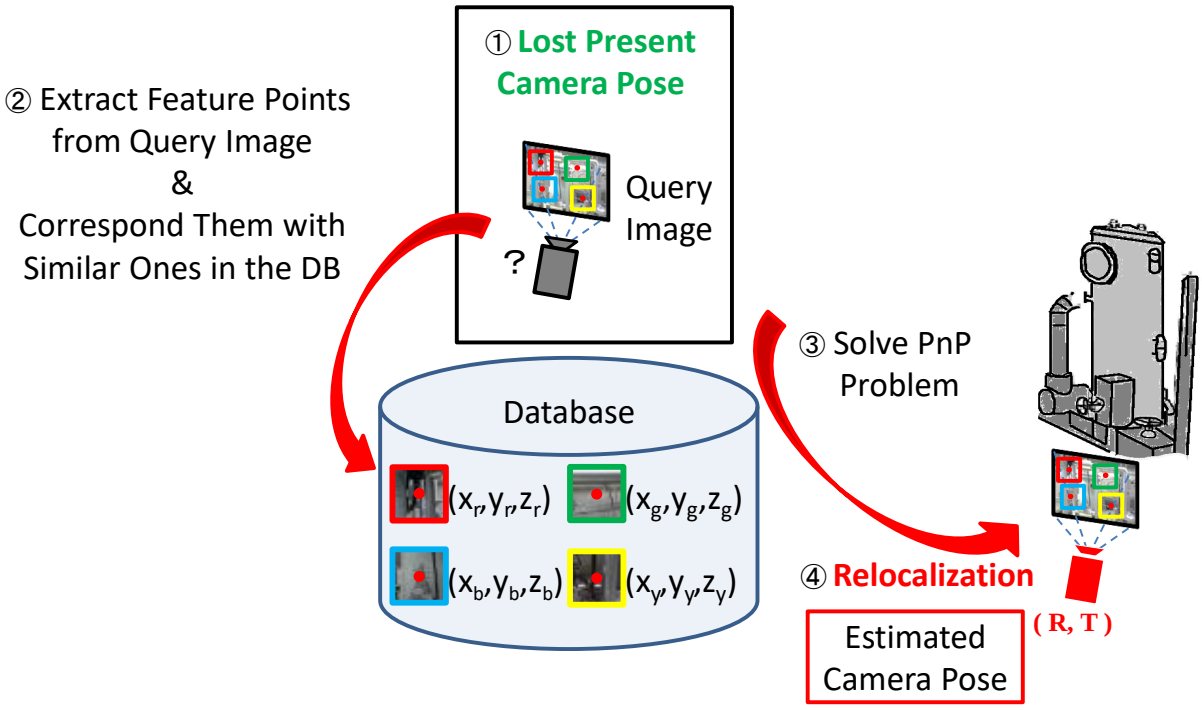


図 2.6: キーポイントベースのリローカリゼーションによるカメラ姿勢の推定

2.1.2.2　キーフレームベースのリローカリゼーション手法

キーフレームベースの手法では、図 2.7 に示すように、事前に撮影されたカメラ姿勢が既知の画像をカメラ姿勢とともに DB に登録しておく。リローカリゼーション実行時には、図 2.8 に示すように、クエリ画像と最も類似する画像を DB から見つけ出し、その画像とともに保存されているカメラ姿勢を現在のカメラ姿勢の推定値とする。この推定値を初期値として、再投影誤差の最小化や ICP アルゴリズムによる位置合わせ [12] など、逐次トラッキングへの復帰処理を行う。一般的な実装では、類似画像を複数枚（5 枚程度）選出し、選出されたそれぞれの画像で順に復帰処理を行うことが多い。選出された画像のいずれかで誤差が基準値以下に収束した場合に、リローカリゼーションに成功したと判定し、その時のカメラ姿勢を現在のカメラ姿勢として逐次トラッキングに復帰する。

クエリ画像と全く同じ画像が DB に含まれる場合には、画像のハッシュ値などを用いることで、容易にその画像を見つけ出すことができる。しかし、DB 作成時と AR アプリケーションの利用時で全く同じカメラ姿勢を取ることは通常起こり得ず、DB にクエリ画像と同一のデータが含まれることはないため、同一ではなく類似する画像を見つけ出す必要がある。類似画像の検索手法（画像検索手法）については数多くの研究が存在し、検索手法の選択は、リローカリゼーションの実行速度や精確性を決定する重要な要因である。

Klein らは、クエリ画像と DB の画像をそれぞれ縮小したうえで、全ピクセルの輝度値の差の二乗和（Sum of Squared Difference : SSD）を計算し、最も SSD が小さくなる画像を類似画像として選出する手法を提案した [13]。Pirchheim らは、画像の正規化相互相関を用いる手法を開発している [14]。しかし、これらの手法は SSD や正規化相互相関の計算処理に掛かる時間が大きいという問題がある。Glocker らは、画像からランダムに抽出したピクセルをランダムな閾値で 2 値化してコード化し、それを類似画像検索に用いることで、高速な検索処理を実現した [15]。木村らは、画像中の線分特徴を Line Segment Detector（LSD）[16] を用いて抽出し、抽出された線分の本数や角度、線分間の距離などの統計情報を利用して段階的に類似画像の候補を絞り込む手法を開発した [17]。この手法では、計算コストが低い単純な方法で類似画像の候補を絞り込んだ後、計算コストが高いが、精度が高い方法を用いて類似画像を選出することで、高速処理と高精度の両立を実現した。この研究により、原子力発電プラント内では、線分特徴を用いた類似画像検索が有効であることが示されている。

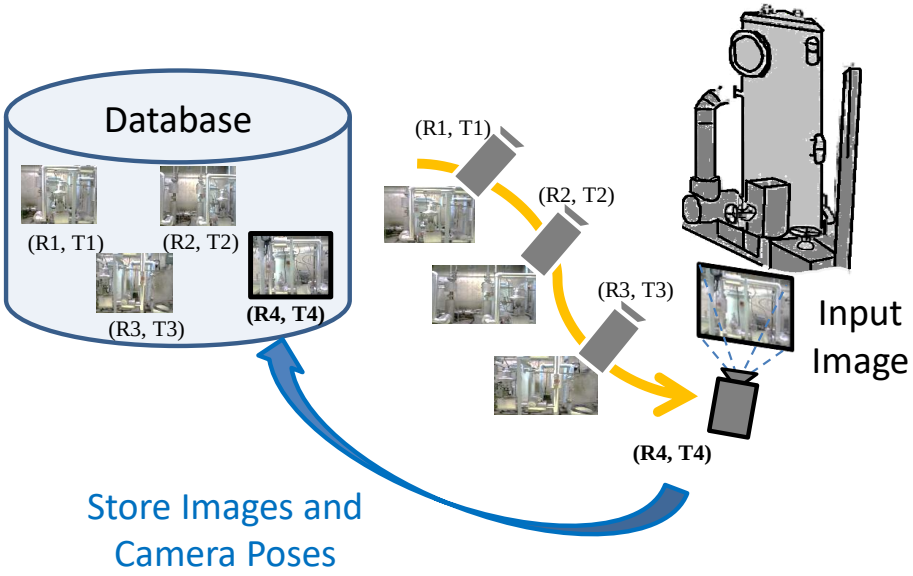


図 2.7: キーフレーム DB の作成

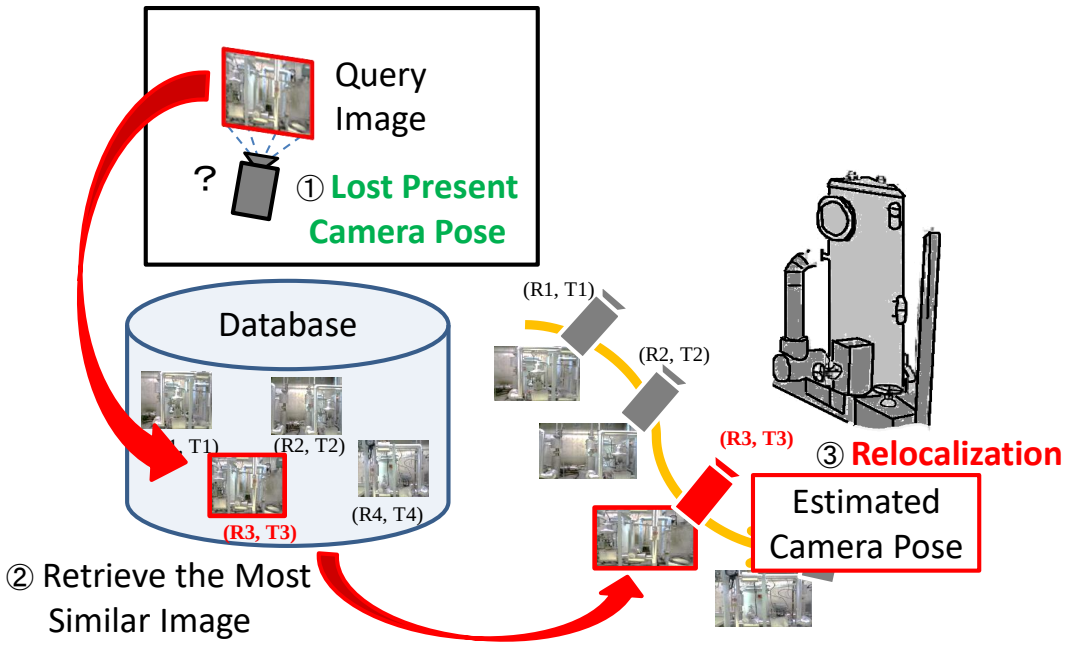


図 2.8: キーフレームベースのリローカリゼーションによるカメラ姿勢の推定

2.2　研究の目的と意義

近年の様々な研究の成果により、リローカリゼーションの性能は大きく向上してきた。しかし、ARを用いた作業支援システムの動作をより確実なものにしていくためには、より広い領域を対象に高速かつ正確に実行可能にするなど、リローカリゼーションの性能をさらに向上させる必要がある。

キーフレームベースのリローカリゼーションの性能を向上させる方法として、複数の画像検索手法を組み合わせて利用することで、類似画像検索の性能を向上させる方法が考えられる。木村らは線分に着目した段階的な絞り込みにより類似画像検索の性能を向上させた[17]が、線分に限らず様々な画像検索手法を用いて段階的な絞り込みを行うことで、処理速度に特化した手法や、計算コストが高い代わりに精度良く類似画像を選出できる手法など、それぞれの手法の利点を併せ持った、より高性能な類似画像検索を実現できる可能性がある。しかし、リローカリゼーションに有効な画像検索手法は、アプリケーションに要求される処理速度や精度、利用する環境によって異なると考えられるため、状況に応じて最適な画像検索手法の組み合わせを選出する必要があるが、人が最適な組合せを予測することは困難である。

そこで本研究では、キーフレームベースのリローカリゼーションにおける類似画像検索の性能を向上させるために、**1. 複数の画像検索手法を用いて段階的に類似画像の候補を絞り込む際の処理の流れ**と、**2. 要求される性能や使用する環境に応じて画像検索手法の組み合わせを自動的に最適化する手法**を定めたフレームワークの開発と評価を行うことを目的とする。

これまでに、画像検索手法の研究は数多く行われているが、複数の画像検索手法を組み合わせた利用方法に関する研究は行われていない。そのため、本研究で提案するフレームワークは、複数の画像検索手法を組み合わせて利用する方法を初めて明確に定めるものであり、これにより画像検索手法の利用可能性が大きく広がると考えられる。

このフレームワークに従って複数の画像検索手法を組み合わせて利用することで、ARを用いた作業支援システムに実装するリローカリゼーションの精度を向上できるため、より確実に作業の支援を行えるようになる。また、新たな画像検索手法が利用可能となった際には、画像検索手法の組み合わせの最適化を再度実行することで、その時点で最も高い性能を発揮する画像検索手法の組み合わせを選択することができるため、継続的にリローカリゼーションの性能を向上させることができると期待される。

第 3 章 多段階絞り込み処理フレームワークの提案

3.1 フレームワークの設計

キーフレームベースのリローカリゼーションで DB からクエリ画像に類似する画像を検索する際には、クエリ画像と DB 中の各画像を比較、画像間の類似度を算出し、類似度が高い画像を類似画像として選出する。多くの画像検索手法では、類似度を高速に算出するためや、画像のより高次な特徴から類似度を算出するためなどに、画像のピクセル値同士を直接比較するのではなく、画像の特徴を数値化した多次元ベクトル（画像特徴量）を抽出し、抽出された画像特徴量を用いて類似度を算出する。このとき、通常、DB 中の全画像との類似度の計算に要する時間は DB に含まれる画像数に比例し、1 画像当たりの類似度の算出に要する時間は類似度（或いは相違度）の定義（L1 距離、L2 距離、交差、相関など）や画像特徴量の次元数などに依存する。画像特徴量の次元が小さいほど、類似度の算出に要する時間は短くなるが、情報量が減少するため、類似度に画像間の細かな差異が反映されづらく、選出される類似画像の精度が低くなりやすい。反対に、画像特徴量の次元が大きいほど、類似度の算出に要する時間は長くなるが、精度の高い類似画像を選出できる傾向がある。そのため、DB のサイズが大きい場合に、精度を優先して次元が大きい画像特徴量で類似画像検索を行うと、検索に多くの時間を要する。一方、時間を優先して次元が小さい画像特徴量を用いた場合、類似画像検索の精度が低くなる。そこで、提案するフレームワークでは、複数の画像検索手法を用いて段階的に類似画像の候補を絞り込む方法を採用する。まず、短い時間で類似度を算出できる手法を用いて、クエリ画像と DB 中の全画像との比較を行い、類似度が一定以上の画像のみを類似画像の候補として残す。その後、絞り込まれた残りの候補に対して別の画像検索手法を適用し、さらに候補を絞り込む。この処理を候補が十分に少なくなるまで繰り返し、最終的に残った画像をクエリ画像に対する類似画像として選出する。この方法では、処理速度が速いが精度が低い手法を、類似画像の候補の大まかな絞り込みに利用することで、精度の低さが問題とならない範囲で素早く類似画像の候補を絞り込むことができると考えられる。また、少なくなった候補に対して、処理速度は遅いが精度が高い手法を適用して類似度の計算を実施すること

で、全体の処理として短い時間で精度良く類似画像を選出できると考えられる。本研究で提案する段階的な絞り込み処理の詳細は 3.2.2 項で述べる。

段階的な絞り込み処理を実行する際の最適な画像検索手法の組み合わせは、アプリケーションに要求される処理速度や精度、利用する環境によって異なると予想される。そのため、アプリケーションに要求される性能や利用する環境に応じて、最適な画像検索手法の組み合わせを選択するための明確な指針、或いは組み合わせの自動最適化アルゴリズムが提案されることが望ましい。そこで、提案するフレームワークには、段階的な絞り込み処理に用いる画像検索手法の組み合わせの自動最適化アルゴリズムを実装する。また、画像検索手法で利用される画像特徴量の次元数や算出に要する時間などは、多くの場合、画像特徴量の算出元の画像のサイズ（特徴量算出元画像サイズ）によって変化する。そのため、特徴量算出元画像サイズを適切に変更することで、類似画像検索の処理速度や検索精度を向上させることが可能であると考えられる。このことから、実装する自動最適化アルゴリズムでは、画像検索手法の組み合わせに加えて、各手法で用いる特徴量算出元画像サイズも最適化の対象とする。自動最適化アルゴリズムの詳細は 3.2.1 項で述べる。

3.2　提案するフレームワークの詳細

図 3.1 に提案するフレームワークの概要を、図 3.2 にフレームワークの詳細を示す。DB は、オフラインの処理 [18] や専用のハードウエアの使用 [19]、3 次元再構成モデルを用いたレンダリング手法 [20] などにより、撮影時のカメラ姿勢が既知の画像で構成されているものとする。本フレームワークは、類似画像検索に用いる複数の画像検索手法と特徴量算出元画像サイズの組み合わせを最適化する最適化パートと、最適化された手法の組み合わせで類似画像検索を実行する実行パートの 2 つのパートで構成される。最適化パートでは、組み合わせる画像検索手法の候補、特徴量算出元画像サイズの候補、アプリケーションに要求される性能、DB 用の画像（利用環境を撮影して取得した画像）を入力として、段階的な絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの最適な組み合わせ、それぞれの画像検索手法で用いる画像特徴量 DB（DB の全画像から抽出した画像特徴量を保存した DB）を出力する。実行パートでは、AR アプリケーションからリローカリゼーション（類似画像検索）の実行を要請された際に、クエリ画像を入力として、DB 中からクエリ画像と類似する画像を検索し、出力する。以下では、まず、それぞれのパートについて説明し、その後、提案するフレームワークの予想される利点と欠点について述べる。

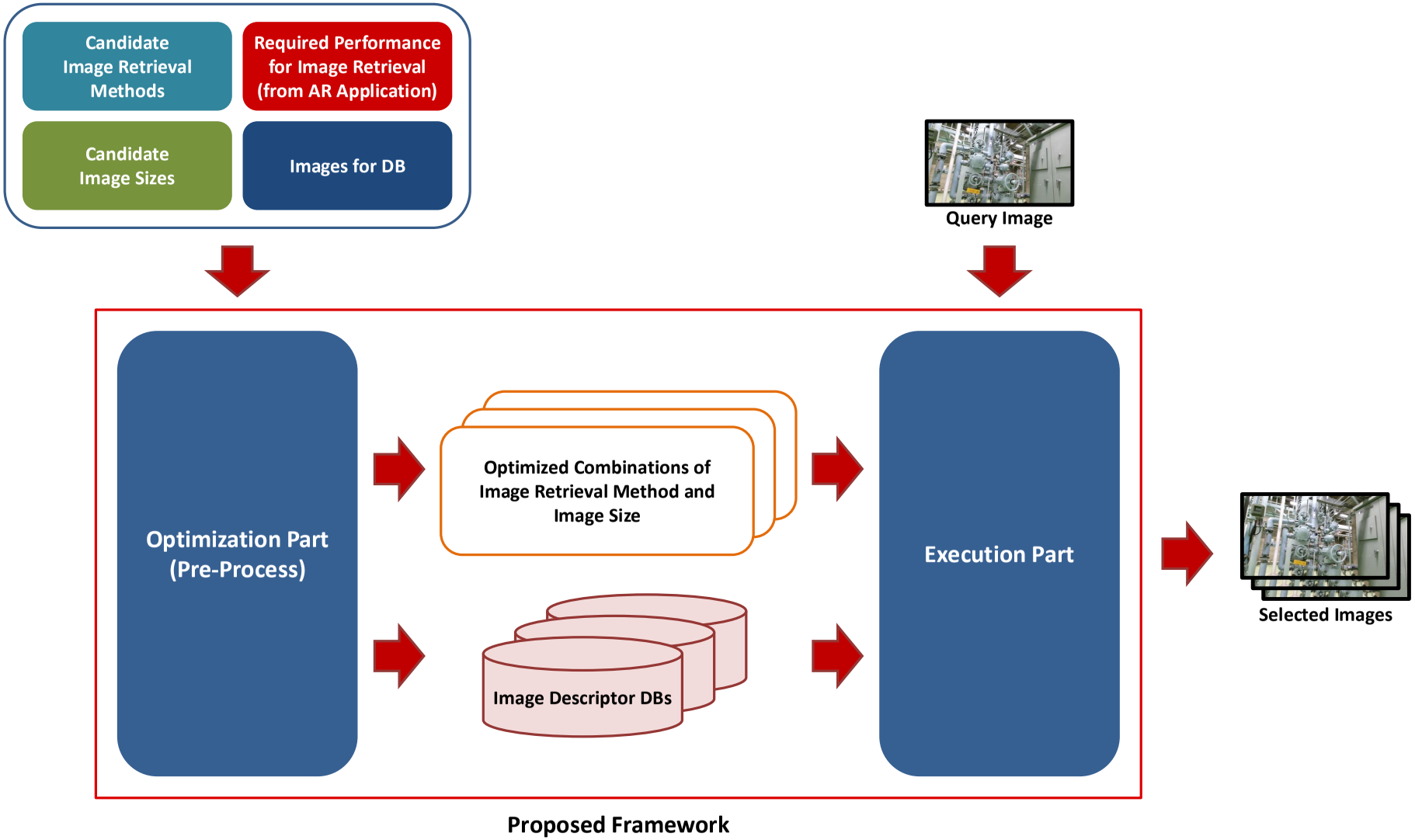


図 3.1: 提案する類似画像検索フレームワークの概要

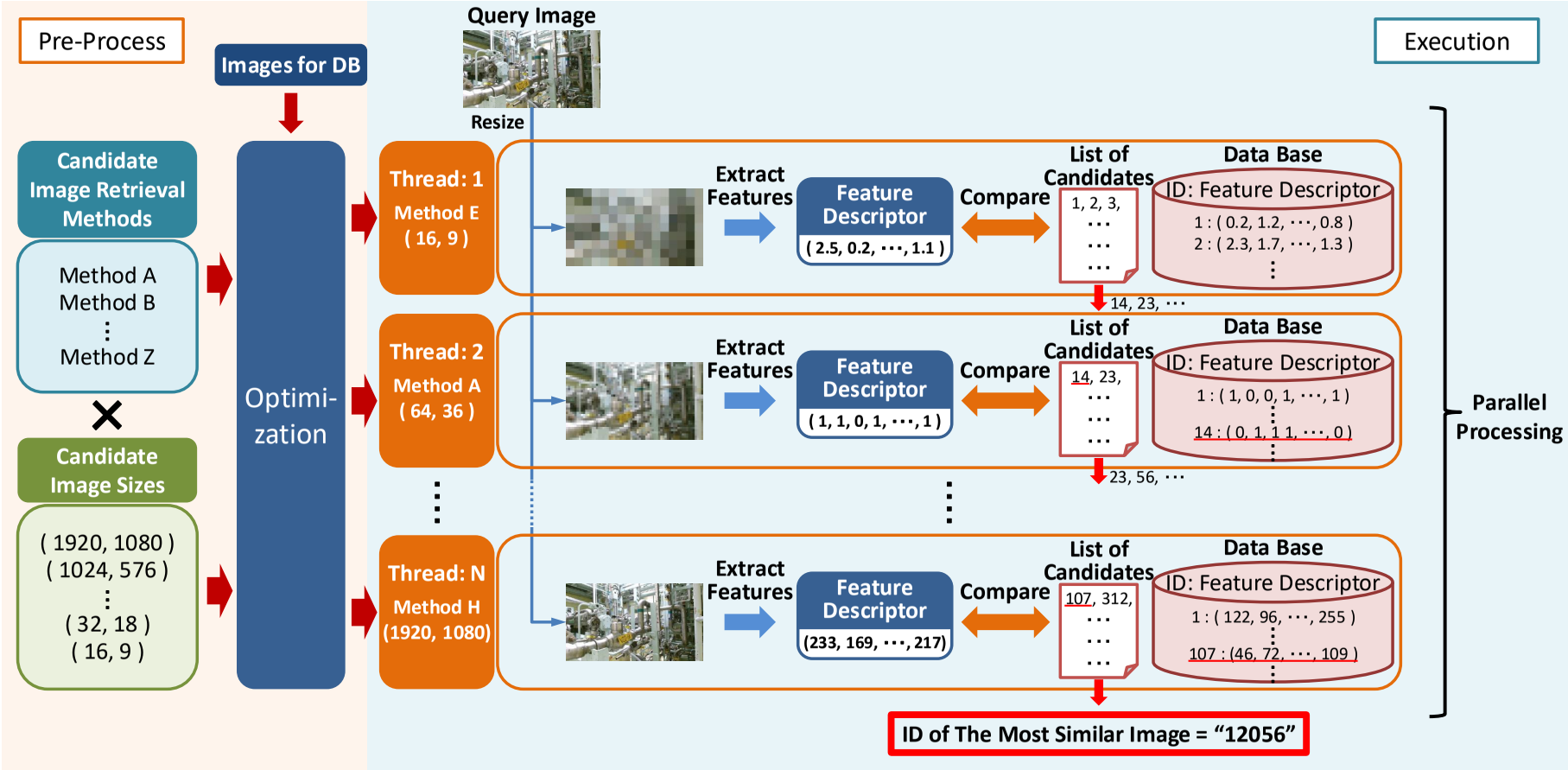


図 3.2: 提案する類似画像検索フレームワークの詳細

3.2.1 画像検索手法の組み合わせの最適化

最適化パートでは、DB に含まれる画像の情報と、許容されるカメラ姿勢の推定誤差や処理速度などのアプリケーションに要求される性能に基づいて、実行パートで使用する画像検索手法と特徴量算出元画像サイズの組み合わせを最適化する。最適化の手順を図 3.3 に示す。最適化では、画像検索手法と特徴量算出元画像サイズの組み合わせを変更しながら、正解が既知のクエリ画像に対して類似画像検索を実行し、処理速度と類似画像の選出精度を評価する。それぞれの組み合わせを評価する際には、その組み合わせでの類似画像検索に用いる画像特徴量を事前に DB の全画像から抽出し、画像特徴量 DB を作成する。画像特徴量 DB は実行パートで利用するため、評価後 HDD に保存する。類似画像検索の実行方法は、3.2.2 項で述べる方法に準ずる。このとき、処理速度は、クエリ画像 1 枚に対して類似画像の選出に要する時間で評価し、類似画像の選出精度は、選出された画像の内、正解の画像の割合を表す適合率や、DB に含まれる正解画像の内、類似画像として正しく選出できた画像の割合を表す再現率、またはそれらの調和平均である F 値 [21] などで評価する。クエリ画像によって評価値にばらつきが生じると予想できるため、クエリ画像を複数枚用意し、全てのクエリ画像に対して類似画像検索を実行した際の評価の合計値を用意したクエリ画像の枚数で除した平均値を最終的な評価値とする。処理速度は AR アプリケーションの他の処理を妨げない程度であれば十分であり、それ以上高速である必要はないと考えられるため、最適化では処理速度を制約条件して扱い、選出精度の評価値を目的関数とする。すなわち、全ての組み合わせの評価が終了した時点で、処理速度がアプリケーションの要求を満たし、かつ、選出精度の評価値が最も高かった組み合わせを最適な組み合わせとする。しかし、最適化は、事前に取得・生成した画像のみを使用して実行する必要があり、アプリケーション利用時の実際のクエリ画像は使用できない。そこで、DB 中の画像の一部を取り出し、それらをクエリ画像の代わりに用いることで最適化を行う。以降、最適化パートでクエリ画像の代わりに利用する DB 中の画像を疑似クエリ画像と呼ぶ。DB 中の画像は撮影時のカメラ姿勢が既知であるため、アプリケーションに要求されるリローカリゼーションの許容誤差（カメラ姿勢の並進誤差が 10cm、回転誤差が 10° 以内など）に応じて、疑似クエリ画像に対してどの画像が類似画像として選出されれば正解として扱えるかを決定できる。疑似クエリ画像を用いた最適化では、DB 作成時の撮影条件と AR アプリケーション利用時の撮影条件が異なる場合に、最適化パートでの評価と実行パートでの性能が異なる可能性がある。この影響については 4.4 節で検討する。

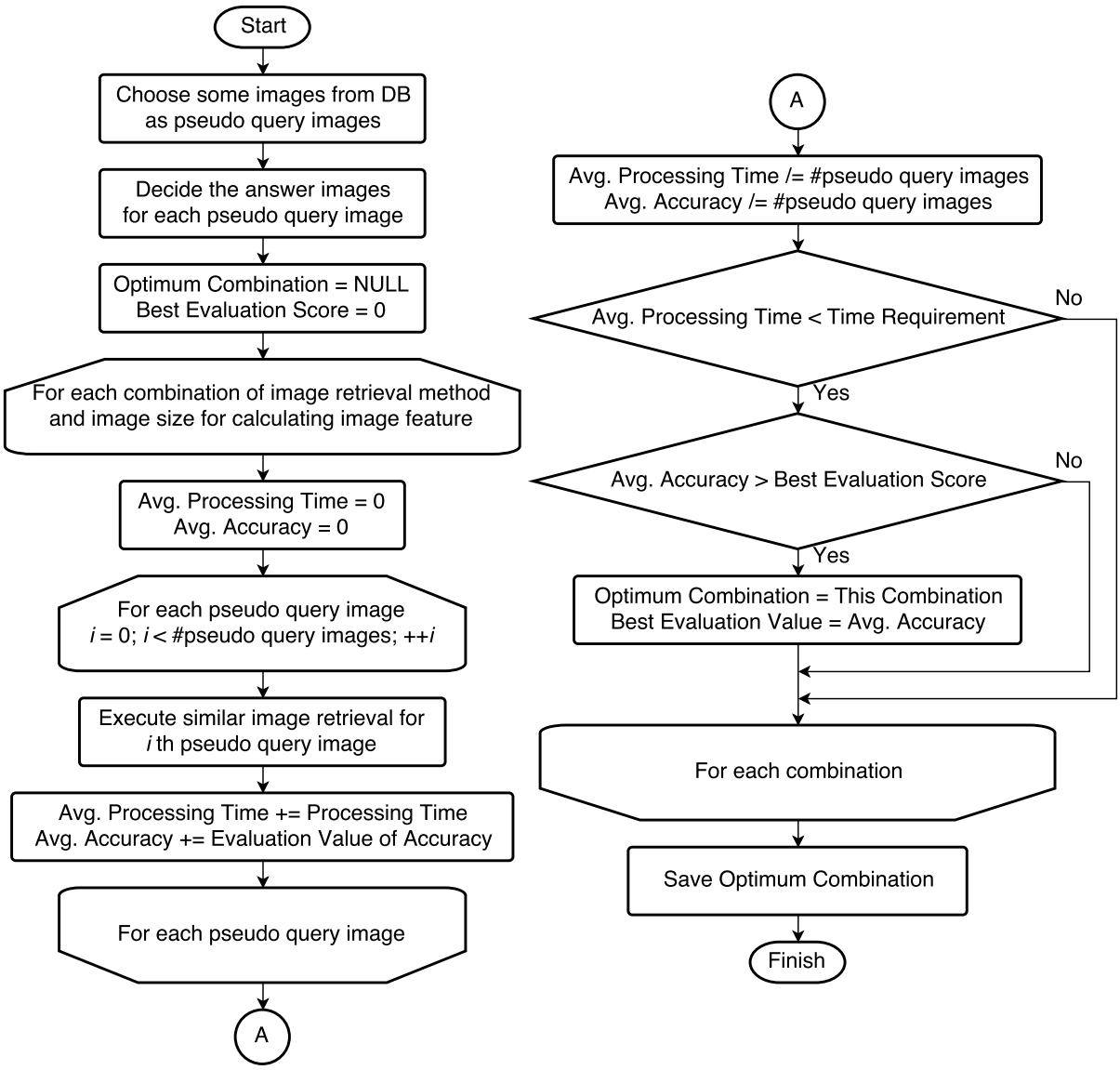


図 3.3: 段階的な絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの最適化手順

3.2.2 類似画像検索の実行

実行パートでは、最適化パートで最適化された画像検索手法と特徴量算出元画像サイズの組み合わせで、段階的な絞り込み処理による類似画像検索を実行する。クエリ画像とDB中の画像との類似度を基に、類似画像を選出する方法として、図3.4に示すように、類似度の閾値を設けて、閾値以上の類似度を持つ画像を類似画像として選出する方法や、図3.5に示すように、類似度が高い順に候補画像のリストをソートして、ソート結果が上位の画像から予め定めた枚数の画像を抜き出して類似画像として選出する方法が一般的である。前者の閾値による選出方法を段階的な絞り込み処理に用いる場合、ある絞り込み段階でクエリ画像と候補画像の類似度が算出でき次第、その画像を次の絞込み段階へ通過させるか否かを決定できるため、各段での絞り込み処理を並列化して全体の処理を高速化できる（DB内の画像をランダムな順番に調べる場合、特定のタイミングに連続して複数枚の画像が絞り込みを通過することは少なく、次段の処理の負荷はほぼ一定になると予想される）。しかし、閾値の設定が性能に大きく影響する、各段での絞り込みを通過する画像の枚数が毎回異なるなどの問題がある。一方で、後者のソートによる選出方法を段階的な絞り込み処理に用いる場合、各段での絞り込みを通過する画像の枚数を任意に決定できるため、安定して同じ枚数の類似画像を選出することができるが、クエリ画像と候補画像の類似度の算出が全て終了した後でなければ、絞り込みを通過させる画像を決定できないため、各段での絞り込み処理を並列化できない。

そこで、絞り込みを通過させる画像を決定する方法（通過画像決定方法: Filtering Strategy）として、図3.6に示すように、DB内の画像を複数のグループ（各グループ内にN枚の候補画像）に分割し、各グループN枚の候補画像に対して類似度を算出する毎に、N枚の中で類似度が上位M枚の画像を通過させる方法を提案する。この提案手法を本論文では部分ソートフィルタリングと名付ける。部分ソートフィルタリングでは、毎回既定の枚数の画像が絞り込みを通過する上、順次通過する画像が決定されていくため、処理の並列化が可能である。部分ソートフィルタリングを用いた段階的な絞り込み処理の流れを図3.7〜図3.10に示す。1段目の絞り込み処理では、クエリ画像をDBの全画像と比較し、2段目の絞り込み処理へ通過させる画像を順次決定していく。2段目の絞り込み処理では、クエリ画像を1段目から通過してきた画像と順次比較し、次の絞り込み処理へ通過させる画像を決定していく。n段目（最後）の絞り込み処理では、クエリ画像を前段から通過してきた画像と順次比較し、類似画像を選出する。

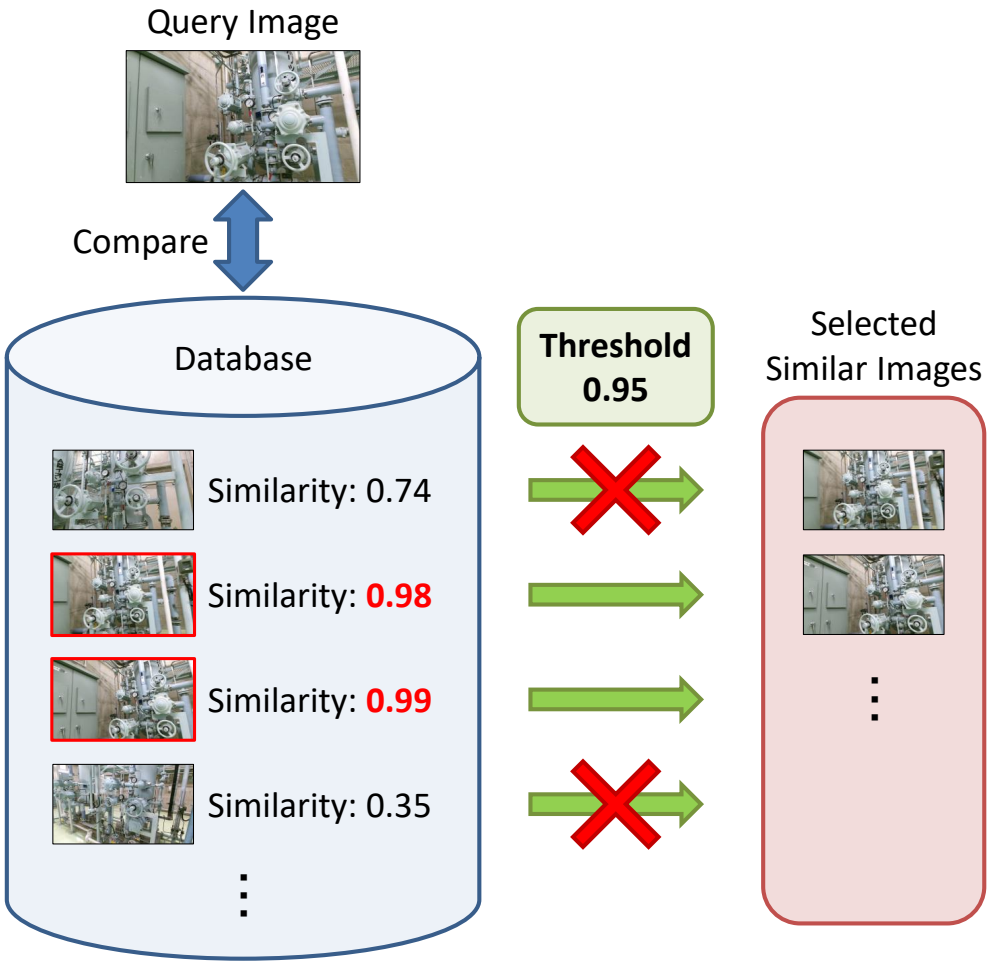


図 3.4: 類似度の閾値を用いた類似画像選出方法

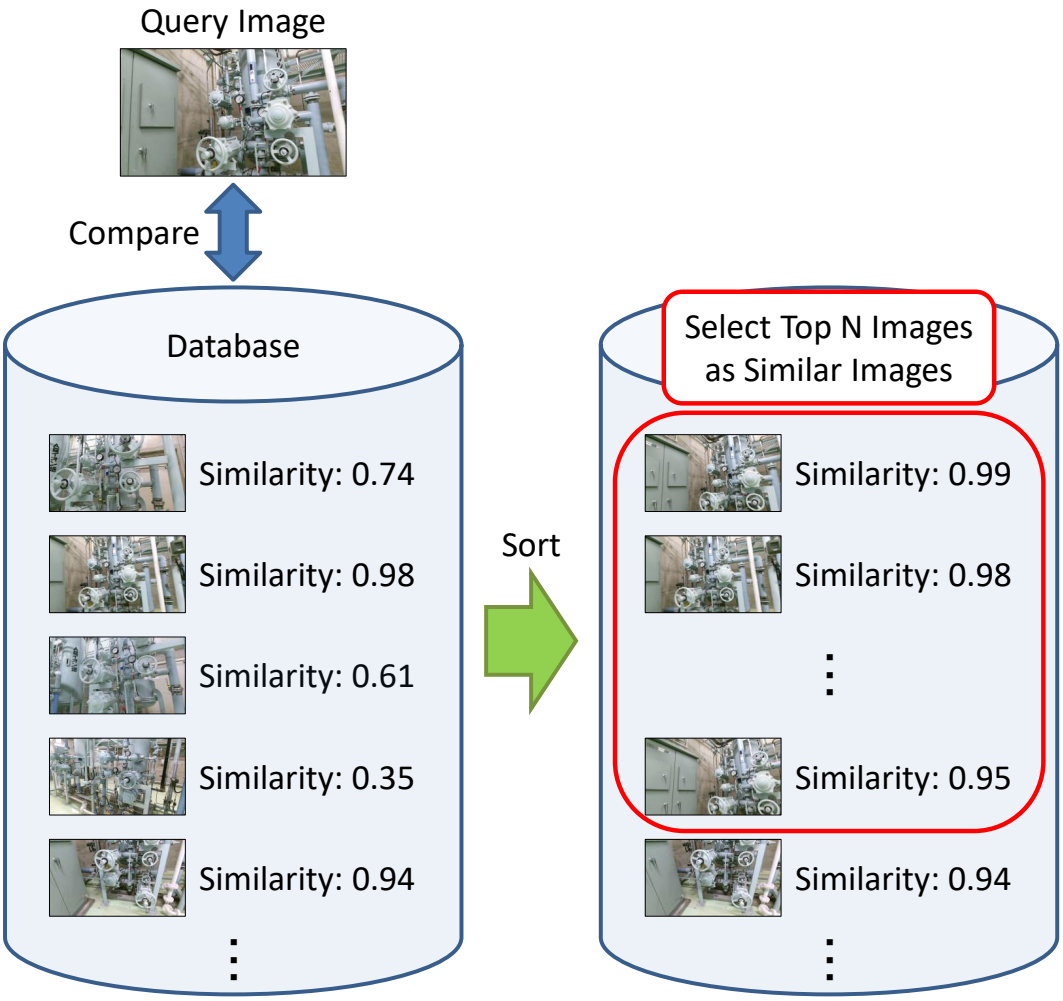


図 3.5: 類似度によるソートを用いた類似画像選出方法

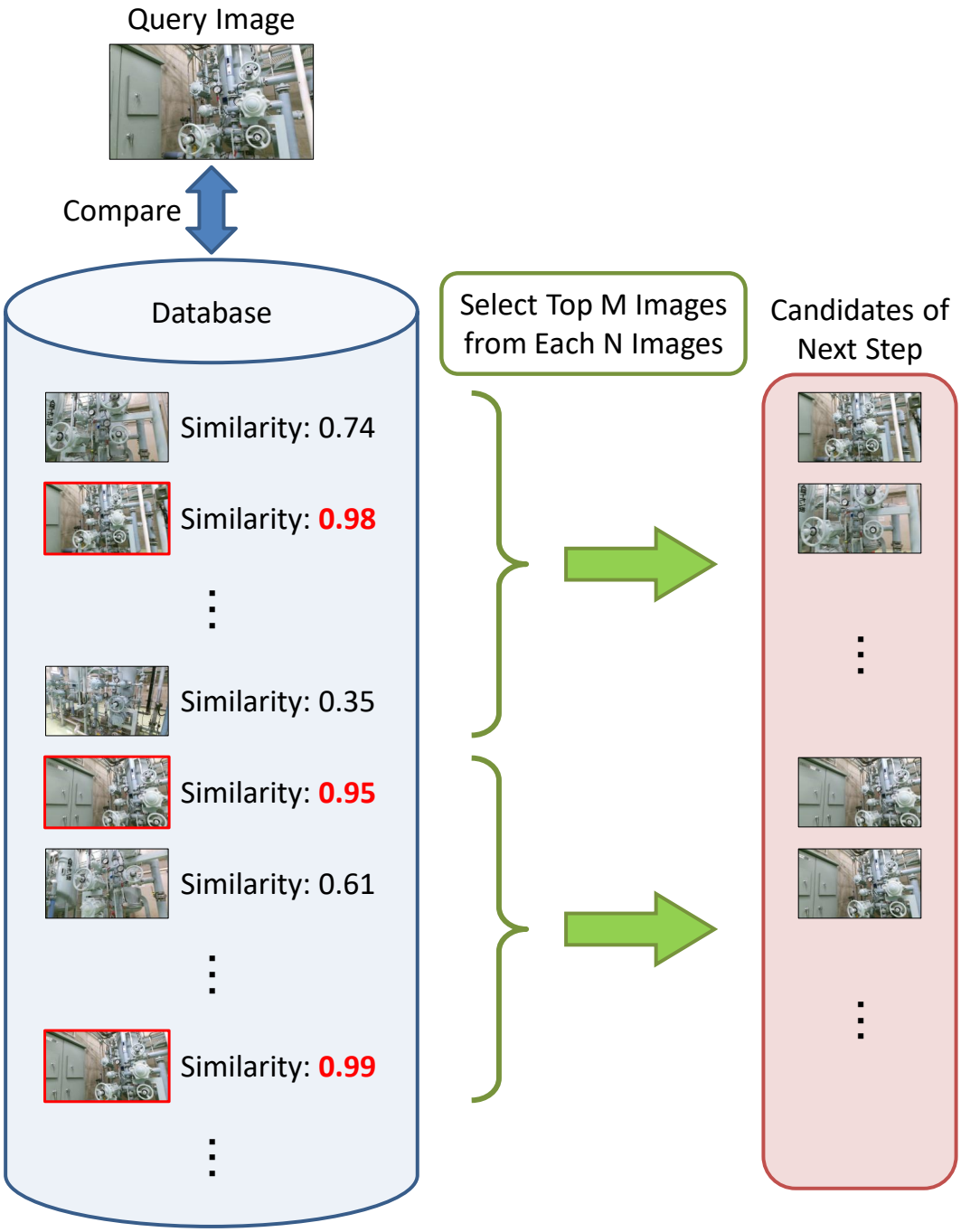


図 3.6: 部分ソートフィルタリングによる類似画像の候補の絞り込み

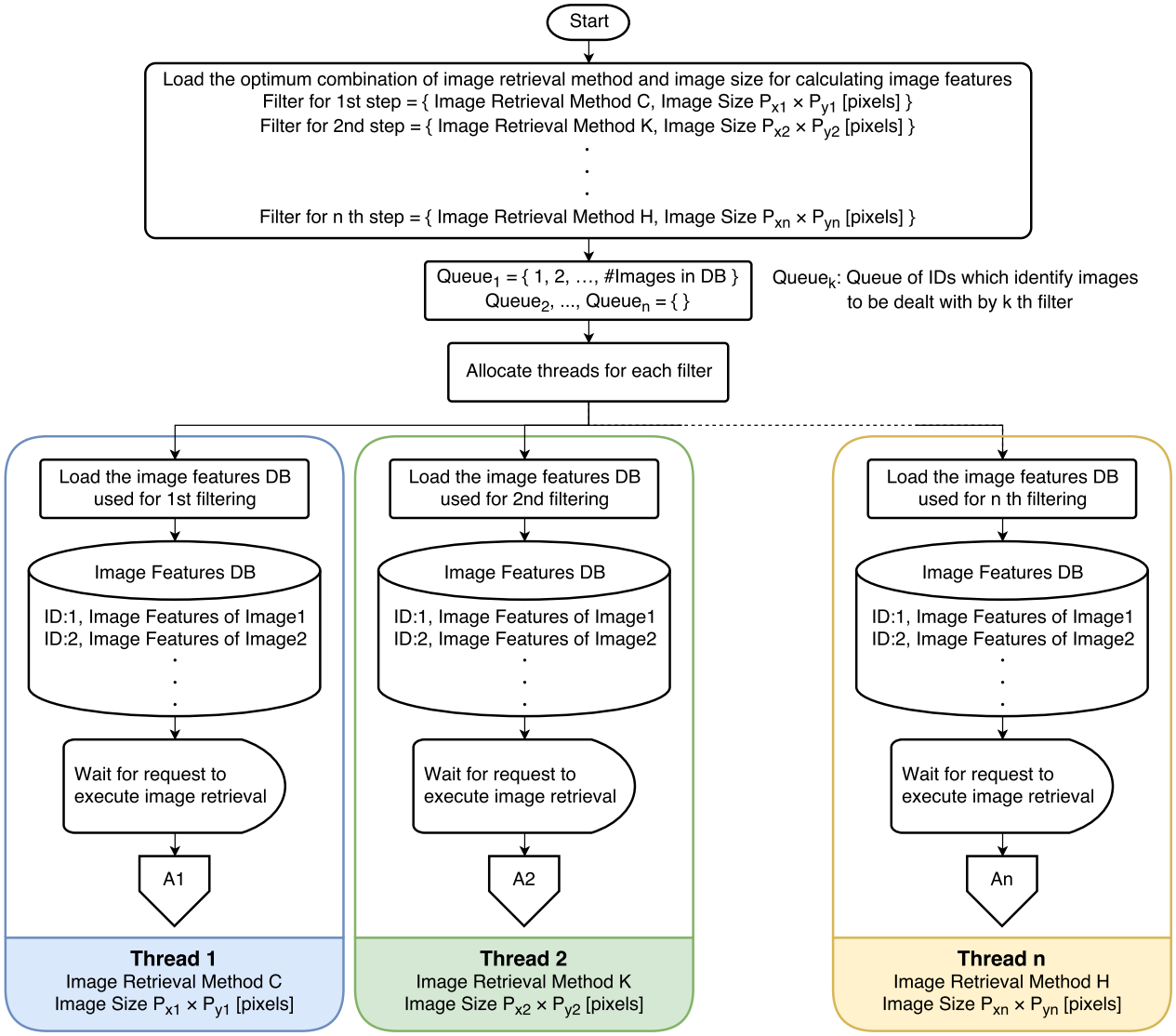


図 3.7: 部分ソートフィルタリングを用いた段階的な絞り込み処理（準備）

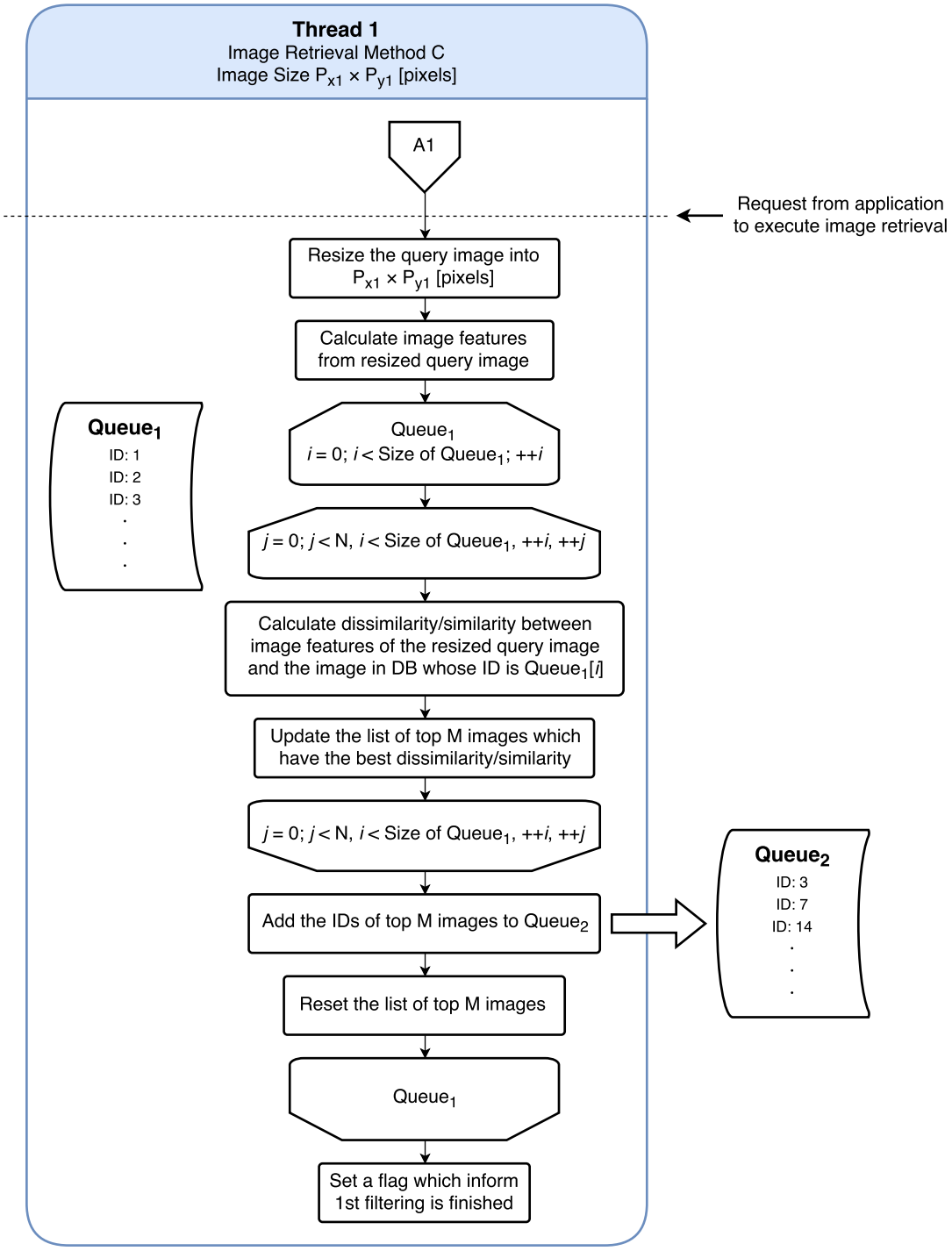


図 3.8: 部分ソートフィルタリングを用いた段階的な絞り込み処理（1 段目の絞り込み処理の実行）

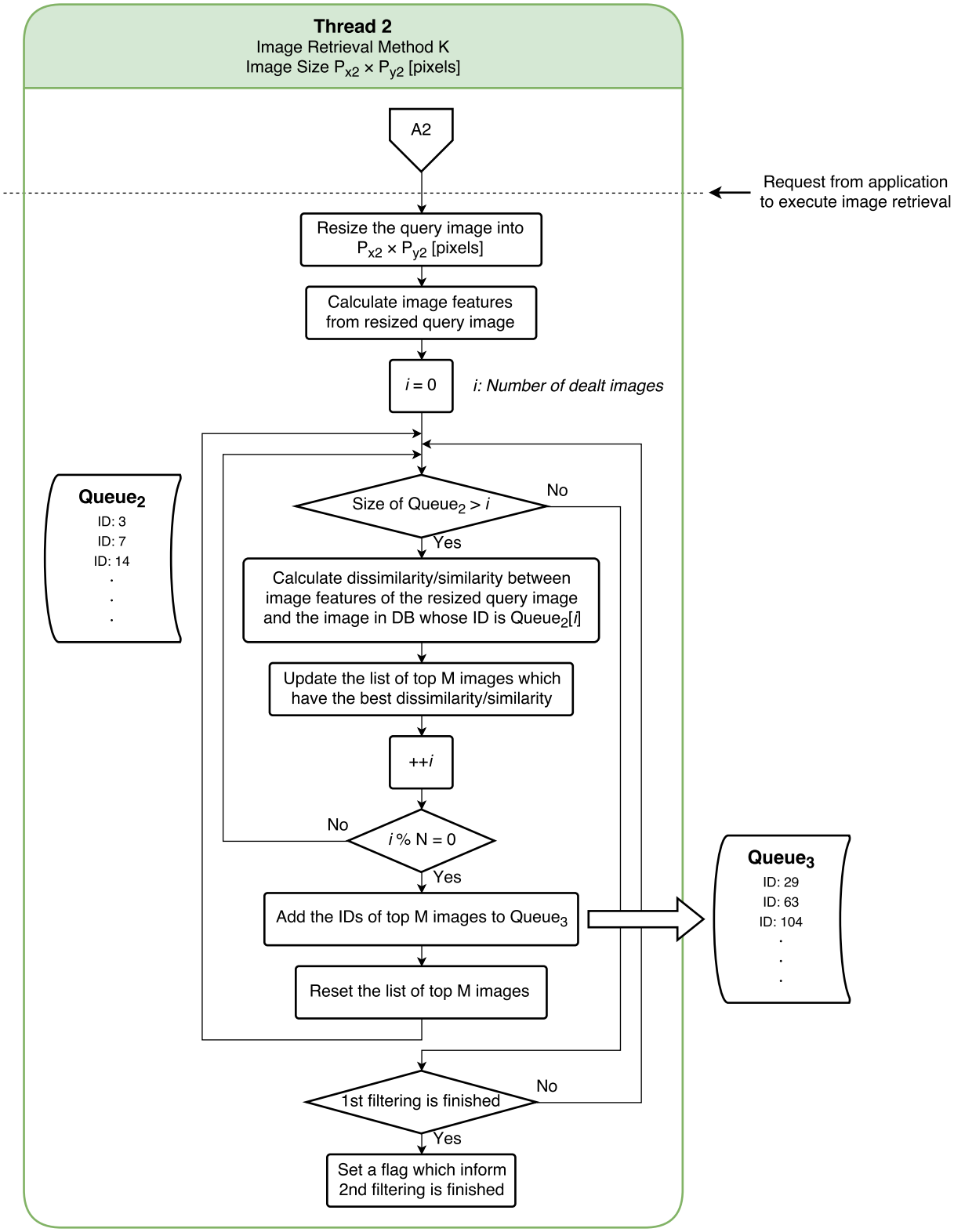


図 3.9: 部分ソートフィルタリングを用いた段階的な絞り込み処理（2 段目の絞り込み処理の実行）

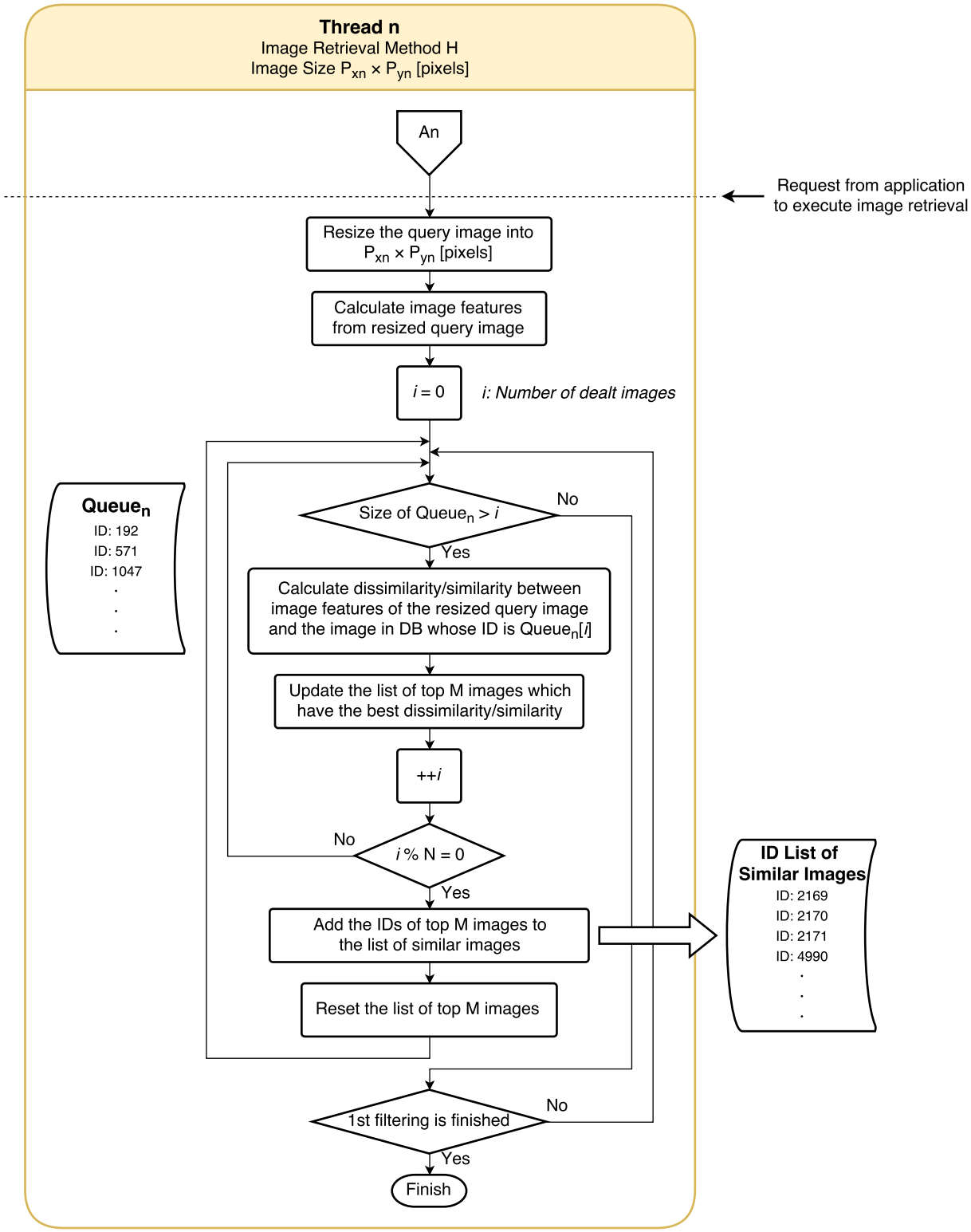


図 3.10: 部分ソートフィルタリングを用いた段階的な絞り込み処理（n 段目の絞り込み処理の実行）

3.2.3　提案手法の予想される利点と欠点

提案するフレームワークを利用することで、複数の画像検索手法の利点を併せ持った、より高性能な類似画像検索が実現できると予想される。また、実際にアプリケーションを利用する環境の画像で構築された DB を利用して、画像検索手法と特徴量算出元画像サイズの組み合わせを最適化するため、利用環境に応じた最適化結果が現れると期待される。加えて、今後新たにリローカリゼーションに有効な可能性のある画像検索手法が開発された際には、最適化の処理によってその手法の性能を評価し、既存手法より高性能であれば段階的な絞り込み処理の一部に組み込むことが可能である。

一方、絞り込みの段数を増やすにしたがって、画像検索手法と特徴量算出元画像サイズの組み合わせの数が指数関数的に増加するため、最適化に多くの時間を要することが懸念される。そのため、最適化前に、候補となる画像検索手法と特徴量算出元画像サイズの組み合わせをある程度絞り込んでおくことなどが必要となる。

第 4 章 提案手法の評価

4.1 評価の目的と概要

本研究では、

1. 通過画像決定方法として、3.2 節で提案した部分ソートフィルタリングが適切であるか

2. 複数の画像検索手法を組み合わせることで、個々の手法を単独で利用する場合よりも高い類似画像検索性能を実現できているか

3. 最適化アルゴリズムによって、利用する環境ごとに適切な画像検索手法と特徴量算出元画像サイズの組み合わせを選択できているか

の 3 点を確認することを目的として、3 種類の評価実験により、提案するフレームワークを評価した。

1 つ目の通過画像決定方法の評価では、複数の画像検索手法を組み合わせて類似画像検索を実行する際に、部分ソートフィルタリングを用いた場合、閾値による通過画像決定方法を用いた場合、全候補画像との類似度をソート（全体ソート）する通過画像決定方法を用いた場合のそれぞれで類似画像検索性能を評価し、最も適切な通過画像決定方法を検討した。

2 つ目の複数の画像検索手法を組み合わせた際の類似画像検索性能の評価では、複数の画像検索手法を組み合わせた多段階絞り込み処理により類似画像検索を実行した場合と、1 つの画像検索手法で類似画像検索を実行した場合の性能を比較し、複数の画像検索手法を組み合わせることで類似画像検索性能が向上するかを確認した。

3 つ目の組み合わせ最適化アルゴリズムの評価では、最適化された画像検索手法と特徴量算出元画像サイズの組み合わせで類似画像検索を実行した場合と、その他の画像検索手法と特徴量算出元画像サイズの組み合わせで類似画像検索を実行した場合の性能を比較し、最適化アルゴリズムで適切な組み合わせを選択できているかを確認した。

4.2　通過画像決定方法の評価

本節では、適切な通過画像決定方法を検討するために行った、1 つ目の評価実験について述べる。

4.2.1　評価の方法

評価の際の DB 画像やクエリ画像には、原子炉廃止措置研究開発センター（ふげん）内で撮影して作成したデータセットの画像を用いた。データセットの内、DB の構築に使用する画像のセットを DB 用データセット、性能評価時にクエリ画像として用いる画像のセットをテスト用データセットと呼ぶ。データセットの詳細は 4.2.2 項で述べる。

まず、DB 用データセットを用いて画像検索手法と特徴量算出元画像サイズの組み合わせの最適化を行った。最適化の際の類似画像検索性能の評価では、通過画像決定方法として部分ソートフィルタリングを用いた。最適化の際に用意した画像検索手法と特徴量算出元画像サイズの候補や、段階的な絞り込み処理の段数など、フレームワークの実装の詳細は 4.2.3 項で述べる。

次に、最適化された画像検索手法と特徴量算出元画像サイズの組み合わせで、テスト用データセットのクエリ画像に対して類似画像検索を実行し、その性能を評価した。このとき、実行時の通過画像決定方法に、部分ソートフィルタリングを用いた場合、閾値による方法を用いた場合、全体ソートによる方法を用いた場合のそれぞれで評価を行った。

4.2.2　評価に用いたデータセット

データセットの画像は、原子炉廃止措置研究開発センター（ふげん）の純水装置室内を撮影して取得した。純水装置室の見取り図と取得した画像の例を図 4.1 に示す。図 4.1 に示す赤枠で囲まれた領域で AR を用いた作業支援システムを利用することを想定し、撮影はこの領域内で行った。撮影には、Microsoft 社の Xbox One Kinect センサー（Kinect v2）を使用した。Kinect v2 の外観を図 4.2 に、仕様を表 4.1 に示す。

DB 用データセットとテスト用データセットは、それぞれ別に撮影した動画から抜き出した画像で構成した。画像には抜き出した順番に連続する番号の ID を与えた。Kinect v2 では RGB 画像と同時に、被写体までの距離を記録した画像（デプス画像）も取得できるが、本研究では RGB 画像のみを利用した。また、実際にリローカリゼーションを実行する際には、DB を構成する画像の撮影時のカメラ姿勢は既知でなければならな

いが、本データセットの画像は撮影時のカメラ姿勢が不明である。そのため、類似画像検索性能を評価する際には、画像間の正規化相互相関を利用した。詳細は 4.2.3.4 条および 4.2.6 項で述べる。

DB 用データセットは、作業支援システムの利用を想定した領域全体でカメラ姿勢をゆっくり変化させながら撮影した。このとき、作業支援システム利用時に取り得るカメラ姿勢を考慮し、カメラは床から約 80〜170cm の高さの範囲を動かした。可能な限り多くのカメラ姿勢でリローカリゼーションに成功できるようにするため、カメラを上下左右様々な方向に向けて撮影を行った。その結果、撮影時間は約 60 分となり、その動画から画像を抜き出したところ、画像は 32,491 枚となった。

テスト用データセットは、多様なクエリ画像に対する類似画像検索性能を評価するために、DB 用データセットと同様に作業支援システムの利用を想定した領域全体を撮影して作成した。テスト用データセットとして約 3 分の動画データから抜き出したところ、画像は 2,329 枚となった。

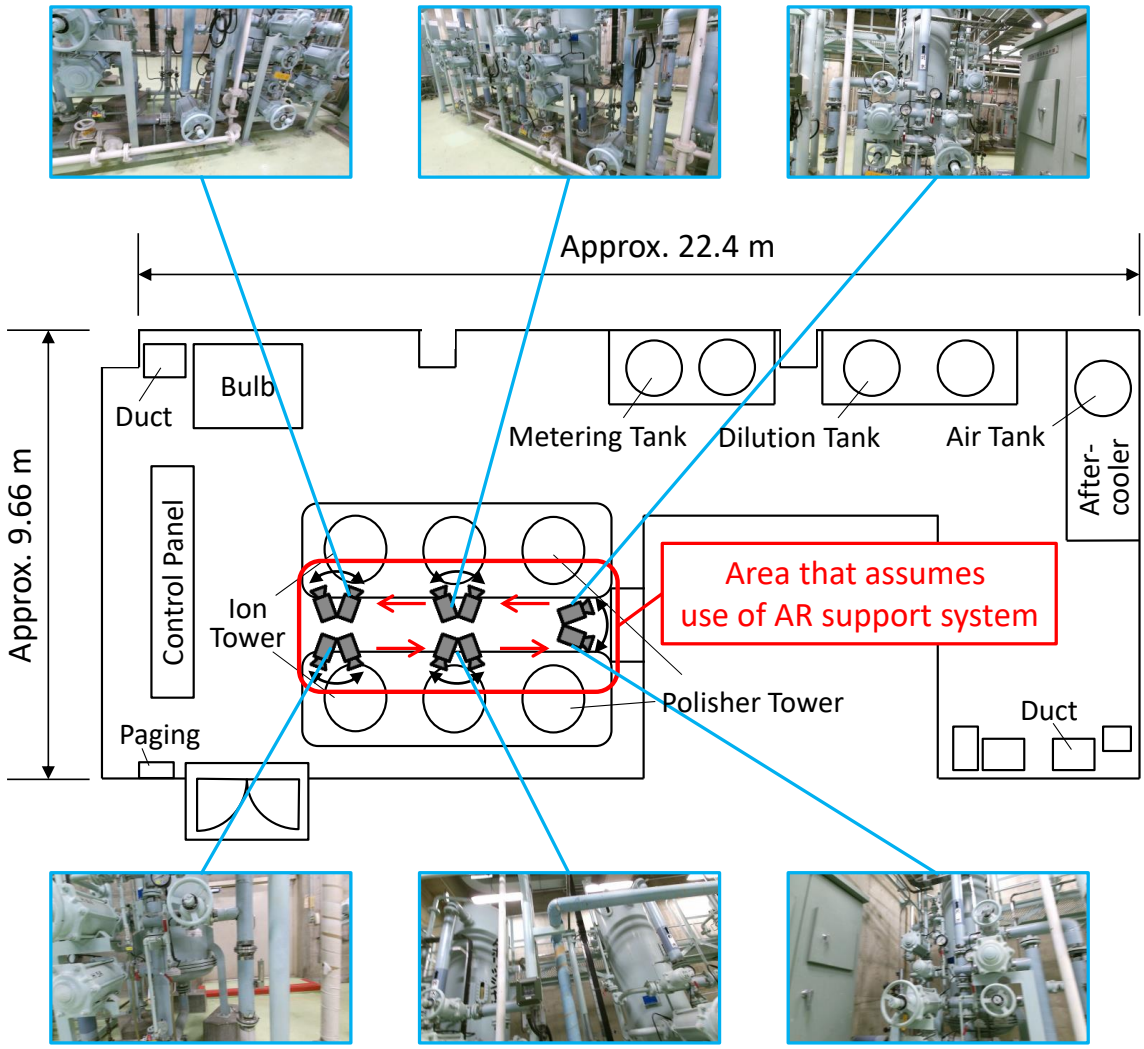


図 4.1: 純水装置室の見取り図と撮影された画像の例

図 4.2: Kinect v2 の外観

表 4.1: Kinect v2 の仕様

Size	H 67mm × W 249mm × L 66mm
Weight	About 1.4 kg
Resolution	1920 × 1080 pixels
Field of View*	Horizontal 84.1° × Vertical 53.8°
Frame Rate	30 fps
Color	24bit RGB

*Retrieved from SDK[22]

4.2.3 多段階絞り込み処理フレームワークの実装

4.2.3.1 実装に用いたハードウェアおよびソフトウェア

フレームワークの実装に用いた PC の仕様と開発環境、開発に利用したライブラリを表 4.2 に示す。DB 画像の画像特徴量の読み書きには Boost の serialization ライブラリを、絞り込み処理の並列化には Boost の thread ライブラリを利用した。また、画像の読み出し、画像サイズの変更などの各種画像処理、画像特徴量の計算、画像特徴量間の類似度の計算には OpenCV を利用した。

表 4.2: フレームワークの実装に用いた PC の仕様と開発環境および開発に利用したライブラリ

PC	CPU	Intel(R) Core(TM) i7-3770 3.40 GHz (4 Cores, 8 Threads)
	RAM	16.0 GB
	OS	Windows 10 Pro (64bit)
	Virtual Memory	TS240GSSD220S(240 GB)×2 RAID 0
Development Environment	IDE	Visual Studio 2017
	Language	C++
Libraries	Boost 1.65.1[23]	
	OpenCV 3.3.0 with contrib[24]	

4.2.3.2　実行パートの実装

最適化パートでは、実行パートの実装に従って類似画像検索を行った際の性能を評価するため、先に実行パートの実装を述べる。

AR アプリケーションを利用する際に用いられる、近年の一般的な PC やタブレットの CPU は、物理コアを 4 個程度持つものが多いが、全てのコアをリローカリゼーションの処理に使用すると、アプリケーションの画面の更新やユーザの操作の処理が遅くなる場合がある。そこで、今回の評価では、並列の処理は 3 つのスレッドで処理することとし、実行パートの段階的な絞り込み処理の段数は 3 段とした。

部分ソートフィルタリングでは類似画像が順次選出されるため、リローカリゼーションに与えられた時間内に逐次トラッキングへの復帰処理を試行できる回数が比較的多いと考えられる。そのため、最終的に選出する類似画像の枚数を、一般的な枚数（5 枚程度）より多い 10 枚に設定し、部分ソートフィルタリングのパラメータ M, N（N 枚ごとに M 枚の画像を通過させる）を、3 段階の絞り込み全てで共通の N=15, M=1 とした。類似画像の候補を 15 枚ずつ処理した際に、最後に残る画像（15 枚未満の画像グループ）からも 1 枚の画像を通過させた。このとき類似画像の候補は、32,941 枚（DB に含まれる画像数）→ 2,197 枚（32,941÷15+1）→ 147 枚（2,197÷15+1）→ 10 枚（147÷15+1）と絞り込まれる。

また、閾値による通過画像決定方法を用いる際には、部分ソートフィルタリングと同程度の類似画像選出枚数で評価を行うため、テスト用データセットのクエリ画像に対して類似画像検索を実行した場合に、1 段目、2 段目、3 段目の絞り込みを通過する画像の平均枚数がそれぞれ 2,197 枚、147 枚、10 枚となるように閾値を設定した。全体ソートによる通過画像決定方法を用いる際も、部分ソートフィルタリングと同様に、32,941 枚→ 2,197 枚→ 147 枚→ 10 枚と類似画像の候補を絞り込んだ。

処理速度を向上させるために、類似画像検索に用いる DB 画像の画像特徴量は事前に読み込んでおくことが望ましい。しかし、今回のデータセットの場合では、読み込みに必要なメモリ量が最大で 64GB と大きく、物理メモリ上に全てを常に格納することができなかったため、類似画像検索の実行中はその一部を仮想メモリ上に保持した。アプリケーションで使用可能なメモリ量が限られる場合は、最適化時の制約条件に必要メモリ量の上限を加えるなどの対応が必要である。

4.2.3.3　最適化パートの実装

最適化パートでは画像検索手法と特徴量算出元画像サイズの組み合わせを変更しながら、疑似クエリ画像に対して類似画像検索を実行した際の性能を評価することで、最適な組み合わせを決定する。

リローカリゼーションは、逐次トラッキングに失敗した場合など現在のカメラ姿勢が不明である場合にのみ実行するため、毎フレーム実行する必要はない。そのため、カメラ画像の標準的な更新速度である 33msec より実行速度が遅くても良いが、前述のように、絞り込まれた複数の画像に対して逐次トラッキングへの復帰を試みるため、ユーザにストレスを感じさせにくい待ち時間とするために、類似画像検索の実行に許容される時間は 1 クエリ画像あたり 300msec 程度が妥当であると考えられる。

絞り込み段数を 3 段とした場合、全ての画像検索手法と特徴量算出元画像サイズの組み合わせの数は、画像検索手法の候補数と特徴量算出元画像サイズの候補数の積の 3 乗となる。そのため、全ての組み合わせの中から最適な組み合わせを求めようとした場合、評価対象となる組み合わせの数が膨大となり、評価に非常に長い時間を要する。そこで本研究では、最長でも数日程度の実用的な時間で最適化を完了するために、1 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズから順に 3 段目まで最適化する。

まず、1 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせを変更しながら、それぞれの組み合わせで 1 段目の絞り込みを実行し、その性能（速度と精度）を評価する。絞り込みの精度は、4.1 式で定義される再現率（Recall）で評価する。

$$\text{再現率} = \frac{\text{絞り込みを通過した画像中の正解画像の枚数}}{\text{DB 中の正解画像の枚数}} \tag{4.1}$$

図 4.3 に示すように、疑似クエリ画像 1 枚あたりに対して、絞り込みに要した時間が 300msec 以下であった画像検索手法と特徴量算出元画像サイズの組み合わせの内、最も精度が高かった上位 3 つの組み合わせを、1 段目の絞り込み処理に最適な組み合わせの候補とする。このとき、上位 3 つの組み合わせの中に、画像検索手法が同じで特徴量算出元画像サイズのみが異なる組み合わせが複数存在した場合、それらの組み合わせの内、精度が最も高かった組み合わせ以外を候補から除外し、画像検索手法が異なる別の組み合わせの中から、最も精度が高かった組み合わせを 1 段目の絞り込み処理

に最適な組み合わせの候補に繰り上げることで、候補に残る画像検索手法の種類を確保する（画像検索手法が同じで特徴量算出元画像サイズのみが異なる組み合わせでは、次の段の画像検索手法と組み合わせた場合の性能の変化が同じ傾向になると予想されるため）。次に、1,2 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせを最適化する。図 4.4 に示すように、1 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせは、最適な組み合わせの候補として選出された 3 つの組み合わせのいずれかとし、2 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせを変更しながら、2 段目までの絞り込みを実行し、その性能を評価する。ここでも 1 段目の最適化時と同様に、絞り込みに要した時間が 300msec 以下であった画像検索手法と特徴量算出元画像サイズの組み合わせの内、最も精度が高かった上位の組み合わせから、画像検索手法が異なる 3 つの組み合わせを 1,2 段目の絞り込み処理に最適な組み合わせの候補とする。最後に、1,2 段目の絞り込み処理に最適な組み合わせの候補に、様々な組み合わせの 3 段目の絞り込み処理を加えて類似画像検索を実行し、その性能を評価する。絞り込みに要した時間が 300msec 以下であった画像検索手法と特徴量算出元画像サイズの組み合わせの内、最も精度が高かった組み合わせを最適な組み合わせとする。

このように、1 段目の絞り込み処理から順に最適化を行うことで、全ての組み合わせを評価する場合と比較して、評価する組み合わせの数を大幅に削減できる。また、各段階で最適な組み合わせを 1 つに決定するのではなく、複数の候補を残すことで、性能が高い組み合わせの中から多様な組み合わせを評価できる。これにより、例えば 1 段目では評価が 2 番目や 3 番目であった手法が、2 段目に組み合わせる手法との相乗効果で高い性能を発揮する場合などに、その組み合わせを見落とすことなく最適化できる。

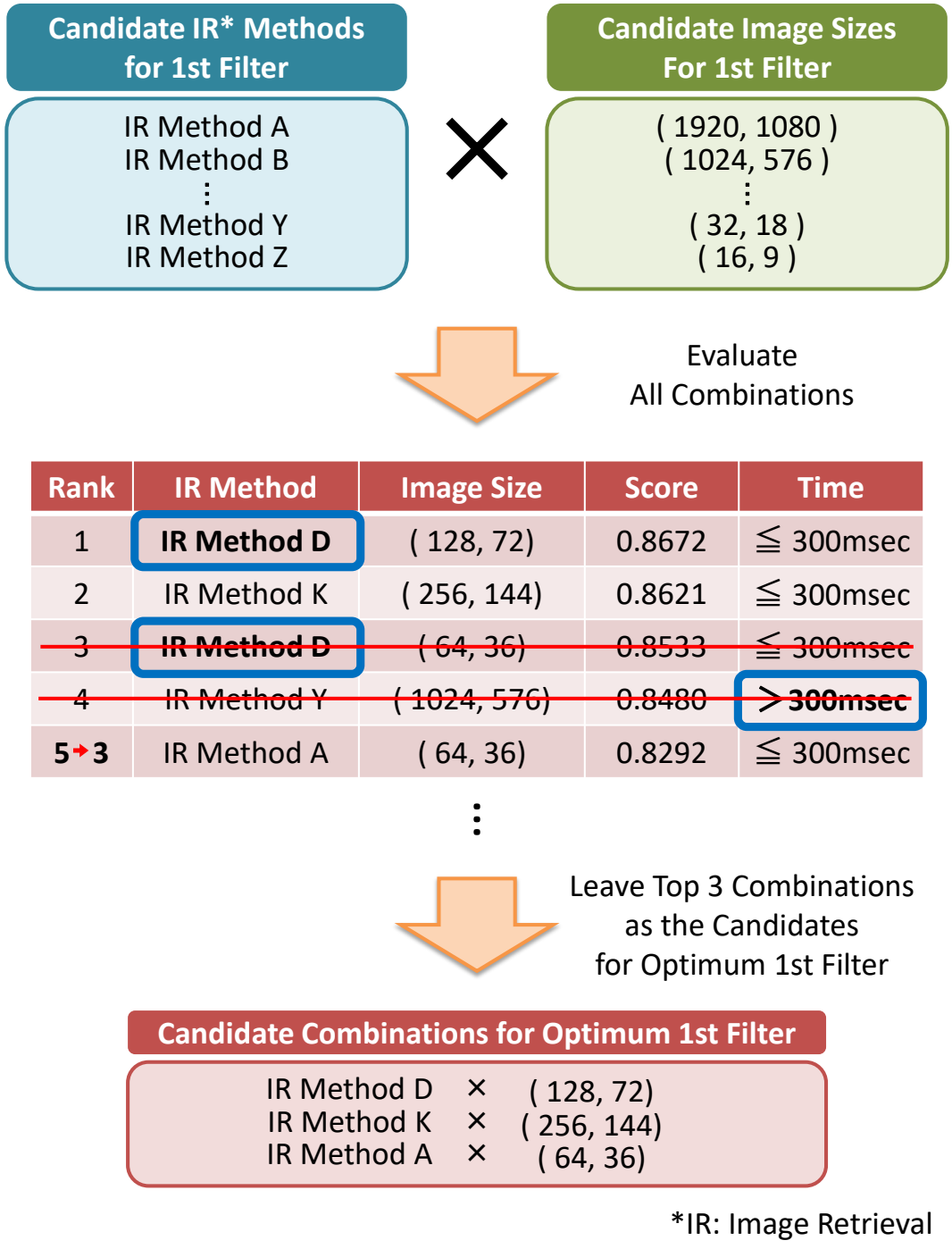


図 4.3: 1段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの最適化

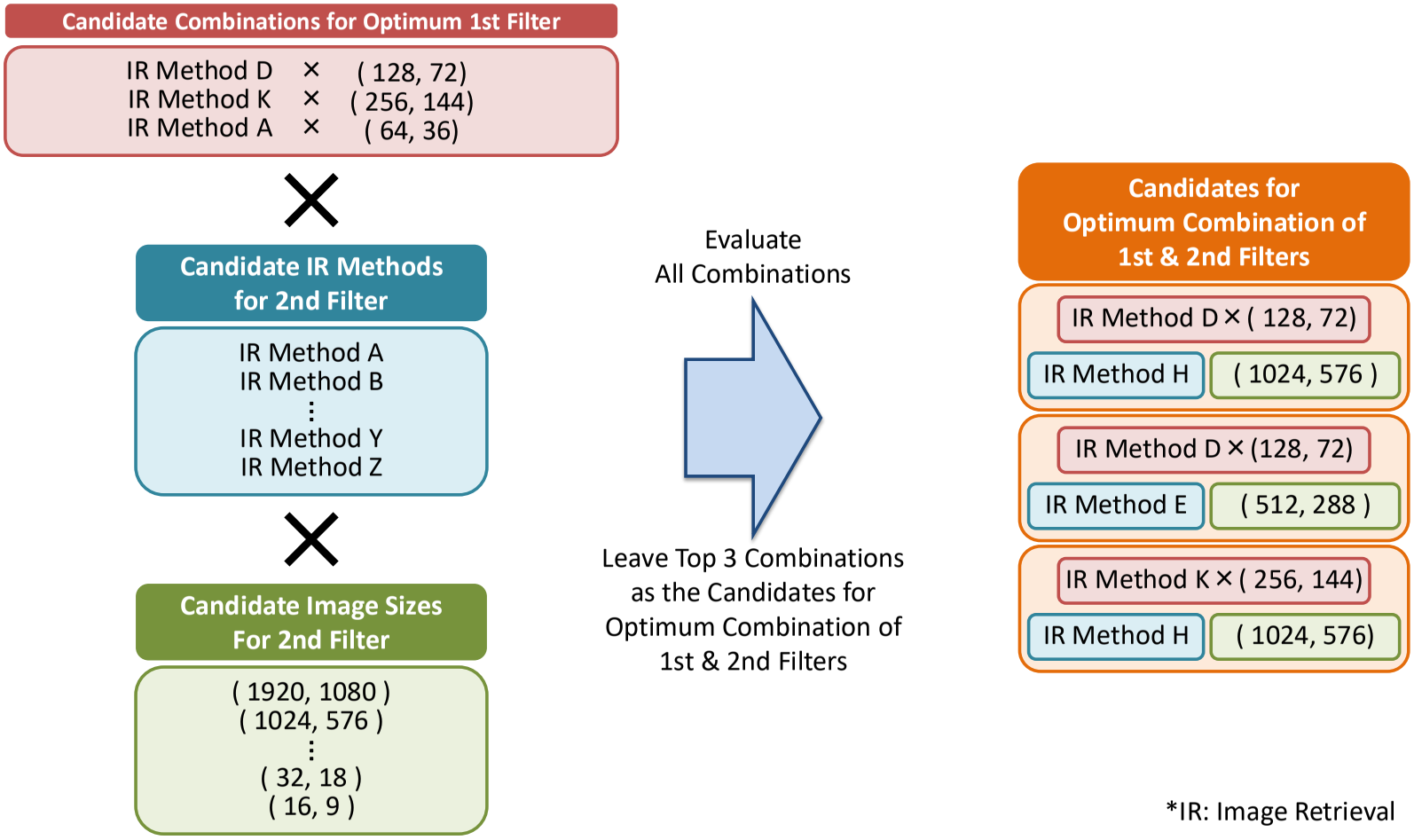


図 4.4: 1,2 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの最適化

4.2.3.4　最適化に用いた疑似クエリ画像と正解画像の決定方法

最適化の際には、DB に含まれる画像全体の 5%の画像を疑似クエリ画像とした。このとき、疑似クエリ画像にできるだけ多くの場所を撮影した画像を含めるために（似た画像が多くなることを防ぐために）、等間隔に離れた ID を持つ画像を疑似クエリ画像に選択した。それぞれの疑似クエリ画像について、DB に含まれる画像の内、疑似クエリ画像との正規化相互相関が 0.90 以上の画像を正解画像とした。しかし、正規化相互相関は画像間の類似度を正確に数値化できる一方、計算コストが大きいため、疑似クエリ画像と DB の全ての画像の正規化相互相関を計算して正解画像を決定した場合、膨大な時間を要する。そこで、それぞれの疑似クエリ画像について、DB に含まれる画像の内、疑似クエリ画像の ID と近い ID の画像から順に正規化相互相関を求めた。このとき正規化相互相関が 0.90 より小さい画像が見つかった時点で、それより離れた ID を持つ画像中に正解画像が存在することは少ないと判断し、DB の残りの画像との正規化相互相関の計算は省略した。

4.2.3.5　画像検索手法と特徴量算出元画像サイズの候補

複数の画像検索手法を組み合わせる際の候補として実装した画像検索手法を表 4.3〜表 4.5 に示す。各手法で類似画像検索に用いる画像特徴量（Feature Descriptor）の概要を上段に、画像間の相違度（Desimilarity）の定義方法を下段に示す。各手法の詳細は付録 A に示す。特徴量算出元画像サイズの候補は、1920×1080[pixels]、1024×576[pixels]、512×288[pixels]、256×144[pixels]、128×72[pixels]、64×36[pixels]、32×16[pixels]、16×9[pixels] の 8 サイズとし、画像の縦横比は画像を縮小する前のオリジナルの縦横比である 16:9 で固定した。

また、画像検索手法と特徴量算出元画像サイズの組み合わせの最適化に要する時間を削減するために、画像の解像度が大きい場合は処理に長い時間を要することが予め分かっているなど、最適化で明らかに選択されないと予想される組み合わせは、事前に除外した。1 段目から 3 段目までの絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補を、それぞれ表 4.6〜表 4.8 に示す。表中でチェックマークが記された画像検索手法と特徴量算出元画像サイズの組み合わせを、各段の絞り込みに用いる組み合わせの候補とした。Glocker らの報告によると、Randomized-Fern は画像サイズを 40×30[pixels] とした場合に最も性能が高かった [15]。そこで、Randomized-Fern のみ画像サイズを 40×30[pixels] とした組み合わせを全ての段の候補に加えた。

表 4.3: 実装した画像検索手法（Part1）

AKAZE-Matching	Feature Descriptor	AKAZE Decriptors[25]
	Definition of Disimilarity	1 - Matching Ratio
CCV	Feature Descriptor	CCV(Color Coherence Vector[26])
	Definition of Disimilarity	L2 Distance between CCVs
Gray-Histogram	Feature Descriptor	Histogram of Grayscale Image
	Definition of Disimilarity	L2 Distance between Histograms
Gray-L2	Feature Descriptor	-
	Definition of Disimilarity	L2 Distance between Grayscale Images
Gray-NCC	Feature Descriptor	-
	Definition of Disimilarity	1 - NCC(Normalized Cross-Correlation) between Grayscale Images
HOG	Feature Descriptor	HOG(Histogram of Oriented Gradient[27])
	Definition of Disimilarity	L2 Distance between HOGs
LBP	Feature Descriptor	Histogram of LBP(Local Binary Pattern[28]) Image
	Definition of Disimilarity	L2 Distance between Histograms

表 4.4: 実装した画像検索手法（Part2）

LSD-Angle	Feature Descriptor	Histogram of Angles between Lines
	Definition of Disimilarity	L2 Distance between Histograms
LSD-2Lines	Feature Descriptor	Histogram of Angle-Distance Pairs between Lines
	Definition of Disimilarity	L2 Distance between Histograms
LSD-Direction	Feature Descriptor	Number of Vartical Lines and Number of Horizontal Lines
	Definition of Disimilarity	L2 Distance between Numbers
LSD-Distance	Feature Descriptor	Histogram of Distances between Lines
	Definition of Disimilarity	L2 Distance between Histograms
LSD-Length	Feature Descriptor	Histogram of Line Lengths
	Definition of Disimilarity	L2 Distance between Histograms
LSD-Number	Feature Descriptor	Number of Detected Lines
	Definition of Disimilarity	Difference between Numbers
LSD-Region	Feature Descriptor	Numbers of Detected Lines in each Devided Image Region
	Definition of Disimilarity	L2 Distance between Numbers
LSD-1Line	Feature Descriptor	Histograms of Horizontal/Vertical Lines' Lengths in each Devided Image Region
	Definition of Disimilarity	L2 Distance between Histograms

表 4.5: 実装した画像検索手法（Part3）

Randomized-Fern	Feature Descriptor	A set of Ferns[15]
	Definition of Disimilarity	Block-wise Hamming Distance[15]
RGB-Histogram	Feature Descriptor	Histogram of RGB Image
	Definition of Disimilarity	L2 Distance between Histograms
RGB-L2	Feature Descriptor	-
	Definition of Disimilarity	L2 Distance between RGB Images
RGB-NCC	Feature Descriptor	-
	Definition of Disimilarity	1 - NCC between RGB Images
SURF-BoK	Feature Descriptor	Histogram Made via BoK(Bag of Keypoints[29]) Method with SURF[30]
	Definition of Disimilarity	L2 Distance between Histograms
SURF-Matching	Feature Descriptor	SURF Decriptors[30]
	Definition of Disimilarity	1 - Matching Ratio

表 4.6: 通過画像決定方法の評価で 1 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補

		Size of Source Image to Calculate Feature Descriptors							
		1920×1080	1024×576	512×288	256×144	128×72	64×36	32×18	16×9
Image Retrieval Method	AKAZE-Matching								
	CCV					✓	✓		
	Gray-Histogram	✓	✓	✓	✓	✓			
	Gray-L2								
	Gray-NCC								
	HOG								
	LBP						✓		
	LSD-Angle								
	LSD-2Lines								
	LSD-Direction			✓					
	LSD-Distance								
	LSD-Length								
	LSD-Number								
	LSD-Region			✓					
	LSD-1Line		✓	✓					
	Randomized-Fern	✓	✓	✓	✓	✓	✓		
	RGB-Histogram	✓	✓	✓	✓	✓			
	RGB-L2								
	RGB-NCC								
	SURF-BoK			✓	✓	✓			
	SURF-Matching								

表 4.7: 通過画像決定方法の評価で 2 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補

		Size of Source Image to Calculate Feature Descriptors							
		1920×1080	1024×576	512×288	256×144	128×72	64×36	32×18	16×9
Image Retrieval Method	AKAZE-Matching								
	CCV	✓	✓						
	Gray-Histogram	✓	✓	✓	✓	✓			
	Gray-L2				✓	✓	✓	✓	✓
	Gray-NCC						✓	✓	✓
	HOG				✓	✓			
	LBP				✓	✓	✓		
	LSD-Angle		✓						
	LSD-2Lines		✓						
	LSD-Direction		✓	✓					
	LSD-Distance		✓						
	LSD-Length		✓						
	LSD-Number		✓						
	LSD-Region	✓	✓	✓					
	LSD-1Line		✓	✓	✓				
	Randomized-Fern	✓	✓	✓	✓	✓	✓		
	RGB-Histogram	✓	✓	✓	✓	✓			
	RGB-L2				✓	✓	✓	✓	✓
	RGB-NCC							✓	✓
	SURF-BoK		✓	✓	✓				
	SURF-Matching								

表 4.8: 通過画像決定方法の評価で 3 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補

		Size of Source Image to Calculate Feature Descriptors							
		1920×1080	1024×576	512×288	256×144	128×72	64×36	32×18	16×9
Image Retrieval Method	AKAZE-Matching		✓	✓	✓	✓			
	CCV								
	Gray-Histogram								
	Gray-L2		✓	✓					
	Gray-NCC				✓	✓			
	HOG								
	LBP								
	LSD-Angle								
	LSD-2Lines								
	LSD-Direction								
	LSD-Distance								
	LSD-Length								
	LSD-Number								
	LSD-Region								
	LSD-1Line								
	Randomized-Fern	✓	✓	✓	✓	✓	✓		
	RGB-Histogram								
	RGB-L2			✓					
	RGB-NCC					✓			
	SURF-BoK								
	SURF-Matching			✓	✓	✓			

4.2.4　最適化パートの実行と評価に用いたPC

最適化パートの実行と通過画像決定方法の評価に使用したPCの仕様を表4.9に示す。高速化のため、525GBのSSDを仮想メモリに割り当てた。

表 4.9: 最適化パートの実行と通過画像決定方法の評価に使用したPCの仕様

CPU	Intel(R) Core(TM) i7-3770 3.40 GHz (4 Cores, 8 Threads)
RAM	32.0 GB
OS	Windows 10 Pro (64bit)
Virtual Memory	CT525MX300SSD1(525 GB)

4.2.5　最適化の結果

画像検索手法と特徴量算出元画像サイズの組み合わせの最適化を行った結果（平均処理時間と再現率）を付録Bの表B.1～表B.10に示す。また、1段目、1,2段目の最適化で選ばれた組み合わせの候補と、3段目までの最適化で選出された最適な組み合わせを表4.10に示す。1段目の最適化では、SURF（画像サイズ512×288）、LSD-1Line（画像サイズ512×288）、LBP（64×36）が上位3つの候補に選ばれた。ふげんの純水装置室内には配管などが多数存在するため、線分を利用した画像検索手法が上位に入ったと考えられる。1,2段目の最適化では、1段目の候補に残ったそれぞれの組み合わせに、2段目の絞り込み処理としてSURF（画像サイズ512×288）を追加したものが、上位3つの候補に選ばれた。3段目までの最適化では、1段目にSURF（画像サイズ512×288）、2段目にも同じくSURF（画像サイズ512×288）、3段目にAKAZE-Matching（画像サイズ512×288）を用いる組み合わせが最適な組み合わせとして選出された。

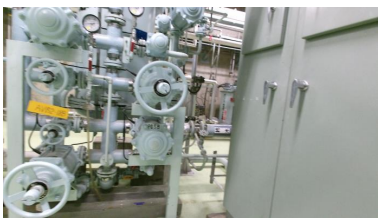
表 4.10: 最適化で選出された画像検索手法と特徴量算出元画像サイズの組み合わせ

	1st Filter		2nd Filter		3rd Filter	
	Method	Image Size	Method	Image Size	Method	Image Size
Candates for Optimum Combination (1st Filter)	SURF-BoK	512×288				
	LSD-1Line	512×288				
	LBP	64×36				
Candates for Optimum Combination (1st and 2nd Filters)	SURF-BoK	512×288	SURF-BoK	512×288		
	LSD-1Line	512×288	SURF-BoK	512×288		
	LBP	64×36	SURF-BoK	512×288		
Optimum Combination	SURF-BoK	512×288	SURF-BoK	512×288	AKAZE-Matching	512×288

4.2.6 評価指標

類似画像検索の速度は、クエリ画像が与えられてから絞り込み処理が終了するまでの処理時間の平均値で評価する。すなわち、テスト用データセットの全クエリ画像に対する処理時間の合計をデータセットに含まれるクエリ画像の数で除した平均処理時間（Avg. Processing Time）を、速度の評価指標とする。

類似画像検索の精度は、テスト用データセットの全クエリ画像に対して類似画像検索を実行したときの成功率（Success Rate）を評価指標とする。1 枚のクエリ画像に対して類似画像検索を実行した際、選出された画像（部分ソートフィルタリング、全体ソートによる通過画像決定方法では 10 枚、閾値による通過画像決定方法ではクエリ画像ごとに異なる枚数）中に 1 枚以上の正解画像が含まれる場合、類似画像検索に「成功」したとみなす。このとき、4.2.3.3 条で述べた疑似クエリ画像に対する正解画像と同様に、DB に含まれる画像の内、クエリ画像との正規化相互相関が 0.90 以上の画像を正解画像とする。しかし、ふげんの純水装置室内には似た外見を持つ機器が多く存在し、図 4.5 に示すように、クエリ画像との正規化相互相関が 0.90 以上であっても、クエリ画像とは異なる対象を写した画像が一部存在する。そこで、正規化相互相関で正解と判定された画像を全て目視で確認し、クエリ画像と異なる対象を写した画像は正解から除外した。

NCC*	Query Image	Selected Image
0.9076		
0.9122		
0.9072		
0.9108		
0.9089		
0.9305		

*Normalized Cross Correlation

図 4.5: 異なる対象を写しているが正規化相互相関が 0.90 以上の画像の例

4.2.7　評価の結果

最適化された画像検索手法と特徴量算出元画像サイズの組み合わせで、テスト用データセットのクエリ画像に対して類似画像検索を実行した結果を、通過画像決定方法に部分ソートフィルタリングを用いた場合、閾値による方法を用いた場合、全体ソートによる方法を用いた場合のそれぞれの場合に分けて表 4.11 に示す。部分ソートフィルタリングを用いた場合に平均処理時間が最も短かった。成功率は全体ソートによる通過画像決定方法を用いた場合に最も高かった。

表 4.11: 部分ソート、全体ソート、閾値による通過画像決定方法の性能比較

Filtering Strategy	Success Rate	Avg. Processing Time
Partial Sort	0.675	**175 msec**
Sort	**0.690**	196 msec
Threshold	0.386	177 msec

4.2.8　考察

表 4.11 に示したように、部分ソートフィルタリングを用いた類似画像検索が最も高速であったが、成功率は全体ソートによる通過画像決定方法を用いた場合の成功率をわずかに下回った。しかし、全体ソートによる通過画像決定方法を用いる場合、類似画像は最後にまとめて選出されるため、一定の時間内に、選出された類似画像で逐次トラッキングへの復帰処理を試行できる回数が少なくなる。そこで、全体ソートによる通過画像決定方法を用いた類似画像検索で、最後に選出する類似画像の枚数を、キーフレームベースのリローカリゼーションを実装する場合の標準的な選出枚数である 5 枚に減らした場合の成功率を求めた。このときの類似画像検索性能の比較結果を表 4.12 に示す。表 4.12 を見ると、逐次トラッキングへの復帰処理を試行できる回数を考慮した評価では、部分ソートフィルタリングの性能が全体ソートによる通過画像決定方法を用いた場合の性能を上回る結果となったことが分かる。

閾値による通過画像決定方法を用いた場合の類似画像検索の成功率は、他の通過画像決定方法を用いた場合の成功率を大きく下回った。この理由として、最後に選出された類似画像の枚数のばらつきが考えられる。そこで、この原因を考察するために、各クエリ画像（計 2,329 枚）に対してそれぞれ選出された類似画像の枚数を調べた。また、

選出された類似画像の枚数や 1,2 段目の絞り込みを通過する画像の枚数のばらつきは、処理時間にも影響すると考えられるため、各クエリ画像に対する類似画像検索の処理時間も調べた。閾値による通過画像決定方法を用いた場合に、各クエリ画像に対してそれぞれ選出された類似画像の枚数と処理時間の関係を図 4.6 に、部分ソートフィルタリングを用いた場合に、各クエリ画像に対してそれぞれ選出された類似画像の枚数と処理時間の関係を図 4.7 に示す。図 4.6 および図 4.7 を見ると、部分ソートフィルタリングでは原理上常に 10 枚の類似画像が選出される一方で、閾値による通過画像決定方法を用いた場合には、選出された類似画像の枚数がばらついていることが分かる。また、閾値による通過画像決定方法を用いた場合で、類似画像検索失敗時に選出された類似画像の枚数と処理時間の関係を図 4.8 に示す。類似画像が 1 枚も選出されなかったことが原因で類似画像検索に失敗したケースも多いことが分かる。さらに、閾値による通過画像決定方法を用いた場合の平均処理時間は、部分ソートフィルタリングと同程度であったが、試行ごとの処理時間には大きなばらつきが存在し、特に時間を要した試行ではアプリケーションの要求（300msec 以内）を満たせていないこともあった。

以上から、多段階絞り込み処理に用いる通過画像決定方法として、提案する部分ソートフィルタリングが適切であることが確認できた。以降の評価では、通過画像決定方法として部分ソートフィルタリングを用いる。

表 4.12: 全体ソートによる通過画像決定方法で選出する類似画像を 5 枚とした場合の類似画像検索性能の比較

Filtering Strategy	Success Rate	Avg. Processing Time
Partial Sort	**0.675**	**175 msec**
Sort	0.619	196 msec
Threshold	0.386	177 msec

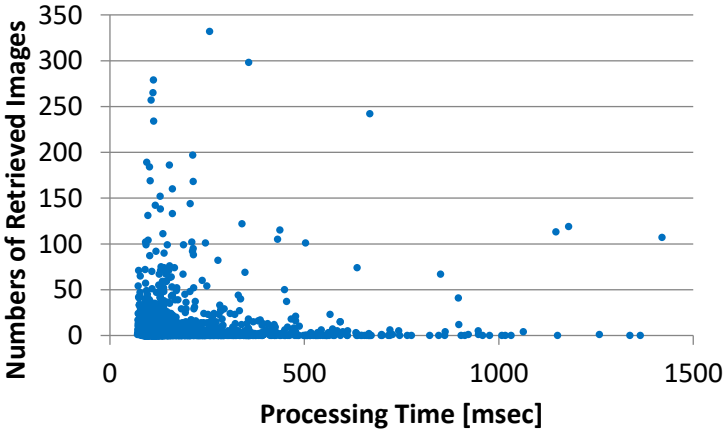


図 4.6: 選出された類似画像の枚数と処理時間の関係（閾値による通過画像決定方法を用いた場合）

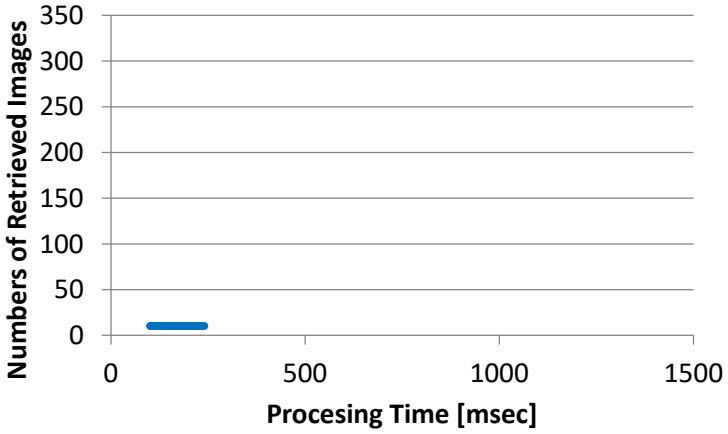


図 4.7: 選出された類似画像の枚数と処理時間の関係（部分ソートフィルタリングを用いた場合）

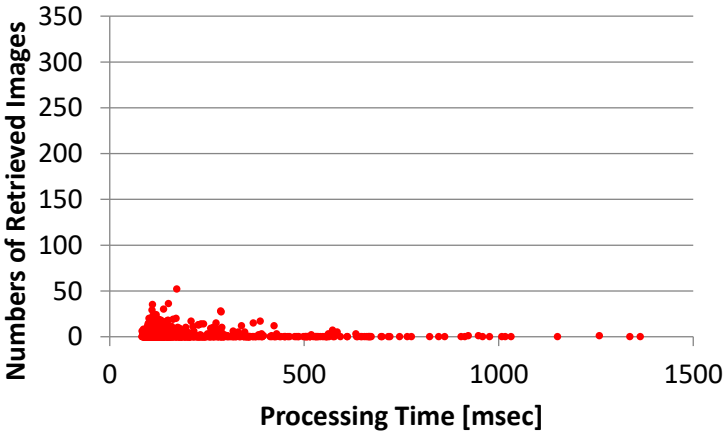


図 4.8: 類似画像検索失敗時に選出された類似画像の枚数と処理時間の関係（閾値による通過画像決定方法を用いた場合）

4.3 複数の画像検索手法を組み合わせた際の類似画像検索性能の評価

本節では、複数の画像検索手法を組み合わせた際に類似画像検索性能が向上することを確認するために行った、2 つ目の評価実験について述べる。

4.3.1 評価の方法

複数の画像検索手法を組み合わせた際に、類似画像検索性能が向上するかを評価するため、最適化された組み合わせを用いた場合の類似画像検索性能（4.2 節で求められた結果）と、比較対象として、新たに 7 つの画像検索手法を単独に用いた 7 つの場合の類似画像検索性能を比較した。評価に用いたデータセット、多段階絞り処理込みフレームワークの実装、最適化パートの実行と評価に使用した PC、および評価指標は、4.2 節で述べた通過画像決定方法の評価時と同じである。そのため、最適化パートは再実行せず、4.2 節での最適化結果を用いた。比較対象とした 7 つの画像検索手法とそれぞれの手法で用いた特徴量算出元画像サイズを表 4.13 に示す。1 段目、2 段目、3 段目それぞれの絞り込み処理に用いる画像検索手法の候補から、最適化の際に評価が高かった手法を選択した。それぞれの画像検索手法で用いる特徴量算出元画像サイズは、最適化の際の評価が最も高かったサイズを選択した。1 つの画像検索手法を単独に用いる場合には、絞り込み段数を 1 段、部分ソートフィルタリングのパラメータを N=3295, M=1 とし、最適化された組み合わせを用いた場合と同様に、10 枚の類似画像が選出されるようにした。

表 4.13: 比較対象とした 7 つの組み合わせ

Method	Size
SURF	512×288
LSD-1Line	512×288
LBP	64×36
Gray-Histogram	1920×1080
RGB-Histogram	256×144
AKAZE-Matching	512×288
SURF-Matching	256×144

4.3.2　評価の結果

最適化された組み合わせを用いた場合と 1 つの画像検索手法を単独に用いた場合の類似画像検索性能の評価結果を表 4.14 に示す。GRAY-Histogram（1920×1080）を単独に用いた場合に最も成功率が高く、また、平均処理速度も最も早かった。

4.3.3　考察

表 4.14 に示した結果から、最適化された組み合わせを用いた場合の類似画像検索性能が、他の手法を単独に用いた場合よりも低下してしまっていることがわかった。この原因として、最適化パートでの性能評価が正しく行えていなかったことが考えられる。最適化パートでは、疑似クエリ画像に対する正解画像を正規化相互相関の値で決定した。しかし、図 4.5 に示したように、正規化相互相関による判定では、外見は似ているが異なる対象を写した画像が正解画像として判定されることがあるため、評価では目視による最終チェックを行った。このとき、正規化相互相関による判定では成功と判定されたが目視による最終チェックで失敗と判定されたケースが、特に、AKAZE や SURF などの局所特徴点を利用した画像検索手法を用いた場合に多かった。実際に、目視による最終チェックを行う前の、正規化相互相関による評価結果では、AKAZE や SURF を利用した画像検索手法の成功率が高く、平均処理時間が目標の 300msec 以内のものの中では、最適化された組み合わせを用いた場合の成功率が最も高かった。つまり、最適化パートでの正解画像の判定に誤りがあったために、最適化時の評価が正しく行えず、本来選択すべき高性能な画像検索手法を選択できなかったと考えられる。

一方で、提案するフレームワークを用いて SURF と AKAZE-Matching を組み合わせた場合には、SURF や AKAZE-Matching をそれぞれ単独に用いた場合よりも類似画像検索性能が向上した。このことから、最適化パートで正しく性能評価が行えるようになれば、GRAY-Histogram などの単独で高性能であった手法が最適な組み合わせに含まれるようになり、GRAY-Histogram などの手法を単独に用いた場合の性能を上回る類似画像検索が実現できると考えられる。この点については、撮影時のカメラ姿勢が既知の画像で DB を作成し、疑似クエリ画像に対する正解画像の決定がより正しく行える状態で評価を行うなどして、さらに検討していく必要がある。

表 4.14: 複数の画像検索手法を最適な組み合わせで用いた場合と 1 つの画像検索手法のみを用いた場合の類似画像検索性能の比較

		Method	Size	Success Rate	Avg. Processing Time
Single Method		SURF	512×288	0.590	51 msec
		LSD-1Line	512×288	0.584	29 msec
		LBP	64×36	0.585	29 msec
		Gray-Histogram	1920×1080	**0.775**	**16 msec**
		RGB-Histogram	256×144	0.769	18 msec
		AKAZE-Matching	512×288	0.521	16,827 msec
		SURF-Matching	256×144	0.699	29,144 msec
		Gray-L2	1024×576	0.149	2,149 msec
Optimum Combination	1st Filter	SURF	512×288	0.675	175 msec
	2nd Filter	SURF	512×288		
	3rd Filter	AKAZE-Matching	512×288		

4.4　組み合わせ最適化アルゴリズムの性能評価

本節では、画像検索手法と特徴量算出元画像サイズの組み合わせ最適化アルゴリズムの性能を評価するために行った、3 つ目の評価実験について述べる。

4.4.1　評価の方法

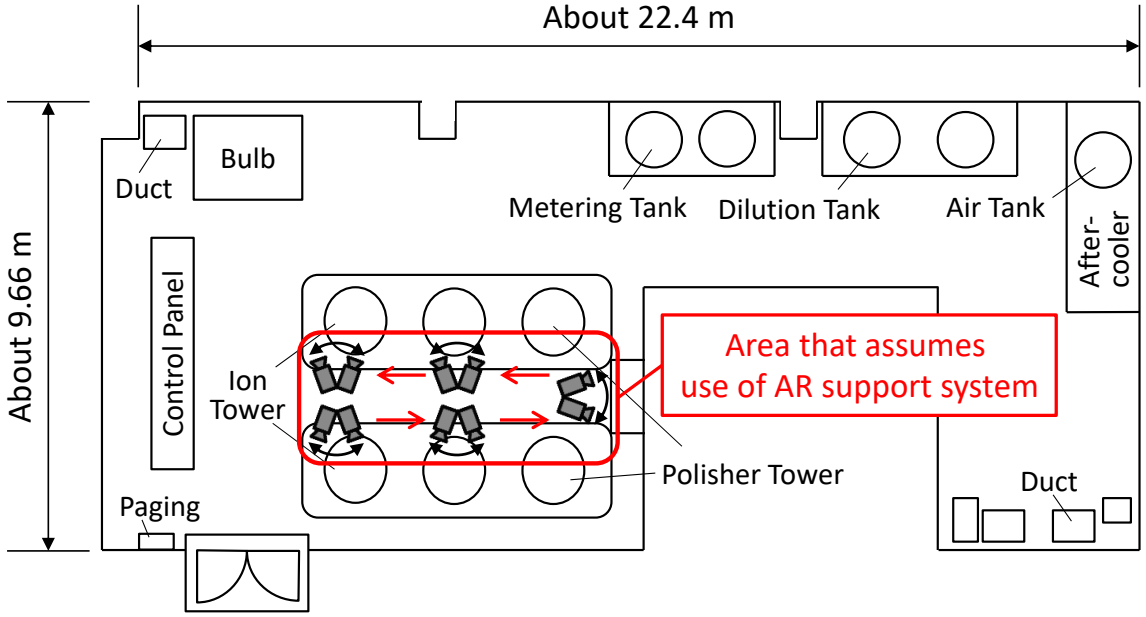
組み合わせ最適化アルゴリズムの評価では、最適化によって選択される画像検索手法と特徴量算出元画像サイズの組み合わせの、環境ごとの違いや共通点を考察するために、ふげんデータセットと 7-Scenes データセット [31,32] の 2 つのデータセットを用いた。それぞれのデータセットの詳細は 4.4.2 項で述べる。それぞれのデータセットでの評価におけるフレームワークの実装の詳細は 4.4.3 項で述べる。

まず、それぞれのデータセットで画像検索手法と特徴量算出元画像サイズの組み合わせを最適化した。次に、最適化アルゴリズムで適切な組み合わせを選択できているかを確認するために、最も評価が高かった組み合わせの類似画像検索性能を、2 番目、3 番目に評価が高かった組み合わせ（3 段全ての画像検索手法が最適な組み合わせと同じである組み合わせは除く）の類似画像検索性能と比較した。

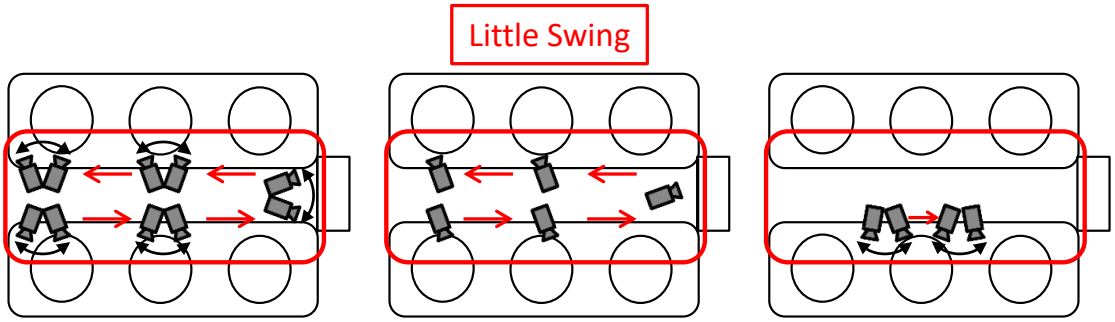
4.4.2　評価に用いたデータセット

4.4.2.1　原子炉廃止措置研究開発センター（ふげん）データセット

本データセットでは、様々な利用シーンを想定した評価を行うために、複数のテスト用データセットを用意した。本データセットは、4.2 節で用いたデータセットに 5 つのテスト用データセットを加えたデータセットで、1 つの DB 用データセットと、6 つのテスト用データセット（テスト用データセット 1～テスト用データセット 6）で構成される。新たに追加した 5 つのテスト用データセットの画像も、ふげんの純水装置室内で撮影した。純水装置室の見取り図と各データセットの撮影条件を図 4.9 に示す。AR を用いた作業支援システムの利用を想定する領域、撮影に使用したカメラは、4.2 節と同じである。追加した 5 つのテスト用データセットの画像も、それぞれ別に撮影した動画から抜き出した画像で構成され、画像には抜き出した順番に連続する番号の ID が与えられている。各データセットの作成に用いた動画の長さと、動画から抜き出した画像の枚数を表 4.15 に、それぞれのデータセットに含まれる画像の例を図 4.10 に示す。



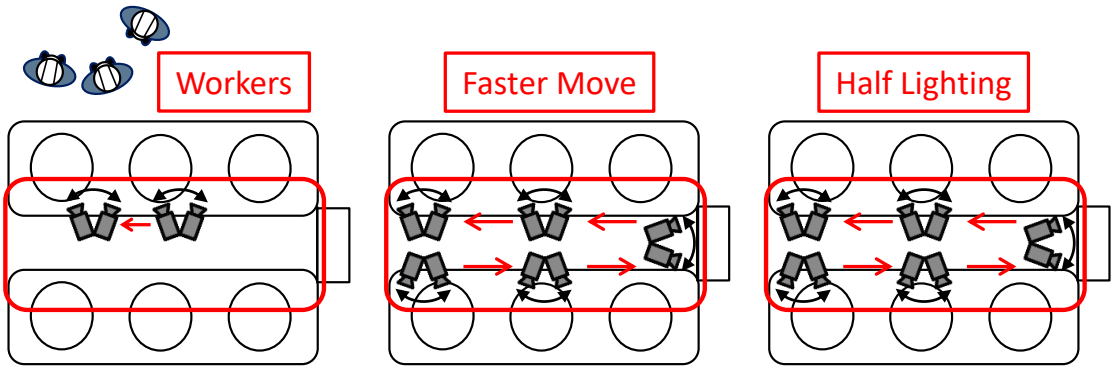
(a) Plane View of Demineralizer Room and Camera Track of DB Dataset



(b) Camera Track of Test Dataset 1

(c) Camera Track of Test Dataset 2

(d) Camera Track of Test Dataset 3



(e) Camera Track of Test Dataset 4

(f) Camera Track of Test Dataset 5

(g) Camera Track of Test Dataset 6

図 4.9: 純水装置室の見取り図と各データセットの撮影条件

(a) Test Dataset 1

(b) Test Dataset 2

(c) Test Dataset 3

(d) Test Dataset 4

(e) Test Dataset 5

(f) Test Dataset 6

図 4.10: ふげんデータセットの各テスト用データセットに含まれる画像の例

表 4.15: ふげんデータセットの各データセットの動画の長さと画像枚数

Dataset	Lengh of Movie	Numbers of Extracted Images
DB Dataset	60 minutes	32,941
Test Dataset 1	3 minutes	2,329
Test Dataset 2	4 minutes	2,363
Test Dataset 3	3.5 minutes	2,346
Test Dataset 4	2 minutes	2,150
Test Dataset 5	1.5 minutes	1,322
Test Dataset 6	2 minutes	1,971

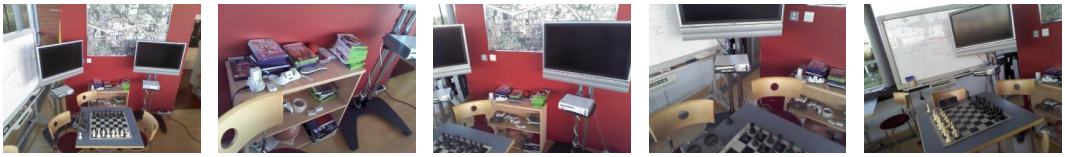
テスト用データセット 1 は、4.2 節で用いたテスト用データセットで、作業支援システムの利用を想定した領域全体を撮影して作成されたものである。テスト用データセット 2 は、テスト用データセット 1 と同様に、作業支援システムの利用を想定する領域全体を撮影しているが、領域内の機器を順番に点検していく場合などを想定し（すでに確認した箇所を再度確認することは少ないと想定し)、カメラ姿勢の左右の変化を少なくして撮影を行った。テスト用データセット 3 は、特定の機器を重点的に点検する場合などを想定し、様々な方向から同じ機器を撮影した。テスト用データセット 4 も、テスト用データセット 3 と同様に、特定の機器を重点的に点検する場合などを想定して、様々な方向から同じ機器を撮影しているが、その際に別の作業を行う作業員が同室に居合わせることを想定し、機器の背後に作業員を模した人が写りこむように撮影した。テスト用データセット 5 は、クエリ画像が手振れなどを多く含む場合の性能を評価するために、比較的速い速度でカメラを動かして撮影した。テスト用データセット 6 は、室内の照明環境が変化した場合の性能を評価するために、純粋装置室内の照明を半分程度に落とした状態で撮影を行った。

4.4.2.2 7-Scenes データセット

原子力発電プラントのような特殊な環境での組み合わせ最適化の結果を、比較的一般的な環境での最適化結果と比較する為に、7-Scenes データセット [31,32] を評価に用いる。7-Scenes データセットは、Microsoft Research が公開している、トラッキングやリローカリゼーションなどの性能評価用のデータセットである。データセットの内容を表 4.16 に示す。データセットは 7 つのシーンに分かれており、それぞれのシーンは複数のシーケンスで構成される。各シーケンスには RGB 画像、デプス画像、及び撮影時のカメラ姿勢の画像セットが 1000 組 (Stairs のみ 500 組) 存在し、それぞれ 0〜999 まで（Stairs のみ 0〜499 まで）の ID が割り振られている。画像は Microsoft 社の Xbox 360 Kinect センサーで撮影されており、カメラ姿勢は KinectFusion[33] を使用して取得されている。本研究では RGB 画像とカメラ姿勢のみを利用し、デプス画像は利用しない。RGB 画像の解像度は 640×480[pixels]、色深度は 24bit（8bit×3colors）である。カメラ姿勢はシーン毎の座標系での回転行列と並進ベクトルで表されており、同一のシーン内では異なるシーケンスでも座標系は同じである。各シーンに含まれる画像の例を図 4.11 に示す。本研究では、表 4.17 に示すように、データセットを DB 用データセットとテスト用データセットに分割し、それぞれのデータセット内で 0 から始まる連続した ID を画像に再度割り振った。

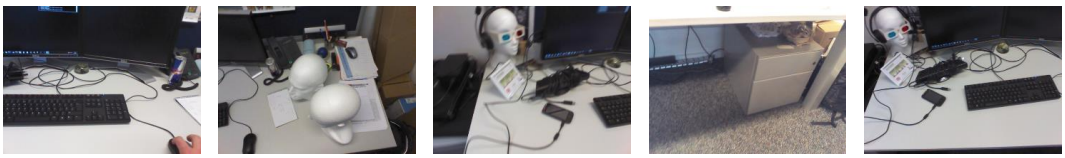
表 4.16: 7-Scenes データセットの内容

Scene	Sequence	Numbers of Image Set
Chess	seq-01 〜 seq-06	6,000
Fire	seq-01 〜 seq-04	4,000
Heads	seq-01 〜 seq-02	2,000
Office	seq-01 〜 seq-10	10,000
Pumpkin	seq-01 〜 seq-08	8,000
RedKitchen	seq-01 〜 seq-14	14,000
Stairs	seq-01 〜 seq-06	3,000
Total	-	**47,000**

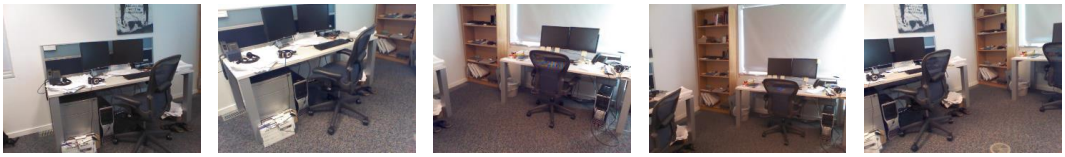


(a) Chess

(b) Fire

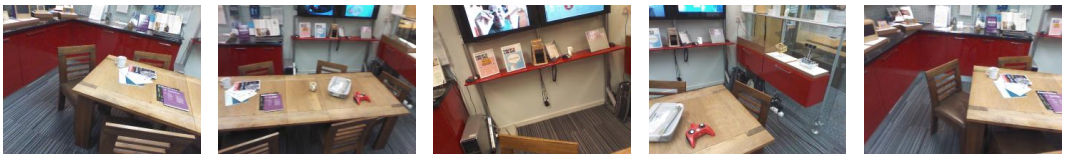


(c) Heads

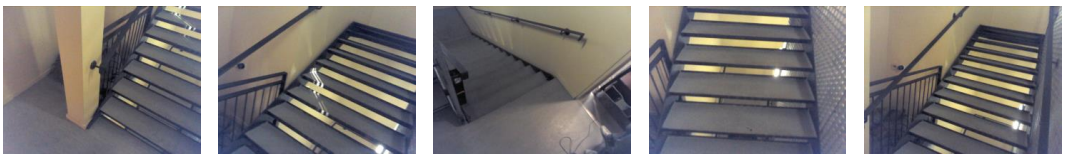


(d) Office

(e) Pumpkin



(f) RedKitchen



(g) Stairs

図 4.11: 7-Scenes データセット [31,32] の各シーンに含まれる画像の例

表 4.17: 7-Scenes データセットの分割

Dataset	Scene	Sequence	Reallocated IDs
DB Dataset	Chess	seq-01, 02, 04, 06	0 - 3,999
	Fire	seq-01, 02	4,000 - 5,999
	Heads	seq-02	6,000 - 6,999
	Office	seq-01, 03, 04, 05, 08, 10	7,000 - 12,999
	Pumpkin	seq-02, 03, 06, 08	13,000 - 1,6999
	RedKitchen	seq-01, 02, 05, 07, 08, 11, 13	17,000 - 2,3999
	Stairs	seq-02, 03, 05, 06	24,000 - 25,999
Test Dataset	Chess	seq-03, 05	0 - 1,999
	Fire	seq-03, 04	2,000 - 3,999
	Heads	seq-01	4,000 - 4,999
	Office	seq-02, 06, 07, 09	5,000 - 8,999
	Pumpkin	seq-01, 07	9,000 - 10,999
	RedKitchen	seq-03, 04, 06, 12, 14	11,000 - 15,999
	Stairs	seq-01, 04	16,000 - 16,999

4.4.3 多段階絞り込み処理フレームワークの実装

4.4.3.1 実装に用いたハードウェアおよびソフトウェア

フレームワークの実装には、4.2 節と同じハードウェアおよびソフトウェアを用いた。

4.4.3.2 実行パートの実装

4.2 節と同様に、段階的な絞り込み処理の段数は 3 段とし、部分ソートフィルタリングのパラメータ M,N は、3 段階の絞り込み全てで共通とした。ふげんデータセットを用いた評価における部分ソートフィルタリングのパラメータ M,N は、4.2 節と同様の N=15, M=1 とした。7-Scenes データセットを用いた評価では、DB の画像数が 2,600 枚であるため、N=14, M=1 とし、類似画像の候補を 26,000 枚→ 1,852 枚（26,000÷14+1）→ 133 枚（1,852÷14+1）→ 10 枚（133÷14+1）と絞り込んだ。

4.4.3.3　最適化パートの実装

最適化パートは、4.2 節と同様に実装した。

4.4.3.4　最適化に用いた疑似クエリ画像と正解画像の決定方法

ふげんデータセットを用いた評価では、4.2 節と同様に、DB に含まれる画像全体の 5%の画像を疑似クエリ画像とし、疑似クエリ画像との正規化相互相関が 0.90 以上の画像を正解画像とした。

7-Scenes データセットを用いた評価でも、同様に DB に含まれる画像全体の 5%の画像を疑似クエリ画像としたが、7-Scenes データセットに含まれる画像は撮影時のカメラ姿勢が既知であるため、カメラ姿勢を基に正解画像を決定した。2 つのカメラ姿勢 $[R_a|T_a]$ 及び $[R_b|T_b]$ が与えられたとき、並進誤差は、

$$\text{並進誤差} = ||T_a - T_b|| \tag{4.2}$$

で表される。また、回転誤差は、

$$\text{回転誤差} = \cos^{-1}\left(\frac{Trace(R_a R_b^{\intercal}) - 1}{2}\right) \tag{4.3}$$

で計算できる。ここで R_a, R_b はそれぞれのカメラ姿勢の回転行列を、T_a, T_b は並進ベクトルを表し、$Trace(A)$ は行列 A の対角成分の和を表す。7-Scenes データセットでは、疑似クエリ画像と、DB 中の疑似クエリ画像と同一シーンの全ての画像の撮影時のカメラ姿勢を比較し、並進誤差が 20cm 以内、回転誤差が 20° 以内の画像を正解画像とした。

4.4.3.5　画像検索手法と特徴量算出元画像サイズの候補

ふげんデータセットを用いた評価での画像検索手法と特徴量算出元画像サイズの候補は、4.2 節と同じである。

7-Scenes データセットを用いた評価でも、ふげんデータセットと同じ画像検索手法を利用した。特徴量算出元画像サイズの候補は、640×480[pixels]、320×240[pixels]、160×120[pixels]、80×60[pixels]、40×30[pixels]、20×15[pixels] の 6 サイズとした。4.2 節と同様に、画像検索手法と特徴量算出元画像サイズの組み合わせの最適化に要する時間を削減するために、最適化で選択されないと予想される組み合わせは、事前に除外した。7-Scenes データセットで、1 段目から 3 段目までの絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補を、それぞれ表 4.18〜表 4.20 に示す。

表 4.18: 7-Scenes データセットで 1 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補

		Size of Source Image to Calculate Feature Descriptors					
		640×480	320×240	160×120	80×60	40×30	20×15
Image Retrieval Method	AKAZE-Matching						
	CCV			✓	✓	✓	
	Gray-Histogram	✓	✓	✓	✓		
	Gray-L2						
	Gray-NCC						
	HOG						
	LBP				✓	✓	
	LSD-Angle						
	LSD-2Lines						
	LSD-Direction	✓	✓				
	LSD-Distance						
	LSD-Length						
	LSD-Number						
	LSD-Region	✓	✓				
	LSD-1Line	✓	✓				
	Randomized-Fern	✓	✓	✓	✓	✓	
	RGB-Histogram	✓	✓	✓	✓		
	RGB-L2						
	RGB-NCC						
	SURF-BoK	✓	✓	✓	✓		
	SURF-Matching						

表 4.19: 7-Scenes データセットで 2 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補

		Size of Source Image to Calculate Feature Descriptors					
		640×480	320×240	160×120	80×60	40×30	20×15
Image Retrieval Method	AKAZE-Matching						
	CCV	✓					
	Gray-Histogram	✓	✓	✓	✓		
	Gray-L2		✓	✓	✓	✓	✓
	Gray-NCC					✓	✓
	HOG		✓	✓			
	LBP		✓	✓	✓	✓	
	LSD-Angle	✓					
	LSD-2Lines	✓					
	LSD-Direction	✓	✓				
	LSD-Distance	✓					
	LSD-Length	✓					
	LSD-Number	✓					
	LSD-Region	✓	✓				
	LSD-1Line	✓	✓	✓			
	Randomized-Fern	✓	✓	✓	✓	✓	
	RGB-Histogram	✓	✓	✓	✓		
	RGB-L2		✓	✓	✓	✓	✓
	RGB-NCC						✓
	SURF-BoK	✓	✓	✓			
	SURF-Matching						

表 4.20: 7-Scenes データセットで 3 段目の絞り込み処理に用いる画像検索手法と特徴量算出元画像サイズの組み合わせの候補

		Size of Source Image to Calculate Feature Descriptors					
		640×480	320×240	160×120	80×60	40×30	20×15
Image Retrieval Method	AKAZE-Matching	✓	✓	✓	✓		
	CCV						
	Gray-Histogram						
	Gray-L2	✓	✓				
	Gray-NCC			✓	✓		
	HOG						
	LBP						
	LSD-Angle						
	LSD-2Lines						
	LSD-Direction						
	LSD-Distance						
	LSD-Length						
	LSD-Number						
	LSD-Region						
	LSD-1Line						
	Randomized-Fern	✓	✓	✓	✓	✓	
	RGB-Histogram						
	RGB-L2		✓				
	RGB-NCC				✓		
	SURF-BoK						
	SURF-Matching	✓	✓	✓	✓		

4.4.4 最適化パートの実行と評価に用いたPC

最適化パートの実行と最適化アルゴリズムの性能評価に使用したPCの仕様を表4.21に示す。最適化パートの実行と様々な評価を短時間で行うために、ふげんデータセットと7-Scenesデータセットでそれぞれ別のPCを用いた。2台のPCの処理性能は同程度である。

表 4.21: 最適化パートの実行と評価に使用したPCの仕様

Evaluation Experiment with Fugen Dataset	CPU	Intel(R) Core(TM) i7-3770 3.40 GHz (4 Cores, 8 Threads)
	RAM	32.0 GB
	OS	Windows 10 Pro (64bit)
	Virtual Memory	CT525MX300SSD1(525 GB)
Evaluation Experimant with 7-Scenes Dataset	CPU	Intel(R) Core(TM) i7-3770 3.40 GHz (4 Cores, 8 Threads)
	RAM	16.0 GB
	OS	Windows 10 Pro (64bit)
	Virtual Memory	TS240GSSD220S(240 GB)×2 RAID 0

4.4.5 最適化の結果

ふげんデータセットを用いた評価時の、DB データセット、多段階絞り処理込みフレームワークの実装、最適化パートの実行と評価に使用した PC は、4.2 節で述べた通過画像決定方法の評価時と同じである。そのため、最適化パートは再実行せず、4.2 節での最適化結果を用いた。画像検索手法と特徴量算出元画像サイズの組み合わせの最適化結果の上位 3 位までを表 4.22 に示す。全体として SURF[30] や AKAZE[25] を用いた手法が多く選ばれた中、最適化結果が 3 位の組み合わせは、1 段目に線分特徴を用いた手法が選ばれていることが特徴的である。

次に、7-Scenes データセットで、画像検索手法と特徴量算出元画像サイズの組み合わせの最適化を行った際の各組合せの評価の結果を付録 C の表 C.1〜表 C.10 に示す。また、1 段目、1,2 段目の最適化で選ばれた組み合わせの候補と、3 段目までの最適化結果が上位 3 位までの組み合わせを表 4.23 に示す。1 段目の最適化では、SURF（画像サイズ 160×120）、LBP（画像サイズ 80×60）、RGB-Histogram（640×480）が上位 3 つの候補に選ばれた。ふげんデータセットで評価が高かった LSD-1Line などの線分特徴を用いた手法は、7-Scenes データセットでは低い評価となった。1,2 段目の最適化では、SURF（画像サイズ 160×120）と LBP（画像サイズ 80×60）を組み合わせた 3 種類の組み合わせが、上位 3 つの候補に選ばれた。3 段目までの最適化では、1 段目に LBP（画像サイズ 80×60）、2 段目に SURF（画像サイズ 160×120）、3 段目に AKAZE-Matching（画像サイズ 640×480）を用いる組み合わせが最適な組み合わせとして選出された。7-Scenes データセットでは、上位 3 つの組み合わせ全てに LBP が含まれている。

表 4.22: 画像検索手法と特徴量算出元画像サイズの組み合わせの最適化結果（上位 3 位まで、ふげんデータセットを対象）

	1st Filter		2nd Filter		3rd Filter	
	Method	Image Size	Method	Image Size	Method	Image Size
Optimum Combination	SURF-BoK	512×288	SURF-BoK	512×288	AKAZE-Matching	512×288
Second Optimum Combination	SURF-BoK	512×288	SURF-BoK	512×288	SURF-Matching	256×144
Third Optimum Combination	LSD-1Line	512×288	SURF-BoK	512×288	AKAZE-Matching	512×288

表 4.23: 画像検索手法と特徴量算出元画像サイズの組み合わせの最適化結果（各段階上位 3 位まで、7-Scenes データセットを対象）

	1st Filter		2nd Filter		3rd Filter	
	Method	Image Size	Method	Image Size	Method	Image Size
Candates for Optimum Combination (1st Filter)	SURF-BoK	160×120				
	LBP	80×60				
	RGB-Histogram	640×480				
Candates for Optimum Combination (1st and 2nd Filters)	LBP	80×60	SURF-BoK	160×120		
	SURF-BoK	160×120	SURF-BoK	160×120		
	SURF-BoK	160×120	LBP	80×60		
Optimum Combination	LBP	80×60	SURF-BoK	160×120	AKAZE-Matching	640×480
Second Optimum Combination	LBP	80×60	SURF-BoK	160×120	SURF-Matching	640×480
Third Optimum Combination	SURF-BoK	160×120	LBP	80×60	SURF-Matching	320×240

4.4.6　評価に用いた指標

類似画像検索性能は、4.2 節と同様に、平均処理速度と成功率で評価した。このとき、7-Scenes データセットではカメラ姿勢を基に正解画像を決定した。4.4.3.4 条で疑似クエリ画像に対する正解画像を決定した際と同様に、クエリ画像と、DB 中のクエリ画像と同一シーンの全ての画像の撮影時のカメラ姿勢を比較し、並進誤差が 20cm 以内、回転誤差が 20° 以内の画像を正解画像とした。一部のクエリ画像は DB 内に正解画像が存在しなかったため、テスト用データセットから除外した。除外後のテスト用データセットに含まれる画像の枚数を表 4.24 に示す。

表 4.24: 7-Scenes のテスト用データセットの画像枚数

Scene	Before Exclusion	→	After Exclusion
Chess	4,000	→	**1,262**
Fire	2,000	→	**585**
Heads	1,000	→	**723**
Office	6,000	→	**1,851**
Pumpkin	4,000	→	**826**
RedKitchen	7,000	→	**2,127**
Stairs	2,000	→	**425**
Total	26,000	→	**7,799**

4.4.7　評価の結果

ふげんデータセットで、最適化結果が上位 3 位までに入った画像検索手法と特徴量算出元画像サイズの組み合わせの類似画像検索性能の評価結果を表 4.25 に示す。全ての評価で平均処理時間は 300msec 以内であった。テスト用データセット 4 を用いた評価では、最適であるとされた組み合わせの成功率が最も高かったが、他の全てのテスト用データセットでは、最適化時の評価が 3 位の組み合わせの成功率が最も高かった。

次に、7-Scenes データセットで、最適化結果が上位 3 位までに入った組み合わせの類似画像検索性能の評価結果を表 4.26 に示す。7-Scenes データセットでも、全ての評価で平均処理時間は 300msec 以内であった。Fire、Pumpkin、RedKitchen の 3 シーンのクエリ画像に対しては、最適であるとされた組み合わせの成功率が最も高かった。Chess、Office のクエリ画像に対しては、最適化時の評価が 2 位の組み合わせの成功率が、最適であるとされた組み合わせの成功率をわずかに上回った。最適化時の評価が 3 位の組み合わせは、Stairs のクエリ画像に対してのみ、最も成功率が高くなった。

表 4.25: 最適化で上位 3 位までに入った組み合わせの類似画像検索性能の比較（ふげんデータセット）

	Optimum Combination		Second Optimum Combination		Third Optimum Combination	
	Success Rate	Time*	Success Rate	Time	Success Rate	Time
TEST Dataset 1 (N=2,329)	0.675	**175 msec**	0.709	238 msec	**0.753**	234 msec
TEST Dataset 2 (N=2,363)	0.838	**175 msec**	0.857	243 msec	**0.904**	239 msec
TEST Dataset 3 (N=2,346)	0.461	**181 msec**	0.426	247 msec	**0.546**	242 msec
TEST Dataset 4 (N=2,150)	0.392	**186 msec**	0.399	252 msec	**0.451**	250 msec
TEST Dataset 5 (N=1,322)	**0.542**	**186 msec**	0.520	251 msec	0.498	258 msec
TEST Dataset 6 (N=1,971)	0.166	**191 msec**	0.184	256 msec	**0.215**	254 msec
Total (N=12,481)	0.522	**187 msec**	0.527	255 msec	**0.579**	252 msec

*Time = Avg. Processing Time

表 4.26: 最適化で上位 3 位までに入った組み合わせの類似画像検索性能の比較（7-Scenes データセット）

	Optimum Combination		Second Optimum Combination		Third Optimum Combination	
	Success Rate	Time*	Success Rate	Time	Success Rate	Time
Chess (N=1,262)	0.831	175 msec	**0.835**	182 msec	0.785	**146 msec**
Fire (N=585)	**0.858**	176 msec	0.809	186 msec	0.786	**147 msec**
Heads (N=723)	0.723	175 msec	0.725	182 msec	0.729	**146 msec**
Office (N=1,851)	0.806	176 msec	**0.807**	183 msec	0.801	**163 msec**
Pumpkin (N=826)	**0.671**	177 msec	0.656	182 msec	0.603	**147 msec**
RedKitchen (N=2,127)	**0.814**	181 msec	0.809	185 msec	0.790	**147 msec**
Stairs (N=425)	0.602	174 msec	0.548	183 msec	**0.621**	**147 msec**
Total (N=7,799)	**0.783**	177 msec	0.775	184 msec	0.757	**151 msec**

*Time = Avg. Processing Time

4.4.8 考察

表 4.10、表 4.22、および表 4.23 の最適化結果を見ると、ふげんデータセットと 7-Scenes データセットで共通して SURF-BoK、LBP、AKAZE-Matching、SURF-Matching が上位に現れており、これらの手法が様々な環境で高性能であることが分かる。LSD-1Line などの線分特徴を利用した手法は、ふげんデータセットでの最適化では評価が高かったが、7-Scenes データセットでは評価が低くなっている。このように、環境ごとに性能が大きく異なる手法も存在するが、最適化では環境に応じて適切な手法を選択できていることが確認できた。

表 4.25 および表 4.26 に示した結果より、フレームワークの最適化アルゴリズムで選出された画像検索手法と画像特徴量算出元画像のサイズの組み合わせが、必ずしも最も高い類似画像性能を発揮するものではないことが分かった。しがしながら（特に 7-Scenes データセットでは)、最適とされた組み合わせの性能が、他の組み合わせの性能に大きく劣る訳ではなく、ある程度は最適に近い組み合わせを選出できていると思われる。今後さらに、最適化の精度を高めるために、最適化時の類似画像検索精度の評価関数を絞り込み段階によって使い分ける（例：１段目は再現率、2 段目は F 値、3 段目は適合率）など、評価方法の見直しが必要である。また、本研究では 1 段目の組み合わせから順に最適化し、各段階で上位 3 位までの組み合わせを候補としたが、表 4.10 や表 4.23 を見ると、1,2 段目の組み合わせの候補の時点で偏った手法の組み合わせに絞り込まれていた。これにより、さらに高い類似画像検索性能を発揮できた組み合わせを除外してしまった可能性も考えられる。そのため、限られた時間の中で効率的により多くのバリエーションを評価するための方法なども検討する必要がある。

次に、ふげんデータセットのテスト用データセットごとの評価結果を見ると、テスト用データセット 3,4,5,6 で成功率が低かったことがわかる。テスト用データセット 3,4 は特定の機器を重点的に撮影して取得した画像から成るデータセットであるが、それぞれの機器は通路を挟んで反対側に対となる機器が存在するため、両者の区別が難しかったことなどが、成功率が低くなった要因であると考えられる。テスト用データセット 5 はカメラを比較的速く動かしながら撮影して取得した画像から成るデータセットで、手ぶれなどの影響により成功率が低くなったと考えられる。また、テスト用データセット 5 では、上位 3 つの組み合わせの成功率の順位が、他のテスト用データセットでの順位と大きく異なる。テスト用データセット 6 は、室内の照明が半分の状態で撮影して取得された画像から成るデータセットである。テスト用データセット 6 で成功率が極めて低くなっていることから、最適化で選ばれた画像検索手法が輝度の変化に

弱いことがわかる。この原因として、最適化に利用したDB中の画像が、全て同じ照明環境で撮影して取得された画像であることが挙げられる。アプリケーション利用時に人の写り込みや照明環境の変化が想定される場合には、それぞれの外乱に強い手法が最適化で選択されるように、それぞれの外乱を含む状態で撮影して取得した画像をDBに含めるなどの対策が必要である。

7-Scenesデータセットを用いた評価では、 最適とされた組み合わせが、ほぼ全てのシーンのクエリ画像に対して最も高い成功率またはそれに近い成功率を示した。一方で、最適化時の評価が3位の組み合わせは、ほぼ全てのシーンで他の2つの組み合わせに劣る性能であったが、Stairsデータセットのクエリ画像に対しては最も高い成功率を示した。1位、2位の組み合わせと3位の組み合わせでは、1,2段目の絞り込みに用いる手法の順序が異なることが結果に大きく影響したと思われる。1位、2位の組み合わせは、1段目のみの最適化では評価が2位であった組み合わせから派生した組み合わせであり、1段目で上位3つまでの組み合わせを残したことで選択された組み合わせである。このことから、各段の最適化で最適な組み合わせを1つに絞らず、より多くのバリエーションを評価する今回の実装は、有効であったと言える。

第 5 章 結論

AR を用いて原子力発電所内での作業を支援するシステムの安定性を向上させるためには、リローカリゼーション中に実行する類似画像検索の性能を向上させる必要がある。そこで、本研究では、複数の画像検索手法を組み合わせた段階的な絞り込み処理により、類似画像検索の性能を向上させるためのフレームワークを開発し、その性能を評価することを目的とした。複数の画像検索手法を組み合わせて利用する際の方法を定めた研究は、本研究が初である。

提案するフレームワークでは、高速な画像検索手法で類似画像の候補を段階的に絞り込んだ後に、高精度な画像検索手法で類似画像を選出することで、類似画像検索の速度と精度の両立を図った。段階的に類似画像の候補を絞り込む処理では、ある絞り込み段階で既定枚数の候補画像とクエリ画像の類似度の計算が終了するごとに、その中で類似度が上位の画像を次の絞り込み段階へと通過させる、部分ソートフィルタリングを用いて、各段の絞り込み処理を並列化した。また、複数の画像検索手法を組み合わせる際に、最も高い性能を発揮する組み合わせは、AR を利用する環境によって異なると考えられるが、環境ごとに最も高い性能を発揮する組み合わせを人間が予測することは難しい。そこで、提案するフレームワークには、環境に応じて画像検索手法の組み合わせを最適化するアルゴリズムを組み込んだ。最適化の際には、類似画像検索性能に影響すると思われる、画像特徴量算出元画像サイズも最適化の対象とした。

提案するフレームワークについて、1. 通過画像の決定方法として部分ソートフィルタリングが適切であるか、2. 複数の画像検索手法を組み合わせることで、単独の手法を使用する場合よりも高い類似画像検索性能を実現できているか、3. 最適化アルゴリズムによって、利用する環境ごとに適切な画像検索手法と特徴量算出元画像サイズの組み合わせを選択できているかの 3 項目を確認するために、3 種類の評価実験を行った。実験では、原子力発電プラント内での性能を評価するために、原子炉廃止措置研究開発センター（ふげん）の純水装置室内で撮影して取得した画像で構成されるデータセットを作成した。また、ふげんのような特殊な環境での最適化結果と、より一般的な環境での最適化結果を比較するために、7-Scenes データセット [31] を利用した。

1. 通過画像決定方法の評価では、通過画像決定方法に部分ソートフィルタリングを用いた場合に、全体をソートする通過画像決定方法や閾値による通過画像決定方法を

用いた場合よりも、類似画像検索性能が高いことが確認できた。2. 複数の画像検索手法を組み合わせた際の類似画像検索性能の評価では、今後最適化パートでの性能評価の信頼性が向上すれば、フレームワークに従って複数の画像検索手法を組み合わせることで、画像検索手法を単独で利用する場合よりも類似画像検索性能を向上できる可能性が示された。3. 最適化アルゴリズムの性能評価では、最適化アルゴリズムで必ずしも最適な組み合わせが選択できているわけではなかったが、ある程度性能が高い組み合わせを環境に応じて選択できていることが分かった。以上の評価結果から、提案するフレームワークの有用性が示された。

今後の課題として、最適化パートの性能を改善することが挙げられる。最適化パートでの性能評価の信頼性を向上させるほか、最適化時にそれぞれの組み合わせの評価に用いる指標の検討や、より短時間の間に様々なバリエーションの組み合わせを評価できるアルゴリズムの開発が必要である。また、本研究では、段階的な絞り込み処理に用いる画像検索手法と画像特徴量算出元画像サイズの組み合わせのみを最適化したが、各画像検索手法や部分ソートフィルタリングで用いるパラメータを含めた最適化アルゴリズムの開発も今後の課題である。

謝 辞

本研究を進めるにあたり、研究会や学会発表の練習など、様々な場面で丁寧にご指導してくださった下田宏教授に深くお礼申し上げます。時折学生部屋に足を運び、就職活動などの相談に乗っていただいたことにも、併せて感謝申し上げます

研究の方針決め、プログラミング、論文の執筆など、多くの面でご指導とお力添えを頂いた石井裕剛准教授に心より感謝申し上げます。また、先生に伝授して頂いたうどん打ちのスキルは、これからの社会を生き抜くうえで大変重宝するものと存じます。重ねて深謝いたします。

共同研究先として研究を支援してくださった原子炉廃止措置研究開発センターの皆様に、この場を借りて深くお礼申し上げます。

同じ AR チームとして、ふげんでのデータセット作成、論文執筆のサポートなどで、助力を頂いたエネルギー科学研究科修士 1 回生の原園友規君、井上純輝君、工学部電気電子工学科 4 回生の三木直也君に厚くお礼申し上げます。

長く研究室に残り研究に関するアドバイスやご指導をくださった、エネルギー科学研究科修士 2 回生の松田宅司さん、浦山大輝さんに心より感謝申し上げます。

研究に関する相談に加え、連休室での雑談や気分転換の遊びにお付き合い頂いた、エネルギー科学研究科修士 2 回生の辻雄太君、上田樹美さん、緒方省吾君、岩崎達郎君に厚くお礼申し上げます。研究室での時間を楽しく過ごすことができたのは、同期の皆様のお陰です。

同じ研究室のメンバーとして、共に楽しい時間を共有してくれた、エネルギー科学研究科修士 1 回生の原園友規君、井上純輝君、川本聡真君、日下部曜君、電気電子工学科 4 回生の三木直也君、木村覚君、竹川和佳子さんに、この場を借りてお礼申し上げます。

また、英語論文執筆のサポートをしてくれた、エネルギー情報学研究室 OB の Huang Bingrong 君に深く感謝いたします。共に切磋琢磨しアグリコラの実力を高めあった日々は、忘れられない良い思い出です。

最後になりましたが、細かな事務手続きや研究生活全般で、様々な支援を頂いた普照郁美秘書と山田美保秘書に深く感謝を申し上げます。修論執筆中に頂いたコーヒーの差し入れはとても温かく、身に沁みました。重ねてお礼申し上げます。

参 考 文 献

[1] 経済産業省・資源エネルギー庁: 日本の原子力発電所の状況, http://www.enecho.meti.go.jp/category/electricity_and_gas/nuclear/001/pdf/001_02_001.pdf, Accessed February 13, 2018.

[2] Hirotake Ishii: Augmented Reality: Fundamentals and Nuclear Related Applications, International Journal of Nuclear Safety and Simulation, **1**(4), pp.316-327(2010).

[3] Ihsan Rabbi, Sehat Ullah: A Survey on Augmented Reality Challenges and Tracking, Acta Graphica, **1**(2), pp.29-46(2013).

[4] Eric Marchand, Hideaki Uchiyama, Fabien Spindler: Pose Estimation for Augmented Reality: A Hands-On Survey, IEEE Transactions on Visualization and Computer Graphics, **22**(12), pp.2633-2651(2016).

[5] Vincent Lepetit, Francesc M. Noguer, Pascal Fua: EPnP: An Accurate O(n) Solution to the PnP Problem, International Journal of Computer Vision, doi:10.1007/s11263-008-0152-6(2009).

[6] Daniel Wagner, Gerhard Reitmayr, Alessandro Mulloni, Tom Drummond, Dieter Schmalstieg: Pose Tracking from Natural Features on Mobile Phones, Proceedings of the 7th IEEE/ACM International Symposium on Mixed and Augmented Reality, pp.125-134(2008).

[7] Gerardo Carrera, Jesus Savage, Walterio M. Cuevas, Robust Feature Descriptors for Efficient Vision-Based Tracking, In: Progress in Pattern Recognition, Luis Rueda, Domingo Mery, Josef Kittler(eds.), Springer-Verlag, pp.251-260(2007).

[8] Sebastián Bronte, Luis M. Bergasa, Daniel Pizarro, Rafael Barea: Model-Based Real-Time Non-Rigid Tracking, Sensors, **17**(10), 2342(2017).

[9] Arnold Irschara, Christopher Zach, Jan M. Frahm, Horst Bischof: From Structure-from-Motion Point Clouds to Fast Location Recognition, IEEE Conference on Computer Vision and Pattern Recognition, pp.2599-2606(2009).

[10] Torsten Sattler, Bastian Leibe, Leif Kobbelt: Fast ImageBased Localization using Direct 2D-to-3D Matching, IEEE International Conference on Computer Vision, pp.667-674(2011).

[11] Martin A. Fischler, Robert C. Bolles: Random Sample Consensus: A Paradigm for Model Fitting with Apphcatlons to Image Analysis and Automated Cartography, Communications of the ACM **24**(6), pp.381-395(1981).

[12] Paul J. Besl, Neil D. McKay: Method for Registration of 3-D Shapes, Proceedings of SPIE 1611, Sensor Fusion IV: Control Paradigms and Data Structures, doi:10.1117/12.57955(1992).

[13] Georg Klein, David Murray: Improving the Agility of Keyframe-Based SLAM, In: Computer Vision - ECCV 2008, David Forsyt, Philip Torr, Andrew Zisserman(eds.), Springer-Verlag, pp.802-815(2008).

[14] Christian Pirchheim, Gerhard Reitmayr: Homography-Based Planar Mapping and Tracking for Mobile Phones, 10th IEEE International Symposium on Mixed and Augmented Reality, doi:10.1109/ISMAR.2011.6092367(2011).

[15] Ben Glocker, Jamie Shotton, Antonio Criminisi, Shahram Izadi: Real-Time RGB-D Camera Relocalization via Randomized Ferns for Keyframe Encoding, IEEE Transactions on Visualization and Computer Graphics, **21**(5), pp.571-583(2015).

[16] Rafael G. von Gioi, Jérémie Jakubowicz, Jean-Michel Morel, Gregory Randall: LSD: a Line Segment Detector, Image Processing On Line, doi:10.5201/ipol.2012.gjmr-lsd(2012).

[17] 木村 太郎, 徳丸 博紀, 石井 裕剛, 下田 宏, 香田 友哉: プラント内の直線を利用した拡張現実感用リローカリゼーション手法の開発, ヒューマンインタフェース学会研究報告集, **18**(3), pp.7-14(2016).

[18] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M. Seitz, Richard Szeliski: Building Rome in a Day，Communications of the ACM，**54**(10)，pp.105-112(2011).

[19] NaturalPoint Inc.: OptiTrack - Motion Capture Systems, `http://optitrack.com/hardware/`, Accessed February 13, 2018.

[20] 大橋 由暉, 木村 太郎, 久留島 隆史, 石井 裕剛, 下田 宏, 香田 有哉: 環境再構成モデルによるレンダリング画像を利用したリローカリゼーション手法の開発, ヒューマンインタフェース学会研究報告集, **19**(5), pp.39-46(2017).

[21] Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: 情報検索の基礎, 岩野和生, 黒川利明, 濱田誠司, 村上明子 訳, 共立出版, pp.138-139(2012).

[22] Microsoft: Kinect for Windows SDK 2.0, `https://www.microsoft.com/en-us/download/details.aspx?id=44561`, Accessed February 13, 2018.

[23] Boost C++ Libraries, `http://www.boost.org/`, Accessed February 13, 2018.

[24] OpenCV library, `https://opencv.org/`, Accessed February 13, 2018.

[25] Pablo F. Alcantarilla, Jesús Nuevo, Adrien Bartoli: Fast Explicit Diffusion for Accelerated Features in Nonlinear Scale Spaces, IEEE Transactions on Pattern Analysis and Machine Intelligence, **34**(7), p.1281(2011).

[26] Greg Pass, Ramin Zabih, Justin Miller: Comparing Images Using Color Coherence Vectors, Proceedings of the 4th ACM International Conference on Multimedia, pp.65-73(1996).

[27] Navneet Dalal, Bill Triggs: Histograms of Oriented Gradients for Human Detection, Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, doi:10.1109/CVPR.2005.177(2005).

[28] Li Wang, Dong-Chen He: Texture Classification Using Texture Spectrum, Pattern Recognition, **23**(8), pp.905-910(1990).

[29] Gabriella Csurka, Christopher R. Dance, Lixin Fan, Jutta Willamowski, Cédric Bray: Visual Categorization with Bags of Keypoints, Workshop on Statistical Learning in Computer Vision, ECCV(2004).

[30] A. C. Murillo, J. J. Guerrero, C. Sagues: SURF features for efficient robot localization with omnidirectional images, Proceedings of 2007 IEEE International Conference on Robotics and Automation, pp.3901-3907(2007).

[31] Microsoft Research: RGB-D Dataset 7-Scenes, https://www.microsoft.com/en-us/research/project/rgb-d-dataset-7-scenes/ Accessed February 13, 2018.

[32] Ben Glocker, Shahram Izadi, Jamie Shotton, Antonio Criminisi: Real-Time RGB-D Camera Relocalization, IEEE International Symposium on Mixed and Augmented Reality, pp.173-179(2013).

[33] Richard A. Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Andrew Fitzgibbon: KinectFusion: Real-Time Dense Surface Mapping and Tracking, IEEE International Symposium on Mixed and Augmented Reality, pp.127-136(2011).

付録 A 実装した画像検索手法

本付録では、4 章の評価実験の際に実装した画像検索手法の詳細を述べる。本付録に記載したソースコードは、処理の手順を示すものであり、本研究で使用したものとは一部異なる場合がある。

A.1 AKAZE-Matching

AKAZE-Matching の画像特徴量算出方法をソースコード A.1 に、相違度の計算方法をソースコード A.2 に示す。AKAZE-Matching では、画像から検出した特徴点（処理の高速化のため強度の高い 200 点まで）を AKAZE 特徴量 [25] で記述したものを画像特徴量とした。相違度の計算では、算出した画像特徴量を用いて特徴点のマッチング（双方向マッチング）を行い、マッチングできた割合（この値が高いほど 2 つの画像は似ている＝類似度）を 1 から引いた値を相違度とした。

ソースコード A.1: AKAZE-Matching の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor; //画像特徴量を格納する行列
6  std::vector<cv::KeyPoint> keypoints; //特徴点を格納する配列
7  cv::Ptr<cv::FeatureDetector> detector = cv::AKAZE::create(); //特徴量検出器・記述子
8
9  detector->detect(src, keypoints);
10
11 //検出された特徴点を強度順に降順ソート、上位 200点以外を消去
12 if (keypoints.size() > 200) {
13         std::sort(keypoints.begin(), keypoints.end(),
14                 [](cv::KeyPoint& l, cv::KeyPoint& r){
15                 return (l == r) ? (l.size > r.size) : (l.response > r.response);
16         });
17         keypoints.erase(keypoints.begin() + 200, keypoints.end());
18 }
```

```
19
20 detector->compute(src, keypoints, descriptor); //特徴点を AKAZE 特徴量で記述
```

ソースコード A.2: AKAZE-Matching の相違度算出方法

```
1  cv::Mat descriptor1, descriptors2; //比較する 2画像の画像特徴量
2
3  cv::Ptr<cv::DescriptorMatcher> matcher
4          = cv::DescriptorMatcher::create("BruteForce-Hamming");
5
6  //特徴点を双方向マッチング
7  vector<cv::DMatch> dmatch, dmatch12, dmatch21;
8  matcher->match(descriptor1, descriptors2, dmatch1);
9  matcher->match(descriptors2, descriptor1, dmatch2);
10 for (size_t i = 0; i < dmatch12.size(); ++i){
11         cv::DMatch forward = dmatch12[i];
12         cv::DMatch backward = dmatch21[forward.trainIdx];
13         if (backward.trainIdx == forward.queryIdx) dmatch.push_back(forward);
14 }
15
16 double similarity = dmatch.size() / (double)descriptor1.rows;
17 double disimilarity = 1 - similarity;
```

A.2 CCV

CCV の画像特徴量算出方法をソースコード A.3 に示す。CCV では、画像の Color Coherence Vector[26]（CCV）を画像特徴量とし、相違度は画像特徴量間の L2 距離とした。CCV を算出する際に、同じ色のピクセルが隣接する領域を”coherent”とするか”incoherent”とするかの領域サイズの閾値 τ は、特徴量算出元画像のピクセル数の 1% のピクセル数とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.3: CCV の画像特徴量算出方法

```
1  //構造体の定義
2  typedef struct{
3          int parent;
4          uchar color;
```

```
5           int count;
6   }label;
7
8   //関数の宣言
9   inline int compress(std::vector<label> &LookUpTable, int a);
10  inline int link(std::vector<label> &LookUpTable, int a, int b);
11  inline uchar Vec3b_to_uchar(cv::Vec3b vec);
12
13  ========================================
14
15  cv::Mat src = cv::imread("ID.color.png"); //特徴量算出元画像
16  cv::Size size = { width, height } //特徴量算出元画像サイズ
17  cv::resize(src, src, size);
18
19  cv::blur(src, src, cv::Size(3, 3));
20  Vec3b *ptr_vec3b = src.ptr<cv::Vec3b>(0);
21
22  cv::Mat reduced = cv::Mat::zeros(size, CV_8UC1);
23  uchar *ptr_uchar = reduced.ptr<uchar>(0);
24  uchar *ptr_uchar_prev;
25
26  cv::Mat labeled = cv::Mat::zeros(size, CV_32SC1);
27  int *ptr_int = labeled.ptr<int>(0);
28  int *ptr_int_prev;
29
30  std::vector<label> LookUpTable;
31  LookUpTable.reserve(width*height / 8);
32
33  std::vector<uchar> color;
34
35  int num_labels = 1;
36
37  //画像のラベリング処理（色を 64分割し、同じ色が連続する領域にラベル付け）
38  //左上隅の処理
39  ptr_uchar[0] = Vec3b_to_uchar(ptr_vec3b[0]);
40  ptr_int[0] = 0;
41  LookUpTable.push_back({ 0, ptr_uchar[0], 1 });
42
43  //上端の処理
44  for (int col = 1; col < cols; ++col){
```

```
45          ptr_uchar[col] = Vec3b_to_uchar(ptr_vec3b[0]);
46
47          if (ptr_uchar[col] == ptr_uchar[col − 1]){
48                  ptr_int[col] = ptr_int[col − 1];
49                  LookUpTable[ptr_int[col]].count += 1;
50          }
51          else{
52                  ptr_int[col] = num_labels;
53                  LookUpTable.push_back({ num_labels, ptr_uchar[col], 1 });
54                  ++num_labels;
55          }
56  }
57
58  for (int row = 1; row < rows; ++row){
59          ptr_uchar_prev = ptr_uchar;
60          ptr_int_prev = ptr_int;
61          ptr_vec3b = src.ptr<cv::Vec3b>(row);
62          ptr_uchar = reduced.ptr<uchar>(row);
63          ptr_int = labeled.ptr<int>(row);
64
65          //左端の処理
66          ptr_uchar[0] = Vec3b_to_uchar(ptr_vec3b[0]);
67          if (ptr_uchar[0] == ptr_uchar_prev[0]){
68                  ptr_int[0] = ptr_int_prev[0];
69                  LookUpTable[ptr_int[0]].count += 1;
70                  if (ptr_uchar[0] == ptr_uchar_prev[1]){
71                          ptr_int[0] = link(LookUpTable, ptr_int[0], ptr_int_prev[1]);
72                  }
73          }
74          else{
75                  if (ptr_uchar[0] == ptr_uchar_prev[1]){
76                          ptr_int[0] = ptr_int_prev[1];
77                          LookUpTable[ptr_int[0]].count += 1;
78                  }
79                  else{
80                          ptr_int[0] = num_labels;
81                          LookUpTable.push_back({ num_labels, ptr_uchar[0], 1 });
82                          ++num_labels;
83                  }
84          }
```

```
85
86          //内部の処理
87          int col = 1;
88          for (; col < cols − 1; ++col){
89                  ptr_uchar[col] = Vec3b_to_uchar(ptr_vec3b[0]);
90
91                  bool flagA = (ptr_uchar[col] == ptr_uchar[col − 1]); //左
92                  bool flagB = (ptr_uchar[col] == ptr_uchar_prev[col − 1]); //左上
93                  bool flagC = (ptr_uchar[col] == ptr_uchar_prev[col]); //上
94                  bool flagD = (ptr_uchar[col] == ptr_uchar_prev[col + 1]); //右上
95
96                  ptr_int[col] = num_labels;
97                  LookUpTable.push_back({ num_labels, ptr_uchar[0], 1 });
98
99                  if (flagA | flagB | flagC | flagD){
100                         if (flagA){
101                                 ptr_int[col] = ptr_int[col − 1];
102                                 LookUpTable[ptr_int[col]].count += 1;
103                         }
104                         if (flagB) ptr_int[col]
105                                 = link(LookUpTable, ptr_int[col], ptr_int_prev[col − 1]);
106                         if (flagC) ptr_int[col]
107                                 = link(LookUpTable, ptr_int[col], ptr_int_prev[col]);
108                         if (flagD) ptr_int[col]
109                                 = link(LookUpTable, ptr_int[col], ptr_int_prev[col + 1]);
110                         LookUpTable.pop_back();
111                 }
112                 else ++num_labels;
113         }
114
115         //右端の処理
116         ptr_uchar[col] = Vec3b_to_uchar(ptr_vec3b[0]);
117
118         bool flagA = (ptr_uchar[col] == ptr_uchar[col − 1]); //左
119         bool flagB = (ptr_uchar[col] == ptr_uchar_prev[col − 1]); //左上
120         bool flagC = (ptr_uchar[col] == ptr_uchar_prev[col]); //上
121
122         ptr_int[col] = num_labels;
123         LookUpTable.push_back({ num_labels, ptr_uchar[0], 1 });
124
```

```
125         if (flagA | flagB | flagC){
126                 if (flagA){
127                         ptr_int[col] = ptr_int[col − 1];
128                         LookUpTable[ptr_int[col]].count += 1;
129                 }
130                 if (flagB) ptr_int[col]
131                         = link(LookUpTable, ptr_int[col], ptr_int_prev[col − 1]);
132                 if (flagC) ptr_int[col] = link(LookUpTable, ptr_int[col], ptr_int_prev[col]);
133                 LookUpTable.pop_back();
134         }
135         else ++num_labels;
136 }
137
138 descriptor = Mat::zeros(Size(128, 1), CV_32S);
139
140 const int tau = width∗height / 100; //”coherent” / ”incoherent” を分ける閾値
141 for (int i = 0; i < num_labels; ++i){
142         if (i == LookUpTable[i].parent){
143                 if (LookUpTable[i].count > thresh){
144                         descriptor.at<int>(0, 2 ∗ LookUpTable[i].color)
145                                 += LookUpTable[i].count;
146                 }
147                 else{
148                         descriptor.at<int>(0, 2 ∗ LookUpTable[i].color + 1)
149                                 += LookUpTable[i].count;
150                 }
151         }
152 }
153
154 ========================================
155
156 //関数の定義
157 inline int compress(std::vector<label> &LookUpTable, int a){
158         while (a != LookUpTable[a].parent){
159                 LookUpTable[a].parent = LookUpTable[LookUpTable[a].parent].parent;
160                 a = LookUpTable[a].parent;
161                 }
162         return a;
163 }
164
```

```
165 inline int link(std::vector<label> &LookUpTable, int a, int b){
166         a = compress(LookUpTable, a);
167         b = compress(LookUpTable, b);
168         if (a == b){
169                 return a;
170         }
171         else if (a < b){
172                 LookUpTable[b].parent = a;
173                 LookUpTable[a].count += LookUpTable[b].count;
174                 return a;
175         }
176         else{
177                 LookUpTable[a].parent = b;
178                 LookUpTable[b].count += LookUpTable[a].count;
179                 return b;
180         }
181 }
182
183 inline uchar Vec3b_to_uchar(cv::Vec3b vec){
184         return ((vec[0] >> 6) << 4) + ((vec[1] >> 6) << 2) + ((vec[2] >> 6) << 0);
185 }
```

A.3 Gray-Histogram

Gray-Histogram の画像特徴量算出方法をソースコード A.4 に示す。Gray-Histogram では、グレースケール化した画像から算出した輝度値のヒストグラムを画像特徴量とし、相違度は画像特徴量間の L2 距離とした。ヒストグラムのビンの数は 32（それぞれのビンの幅は一様）とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.4: Gray-Histogram の画像特徴量算出方法

```
1 cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2 cv::Size size = { width, height } //特徴量算出元画像サイズ
3 cv::resize(src, src, size);
4
5 cv::Mat descriptor; //画像特徴量を格納する行列
6
```

```
7  int histSize[] = { 32 };
8  float range[] = { 0, 256 };
9  const float* ranges[] = { range };
10 int channels[] = { 0 };
11
12 cv::calcHist(src, 1, channels, cv::Mat(), descriptor, 1, histSize, ranges, true, false);
```

A.4 Gray-L2

Gray-L2 では、グレースケール化した画像間の L2 距離を相違度とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

A.5 Gray-NCC

Gray-NCC では、グレースケール化した画像間の正規化相互相関（Normalized Cross Correlation: NCC）を 1 から引いた値を相違度とした。NCC の計算には OpneCV の matchTemplate 関数（method=CV_TM_CCORR_NORMED）を用いた。

A.6 HOG

HOG の画像特徴量算出方法をソースコード A.5 に示す。HOG では、画像の Histogram of Oriented Gradients[27]（HOG）を画像特徴量とし、相違度は画像特徴量間の L2 距離とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.5: HOG の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor; //画像特徴量を格納する行列
6
7  //画像の輝度勾配を計算
8  cv::Mat gx, gy;
9  cv::Sobel(src, gx, CV_32F, 1, 0, 1);
10 cv::Sobel(src, gy, CV_32F, 0, 1, 1);
```

```
11
12 //勾配ベクトルの大きさと角度を計算
13 cv::Mat mag, angle;
14 cv::cartToPolar(gx, gy, mag, angle, 1);
15
16 //角度が 0〜180度に収まるように再計算
17 cv::convertScaleAbs(angle, angle, 1.0, −180);
18
19 std::vector<cv::Mat> cells(width ∗ height / 64, cv::Mat::zeros(Size(9, 1), CV_32F));
20 float ∗ptr_float;
21 uchar ∗ptr_uchar;
22
23 //8x8 ピクセルのセルごとに勾配方向のヒストグラムを計算
24 for (int row = 0; row < height; ++row){
25         ptr_float = mag.ptr<float>(row);
26         ptr_uchar = angle.ptr<uchar>(row);
27
28         for (int col = 0; col < width; ++col){
29                 int cell = (width / 8)∗(row / 8) + (col / 8);
30                 int bin = int(ptr_uchar[col]) / 20;
31                 float ratio = ptr_uchar[col] / 20.0 − bin;
32
33                 switch (bin){
34                         case 0:
35                                 cells[cell].at<float>(0, 0) += (1.0 − ratio)∗ptr_float[col];
36                                 cells[cell].at<float>(0, 1) += ratio∗ptr_float[col];
37                                 break;
38
39                         case 1:
40                                 cells[cell].at<float>(0, 1) += (1.0 − ratio)∗ptr_float[col];
41                                 cells[cell].at<float>(0, 2) += ratio∗ptr_float[col];
42                                 break;
43
44                         case 2:
45                                 cells[cell].at<float>(0, 2) += (1.0 − ratio)∗ptr_float[col];
46                                 cells[cell].at<float>(0, 3) += ratio∗ptr_float[col];
47                                 break;
48
49                         case 3:
50                                 cells[cell].at<float>(0, 3) += (1.0 − ratio)∗ptr_float[col];
```

```
51                                  cells[cell].at<float>(0, 4) += ratio*ptr_float[col];
52                                  break;
53
54                          case 4:
55                                  cells[cell].at<float>(0, 4) += (1.0 - ratio)*ptr_float[col];
56                                  cells[cell].at<float>(0, 5) += ratio*ptr_float[col];
57                                  break;
58
59                          case 5:
60                                  cells[cell].at<float>(0, 5) += (1.0 - ratio)*ptr_float[col];
61                                  cells[cell].at<float>(0, 6) += ratio*ptr_float[col];
62                                  break;
63
64                          case 6:
65                                  cells[cell].at<float>(0, 6) += (1.0 - ratio)*ptr_float[col];
66                                  cells[cell].at<float>(0, 7) += ratio*ptr_float[col];
67                                  break;
68
69                          case 7:
70                                  cells[cell].at<float>(0, 7) += (1.0 - ratio)*ptr_float[col];
71                                  cells[cell].at<float>(0, 8) += ratio*ptr_float[col];
72                                  break;
73
74                          case 8:
75                                  cells[cell].at<float>(0, 8) += (1.0 - ratio)*ptr_float[col];
76                                  cells[cell].at<float>(0, 0) += ratio*ptr_float[col];
77                                  break;
78                  }
79          }
80  }
81
82  std::vector<cv::Mat> blocks((width / 8 - 1)*(height / 8 - 1));
83
84  //2x2 セルのブロックごとにヒストグラムを正規化
85  for (int row = 0; row < height / 8 - 1; ++row){
86          for (int col = 0; col < width / 8 - 1; ++col){
87                  int cell, block;
88                  cell = block = (width / 8 - 1)*row + col;
89                  vconcat(std::vector<cv::Mat>{ cells[cell], cells[cell + 1],
90                      cells[cell + width / 8],   cells[cell + width / 8 + 1] }, blocks[block]);
```

```
91                  blocks[block] /= norm(blocks[block], cv::NORM_L2);
92          }
93  }
94
95  hconcat(blocks, descriptor);
```

A.7 LBP

LBP の画像特徴量算出方法をソースコード A.6 に示す。LBP では、画像の Local Binary Pattern[28]（LBP）のヒストグラムを画像特徴量とし、相違度は画像特徴量間の L2 距離とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.6: LBP の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor; //画像特徴量を格納する行列
6
7  cv::Mat LBP = cv::Mat::zeros(src.rows − 2, src.cols − 2, CV_8UC1);
8
9  for (int row = 1; row   <   src.rows − 1; ++row) {
10         for (int col = 1; col   <   src.cols − 1; ++col) {
11                 uchar center = src.at<uchar>(row, col);
12                 uchar code = 0;
13                 code |= (src.at<uchar>(row − 1, col − 1) > center) << 7;
14                 code |= (src.at<uchar>(row − 1, col) > center) << 6;
15                 code |= (src.at<uchar>(row − 1, col + 1) > center) << 5;
16                 code |= (src.at<uchar>(row, col + 1) > center) << 4;
17                 code |= (src.at<uchar>(row + 1, col + 1) > center) << 3;
18                 code |= (src.at<uchar>(row + 1, col) > center) << 2;
19                 code |= (src.at<uchar>(row + 1, col − 1) > center) << 1;
20                 code |= (src.at<uchar>(row, col − 1) > center) << 0;
21                 LBP.at<uchar>(row − 1, col − 1) = code;
22         }
23 }
24
```

```
25 int histSize[] = { 256 };
26 float range[] = { 0, 256 };
27 const float* ranges[] = { range };
28 int channels[] = { 0 };
29 calcHist(&LBP, 1, channels, Mat(), descriptor, 1, histSize, ranges, true, false);
```

A.8 LSD-Angle

LSD-Angle の画像特徴量算出方法をソースコード A.7 に示す。LSD-Angle では、まず、画像中の線分を Line Segment Detector[16]（LSD）で検出した。次に、検出された線分（を延長した直線）同士が画像上で成す角を全ての線分の組み合わせで求め、成す角のヒストグラム（角度が 0°〜22.5°、22.5°〜45°、45°〜67.5°、67.5°〜90° の 4 ビン）を画像特徴量とした。相違度は画像特徴量間の L2 距離とし、L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.7: LSD-Angle の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor= cv::Mat::zeros(cv::Size(4, 1), CV_32S); //画像特徴量を格納する行列
6  int* des = descriptor.ptr<int>(0);
7
8  //成す角のビン分けの閾値
9  double threshold[3];
10 threshold[0] = cosf(CV_PI / 8.)*cosf(CV_PI / 8.);   //cos( 22.5° )の 2乗
11 threshold[1] = cosf(CV_PI / 4.)*cosf(CV_PI / 4.);   //cos( 45° ) の 2乗
12 threshold[2] = cosf(CV_PI*3 / 8.)*cosf(CV_PI*3 / 8.);   //cos( 67.5° )の 2乗
13
14 cv::Ptr<cv::LineSegmentDetector> detector = createLineSegmentDetector();
15 detector->detect(src, lines);
16
17 cv::Vec2f vec1, vec2;
18 float norm1, norm2;
19 double angle;
20
21 for (int i = 0; i < (int)lines.size() - 1; ++i){
```

```
22          vec1[0] = lines[i][0] − lines[i][2];
23          vec1[1] = lines[i][1] − lines[i][3];
24          norm1 = vec1[0] ∗ vec1[0] + vec1[1] ∗ vec1[1]; //norm の 2 乗
25
26          for (int j = i + 1; j < (int)lines.size(); ++j) {
27                  vec2[0] = lines[j][0] − lines[j][2];
28                  vec2[1] = lines[j][1] − lines[j][3];
29                  norm2 = vec2[0] ∗ vec2[0] + vec2[1] ∗ vec2[1]; //norm の 2 乗
30
31                  //vec1 と vec2 の成す角の余弦の 2 乗
32                  angle = (vec1[0] ∗ vec2[0] + vec1[1] ∗ vec2[1])∗
33                          (vec1[0] ∗ vec2[0] + vec1[1] ∗ vec2[1]) / norm1 / norm2;
34
35                  if (angle > threshopld[1]) {
36                          if (angle > threshopld[0]) des[0] += 1;
37                          else des[1] += 1;
38                  }
39                  else {
40                          if (angle > threshopld[2]) des[2] += 1;
41                          else des[3] += 1;
42                  }
43          }
44  }
```

A.9 LSD-2Lines

LSD-2Lines の画像特徴量算出方法をソースコード A.8 に示す。LSD-2Lines では、まず、画像中の線分を LSD で検出した。次に、検出された線分（を延長した直線）同士が画像上で成す角と線分間の画像上での距離（最も距離が近い端点間の距離）を全ての線分の組み合わせで求め、成す角と距離のヒストグラム（角度、距離をそれぞれ 4 つのビンに分けた計 16 ビン）を画像特徴量とした。相違度は画像特徴量間の L2 距離とし、L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.8: LSD-2Lines の画像特徴量算出方法

```
1 cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2 cv::Size size = { width, height } //特徴量算出元画像サイズ
3 cv::resize(src, src, size);
```

```
4
5  cv::Mat descriptor= cv::Mat::zeros(cv::Size(16, 1), CV_32S); //画像特徴量を格納する行列
6  int* des = descriptor.ptr<int>(0);
7
8  //成す角のビン分けの閾値
9  double threshold_angle[3];
10 threshold_angle[0] = cosf(CV_PI / 8.)*cosf(CV_PI / 8.);   //cos( 22.5° )の 2乗
11 threshold_angle[1] = cosf(CV_PI / 4.)*cosf(CV_PI / 4.);   //cos( 45° ) の 2乗
12 threshold_angle[2] = cosf(CV_PI*3 / 8.)*cosf(CV_PI*3 / 8.);   //cos( 67.5° )の 2乗
13
14 //距離のビン分けの閾値
15 double threshold_distance[4];
16 threshold_distance[0] = 25; //5^2
17 threshold_distance[1] = 100; //10^2
18 threshold_distance[2] = 400; //20^2
19 threshold_distance[3] = 1600; //40^2
20
21 cv::Ptr<cv::LineSegmentDetector> detector = createLineSegmentDetector();
22 detector->detect(src, lines);
23
24 cv::Vec2f vec1, vec2;
25 float norm1, norm2;
26 double angle;
27
28 double distance[4]; //2つ線分の端点間の距離(端点の組み合わせ 4通りのそれぞれで)
29 double distanceMin1, distanceMin2;
30
31 int bin;
32
33 for (int i = 0; i < (int)lines.size() - 1; ++i){
34         vec1[0] = lines[i][0] - lines[i][2];
35         vec1[1] = lines[i][1] - lines[i][3];
36         norm1 = vec1[0] * vec1[0] + vec1[1] * vec1[1]; //norm の 2 乗
37
38         for (int j = i + 1; j < (int)lines.size(); ++j) {
39                 vec2[0] = lines[j][0] - lines[j][2];
40                 vec2[1] = lines[j][1] - lines[j][3];
41                 norm2 = vec2[0] * vec2[0] + vec2[1] * vec2[1]; //norm の 2 乗
42
43                 //vec1 と vec2 の成す角の余弦の 2 乗
```

```
44                  angle = (vec1[0] * vec2[0] + vec1[1] * vec2[1])*
45                  (vec1[0] * vec2[0] + vec1[1] * vec2[1]) / norm1 / norm2;
46
47                  if (angle > threshold_angle[1]) {
48                          if (angle > threshold_angle[0]) bin = 0;
49                          else bin = 4;
50                  }
51                  else {
52                          if (angle > threshold_angle[2]) bin = 8;
53                          else bin = 12;
54                  }
55
56                  distance[0] = (lines[i][0] - lines[j][0])*(lines[i][0] - lines[j][0])
57                          + (lines[i][1] - lines[j][1])*(lines[i][1] - lines[j][1]);
58                  distance[1] = (lines[i][0] - lines[j][0])*(lines[i][2] - lines[j][2])
59                          + (lines[i][1] - lines[j][1])*(lines[i][3] - lines[j][3]);
60                  distance[2] = (lines[i][2] - lines[j][2])*(lines[i][0] - lines[j][0])
61                          + (lines[i][3] - lines[j][3])*(lines[i][1] - lines[j][1]);
62                  distance[3] = (lines[i][2] - lines[j][2])*(lines[i][2] - lines[j][2])
63                          + (lines[i][3] - lines[j][3])*(lines[i][3] - lines[j][3]);
64                  distanceMin1 = min(distance[0], distance[1]);
65                  distanceMin2 = min(distance[2], distance[3]);
66                  distanceMin1 = min(distanceMin1, distanceMin2);
67
68                  if (distanceMin1 < threshold_distance[3]) {
69                          if (distanceMin1 < threshold_distance[1]) {
70                                  if (distanceMin1 < threshold_distance[0]) des[bin] += 1;
71                                  else des[bin + 1] += 1;
72                          }
73                          else {
74                                  if (distanceMin1 < threshold_distance[2]) des[bin + 2] +=
                                        1;
75                                  else des[bin + 3] += 1;
76                          }
77                  }
78          }
79  }
```

A.10 LSD-Direction

LSD-Direction の画像特徴量算出方法をソースコード A.9 に示す。LSD-Direction では、まず、画像中の線分を LSD で検出した。次に、検出されたそれぞれの線分の画像上での傾きを求め、水平に近い線分（傾きが 0°〜10°）と垂直に近い線分（傾きが 80°〜90°）の本数を画像特徴量とした。相違度は画像特徴量間の L2 距離とし、L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.9: LSD-Direction の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor= cv::Mat::zeros(cv::Size(2, 1), CV_32S); //画像特徴量を格納する行列
6  int* des = descriptor.ptr<int>(0);
7
8  //水平、垂直に近いとみなす傾きの閾値
9  float threshold_horizontal = tanf(CV_PI / 18.);
10 float threshold_vertical = tanf(CV_PI*4. / 9.);
11
12 cv::Ptr<cv::LineSegmentDetector> detector = createLineSegmentDetector();
13 detector->detect(src, lines);
14
15 double slope;
16
17 for (int i = 0; i < (int)lines.size(); ++i){
18         slope = abs((lines[i][1] - lines[i][3]) / (lines[i][0] - lines[i][2]));
19
20         if (slope < threshold_horizontal) des[0] += 1;
21         if (slope > threshold_vertical) des[1] += 1;
22 }
```

A.11 LSD-Distance

LSD-Distance の画像特徴量算出方法をソースコード A.10 に示す。LSD-Distance では、まず、画像中の線分を LSD で検出した。次に、検出された線分間の画像上での距離（最も距離が近い端点間の距離）を全ての線分の組み合わせで求め、距離のヒスト

グラム（0〜5[pixels]、5〜10[pixels]、10〜20[pixels]、20〜40[pixels] の 4 ビン）を画像特徴量とした。相違度は画像特徴量間の L2 距離とし、L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.10: LSD-Distance の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor= cv::Mat::zeros(cv::Size(4, 1), CV_32S); //画像特徴量を格納する行列
6  int* des = descriptor.ptr<int>(0);
7
8  //距離のビン分けの閾値
9  double threshold_distance[4];
10 threshold_distance[0] = 25; //5^2
11 threshold_distance[1] = 100; //10^2
12 threshold_distance[2] = 400; //20^2
13 threshold_distance[3] = 1600; //40^2
14
15 cv::Ptr<cv::LineSegmentDetector> detector = createLineSegmentDetector();
16 detector->detect(src, lines);
17
18 double distance[4]; //2つ線分の端点間の距離(端点の組み合わせ 4通りのそれぞれで)
19 double distanceMin1, distanceMin2;
20
21 for (int i = 0; i < (int)lines.size() - 1; ++i){
22         for (int j = i + 1; j < (int)lines.size(); ++j) {
23                 distance[0] = (lines[i][0] - lines[j][0])*(lines[i][0] - lines[j][0])
24                         + (lines[i][1] - lines[j][1])*(lines[i][1] - lines[j][1]);
25                 distance[1] = (lines[i][0] - lines[j][0])*(lines[i][2] - lines[j][2])
26                         + (lines[i][1] - lines[j][1])*(lines[i][3] - lines[j][3]);
27                 distance[2] = (lines[i][2] - lines[j][2])*(lines[i][0] - lines[j][0])
28                         + (lines[i][3] - lines[j][3])*(lines[i][1] - lines[j][1]);
29                 distance[3] = (lines[i][2] - lines[j][2])*(lines[i][2] - lines[j][2])
30                         + (lines[i][3] - lines[j][3])*(lines[i][3] - lines[j][3]);
31                 distanceMin1 = min(distance[0], distance[1]);
32                 distanceMin2 = min(distance[2], distance[3]);
33                 distanceMin1 = min(distanceMin1, distanceMin2);
34
```

```
35                  if (distanceMin1 < threshold_distance[3]) {
36                          if (distanceMin1 < threshold_distance[1]) {
37                                  if (distanceMin1 < threshold_distance[0]) des[0] += 1;
38                                  else des[1] += 1;
39                          }
40                          else {
41                                  if (distanceMin1 < threshold_distance[2]) des[2] += 1;
42                                  else des[3] += 1;
43                          }
44                  }
45          }
46  }
```

A.12 LSD-Number

LSD-Number では、LSD で検出された画像中の線分の本数を画像特徴量とし、相違度は画像特徴量間の差とした。LSD は OpenCV に実装されているものを用いた。

A.13 LSD-Region

LSD-Region の画像特徴量算出方法をソースコード A.11 に示す。LSD-Region では、まず、画像中の線分を LSD で検出した。次に、画像を上下左右 4 つの領域に分割し、それぞれの領域に存在する線分（線分の中点の座標で判定）の本数を画像特徴量とした。相違度は画像特徴量間の L2 距離とし、L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.11: LSD-Region の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor= cv::Mat::zeros(cv::Size(4, 1), CV_32S); //画像特徴量を格納する行列
6  int* des = descriptor.ptr<int>(0);
7
8  cv::Ptr<cv::LineSegmentDetector> detector = createLineSegmentDetector();
9  detector->detect(src, lines);
```

```
10
11 double length;
12
13 for (int i = 0; i < (int)lines.size(); ++i){
14         if ((lines[i][0] + lines[i][2]) < width) {
15                 if ((lines[i][1] + lines[i][3]) < height) des[0] += 1;
16                 else des[1] += 1;
17         }
18         else {
19                 if ((lines[i][1] + lines[i][3]) < height) des[2] += 1;
20                 else des[3] += 1;
21         }
22 }
```

A.14 LSD-1Line

LSD-1Line の画像特徴量算出方法をソースコード A.12 に示す。LSD-1Line では、まず、画像中の線分を LSD で検出した。次に、画像を上下左右 4 つの領域に分割し、それぞれの領域に存在する水平に近い線分と垂直に近い線分それぞれの長さのヒストグラムを画像特徴量とした。相違度は画像特徴量間の L2 距離とし、L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.12: LSD-1Line の画像特徴量算出方法

```
1 cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2 cv::Size size = { width, height } //特徴量算出元画像サイズ
3 cv::resize(src, src, size);
4
5 cv::Mat descriptor= cv::Mat::zeros(cv::Size(32, 1), CV_32S); //画像特徴量を格納する行列
6 int* des = descriptor.ptr<int>(0);
7
8 //距離のビン分けの閾値
9 double threshold_length[3];
10 threshold_length[0] = 25; //5^2
11 threshold_length[1] = 100; //10^2
12 threshold_length[2] = 400; //20^2
13
14 //水平、垂直に近いとみなす傾きの閾値
```

```
15 float threshold_horizontal = tanf(CV_PI / 18.);
16 float threshold_vertical = tanf(CV_PI*4. / 9.);
17
18 cv::Ptr<cv::LineSegmentDetector> detector = createLineSegmentDetector();
19 detector->detect(src, lines);
20
21 double length;
22 double slope;
23 int bin;
24
25 for (int i = 0; i < (int)lines.size(); ++i){
26         if ((lines[i][0] + lines[i][2]) < width) {
27                 if ((lines[i][1] + lines[i][3]) < height) bin = 0;
28                 else bin = 8;
29         }
30         else {
31                 if ((lines[i][1] + lines[i][3]) < height) bin = 16;
32                 else bin = 24;
33         }
34
35         length = (lines[n][0] - lines[n][2])*(lines[n][0] - lines[n][2])
36                 + (lines[n][1] - lines[n][3])*(lines[n][1] - lines[n][3]);
37         if (length < threshold_length[1]) {
38                 if (length >= threshold_length[0]) bin += 2;
39         }
40         else {
41                 if (length < threshold_length[2]) bin += 4;
42                 else bin += 6;
43         }
44
45         slope = abs((lines[i][1] - lines[i][3]) / (lines[i][0] - lines[i][2]));
46         if (slope < threshold_horizontal) des[bin] += 1;
47         if (slope > threshold_vertical) des[bin + 1] += 1;
48 }
```

A.15 Randomized-Fern

Randomized-Fern では、画像からランダムに選択したピクセルの輝度値をランダムな閾値で二値化した値を連ねたコードを画像特徴量とする [15]。Randomized-Fern の画像特徴量算出方法をソースコード A.13 に示す。本研究では、デプス画像は用いず、RGB の 3 チャンネルのみから画像特徴量を計算した。fern[15] 数は 500 とし、それぞれの fern は異なる座標の 4 ピクセル（チャンネルはランダム）で構成される。画像特徴量間の Block Hamming Distance[15]（BlockHD）を画像間の相違度とした。BlockHD の計算には、各 fern のコード表（各コードを持つ DB 画像の ID を記録したもの）を作成し、コードブロックの共起回数を数える方法 [15] を用いた。

ソースコード A.13: Randomized-Fern の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat kernel = cv::getGaussianKernel(15, 2.5, CV_64F);
6  filter2D(src, src, CV_32F, kernel);
7
8  int numFerns = 500;
9  uchar *descriptor = new uchar[numFerns];
10 int positions[4 * numFerns]; //ランダムに選択されたピクセル番号
11 uchar thresholds[4 * numFerns]; //各ピクセルに対するランダムな閾値
12
13 for (int i = 0; i < numFerns; ++i) {
14         descriptor[i] = 0;
15         for (int j = 0; j < 4; ++j) {
16                 if ( *((float*)src.data + positions[4*i + j])) < thresholds[4*i + j] ){
17                         descriptor[i] |= 1 << j;
18                 }
19         }
20 }
```

A.16 RGB-Histogram

RGB-Histogram の画像特徴量算出方法をソースコード A.14 に示す。RGB-Histogram では、RGB 画像の RGB 各チャンネルで算出した輝度値のヒストグラムを統合したものを画像特徴量とし、相違度は画像特徴量間の L2 距離とした。ヒストグラムのビンの数はそれぞれのチャンネルで 32（それぞれのビンの幅は一様）とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

ソースコード A.14: RGB-Histogram の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png"); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor; //画像特徴量を格納する行列
6
7  cv::Mat hist[3];
8  int histSize[] = { 32 };
9  float range[] = { 0, 256 };
10 const float* ranges[] = { range };
11 int channels[] = { 0 };
12
13 for(int i = 0; i < 3; ++i){
14         cv::calcHist(src, 1, &i, cv::Mat(), hist[i], 1, histSize, ranges, true, false);
15 }
16
17 cv::hconcat(hist, 3, descriptor);
```

A.17 RGB-L2

RGB-L2 では、RGB 画像の画像間の L2 距離を相違度とした。L2 距離の計算には OpneCV の norm 関数（normType=NORM_L2）を用いた。

A.18 RGB-NCC

RGB-NCC では、RGB 画像の画像間の NCC を 1 から引いた値を相違度とした。NCC の計算には OpneCV の matchTemplate 関数（method=CV_TM_CCORR_NORMED）

を用いた。

A.19 SURF-BoK

SURF-BoK では、画像を特徴点の集まりとして表現する Bags of Keypoints[29]（BoK）を用いて画像間の相違度を計算する。まず、事前に特徴点のクラス分けを記した辞書（各クラスを表す代表的な特徴量が記録されている）を作成する。この辞書を用いることで、画像を、辞書に記された各クラスの特徴点が何個ずつ存在するかを示したベクトル（BoK）で表現することができる。SURF-BoK では、画像の BoK を画像特徴量とし、画像特徴量間の L2 距離（OpenCV の norm 関数で算出）を相違度とした。BoK を求めるための特徴点の表現には SURF 特徴量 [30] を用いた。SURF-BoK の辞書作成方法、画像特徴量算出方法をそれぞれソースコード A.15、ソースコード A.16 に示す。辞書は DB から等間隔に抜き出した画像 1,000 枚から作成し、辞書のサイズ（特徴点のクラス分けの数）は最大で 200 とした。また、処理の高速化のため、辞書作成時には画像から検出した特徴点の内、強度が高い 100 点までを、BoK の算出時には 500 点までを使用した。

ソースコード A.15: SURF-BoK の辞書作成方法

```
1  cv::Size size = { width, height } //特徴量算出元画像サイズ
2  int DBsize = Number of Images in DB; //DB に含まれる画像数
3
4  cv::Ptr<cv::FeatureDetector> detector = cv::xfeatures2d::SURF::create();
5  cv::Ptr<cv::DescriptorMatcher> matcher = cv::DescriptorMatcher::create("BruteForce");
6  cv::Ptr<cv::BOWImgDescriptorExtractor> extractor
7          = new cv::BOWImgDescriptorExtractor(detector, matcher);
8  cv::BOWKMeansTrainer trainer(200,
9      cv::TermCriteria(CV_TERMCRIT_ITER, 100, 0.001), 3, KMEANS_PP_CENTERS);
10
11 cv::Mat dictionary;
12 vector<cv::KeyPoint> keypoints;
13
14 for (int i = 0; i < 1000; ++i){
15         //DB の ID が i*DBsize/1000の画像をグレースケールで読み込み
16         cv::Mat src = imread("ID.color.png", 0);
17         cv::resize(src, src, size);
18
```

```
19          detector->detect(src, keypoints);
20          if (keypoints.size() > 0) {
21                  if (keypoints.size() > 100){
22                          std::sort(keypoints.begin(), keypoints.end(),
23                          [](cv::KeyPoint& l, cv::KeyPoint& r){
24                          return (l == r) ? (l.size > r.size) : (l.response > r.response);});
25                          keypoints.erase(keypoints.begin() + 100, keypoints.end());
26                  }
27          detector->compute(src, keypoints, descriptor);
28          trainer.add(descriptor);
29  }
30
31  dictionary = trainer.cluster();
32
33  cv::Mat descriptor; //画像特徴量を格納する行列
34  std::vector<cv::KeyPoint> keypoints; //特徴点を格納する配列
35  cv::Ptr<cv::FeatureDetector> detector = cv::xfeatures2d::SURF::create();
36
37  detector->detect(src, keypoints);
```

ソースコード A.16: SURF-BoK の画像特徴量算出方法

```
1   cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2   cv::Size size = { width, height } //特徴量算出元画像サイズ
3   cv::resize(src, src, size);
4
5   cv::Mat descriptor;
6   vector<cv::KeyPoint> keypoints;
7
8   cv::Ptr<cv::FeatureDetector> detector = cv::xfeatures2d::SURF::create();
9   cv::Ptr<cv::DescriptorMatcher> matcher = cv::DescriptorMatcher::create("BruteForce");
10  cv::Ptr<cv::BOWImgDescriptorExtractor> extractor
11          = new cv::BOWImgDescriptorExtractor(detector, matcher);
12
13  detector->detect(src, keypoints);
14  if (keypoints.size() > 500) {
15          std::sort(keypoints.begin(), keypoints.end(),
16                  [](cv::KeyPoint& l, cv::KeyPoint& r){
17                          return (l == r) ? (l.size > r.size) : (l.response > r.response);
18          });
```

```
19          keypoints.erase(keypoints.begin() + 500, keypoints.end());
20  }
21  if (keypoints.size() == 0) descriptor = Mat::zeros(Size(200, 1), CV_32F);
22  else extractor->compute(src, keypoints, descriptor);
```

A.20 SURF-Matching

SURF-Matching の画像特徴量算出方法をソースコード A.17 に、相違度の計算方法をソースコード A.18 に示す。SURF-Matching では、画像から検出した特徴点（処理の高速化のため強度の高い 200 点まで）を SURF 特徴量 [30] で記述したものを画像特徴量とした。相違度の計算では、算出した画像特徴量を用いて特徴点のマッチング（双方向マッチング）を行い、マッチングできた割合を 1 から引いた値を相違度とした。

ソースコード A.17: SURF-Matching の画像特徴量算出方法

```
1  cv::Mat src = cv::imread("ID.color.png", 0); //特徴量算出元画像
2  cv::Size size = { width, height } //特徴量算出元画像サイズ
3  cv::resize(src, src, size);
4
5  cv::Mat descriptor; //画像特徴量を格納する行列
6  std::vector<cv::KeyPoint> keypoints; //特徴点を格納する配列
7  cv::Ptr<cv::FeatureDetector> detector = cv::xfeatures2d::SURF::create();
8
9  detector->detect(src, keypoints);
10
11 //検出された特徴点を強度順に降順ソート、上位200点以外を消去
12 if (keypoints.size() > 200) {
13          std::sort(keypoints.begin(), keypoints.end(),
14                  [](cv::KeyPoint& l, cv::KeyPoint& r){
15                  return (l == r) ? (l.size > r.size) : (l.response > r.response);
16          });
17          keypoints.erase(keypoints.begin() + 200, keypoints.end());
18 }
19
20 detector->compute(src, keypoints, descriptor); //特徴点を SURF 特徴量で記述
```

ソースコード A.18: SURF-Matching の相違度算出方法

```
1  cv::Mat descriptor1, descriptors2; //比較する 2画像の画像特徴量
2
3  cv::Ptr<cv::DescriptorMatcher> matcher = cv::DescriptorMatcher::create("BruteForce");
4
5  //特徴点を双方向マッチング
6  vector<cv::DMatch> dmatch, dmatch12, dmatch21;
7  matcher->match(descriptor1, descriptors2, dmatch1);
8  matcher->match(descriptors2, descriptor1, dmatch2);
9  for (size_t i = 0; i < dmatch12.size(); ++i){
10         cv::DMatch forward = dmatch12[i];
11         cv::DMatch backward = dmatch21[forward.trainIdx];
12         if (backward.trainIdx == forward.queryIdx) dmatch.push_back(forward);
13 }
14
15 double similarity = dmatch.size() / (double)descriptor1.rows;
16 double disimilarity = 1 - similarity;
```

付録 B 最適化の結果（ふげんデータセット）

表 B.1: 1段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット）

1st Filter		Recall	Processing Time (Average)
Method	Image Size		
CCV	128×72	0.4355	22 msec
	64×36	0.3766	21 msec
Gray-Histogram	1920×1080	0.7953	16 msec
	1024×576	0.7959	16 msec
	512×288	0.7956	15 msec
	256×144	0.7945	14 msec
	128×72	0.7907	14 msec
LBP	64×36	**0.8214**	28 msec
LSD-Direction	512×288	0.5039	31 msec
LSD-Region	512×288	0.7576	31 msec
LSD-1Line	1024×576	0.7645	65 msec
	512×288	**0.8290**	32 msec
Randomized-Fern	1920×1080	0.7102	63 msec
	1024×576	0.7137	25 msec
	512×288	0.7159	12 msec
	256×144	0.7396	9 msec
	128×72	0.7485	8 msec
	64×36	0.7549	9 msec
	40×30	0.7378	9 msec
RGB-Histogram	1920×1080	0.8107	30 msec
	1024×576	0.8086	24 msec
	512×288	0.8089	19 msec
	256×144	0.8094	19 msec
	128×72	0.8067	18 msec
SURF-BoK	512×288	**0.8684**	50 msec
	256×144	0.8397	30 msec
	128×72	0.8108	21 msec

表 B.2: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part1）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
SURF-BoK 512×288	CCV	1920×1080	0.3479	71 msec
		1024×576	0.3461	65 msec
	Gray-Histogram	1920×1080	0.6360	60 msec
		1024×576	0.6334	65 msec
		512×288	0.6330	59 msec
		256×144	0.6317	59 msec
		128×72	0.6235	60 msec
	Gray-L2	256×144	0.6128	60 msec
		128×72	0.6134	63 msec
		64×36	0.6117	63 msec
		32×18	0.5921	65 msec
		16×9	0.5308	65 msec
	Gray-NCC	64×36	0.5979	143 msec
		32×18	0.5825	98 msec
		16×9	0.5255	81 msec
	HOG	256×144	0.5359	96 msec
		128×72	0.5401	54 msec
	LBP	256×144	0.5993	87 msec
		128×72	0.6374	84 msec
		64×36	0.5953	85 msec
	LSD-Angle	1024×576	0.3697	156 msec
	LSD-2Lines	1024×576	0.3879	162 msec
	LSD-Direction	1024×576	0.3100	107 msec
		512×288	0.3729	114 msec
	LSD-Distance	1024×576	0.3005	134 msec
	LSD-Length	1024×576	0.3621	406 msec
	LSD-Number	1024×576	0.2613	98 msec

表 B.3: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part2）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
SURF-BoK 512×288	LSD-Region	1920×1080	0.5734	181 msec
		1024×576	0.5967	119 msec
		512×288	0.5785	124 msec
	LSD-1Line	1024×576	0.5813	176 msec
		512×288	0.6404	127 msec
		256×144	0.5915	118 msec
	Randomized-Fern	1920×1080	0.5792	129 msec
		1024×576	0.5833	133 msec
		512×288	0.5821	132 msec
		256×144	0.5903	130 msec
		128×72	0.5907	129 msec
		64×36	0.5836	136 msec
		40×30	0.5708	137 msec
	RGB-Histogram	1920×1080	0.6446	71 msec
		1024×576	0.6446	77 msec
		512×288	0.6452	71 msec
		256×144	0.6460	71 msec
		128×72	0.6411	72 msec
	RGB-L2	256×144	0.6129	105 msec
		128×72	0.6121	67 msec
		64×36	0.6097	78 msec
		32×18	0.5936	79 msec
		16×9	0.5387	78 msec
	RGB-NCC	32×18	0.5890	144 msec
		16×9	0.5316	109 msec
	SURF-BoK	1024×576	0.6876	162 msec
		512×288	**0.6881**	79 msec
		256×144	0.6742	69 msec

表 B.4: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part3）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LSD-1Line 512×288	CCV	1920×1080	0.3202	70 msec
		1024×576	0.3214	52 msec
	Gray-Histogram	1920×1080	0.6032	50 msec
		1024×576	0.6011	49 msec
		512×288	0.5998	48 msec
		256×144	0.6025	46 msec
		128×72	0.5893	45 msec
	Gray-L2	256×144	0.5743	38 msec
		128×72	0.5699	43 msec
		64×36	0.5700	45 msec
		32×18	0.5487	44 msec
		16×9	0.4919	44 msec
	Gray-NCC	64×36	0.5619	175 msec
		32×18	0.5418	79 msec
		16×9	0.4864	62 msec
	HOG	256×144	0.4945	66 msec
		128×72	0.5025	37 msec
	LBP	256×144	0.5591	52 msec
		128×72	0.6049	51 msec
		64×36	0.5539	52 msec
	LSD-Angle	1024×576	0.3285	151 msec
	LSD-2Lines	1024×576	0.3734	157 msec
	LSD-Direction	1024×576	0.2068	96 msec
		512×288	0.2261	68 msec
	LSD-Distance	1024×576	0.2924	126 msec
	LSD-Length	1024×576	0.3304	94 msec
	LSD-Number	1024×576	0.2321	92 msec

表 B.5: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part4）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LSD-1Line 512×288	LSD-Region	1920×1080	0.5262	183 msec
		1024×576	0.5452	100 msec
		512×288	0.5131	71 msec
	LSD-1Line	1024×576	0.5119	102 msec
		512×288	0.5750	72 msec
		256×144	0.5068	67 msec
	Randomized-Fern	1920×1080	0.5420	86 msec
		1024×576	0.5432	69 msec
		512×288	0.5455	63 msec
		256×144	0.5518	68 msec
		128×72	0.5477	63 msec
		64×36	0.5414	63 msec
		40×30	0.5230	64 msec
	RGB-Histogram	1920×1080	0.6132	59 msec
		1024×576	0.6155	56 msec
		512×288	0.6121	57 msec
		256×144	0.6128	56 msec
		128×72	0.6069	49 msec
	RGB-L2	256×144	0.5731	75 msec
		128×72	0.5718	38 msec
		64×36	0.5690	50 msec
		32×18	0.5519	47 msec
		16×9	0.4991	48 msec
	RGB-NCC	32×18	0.5452	116 msec
		16×9	0.4937	80 msec
	SURF-BoK	1024×576	0.6693	450 msec
		512×288	**0.6772**	155 msec
		256×144	0.6523	77 msec

表 B.6: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part5）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LBP 64×36	CCV	1920×1080	0.3037	57 msec
		1024×576	0.3033	31 msec
	Gray-Histogram	1920×1080	0.5860	30 msec
		1024×576	0.5912	37 msec
		512×288	0.5886	29 msec
		256×144	0.5885	29 msec
		128×72	0.5746	29 msec
	Gray-L2	256×144	0.5699	30 msec
		128×72	0.5640	29 msec
		64×36	0.5635	29 msec
		32×18	0.5426	29 msec
		16×9	0.4895	29 msec
	Gray-NCC	64×36	0.5578	100 msec
		32×18	0.5398	54 msec
		16×9	0.4851	32 msec
	HOG	256×144	0.4842	35 msec
		128×72	0.4788	30 msec
	LBP	256×144	0.5345	30 msec
		128×72	0.5809	30 msec
		64×36	0.5026	29 msec
	LSD-Angle	1024×576	0.3347	108 msec
	LSD-2Lines	1024×576	0.3707	112 msec
	LSD-Direction	1024×576	0.2676	58 msec
		512×288	0.3153	30 msec
	LSD-Distance	1024×576	0.2843	83 msec
	LSD-Length	1024×576	0.3320	58 msec
	LSD-Number	1024×576	0.2361	56 msec

表 B.7: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part6）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LBP 64×36	LSD-Region	1920×1080	0.5237	139 msec
		1024×576	0.5451	59 msec
		512×288	0.5206	31 msec
	LSD-1Line	1024×576	0.5321	60 msec
		512×288	0.5885	31 msec
		256×144	0.5354	30 msec
	Randomized-Fern	1920×1080	0.5327	67 msec
		1024×576	0.5349	33 msec
		512×288	0.5352	30 msec
		256×144	0.5418	30 msec
		128×72	0.5392	30 msec
		64×36	0.5276	30 msec
		40×30	0.5116	30 msec
	RGB-Histogram	1920×1080	0.5980	31 msec
		1024×576	0.6001	31 msec
		512×288	0.6011	29 msec
		256×144	0.6000	29 msec
		128×72	0.5953	29 msec
	RGB-L2	256×144	0.5676	40 msec
		128×72	0.5663	30 msec
		64×36	0.5648	29 msec
		32×18	0.5460	29 msec
		16×9	0.4917	29 msec
	RGB-NCC	32×18	0.5408	89 msec
		16×9	0.4898	50 msec
	SURF-BoK	1024×576	0.6539	95 msec
		512×288	**0.6643**	45 msec
		256×144	0.6447	35 msec

表 B.8: 全段の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part1）

1st Filter	2nd Filter	3rd Filter		Recall	Processing Time (Average)
		Method	Image Size		
SURF-BoK 512×288	SURF-BoK 512×288	AKAZE-Matching	1024×576	0.2656	286 msec
			512×288	**0.2675**	172 msec
			256×144	0.2626	139 msec
			128×72	0.1789	92 msec
		Gray-L2	1024×576	0.2532	117 msec
			512×288	0.2504	117 msec
		Gray-NCC	256×144	0.2514	322 msec
			128×72	0.2518	238 msec
		Randomized-Fern	1920×1080	0.2455	128 msec
			1024×576	0.2450	144 msec
			512×288	0.2465	137 msec
			256×144	0.2462	129 msec
			128×72	0.2401	128 msec
			64×36	0.2353	127 msec
			40×30	0.2307	131 msec
		RGB-L2	512×288	0.2531	134 msec
		RGB-NCC	128×72	0.2500	309 msec
		SURF-Matching	512×288	0.2649	177 msec
			256×144	0.2670	182 msec
			128×72	0.2299	178 msec

表 B.9: 全段の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part2）

1st Filter	2nd Filter	3rd Filter		Recall	Processing Time (Average)
		Method	Image Size		
LSD-1Line 512×288	SURF-BoK 512×288	AKAZE-Matching	1024×576	0.2624	514 msec
			512×288	0.2658	315 msec
			256×144	0.2626	210 msec
			128×72	0.1765	108 msec
		Gray-L2	1024×576	0.2440	130 msec
			512×288	0.2439	112 msec
		Gray-NCC	256×144	0.2428	480 msec
			128×72	0.2433	450 msec
		Randomized-Fern	1920×1080	0.2376	152 msec
			1024×576	0.2387	157 msec
			512×288	0.2391	154 msec
			256×144	0.2371	155 msec
			128×72	0.2308	162 msec
			64×36	0.2237	170 msec
			40×30	0.2188	171 msec
		RGB-L2	512×288	0.2449	151 msec
		RGB-NCC	128×72	0.2439	484 msec
		SURF-Matching	512×288	0.2658	272 msec
			256×144	0.2633	242 msec
			128×72	0.2287	71 msec

表 B.10: 全段の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（ふげんデータセット Part3）

1st Filter	2nd Filter	3rd Filter		Recall	Processing Time (Average)
		Method	Image Size		
LBP 64×36	SURF-BoK 512×288	AKAZE-Matching	1024×576	0.2613	794 msec
			512×288	0.2651	548 msec
			256×144	0.2619	354 msec
			128×72	0.1758	206 msec
		Gray-L2	1024×576	0.2432	242 msec
			512×288	0.2436	186 msec
		Gray-NCC	256×144	0.2448	687 msec
			128×72	0.2425	744 msec
		Randomized-Fern	1920×1080	0.2361	266 msec
			1024×576	0.2383	287 msec
			512×288	0.2366	286 msec
			256×144	0.2348	279 msec
			128×72	0.2291	276 msec
			64×36	0.2229	292 msec
			40×30	0.2187	219 msec
		RGB-L2	512×288	0.2444	257 msec
		RGB-NCC	128×72	0.2412	693 msec
		SURF-Matching	512×288	0.2638	356 msec
			256×144	0.2614	322 msec
			128×72	0.2258	107 msec

付録 C 最適化の結果（7-Scenesデータセット）

表 C.1: 1 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット）

1st Filter		Recall	Processing Time (Average)
Method	Image Size		
CCV	160×120	0.5453	17 msec
	80×60	0.5290	17 msec
	40×30	0.4986	17 msec
Gray-Histogram	640×480	0.7906	12 msec
	320×240	0.7863	11 msec
	160×120	0.7876	11 msec
	80×60	0.7874	11 msec
LBP	80×60	**0.8747**	23 msec
	40×30	0.8595	23 msec
LSD-Direction	640×480	0.3938	28 msec
	320×240	0.4399	17 msec
LSD-Region	640×480	0.6798	27 msec
	320×240	0.6822	17 msec
LSD-1Line	640×480	0.7024	29 msec
	320×240	0.7413	18 msec
Randomized-Fern	640×480	0.7925	13 msec
	320×240	0.7871	7 msec
	160×120	0.7936	6 msec
	80×60	0.8046	6 msec
	40×30	0.8234	6 msec
RGB-Histogram	640×480	**0.8314**	15 msec
	320×240	0.8278	15 msec
	160×120	0.8289	14 msec
	80×60	0.8288	14 msec
SURF-BoK	640×480	0.8169	66 msec
	320×240	0.8518	33 msec
	160×120	**0.8859**	20 msec
	80×60	0.7943	16 msec

表 C.2: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part1）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
SURF-BoK 160×120	CCV	640×480	0.3945	21 msec
	Gray-Histogram	640×480	0.5870	21 msec
		320×240	0.5853	21 msec
		160×120	0.5858	21 msec
		80×60	0.5857	21 msec
	Gray-L2	320×240	0.6038	32 msec
		160×120	0.6023	21 msec
		80×60	0.6020	21 msec
		40×30	0.6037	21 msec
		20×15	0.5965	21 msec
	Gray-NCC	40×30	0.5975	69 msec
		20×15	0.5941	40 msec
	HOG	320×240	0.5270	48 msec
		160×120	0.5532	34 msec
	LBP	320×240	0.6249	23 msec
		160×120	0.6339	23 msec
		80×60	**0.6431**	23 msec
		40×30	0.6223	23 msec
	LSD-Angle	640×480	0.3628	30 msec
	LSD-2Lines	640×480	0.4028	31 msec
	LSD-Direction	640×480	0.3057	27 msec
		320×240	0.3340	24 msec
	LSD-Distance	640×480	0.3084	30 msec
	LSD-Length	640×480	0.3708	27 msec
	LSD-Number	640×480	0.2507	26 msec

表 C.3: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part2）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
SURF-BoK 160×120	LSD-Region	640×480	0.5118	28 msec
		320×240	0.5131	24 msec
	LSD-1Line	640×480	0.5420	27 msec
		320×240	0.5670	24 msec
		160×120	0.5046	23 msec
	Randomized-Fern	640×480	0.6141	24 msec
		320×240	0.6137	25 msec
		160×120	0.6134	23 msec
		80×60	0.6196	23 msec
		40×30	0.6284	24 msec
	RGB-Histogram	640×480	0.6114	22 msec
		320×240	0.6116	22 msec
		160×120	0.6090	21 msec
		80×60	0.6109	21 msec
	RGB-L2	320×240	0.6116	64 msec
		160×120	0.6120	28 msec
		80×60	0.6110	22 msec
		40×30	0.6094	22 msec
		20×15	0.6062	22 msec
	RGB-NCC	20×15	0.6057	66 msec
	SURF-BoK	640×480	0.6052	75 msec
		320×240	0.6269	29 msec
		160×120	**0.6449**	25 msec

表 C.4: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part3）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LBP 80×60	CCV	640×480	0.3968	27 msec
	Gray-Histogram	640×480	0.5879	28 msec
		320×240	0.5873	25 msec
		160×120	0.5870	25 msec
		80×60	0.5842	26 msec
	Gray-L2	320×240	0.5780	33 msec
		160×120	0.5799	26 msec
		80×60	0.5786	26 msec
		40×30	0.5778	26 msec
		20×15	0.5713	26 msec
	Gray-NCC	40×30	0.5785	67 msec
		20×15	0.5718	93 msec
	HOG	320×240	0.5084	70 msec
		160×120	0.5336	30 msec
	LBP	320×240	0.5958	25 msec
		160×120	0.6082	25 msec
		80×60	0.6201	25 msec
		40×30	0.5882	25 msec
	LSD-Angle	640×480	0.3521	38 msec
	LSD-2Lines	640×480	0.3909	39 msec
	LSD-Direction	640×480	0.2848	36 msec
		320×240	0.3097	29 msec
	LSD-Distance	640×480	0.2969	37 msec
	LSD-Length	640×480	0.3504	35 msec
	LSD-Number	640×480	0.2359	34 msec

表 C.5: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part4）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LBP 80×60	LSD-Region	640×480	0.5008	39 msec
		320×240	0.5091	29 msec
	LSD-1Line	640×480	0.5221	37 msec
		320×240	0.5533	30 msec
		160×120	0.4855	25 msec
	Randomized-Fern	640×480	0.6009	28 msec
		320×240	0.5986	29 msec
		160×120	0.6011	25 msec
		80×60	0.6111	25 msec
		40×30	0.6166	25 msec
	RGB-Histogram	640×480	0.6102	27 msec
		320×240	0.6081	28 msec
		160×120	0.6110	25 msec
		80×60	0.6090	25 msec
	RGB-L2	320×240	0.5911	65 msec
		160×120	0.5886	30 msec
		80×60	0.5913	25 msec
		40×30	0.5900	25 msec
		20×15	0.5835	25 msec
	RGB-NCC	20×15	0.5852	62 msec
	SURF-BoK	640×480	0.6042	131 msec
		320×240	0.6316	105 msec
		160×120	**0.6550**	30 msec

表 C.6: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part5）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LBP 80×60	CCV	640×480	0.3629	47 msec
	Gray-Histogram	640×480	0.5562	38 msec
		320×240	0.5566	31 msec
		160×120	0.5550	31 msec
		80×60	0.5546	32 msec
	Gray-L2	320×240	0.5721	49 msec
		160×120	0.5714	28 msec
		80×60	0.5705	32 msec
		40×30	0.5715	32 msec
		20×15	0.5643	32 msec
	Gray-NCC	40×30	0.5681	142 msec
		20×15	0.5602	52 msec
	HOG	320×240	0.5146	113 msec
		160×120	0.5437	55 msec
	LBP	320×240	0.5988	43 msec
		160×120	0.6106	41 msec
		80×60	0.6180	41 msec
		40×30	0.5973	42 msec
	LSD-Angle	640×480	0.3269	65 msec
	LSD-2Lines	640×480	0.3757	65 msec
	LSD-Direction	640×480	0.2913	60 msec
		320×240	0.3221	59 msec
	LSD-Distance	640×480	0.2891	63 msec
	LSD-Length	640×480	0.3375	60 msec
	LSD-Number	640×480	0.2162	59 msec

表 C.7: 1,2 段目の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part6）

1st Filter	2nd Filter		Recall	Processing Time (Average)
	Method	Image Size		
LBP 80×60	LSD-Region	640×480	0.4850	62 msec
		320×240	0.4903	60 msec
	LSD-1Line	640×480	0.5132	63 msec
		320×240	0.5422	60 msec
		160×120	0.4809	41 msec
	Randomized-Fern	640×480	0.6009	28 msec
		320×240	0.5986	29 msec
		160×120	0.6011	25 msec
		80×60	0.6111	25 msec
		40×30	0.6166	25 msec
	RGB-Histogram	640×480	0.5870	43 msec
		320×240	0.5871	52 msec
		160×120	0.5878	35 msec
		80×60	0.5858	35 msec
	RGB-L2	320×240	0.5788	86 msec
		160×120	0.5804	54 msec
		80×60	0.5820	37 msec
		40×30	0.5786	36 msec
		20×15	0.5729	35 msec
	RGB-NCC	20×15	0.5715	81 msec
	SURF-BoK	640×480	0.5821	365 msec
		320×240	0.6090	111 msec
		160×120	0.6291	40 msec

表 C.8: 全段の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part1）

1st Filter	2nd Filter	3rd Filter		Recall	Processing Time (Average)
		Method	Image Size		
LBP 80×60	SURF-BoK 160×120	AKAZE-Matching	640×480	**0.1938**	176 msec
			320×240	0.1891	80 msec
			160×120	0.1692	41 msec
			80×60	0.0217	28 msec
		Gray-L2	640×480	0.1826	29 msec
			320×240	0.1821	28 msec
		Gray-NCC	160×120	0.1810	75 msec
			80×60	0.1822	41 msec
		Randomized-Fern	640×480	0.1816	29 msec
			320×240	0.1811	28 msec
			160×120	0.1817	29 msec
			80×60	0.1838	28 msec
			40×30	0.1835	28 msec
		RGB-L2	320×240	0.1832	30 msec
		RGB-NCC	80×60	0.1827	253 msec
		SURF-Matching	640×480	**0.1937**	179 msec
			320×240	0.1936	147 msec
			160×120	0.1915	67 msec
			80×60	0.1657	41 msec

表 C.9: 全段の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part2）

1st Filter	2nd Filter	3rd Filter		Recall	Processing Time (Average)
		Method	Image Size		
LBP 80×60	SURF-BoK 160×120	AKAZE-Matching	640×480	0.1923	299 msec
			320×240	0.1883	174 msec
			160×120	0.1675	44 msec
			80×60	0.0211	36 msec
		Gray-L2	640×480	0.1839	41 msec
			320×240	0.1827	61 msec
		Gray-NCC	160×120	0.1828	294 msec
			80×60	0.1827	101 msec
		Randomized-Fern	640×480	0.5809	44 msec
			320×240	0.5780	50 msec
			160×120	0.5811	41 msec
			80×60	0.5896	40 msec
			40×30	0.5961	41 msec
		RGB-L2	320×240	0.1838	49 msec
		RGB-NCC	80×60	0.1819	708 msec
		SURF-Matching	640×480	0.1923	309 msec
			320×240	0.1933	185 msec
			160×120	0.1911	75 msec
			80×60	0.1648	41 msec

表 C.10: 全段の画像検索手法と特徴量算出元画像サイズの組み合わせの評価結果（7-Scenes データセット Part3）

1st Filter	2nd Filter	3rd Filter		Recall	Processing Time (Average)
		Method	Image Size		
SURF-BoK 160×120	LBP 80×60	AKAZE-Matching	640×480	0.1917	456 msec
			320×240	0.1883	325 msec
			160×120	0.1675	54 msec
			80×60	0.0221	33 msec
		Gray-L2	640×480	0.1801	38 msec
			320×240	0.1800	81 msec
		Gray-NCC	160×120	0.1812	698 msec
			80×60	0.1809	118 msec
		Randomized-Fern	640×480	0.1804	46 msec
			320×240	0.1799	47 msec
			160×120	0.1803	39 msec
			80×60	0.1815	40 msec
			40×30	0.1825	40 msec
		RGB-L2	320×240	0.1815	53 msec
		RGB-NCC	80×60	0.1816	1,342 msec
		SURF-Matching	640×480	0.1923	419 msec
			320×240	0.1937	248 msec
			160×120	0.1909	95 msec
			80×60	0.0647	40 msec