The International Journal of Robotics Research
XX(X):1–??
©The Author(s) 2019
Reprints and permission:
sagepub.co.uk/journalsPermissions.nav
DOI: 10.1177/ToBeAssigned
www.sagepub.com/
SAGE

Distributed and Consistent Multi-Image Feature Matching via QuickMatch

Zachary Serlin[1], Guang Yang[2], Brandon Sookraj[1], Calin Belta[1], and Roberto Tron[1]

Abstract
In this work we consider the multi-image object matching problem in distributed networks of robots. Multi-image feature matching is a keystone of many applications, including simultaneous localization and mapping, homography, object detection, and structure from motion. We first review the QuickMatch algorithm for multi-image feature matching. We then present NetMatch, an algorithm for distributing sets of features across computational units (agents) that largely preserves feature match quality and minimizes communication between agents (avoiding, in particular, the need of flooding all data to all agents). Finally, we present an experimental application of both QuickMatch and NetMatch on an object matching test with low quality images. The QuickMatch and NetMatch algorithms are compared to other standard matching algorithms in terms of preservation of match consistency. Our experiments show that QuickMatch and Netmatch can scale to larger numbers of images and features, and match more accurately than standard techniques.

Keywords
Computer vision, Feature matching, Object matching, Distributed matching, Multi-image matching

Introduction

Matching is a fundamental operation common in robotics and computer vision algorithms such as Structure from Motion (SfM), Simultaneous Localization and Mapping (SLAM), and object detection and tracking. Matching in these domains is typically performed on feature descriptors (such as SIFT, ORB, and SURF by Lowe (2004), Rublee (2011), and Bay (2008) respectively) which extract discriminative characteristics from high dimensional data (i.e. an image). At a fundamental level, these features are used to associate unique objects in the universe to their appearance in multiple views. These different views might be acquired by a geographically dispersed camera or robotic network, which can produce large amounts of data. It is therefore important to have the ability to consistent match features across multiple images (i.e, perform multi-image matching) efficiently, consistently, accurately, and, ideally, in a distributed manner. For instance, match quality is key in settings such as SfM and SLAM, because the quality of the reconstruction is a direct result of how well the features match across images. The speed of image matching is extremely critical in applications such as real-time object detection. Matching consistency of features across many images is also critical for SfM and SLAM since feature points must be tracked not just between images, but across multi-image sequences for the best results. The pervasiveness of graphical processing units (GPUs), networked systems, distributed robotics, cloud computing, and multi-core processors, has made the distributability of computer vision solutions relevant to their applicability. These tools have also shown great advantages in speed and bandwidth for other classic algorithms (Garcia (2010); Warn (2009)). To this end, we introduce a novel version of our matching algorithm that distributes the computational load of the feature matching problem over multiple agents; the resulting algorithm can handle considerably more features, with only a slight performance loss, while minimizing the amount of necessary communications across nodes.

Problem overview and contributions

We propose a solution to the following problem: given a set of images taken from a team of robots or camera network, match unique object features in a distributed manner, as they enter and exit the images from multiple perspectives. This problem arises often in object detection, localization, and tracking (Bradski (2000); Cunningham (2012); Zhou (2015)), homography estimation (Szeliski (2010)), Structure from Motion (Hartley (2017)), and formation control (Montijano (2016)). Solutions to this problem are traditionally computationally complex, and often mismatch features when considering more than two images (Bradski (2000); Lowe (2004)), by either missing matches between two or more views of the same entity in the universe, or by introducing associations between separate universe entities. Multi-image correspondences allow for greater match reliability and a more accurate representation of objects in the universe. Our proposed solution leverages a relatively recent algorithm, QuickMatch (Tron (2017)), to quickly and reliably discover correspondences across multiple images. The experiments presented in this paper

[1]Boston University, Department of Mechanical Engineering, USA
[2] Boston University, Division of Systems Engineering, USA

Corresponding author:
Zachary Serlin, Boston University, Department of Mechanical Engineering, 110 Cummington Mall, Boston, Massachusetts, USA.
Email: zserlin@bu.edu

benchmark QuickMatch's performance by tracking a target object and estimating its trajectory under realistic conditions (i.e. over images with clutter, repeated structures, and poor image quality). We then propose NetMatch, and extension of QuickMatch to a distributed computation setting, which largely preserves match quality while minimizing communication across agents.

Related Work

Feature matching is a basic step in many computer vision algorithms. *Pairwise matching* is the classical approach to this task, where feature descriptors between two images are compared based on a distance metric (e.g. Euclidean or Manhattan distance), and declared a match if this distance is below some threshold (Lowe (2004); Vedaldi and Fulkerson (2008)). Pairwise matching is often posed as a nearest neighbor search problem (Garcia (2010); Dollar and Zitnick (2013)). This method is used in two standard algorithms, *Brute Force* (BF) matching (Bradski (2000)), and *Fast Library for Approximate Nearest Neighbors* (FLANN) matching (Muja (2014)). Pairwise matching has difficulties consistently matching entities with repetitive structure or similar appearance (e.g. adjacent windows) because the distance metric alone does not consider the *distinctiveness* (smallest distance between features from the same image) of the features. Including distinctiveness of features during matching has been shown to be beneficial (Lowe (2004)). For multi-image matching, pairwise matches scale poorly with the number of images and across multiple images, match correspondences often do not belong to the same ground truth object (and hence link together correspondences between different entities). *Graph matching* has also been used for pairwise matching. This approach attempts to match vertices (features) and edges (matches) simultaneously to determine better pairwise matches shown by Yan and Cho (2015) and Yan and Wang (2015), but it cannot handle the multi-image setting for the same reasons as above.

Beyond pairwise matching, a number of other approaches exist for multi-image matching (where multiple images are directly considered) that are based on optimization, graphs, and clustering. *Optimization* based approaches are based upon non-convex problems where optimization constraints must often be relaxed to reliably obtain solutions (Oliveira (2005); Yan and Cho (2015); Chen (2014); Leonardos (2017); Pachauri (2013)). Moreover, these approaches typically require *a priori* knowledge the number of objects, which is often not available, and they do not consider distinctiveness of the features. Approaches based on the explicit consideration of *cycles in graphs* are early predecessors to the QuickMatch algorithm and have largely been used to remove inconsistent matches as shown by Huang (2013). Cycle consistency has also recently been used in a distributed manner to perform matching by Hu (2018) and Aragues (2011). *Clustering* can be cast as finding clusters of similar features. Algorithms such as k-means (Mackay (2003)) and spectral clustering (Ng (2002)) have been explored to this end, but also often require a predefined number of objects, and do not consider that a unique feature only occurs once in an image, meaning repeat structures are also often called a match.

Pairwise matching can also be approached as an unsupervised learning problem, and therefore there are a number of distributed approaches to the classic algorithms, especially for the k-nearest neighbor problem. The k-nearest neighbor problem has been extensively explored in settings such as friend suggestion on Facebook, image classification, and recommendation systems. For this reason, many parallelized approaches have been introduced by Johnson (2019), Andre (2017), and Gieseke (2014) and others. In a feature matching setting, a CUDA based distributed approach to the BF algorithm has been proposed that is approximately 100x faster than its centralized counterpart. Beyond matching, parallel computing has also been applied to feature extraction algorithms such as SIFT (Warn (2009)), offering speed increases along the entire feature matching pipeline. These approaches still do not consider the multi-image matching problem, however they offer a strong motivation for distributing existing multi-image matching techniques. Machine learning techniques have also been used to extract more meaningful features that are characteristic of objects in general, which adds a layer of semantic understanding to the multi-image matching problem (Hariharan (2015); Wang (2018); Novotny (2017)).

QuickMatch and NetMatch are grounded in *density-based clustering* algorithms (see work by Fukunaga (1975); Vedaldi and Soatto (2008); Ester (1996); Hu (2017) for examples), which find clusters by estimating a non-parametric density distribution of data (Parzen (1962); Rosenblatt (1956)). These approaches do not require prior knowledge of the number or shape of clusters, and can be modified to include feature distinctiveness by construction.

This article is an extension of previous work presented at the International Symposium of Experimental Robotics Serlin (2018) and the International Conference on Computer Vision Tron (2017). The primary extensions from this prior work are:

- We introduce a method for distributing the matching problem across multiple agents based on a feature space partition. While this method is agnostic to the choice of the local clustering method, in this paper we propose NetMatch, which extends QuickMatch to the distributed setting.
- We experimentally demonstrate that QuickMatch and NetMatch outperform traditional methods in a realistic robotic application (tracking). Moreover, we show that the performance of NetMatch is comparable to its centralized counterpart, while using minimal network communications.

Outline

The remainder of this paper is structured as follows. We begin by introducing preliminary concepts for features, feature space partitions, and multi-image matching. We then present the multi-image matching problem. Next, we present first the centralized, and then decentralize solutions to the presented problem, and we detail the theory of the approach preformed in the experiment. We then present simulations to compare the centralized and distributed solutions. Finally, we present the experiment preformed and its results.

Preliminaries

Images and Features

We consider a set of images $\mathcal{I} = \{1, \ldots, i, \ldots, N\}$. From each image $i \in \mathcal{I}$, we extract a set of features K_i with a single feature vector denoted $x_{ik} \in \mathbb{R}^F$, where F is the dimension of the feature space (eg. for SIFT, $F = 128$), i is the image index, and $k \in K_i$ is the feature index in that image. We denote the set of all features as $K_\mathcal{I}$ and the cardinality of a set as $|K_\mathcal{I}|$.

Agents and Feature Space Partitions

In the distributed setting, we consider a set of computational units, or agents, $A = \{1, 2, \ldots, a, \ldots, m\}$ where $a \in \mathcal{A}$ denotes a single agent. The feature space is denoted as $\mathcal{Y} = \mathbb{R}^F$, and is partitioned into m convex Voronoi tessellation subspaces, where each space is assigned to a single agent as $\mathcal{Y}_a \subseteq \mathcal{Y}$ with $\bigcup\limits_{a \in A} \mathcal{Y}_a = \mathcal{Y}$. The Voronoi partition seed of each $\mathcal{Y}_a$ is given as $P_a \in \mathbb{R}^F$. We define a labeling function $\ell : x_{ik} \mapsto a$ that maps a feature x_{ik} to a given agent depending on which $\mathcal{Y}_a$ it exists in. This labeling function has a set-valued inverse, $\ell^{-1} : a \mapsto \{x_{ik}\}$, which returns the set of features contained in any given agent partition.

Multi-Image Matching

The multi-image matching problem presented here is summarized from Tron (2017), and is posed in the light of both cycle consistency in a graph of features and clustering. We refer the reader to the original paper for detailed proofs regarding the equivalence between thees two interpretations.

We begin by assuming that the feature space $\mathbb{R}^F$ has a distance metric defined as $d(x_{ik}, x_{i'k'}) \mapsto \mathbb{R} \geq 0$ between two features. We denote the set of all features $\mathcal{X} = \{x_{ik}\}_{k \in 1 \ldots N}^{i \in \mathcal{I}}$. When two features are a match, i.e., $x_{i_1k_1} \to x_{i_2k_2}$, $i_1 \neq i_2$, we mean that the two features represent the same entity. Based on this type of correspondence, we define a directed graph of the matches as $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ with $\mathcal{V} = \{x_{ik}\}$ and $\mathcal{E} \subset \mathcal{V} \times \mathcal{V}$ is the set of matches $x_{i_1k_1} \to x_{i_2k_2} \in \mathcal{E}$.

We define multi-image matches as subsets of the features in $\mathcal{X}$. We therefore define, given a subset of features $\mathcal{C} \subset \mathcal{V}$, a subgraph of $\mathcal{G}$ restricted to $\mathcal{C}$ as $\mathcal{G}|_\mathcal{C} = (\mathcal{C}, \mathcal{E}')$ with $\mathcal{E}' = \{x_{i_1k_1} \to x_{i_2k_2} \in \mathcal{E} : x_{i_1k_1}, x_{i_2k_2} \in \mathcal{C}\}$. A graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is connected if for any $x_{i_1k_1}, x_{i_nk_n} \in \mathcal{V}$ there exist a sequence $x_{i_1k_1}, x_{i_2k_2}, \ldots, x_{i_nk_n}$, called a path, such that $x_{i_j,k_j} \to x_{i_{j+1},k_{j+1}} \in \mathcal{E}\ \forall j \in \{1, \ldots, n-1\}$ and a cycle is a path where the start and end features are the same, or $x_{i_1k_1} = x_{i_nk_n}$. We define a clique as a graph $\mathcal{G}$ with $x_{i_1k_1} \to x_{i_2k_2} \in \mathcal{E}\ \forall x_{i_1k_1}, x_{i_nk_n} \in \mathcal{V}$.

A set of multi-image matches is a set $\mathcal{M} = \{\mathcal{C}_c\}$ of clusters $\mathcal{C}_c = \{x_{i_1k_1}, \ldots, x_{i_nk_n}\}$, where each one corresponds to a single entity in the universe, such that

(C1) $\mathcal{M}$ is a partition of $\mathcal{X}$ (i.e., each feature x_{ik} appears exactly in one set $\mathcal{C}_c$);

(C2) Each set $\mathcal{C}_c$ has at most one feature per image;

(C3) There is an induced directed graph $\mathcal{G}_\mathcal{M} = (\mathcal{X}, \mathcal{E}_\mathcal{M})$ of pairwise mathces such that, for any $\mathcal{C}_c \in \mathcal{M}$, the subgraph $\mathcal{G}_\mathcal{M}|\mathcal{C}_c$ is a clique (i.e., $\mathcal{G}_\mathcal{M}$ is a union of cliques).

Conditions (C1) and (C2) ensure that the same entity cannot appear in multiple clusters, and an entity cannot appear more than once per image. Condition (C3) requires that features from the same cluster always match.

Given the above statements, three properties are implied about $\mathcal{G}_\mathcal{M}$,

(P1) Symmetry: $x_{i_1k_1} \to x_{i_2k_2} \in \mathcal{E}_\mathcal{M}$ implies $x_{i_2k_2} \to x_{i_1k_1} \in \mathcal{E}_\mathcal{M}$;

(P2) Cycle Constraint: Given a path $x_{i_1k_1}, \ldots, x_{i_nk_n}$ in $\mathcal{G}_\mathcal{M}$, having $i_1 = i_n$ implies $k_1 = k_n$ meaning the path is a cycle;

(P3) Single match: A feature cannot correspond to two different features in another image, meaning that $x_{i_1k_1} \to x_{i_2k_2}$ and $x_{i_1k_1} \to x_{i_2k'_2}$ belong to $\mathcal{G}_\mathcal{M}$ then $k_2 = k'_2$.

Problem Statement

Given a set of images $\mathcal{I} = \{1, 2, \ldots, i, \ldots, N\}$ and a set of K_i feature vectors, x_{ik}, extracted from each image, determine matches $(x_{i_1k_1} \leftrightarrow x_{i_2k_2} : i_1 \neq i_2)$ between features from separate images, such that matched features represent the same point in the scene and build a set of multi-image matches $\mathcal{M}$.

Multi-image Matching via QuickMatch and NetMatch

We present both a centralized and decentralized solution to the stated problem. It is important to note that this problem cannot be solved with pairwise matches alone, as they are not capable of considering cliques directly; specifically, they cannot prevent the violation of (C3). The centralized solution introduces the QuickMatch algorithm and demonstrates its effectiveness with a two-stage, offline, centralized implementation on a system of distributed ground robots and a central computer. Features are first extracted using off-the-shelf feature extraction methods (SIFT), and the features are then matched using the QuickMatch algorithm to find a given reference object. These matches are used to perform homography estimation between the reference object and the camera network to generate target trajectories. The decentralized solution, NetMatch, then augments the QuickMatch algorithm with a framework for distributing the features across multiple computational units while maintaining match consistency. The same framework could be used for other matching algorithms as well, however in this paper we compare its results with the centralized solution to demonstrate near equivalence in terms of accuracy.

Feature Extraction

Feature extraction aims to find and describe representative points from high dimensional data, such as an image (Bay (2008); Rublee (2011); Lowe (2004); Zhou (2015)). Features themselves are also high dimensional vectors but typically of much lower dimension then the original data. In this experiment, the *Scale Invariant Feature Transform* (SIFT) feature is used, which extracts a set K_i of 128-dimensional vectors that represent the appearance of each feature point. See Lowe (2004), and Vedaldi and Fulkerson (2008) for more details on this standard feature extraction algorithm. Other feature types can be used, and we also tested with Oriented FAST and Rotated BRIEF (ORB) features and Speeded-Up Robust Features (SURF), however SIFT was qualitatively the most reliable.

QuickMatch

The QuickMatch algorithm begins by calculating the distance between all features (here we use the Euclidean distance). For each image, the minimum distance, σ_i, between any two features is used as the distinctiveness of features for that image. As defined above, x_{ik} is a point in the high dimensional feature space. The feature density $D(x_{ik})$ is then calculated for each point using the formula

$$D(x) = \sum_{ik \in K_{\mathcal{I}}} h(x, x_{ik}; \sigma_i), \tag{1}$$

$$h(x_1, x_2; \sigma_i) = \exp(-\frac{\|x_1 - x_2\|}{2\sigma_i^2}), \tag{2}$$

$$\sigma_i = \min_{\forall k, i=i'} d(x_{ik}, x_{i'k'}), \tag{3}$$

where h is a non-negative kernel function, and σ is the distinctiveness of image i. With this feature density, the features are organized into a tree structure, with parent nodes being the nearest feature with a higher density,

$$T : parent(x_{ik}) = \arg\min_{x_{i'k'} \in J} d(x_{ik}, x_{i'k'}), \tag{4}$$

$$J = \{i'k' \in K_{\mathcal{I}} : k \neq k', D(x_{i'k'}) > D(x_{ik})\}. \tag{5}$$

Edges are directed to parents along the gradient of feature density, and ultimately toward the center of the parent cluster or to another distant cluster. Once the tree has been constructed, edges are broken if either of two criteria are met (based on (C1)-(C3) above);

1. If parent and child groups have nodes from the same image (i.e. $i_1 = i_2$).
2. If the edge is larger than a user defined threshold (ρ) times the distinctiveness σ_i (i.e. $d(x_{ik}, parent(x_{ik})) \geq \rho\sigma_i$).

This method results in a forest of trees ($\mathcal{M}$), where each tree is a cluster (C_c) representing a unique entity in the universe. In practice, each tree represents a point that is common among images, meaning the algorithm discovers common features among very similar objects. Feature discovery will be explored further in the results section, where groups of matching points are employed for object detection and homography transformations.

NetMatch

The centralized QuickMatch algorithm is limited in the number of features it can process, because a single computational unit must handle all of the processing. This limitation motivates distributing the computation across many computational nodes in order to increase the number of features considered. A similar situation occurs when the data is acquired and naturally distributed across different nodes. To this end, we introduce the NetMatch algorithm, which distributes feature over a network of computational nodes such that each can run QuickMatch (or another equivalent algorithm) on only a subset of the feature space.

Algorithm 1 QuickMatch

```
Input: $K, \rho$
Output: Clusters $\mathcal{C}_c$ ($\mathcal{M}$)

for all $x_{ik}, x_{i'k'}$ do
    Compute $h(x_{ik}, x_{i'k'}, \sigma)$
for all $x_{ik}$ do
    Compute $D(x_{ik})$
for all $x_{ik}$ do
    Compute $parent(x_{ik})$ to build $\mathcal{T}$
for edges in $T$ do                              ▷ From shortest to longest
    $x_{ik} \in \mathcal{C}_c, x_{i'k'} \in \mathcal{C}_{c'}$ are ends of edge
    if $\{i\} \in \mathcal{C}_c \cap \{i\} \in \mathcal{C}_{c'} = \emptyset$ and $d(x_{ik}, x_{i'k'}) \leq \rho \arg\min_{\sigma}(\mathcal{C}_c, \mathcal{C}_{c'})$ then
        Merge $\mathcal{C}_c, \mathcal{C}_{c'}$
    else
        Remove edge
return $\mathcal{M}$
```

NetMatch follows three steps:

1. Each node transmits its computed features in such a way that one node receives all the features belonging to a given partition of the feature space; for instance, in a one-dimensional feature space, if a node is assigned the partition element $[1, 5]$, and another node had a feature $x = 3$, the feature would be transmitted from the latter to the former.
2. Each node performs QuickMatch independently on all received features in its assigned partition, stopping, by default, at the tree-building phase (i.e., without breaking the trees into clusters).
3. Computational nodes identify clusters falling near the edges of the partition. We refer to such clusters as *contested*.
4. If necessary, the contested are transferred to other nodes for re-processing.

At a high level, each node needs to process only a subset of the entire dataset, and, more importantly, each point is typically transferred only once, without having to flood the entire network with the entire dataset (additional transmissions are required only for edge cases, which are usually a minority, see also Table 1 for a quantitative analysis in our experiments). We next present the details of each step.

Feature Space Partitioning The goal of partitioning the feature space is to split the computation load of QuickMatch among m agents, ultimately allowing for more features to be matched, while approximately maintaining multi-image match quality. We begin by partitioning the feature space with the k-means algorithm and send features to their nearest partition center for processing. This initial partition can split a cluster between multiple partitions. Finding and merging these split clusters will be the focus of the following sections.

We modify the labeling function presented in the preliminary section to denote the round being considered as $\ell(x_{ik}, r) \mapsto a$ and $\ell^{-1}(a, r) \mapsto \{x_{ik}\}$. In practice we consider only $r = \{0, 1\}$, where 0 is the initial partitioning, and 1 is the labeling after partition reassignment because only one round is required by the NetMatch algorithm. The algorithm begins with a naive partitioning of the features

$$\ell(x_{ik}, 0) = \arg\min_{a \in A} d(x_{ik}, P_a) \; \forall x_{ik}, ik \in K_{\mathcal{I}}, \tag{6}$$

where the set of Voronoi seeds $P_A = \{P_a\}_{a \in A}$ can be found using, for instance, an unsupervised k-means algorithm (MacQueen (1967)), either on a training set, or on the actual data with a distributed version of k-means (the latter, under our all-to-all communication model, it can be implemented with a small amount of communications and, more importantly, without sharing all the data across all the nodes).

As an alternative, in larger data sets, and in a more distributed setting, the choice of P_A can be done at random (using a common seed for a pseudorandom generator, all of the nodes can agree on the same set of P_A without explicit communication); however, the random partition seeding procedure can have an effect on the overall match quality (see also Figure 6 for a quantitative analysis with our experimental setup).

Example 1: Given a set of features shown in Figure 1 (in gray), we generate three Voronoi seeds, P_0, P_1, P_2 (in blue) as shown. The Voronoi partition boundary is represented by the dashed lines. □

Tree Building Once P_A is determined, we send features to their respective agents based on $\ell(x_{ik}, 0)$, and at each agent. We then calculate the density of each feature as in Equation 1; unlike QuickMatch however, we propose to explicitly require that use of a density kernel with finite support, in order to limit interaction of density across the partition boundaries. Specifically, in our implementation we use a truncated quadratic kernel,

$$h(x_1, x_2; \sigma) = \begin{cases} -\frac{(\|x_1 - x_2\|)^2}{\sigma} + 1 & \|x_1 - x_2\| < \sigma. \\ 0 & \text{otherwise,} \end{cases} \tag{7}$$

Once the density is calculated, we build a tree T_a for each agent $a \in A$ according to

$$T_a : parent(x_{ik}) = \arg\min_{x_{i'k'} \in J} d(x_{ik}, x_{i'k'}), \tag{8}$$

$$J = \{i'k' : k \neq k', D(x_{i'k'}) > D(x_{ik}), x_{ik} \in \mathcal{Y}_a\}. \tag{9}$$

Similarly to the distinctiveness in QuickMatch, we now introduce an agent-specific distinctiveness metric, defined as

$$\sigma_a = \max_{p \in K_{\mathcal{I}} \cap \ell^{-1}(a)} (\sigma_p) \; \forall i \in \ell^{-1}(a, 0), \tag{10}$$

where

$$\sigma_p = d(x_p, parent(x_p)) \tag{11}$$

(in practice, p indexes the features in a given partition).

We next use $\{\sigma_a\}$ to determine if individual features may belong to clusters split by the artificial boundaries of the Voronoi partition, as detailed in the next step.

Note (NetMatch Lite): If the NetMatch algorithm is stopped at this point, and the trees built (T_a) are broken based on the rules in QuickMatch above, this algorithm is denoted NetMatch Lite. We use this algorithm in our experiments to benchmark the improvements gained by evaluating contested features (i.e., the next steps) in the full NetMatch algorithm. The NetMatch Lite algorithm can be considered as naively running k-means to partition the feature space, and then running QuickMatch individually in each partition.

Example 1 Cont.: With the features (in gray) in Figure 1, three trees are constructed (one in each partition) based on the density of features. This means, in this example, the features nearest the seed points would have a higher density and would therefore be closer to the root of the tree. □

Partition Boundary Distance To find the minimum distance between a feature and the boundaries that are generated by the Voronoi partition, we formulate the following problem: Consider a set of root nodes $\{P_0, ..., P_m\}$ and their corresponding regions $\{\mathcal{Y}_0, ..., \mathcal{Y}_m\}$, given a test node $x_t \in \mathcal{Y}_t$ and evaluation node $P_e \in \mathcal{Y}_e$, where $\mathcal{Y}_e \subset \mathcal{Y} \setminus \mathcal{Y}_t$, find the minimum distance d_{min}, and the corresponding hyper-plane, to x_t. A graphical illustration of this problem can be found below in Figure 1.

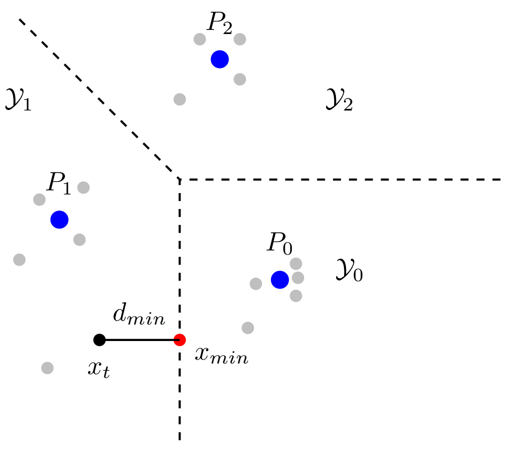


Figure 1. Illustration of Voronoi boundary distance problem.

Given the test feature node x_t, its current root node P_t and the root node that we want to evaluate P_e, we formulate a *Quadratic Program* (QP) as the following:

$$\begin{aligned} \min_{x \in \mathbb{R}^F} \quad & (x_t - x)^T (x_t - x) \\ \text{s.t.} \quad & (\frac{P_e - P_t}{\|P_e - P_t\|})^T (x - P_t) - \frac{d(P_t, P_e)}{2} \geq 0, \end{aligned} \tag{12}$$

where $d(P_t, P_e)$ is the pairwise distance between P_e and P_t. We obtain the closet point x_{min} on the Voronoi boundary that has the minimum distance to x_t. By iterating through all possible root nodes P_e, $\forall e \neq t$, we find the global minimum distance and its corresponding root node and its partition index

$$d_{ika} = d(x_{ika}, x_{min}). \tag{13}$$

The boundary distance calculation could be simplified by considering only the partition boundary between the two nearest points in P_A since the boundary is a Voronoi partition (however, this optimization is not present in our current implementation for the experiments below).

Example 1 Cont.: Feature x_t in Figure 1, is singled out to determine its minimum distance to a boundary. That distance, d_{min} is shown, along with the nearest point on the boundary x_{min}, which is used for computing d_{ika}. □

Contested Feature Re-assignment Once we are able to determine the distance between a feature and the partition boundary, we can then determine the contested features in each region. These features are the ones that may require reassignment to another partition's tree and are in danger of being mismatched due to the feature space partitioning splitting their cluster. This contested set of features S_a is defined as

$$S_a = \{a' \in A \setminus a : (d_{ika} + d_{aa'}) < \sigma_p\} \quad \forall x_{i,k} \in \ell^{-1}(a,r)\ \forall a \in A, \tag{14}$$

where

$$d_{aa'} = \min_{x_{ik} \in \ell^{-1}(a',0)} d_{ika'} \tag{15}$$

$$d_{ika'} = d(x_{ika'}, \mathcal{Y}_a) \quad a' \neq \ell(x_{ik}, 0) \tag{16}$$

where d_{ika} is the distance between each feature x_{ika} and the boundary of the partition of a', and $d_{aa'}$ is the minimum distance between the feature in a' and the $\mathcal{Y}_a$ boundary.

Intuitively, if the inequality in Equation 14 is true, there could be a feature in a' that could belong to the same cluster as x_{ik}. Note that this test is conservative, because it considers a worst-case analysis, but it also requires only a single value $d_{aa'}$ be communicated between agents. Figure 2 illustrates an example of this check; in this case, x_{ika} is not contested with a' because the inequality in Equation 14 is false (i.e. $(d_{ika} + d_{aa'}) \geq \sigma_p$).

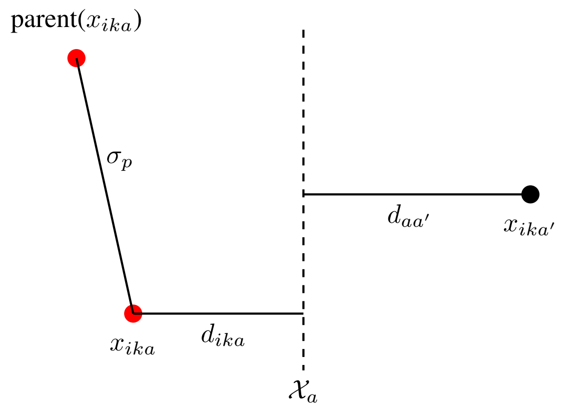


Figure 2. Example of comparison to determine if a point is contested.

Example 1 Cont.: For each feature, we check the inequality $(d_{ika} + d_{aa'}) \geq \sigma_p$, which indicates if there could be a nearer point on the other side of the boundary than to the feature's current parent feature. This comparison is illustrated in Figure 2. □

Once the contested features are determined, we break the agent tree into a forest of trees (clusters) in the same manner as QuickMatch, except the distinctiveness σ_i is calculated over only the features in the agent's own partition.

With the clusters formed, each agent then determines which clusters have contested points. Formally, we check $S_a \cap C_c \neq \emptyset$ (i.e. the intersection between a cluster and the contested points is not empty). For each cluster with a contested point we determine the minimum partition index among the contested points, and then send the cluster to that agent now denoted a'.

Upon arrival at a', that agent checks if

$$(\arg\min_{x_{ik} \in a'} d(x_{ik}, x_{i'k'})\ \forall x_{i'k'} \in C_c) \in C_{a'} \tag{17}$$

where C_c is the newly acquired cluster. If the nearest point to $C_c \in \mathcal{Y}_{a'}$ is also contested, both C_c and the cluster containing that nearest point from Equation 17 ($C_{c'}$) are sent to the lowest partition index if that index is lower than that agent's index (i.e. clusters are only transferred to agents with a lower contested partition index). This lower index requirement prevents switching incomplete clusters repeatedly between agents. This process is performed by decreasing agent index, reducing communication requirements, and the transfer of clusters to at most once per agent.

Example 1 Cont.: Once the partition tree is broken into a forest, for each new cluster tree we check if any features in that cluster violated the $(d_{ika} + d_{aa'}) \geq \sigma_p$ inequality. If any feature has, the cluster is marked for transfer and sent to the nearest agent of lower index. Each agent will do the same, meaning the transferred features can only go to a lower-indexed agent.

Upon receiving a new cluster, an agent checks if the cluster contains any points that it would consider contested. If so, the agent sends the whole cluster to the nearest agent of lower index, if one exists. □

The final step in reassignment is to rerun the tree building and breaking routines from above to reform all of the clusters. Although this is computationally intensive, it does not require any further inter-agent communication. It was also found to perform better than only reassigning the contested points to the appropriate tree and breaking it accordingly. For a simple example of the complete NetMatch framework, see Figure 10 in Appendix 1.

Simulations

This paper looks to compare primarily QuickMatch with NetMatch in simulation, and with off-the-shelf matching tools in an experimental setting. For a detailed comparison of the performance of QuickMatch to other state-of-the-art matching methods (particularly multi-image matching methods), see Tron (2017).

We begin with an illustrative example of NetMatch on a synthetic 2 dimensional data set shown in Figures 3 and 8. Our test case includes 4 agents, and 250 feature points. There are 25 underlying clusters, generated with random Gaussian distributions around 25 evenly spaced points, each

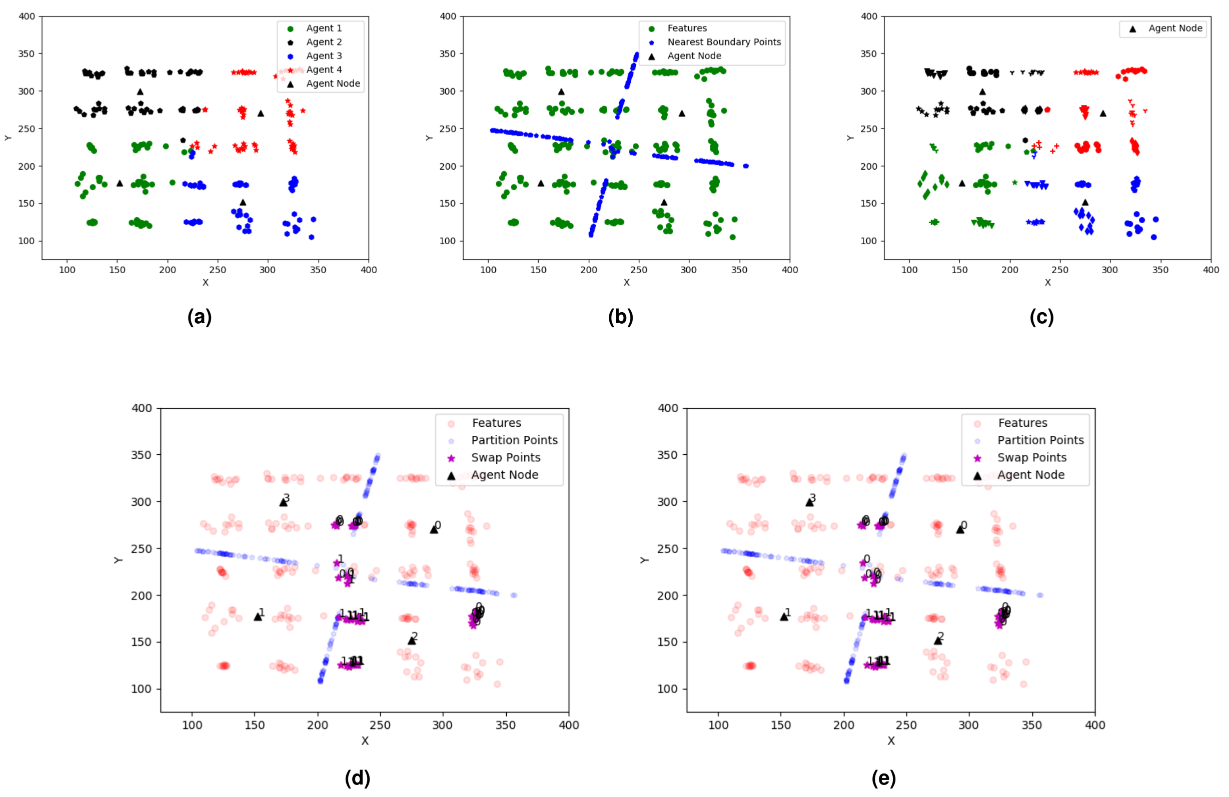


Figure 3. (a) Synthetic data set features with naive k-means partition assignments. (b) Synthetic data set with x_{min} boundary points plotted to highlight partition. (c) QuickMatch solution for each agent with k-means partition. (d) Initial proposed feature switches of contested clusters. Note the values on the highlighted features denote the proposed a' partition index. (e) Final proposed switch assignments after partitioning section of NetMatch is complete.

with 10 sample features. Figure 3 illustrates many of the steps in Algorithm 2. Figure 3 (a) shows the $\ell^{-1}(a, 0)$ labels for all of the agents. The Voronoi partition seeds for the partition are shown as triangles. Note that the clusters along the boundaries have features of different colors, meaning these clusters would be split and improperly matched with just the naive partitioning approach.

The partition generated by the Voronoi seeds can be seen in Figure 3 (b). Here, the x_{min} point is plotted for each feature. It can be seen that at least three of the clusters are split by this partition. This is further shown in Figure 3 (c), which shows the result of QuickMatch being run on each agent partition individually. Most notably, the central cluster is split into three clusters. Each marker style represents a different cluster membership (i.e. points with the same marker belong to the same cluster).

Figure 3 (d) shows the clusters staged for initial agent switching. Each feature that is switched has a label of where it is being sent. Note that the central cluster has multiple labels, meaning even after the switch, the cluster will be segmented. Also, note that whole clusters are staged for transfer, and that the contested region is very conservative, since clusters even somewhat far from the boundary are being swapped. After the agents are switched, the agents check if the switched clusters are nearest to other contested points. Figure 3 (e) shows the reassignment of the clusters after this check is performed. Note that after the first swap, the points in the central cluster are all moving toward a_0. This highlights one drawback to this approach, if the boundaries are drawn such that higher index agents have many contested clusters, the lower index agents end up getting assigned more features.

Experiments

Homography and Localization

Homography is a projective transformation between two perspective images of a planar scene that can also be used to determine the relative pose of an object with respect to a given reference image. The study of homography and localization of images and objects is textbook material; however, a brief overview is provided below as this process is used to experimentally test QuickMatch's performance. More information on both homography and object localization can be found in Hartley (2017) and Sankaranarayanan (2008).

Given a the image coordinates $\widetilde{x}$ (expressed in homogeneous coordinates) of a point belonging to a planar surface as seen in a reference view, the image of the same point in a novel view can be found given the homography matrix $\overline{H}$ as $\overline{H}\widetilde{x}$ (again, expressed using homogeneous coordinates). The $\overline{H}$ matrix can be estimated with a set of known relative points (or matched features) between two views. To improve the estimate of $\overline{H}$, random sample consensus (RANSAC) is used to remove match outliers by randomly sampling the matches, finding a fit of the data, and then removing any matches that fall outside of a user defined region (see Szeliski (2010)).

Given $\overline{H}$, it is also possible to recover the relative pose between the two views; this can then be used to indirectly localize (recover the translation and rotation) of an approximately planar object in a relative coordinate system up to a distance scale factor, as shown in Figure 4 (a). Given some known size of the target object (e.g., height), the scale ambiguity can be resolved, recovering the full object relative position. In our applications, where each image is taken together with images from other cameras in the network, and where the pose of each camera is known, the target object can be accurately positioned in the global reference frame, allowing for the generation of a target's trajectory (e.g., Figure 7).

Since localization using homography is limited to approximately planar surfaces, multiple reference images (as used here) are required to indentify different sides of an object. Secondly, despite the use of robust estimation (RANSAC), inaccurate matches are still possible, resulting in outlier measurements in distance and bearing. These inaccuracies are amplified by the sensitivity to errors in the estimate of the object's height when calculating the target distance. To account for these errors, in practice, multiple measurements can be used to estimate each position, and then a filter can be used to smooth the target's trajectory (e.g. a Kalman Filter).

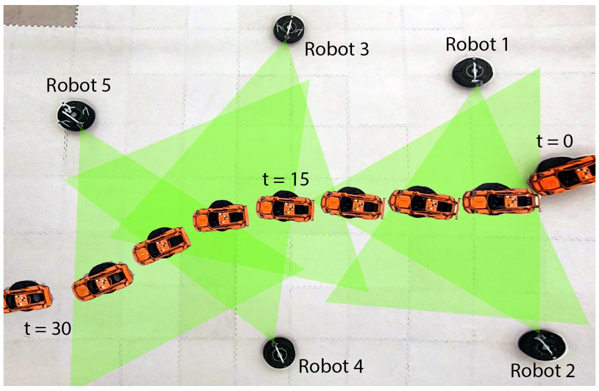
Experimental Setup

The experiment consists of a team of five iRobot Create2 ground robots, each with a forward facing camera, distributed throughout the experimental area shown in Figure 4. Each camera has a $62° \times 48°$ field of view, and takes a 640×480 px image at 2 Hz. Through the center of the area, the target object is driven along the trajectory shown in Figure 4 (a) over approximately thirty seconds. All cameras are triggered simultaneously and the images are sent to a central computer for feature extraction and matching. The central computer has an Intel i7-7800x 3.5GHz processor, and runs Ubuntu 16.04 LTS and ROS Kinetic. Features are extracted using SIFT with an octave layer of 6, a contrast threshold of 0.10, an edge threshold of 15, and sigma of 1.0. The matches from QuickMatch (using $\rho = 1.1$) and NetMatch (using $\rho = 1.1$ and $m = 6$) are used to determine which cameras observe the target object at each time step, based on the number of matches with a target image (in this experiment 10 matches are required). The matches between each reference images and the current images are used to determine the homography between them, using RANSAC with a threshold of 10.0. The homography is used to generate a bounding box around the target object using a perspective transformation on the target image corners. The relationship between pixel height of this box and distance from the camera is calibrated beforehand using an object of known size (in this case a checkerboard pattern of know dimensions). The localization points are recorded to build a target trajectory, which is then compared to ground truth measurements from an OptiTrack motion capture system (Figure 4 (b)). In an offline setting, the recorded images are fed into the NetMatch algorithm to compare its performance to QuickMatch in the experimental case. Once the feature matches are generated, the same pipeline is used from above to generate target positions.

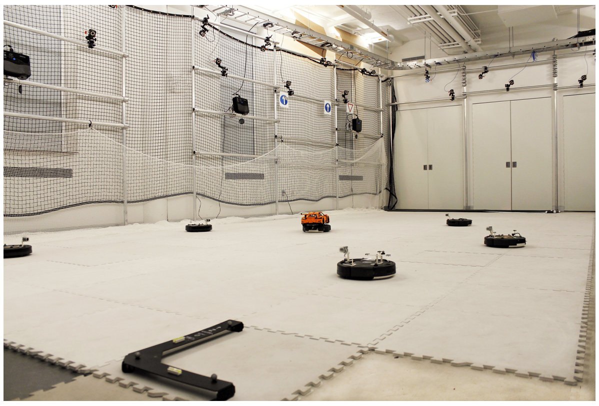
Results

QuickMatch and NetMatch are evaluated in two ways: pure matching performance, and in the context of a target localization application. The algorithms are first compared to standard matching algorithms in the OpenCV Software Package (Bradski (2000)), Brute Force (BF), and FLANN. These algorithms use the Euclidean distance metric and a threshold match distance of 0.75 (Bradski (2000); Lowe (2004)). Unlike QuickMatch and NetMatch, these algorithms cannot consider matches across more than two images but do have very low execution times.

QuickMatch is implemented in Python and takes 5.6 seconds to find matches between 6254 SIFT features (from 115 images), while BF and FLANN are both implemented in C++, and both take approximately 0.05 seconds to find the matches between the reference image features, and the same 6254 features. This time difference arises from two factors: the inherently slower run time of Python compared



(a)



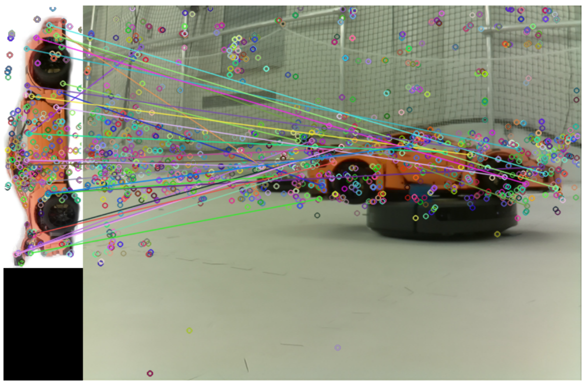
(b)

Figure 4. (a) Overhead view of experimental area with trajectory of the target object, position of the robots, and the approximate field of view for the camera network (shown in yellow). (b) Prospective view of experimental area with modified iRobot Create2 platform, target object, and overhead OptiTrack© motion capture system.

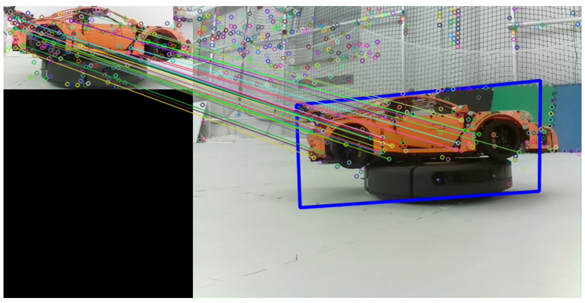
Algorithm 2 NetMatch

```
Input: $K, \rho, |A|$
Output: Clusters $\mathcal{C}_c$
...............................................
Compute $P_A$ given $K$
Compute $\ell(x_{ik}, 0)$                                ▷ Eq. 6
for all $a \in A$ do
    for all $x_{ika}, x_{i'k'a}$ do
        Compute $h(x_{ika}, x_{i'k'a}, \sigma_i)$        ▷ Eq. 7
    for all $x_{ika}$ do
        Compute $D(x_{ika})$                             ▷ Eq. 1
    for all $x_{ik}$ do
        Compute $parent(x_{ika})$ to build $\mathcal{T}_a$   ▷ Eq. 8
    for all $x_{ik}$ do
        Compute $d_{ika}$                                ▷ Eq. 13
    for all $a \in A$ do
        for all $a' \in A, a' \neq a$ do
            Compute $d_{aa'}$                            ▷ Eq. 15
    for all $x_{ika}$ do
        if $d_{ijk} + d_{aa'} \leq \sigma_p$ then
            $x_{ika} \in S_a$                            ▷ Eq. 14
    for all edges in $\mathcal{T}_a$ do
        if $\{i\} \in \mathcal{C}_c \cap \{i\} \in \mathcal{C}_{c'} = \emptyset$ and $d(x_{ika}, x_{i'k'a}) \leq \rho \arg\min_{\sigma}(\mathcal{C}_c, \mathcal{C}_{c'})$ then
            Merge $\mathcal{C}_c, \mathcal{C}_{c'}$
        else
            Remove edge from $\mathcal{T}_a$
    for all $C_c \in \mathcal{Y}_a$ do
        if $C_c \cap S_a \neq \emptyset$ then
            Compute $\min index(a')$ of $C_c \cap S_a$
            if $index(a') < index(a)$ then
                Send $C_c$ to $a'$
for all $a \in A$ do                          ▷ From high to low index
    if $C_c$ received from any $a'$ then
        if $(\arg\min_{x_{ika} \in a} d(x_{ik}, x_{i'k'})\ \forall x_{i'k'} \in C_c) \in S_a$ then
            Compute $\min index(a')$ of $C_c \cap S_a$
            if $index(a') < index(a)$ then
                Send $C_c$ to $a'$
for all $a \in A$ do
    for all $x_{ika}, x_{i'k'a}$ do
        Compute $h(x_{ika}, x_{i'k'a}, \sigma_i)$        ▷ Eq. 7
    for all $x_{ika}$ do
        Compute $D(x_{ika})$                             ▷ Eq. 1
    for all $x_{ik}$ do
        Compute $parent(x_{ika})$ to build $\mathcal{T}_a$   ▷ Eq. 8
    for all edges in $\mathcal{T}_a$ do
        if $\{i\} \in \mathcal{C}_c \cap \{i\} \in \mathcal{C}_{c'} = \emptyset$ and $d(x_{ika}, x_{i'k'a}) \leq \rho \arg\min_{\sigma}(\mathcal{C}_c, \mathcal{C}_{c'})$ then
            Merge $\mathcal{C}_c, \mathcal{C}_{c'}$
        else
            Remove edge from $\mathcal{T}_a$
return All $C_c(\mathcal{M})$
```

to C++ (Fourment and Gillings (2008)), and the extra comparisons done by QuickMatch to solve the entire Multi-match problem. If BF and FLANN compared all images with all other images combinatorially (as QuickMatch implicitly does) their computation times would be $\sim 5.75s$ seconds, which is comparable to QuickMatch's slower Python implementation. This time also does not account for the post processing time necessary to reconcile inconsistent matches from both BF and FLANN, which is not required in QuickMatch. NetMatch's computation time is typically one to two orders of magnitude larger than QuickMatch because it is currently being run as a single thread process.



(a)

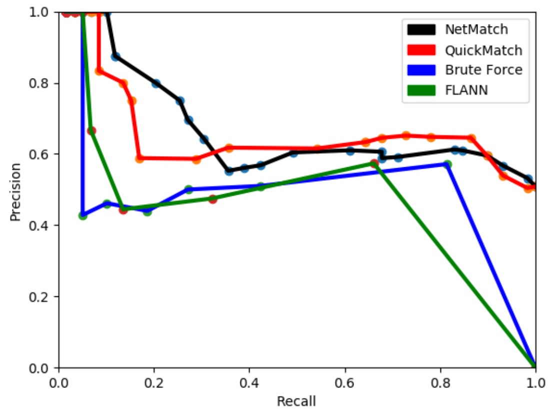


(b)

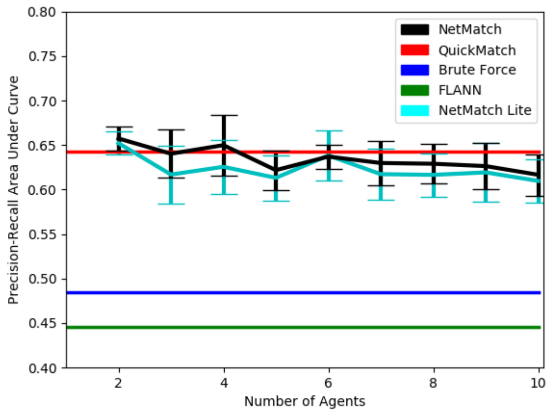
Figure 5. (a) Example image matches between the reference object image (left) and an experimental image (right): Circles represent features, and lines indicate matches. (b) Homography and localization of car with prospective transform of bounding box.

Precision Versus Recall

Although QuickMatch and NetMatch are slower, they outperform both BF and FLANN in the number of matches correctly found, and generally in terms of precision vs. recall (PR) and precision-recall area under the curve (PR AUC), which are common metrics for evaluating matching algorithms Ting (2011). Figure 6 (a) shows the precision (fraction of correctly matched images) versus recall (fraction of possible matches found) curves for QuickMatch, NetMatch, BF, and FLANN. For any recall level, QuickMatch and NetMatch maintain a higher precision level than either BF or FLANN. Both BF and FLANN have terminations before a recall of 0.9 because at that level of discrimination, they are unable to find any matches in the data. QuickMatch and NetMatch on the other hand is still able to find some matches. These curves are non-monotonic because mismatched features appear at a higher rate than correctly matched features at higher thresholds. Note that NetMatch has higher precision than QuickMatch at lower



(a)



(b)

Figure 6. (a) Precision vs. recall curves for the QuickMatch, NetMatch ($m = 6$), Brute Force, and FLANN algorithms. All algorithms are run on the same feature vectors. A match is considered to exist if the number of matched features is above a threshold. NetMatch is shown using 6 computation units. (b) PRAUC calculation for NetMatch and NetMatch Lite with increasing computation unit distribution. Error bars represent one standard deviation values over 10 trials.

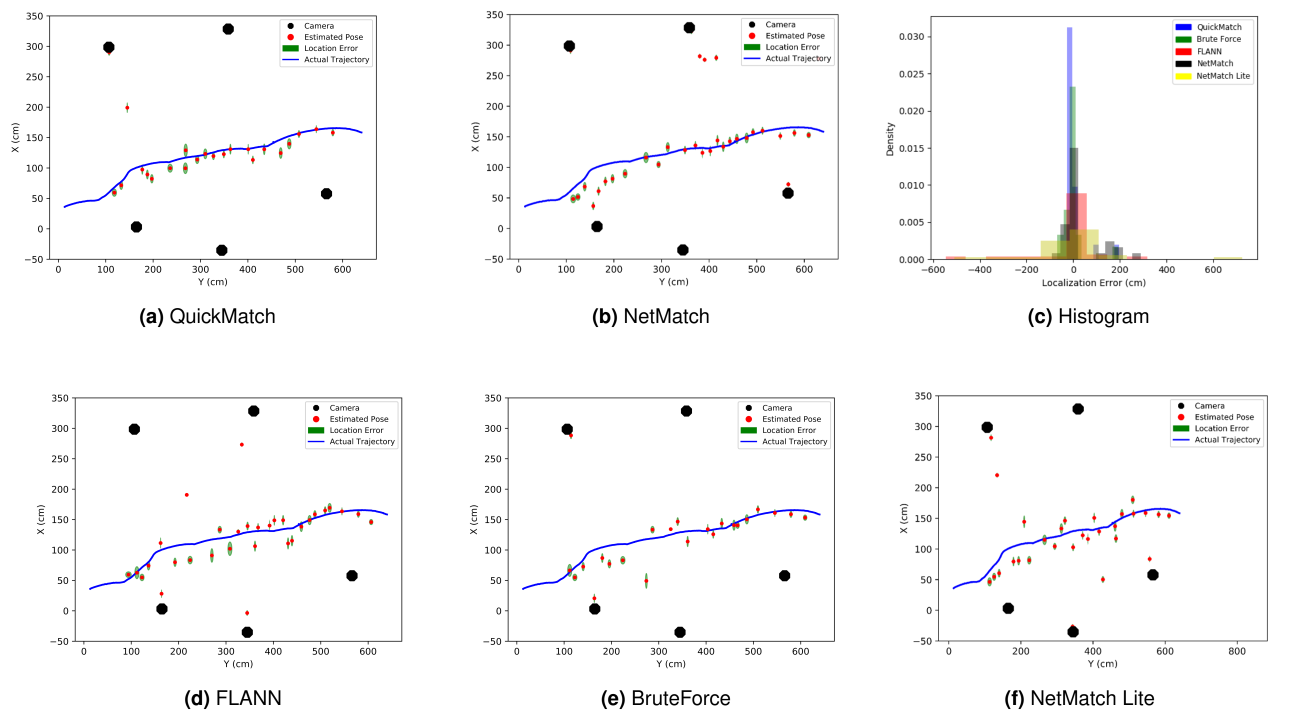
recall levels; we hypothesize that this is due to the fact that the introduction of partitions in the dataset reduces the interference between modes of different clusters in the feature density. At higher recalls however, the partitioning limits its ability to find all of the possible matches. PR AUC is a threshold agnostic metric used for comparing overall performance of matching algorithms (Ting (2011)). In terms of PR AUC, QuickMatch achieves 0.64, while BF and FLANN reach 0.49 and 0.45 respectively. The overall increase in precision stems for QuickMatch's ability to consider more instances of the reference object, by matching cycles of features across multiple images. It is therefore able to find the reference object not only more consistently, but with many more matched features. An example of these matches is shown in Figure 5.

NetMatch and NetMatch Lite (see note in the Tree Building Section above on NetMatch Lite for details) are also evaluated on the same data set as above. There are variables in NetMatch (random seed positions and number of agents) that result in a variable PR curve and therefore PR AUC. Figure 6 (b) shows the effect of these variables on the PR AUC curve. First, note that as the number of computational agents increases, the PR AUC for NetMatch decreases slightly due to losses from split clusters. Second, note that the error bars (which represent one standard deviation of PR AUC over ten trials with different partition seeds) show that the random seed position have a large effect on match quality. Specifically, a good partition can increase the PR AUC by as much as 13% in the best case and conversely a poor partition can decrease the PR AUC by as much as 11% compared to QuickMatch. The NetMatch Lite formulation (removing the contested feature swap) does not greatly reduce PR AUC (on average 3.5%) because many matches are still found (since the percent of contested clusters is small). However, the match quality of NetMatch Lite is shown below to be much worst than NetMatch in an experimental setting (see the Experiments Section below). Also, the standard deviations show that NetMatch Lite has a generally lower PR AUC and may be more susceptible to poor Voronoi seeding.

Homography and Localization

In order to further demonstrate the utility of the algorithms, matches are used to localize a target object in relation to the camera network, and then estimate its global trajectory. This was done using all of the above algorithms with again an identical set of SIFT features. QuickMatch and NetMatch consider multi-image matches between the set of target images and the set of five robot images at each time step, while BF and FLANN consider matches between each target image and the robot image individually. Once feature matches are generated, RANSAC is used to estimate the homography matrix $\overline{H}$ for each pair of images while also removing outliers from the matches. The homography between the reference image and each robot image is used to generate a bounding box around the target in the robot image as shown in Figure 5 (b). This bounding box, given a known camera calibration, provides bearing and height information for the target. The target height is known and is used to find the relate distance to the target with the bounding box height. With these two values, a distance and a bearing, the object can be localized with respect to each robot.

The above steps are performed using the match data from each of the above algorithms. Figures 7 (a-e) show the results of the localization estimation for each algorithm. Red points are estimate target poses for each time step, blue points denote the ground truth measurements, black octagons are the camera network positions, and the green regions are the one standard deviation error between all localization estimates at each time step. The localization error was found by taking the absolute distance between the estimated and ground truth position at each time step. The average error is partly a factor of object height estimation, and the variance is indicative of the match quality. QuickMatch had an error of 0.2118 ± 0.4254 meters, BF had an error of 0.2349 ± 0.4027 meters, NetMatch had an error of 0.1167 ± 0.8461 meters, FLANN had an error of 0.6232 ± 1.1722 meters, and NetMatch Lite had an error of 0.1324 ± 1.7818 meters. QuickMatch outperforms NetMatch, NetMatch Lite, and FLANN in terms of variance, which is indicative of its higher match quality. BF matcher also performs well and maintains a low variance, but does not find as many matches. FLANN is the worst performing of the standard algorithms, and



(a) QuickMatch **(b)** NetMatch **(c)** Histogram **(d)** FLANN **(e)** BruteForce **(f)** NetMatch Lite

Figure 7. (a) QuickMatch trajectory estimates. (b) NetMatch trajectory estimates with 6 agents. (c) Histogram of estimate error for each algorithm. (d) FLANN trajectory estimates. (e) BruteForce trajectory estimates. (f) NetMatch Lite trajectory estimates.

has a number of extremely erroneous estimates. Generally, monocular camera distance measurements are very sensitive to match errors, meaning target localization error is an indirect method for testing the overall accuracy of each method. Figure 7 (c) shows a histogram of the localization error, which is found by comparing the localization estimate to the ground truth pose at each time step. The histogram makes it clear that QuickMatch maintains a higher number of accurate matches and has a small number of highly erroneous estimates. It also shows that NetMatch has a large number of estimates around zero, and that NetMatch Lite does very poorly when estimating the target object's position. In practical applications, a Kalman filter would be employed to smooth the estimates, but the values are left unaltered here to demonstrate the algorithm's output.

Centralized vs Distributed Comparison

To test the difference between the distributed and centralized approaches, we look at how many clusters are split by naive partitions in the feature space to determine if the distribution scheme above is even warranted. This test is performed on images from the Graffiti data set (http://www.robots.ox.ac.uk/~vgg/data/data-aff.htm), in order to simulate realistic conditions where the ground truth is unknown. Given the clusters of features determined by QuickMatch, $C_c \in \mathcal{M}$, we define the count of each agent membership in the cluster as

$$q_a(C_c) = |C_c \cap \ell^{-1}(a)|, \tag{18}$$

where q_a is the number of features in C_c with label a. With this we can define the split quality Q of a cluster C_c as

$$Q(C_c) = \frac{\max\limits_{a \in A}(q_a)}{|C_c|}. \tag{19}$$

With this quality metric, we can quantify the number of contested clusters in a given partition as $C_c : Q(C_c) < 1$ and the percent of contested clusters as

$$p_{\text{contested}} = \frac{|C_c \in \mathcal{M} : Q(C_c) < 1|}{|C_c|}. \tag{20}$$

With this metric $p_{\text{contested}}$, we evaluate both two methods for creating the initial seeds for the Voronoi partition, as well as the ability for NetMatch to find and appropriately reassign contested clusters. For the synthetic data set, the ground truth is known, however for the graffiti data set, we assume that the QuickMatch clusters are the ground truth.

To determine how many contested features, and hence split clusters, are missed by NetMatch, we calculate the following precision

$$p_{\text{split}} = \frac{|S_A|}{\sum_{C_c : Q(C_c) < 1} |C_c|} \tag{21}$$

where S_A is the set of all contested features in $K_{\mathcal{I}}$. In other words, we can consider NetMatch to be in part as a classifier that needs to detect which features are contested; then, p_{split} represents the precision of the classifier (the number of features that are declared as contested over the number of features that ought to be declared).

The results of these tests are shown in Table 1. The first column in Table 1 shows the results of the centralized

QuickMatch algorithm. Row five shows that as the number of agents increases, intuitively, the percentage of contested clusters also increases. At the same time however, the post-QP computation time decreases as agent number decreases, because each agent has to work on considerably fewer features. The largest computational requirement in NetMatch is finding the boundary distances with the QP. The time reported for this is per-agent, however it is worth noting the implementation of this QP is sub-optimal. In the QP formulation, we consider each partition individually, meaning we solve m QPs for each feature. In the future, we plan to reduce this to combining all of the constraints to solve a single QP and this is a focus of future work. One key aspect of NetMatch to note is that it finds around 99 percent of all contested points, meaning it is very good at finding split clusters. One interesting result of NetMatch is that it matches the features into fewer, larger clusters due to the use of its finite density kernel. This is a counter-intuitive result and will be a subject of future study for this algorithm.

The final clustering results from the synthetic data set in the Simulation section are shown in Figure 8. Figure 8 (a) shows the result of the centralized QuickMatch algorithm run on the synthetic data, while Figure 8 (b) shows the final result from NetMatch. Note that many of the features from the higher index agents have been shifted to the lower index agents, but ultimately each cluster has features belonging to only one agent. Ultimately both algorithms are able to cluster all of the features correctly.

Feature Discovery

The QuickMatch and NetMatch algorithms implicitly discovers common features among images by creating clusters of similar features. These clusters correspond to specific locations in the universe, and therefore can be used to find both targets and landmarks across images. Landmarks, although not used in this paper, are points that occur commonly across all images (except when occluded), and are useful for multi-agent localization tasks. In the experimental images collected, landmarks were the clusters with the largest number of features, because many of this images did not contain the target object. An example landmark cluster is shown in Figure 9 (a). Features belonging to the target object are generally smaller than the landmark clusters, but can still be extracted, and show key features of the target. Figure 9 (b) shows one such cluster, which is the front hood of the car model. Feature discovery is one attribute of QuickMatch and NetMatch that does not exist in either BF or FLANN and can be useful for discerning what features are most descriptive of images from the network.

Table 1. Comparison of QuickMatch to NetMatch by agent number on graffiti data set.

Number of Agents	1	2	3	4	5	6	7	8	9	10	15	20	25
Compute Time Per Agent (s)	1.37	21.53	24.62	25.64	27.72	27.93	29.40	29.06	29.29	30.00	31.21	30.59	30.68
Post-QP Compute Time Per Agent (s)	1.37	5.14	3.81	2.25	2.53	1.95	1.85	1.05	1.12	1.18	0.77	0.72	0.67
QP Time Per Agent (s)	NA	16.39	20.81	23.39	25.18	25.98	27.53	27.46	27.76	28.59	30.08	29.82	30.00
Percent Contested Clusters	0	18.21	28.62	27.51	21.43	29.24	35.42	46.34	40.13	39.78	42.87	48.71	52.16
Number of Clusters Found	1320	1313	1314	1278	1265	1264	1281	1258	1269	1264	1246	1259	1309
% Contested Features Missed	NA	0.20	0.13	0.13	0.17	0.36	0.10	0.23	0.09	0.37	0.43	0.76	0.90

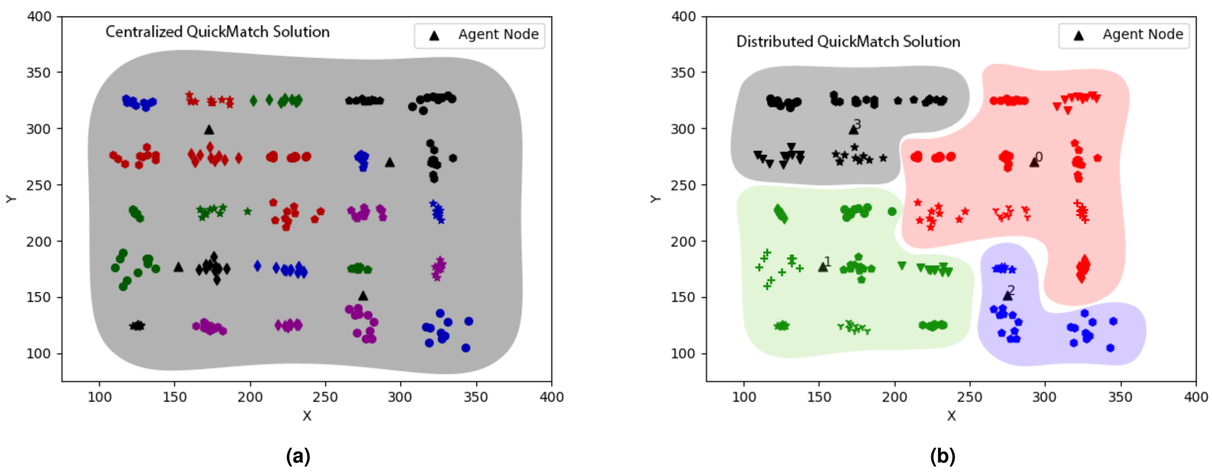
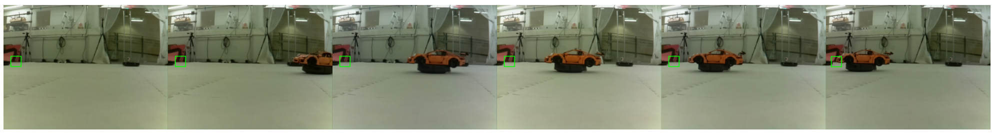
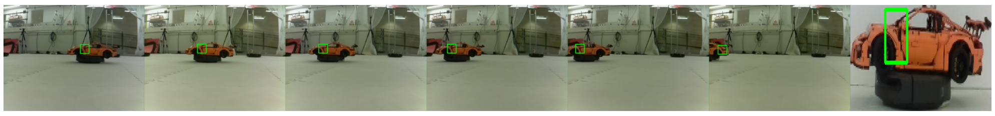


Figure 8. (a) Clustering result from the centralized QuickMatch algorithm on the synthetic data set. Feature color denotes cluster membership. (b) Clustering result from the NetMatch algorithm on the synthetic data set. Feature color denotes agent membership.



(a)



(b)

Figure 9. (a) Landmark feature cluster. (b) Target feature cluster.

Conclusion

This paper highlights the utility of QuickMatch multi-image matching for object matching and presents the NetMatch algorithm. QuickMatch is able to find many more object feature matches than standard methods by considering matches across all images, not just pairwise matches. The presented experiment tests the QuickMatch algorithm in an experimental setting with realistic conditions, and shows that multi-image matching is superior to standard methods at matching the reference object (even as it enters and exits images across the entire camera network). QuickMatch is also tested with a target object localization and again outperforms both the BF and FLANN algorithms. Beyond testing QuickMatch, we demonstrate the capabilities of NetMatch as a distributed solution to the same problem. We also demonstrate QuickMatch's feature discovery ability by showing a characteristic landmark and target feature cluster from the test images. This approach is the precursor to an online and decentralized approach. Our future work will focus on the online version of object discovery and localization and multi-camera homography. We also plan to decrease the Distributed QuickMatch's QP implementation computation time. Overall, QuickMatch is shown to be a versatile multi-feature matching algorithm that outperforms standard pairwise matching algorithms, and NetMatch offers an avenue for the QuickMatch framework to handle a large volume of features with minimal inter-agent communication.

Acknowledgements

This work was supported by the National Science Foundation under grants NRI-1734454, and IIS-1717656.

References

Andre F, Kermarrec A, Le Scouarnec N (2017) Accelerated nearest neighbor search with quick adc. The 2017 ACM on International Conference on Multimedia Retrieval (pp. 159-166). ACM.

Aragues R, Montijano E, Sagues C (2011) Consistent data association in multi-robot systems with limited communications. Robotics: Science and Systems, pages 97–104.

Bay H, Ess A, Tuytelaars T, Gool LV (2008) Speededup robust features (SURF). Computer Vision and Image Understanding, 110(3):346–359.

Bradski G (2000) The OpenCV Library. Dr. Dobb's Journal of Software Tools.

Chen Y, Guibas L, Huang Q (2014) Near-optimal joint object matching via convex relaxation. In International Conference on Machine Learning.

Cunningham A, Wurm K, Burgard W, Dellaert F (2012) Fully distributed scalable smoothing and mapping with robust multi-robot data association. IEEE International Conference on Robotics and Automation, 1093–1100.

Dollar P, Zitnick CL (2013) Structured forests for fast edge detection. International Conference on Computer Vision.

Ester M, Kriegel HP, Sander J, Xu X (1996) A Density-based Algorithm for Discovering Clusters a Density-based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proceedings of the Second International Conference on Knowledge Discovery and Data Mining, Portland, Oregon, AAAI Press, 226–231.

Fourment M, Gillings M (2008) A comparison of common programming languages used in bioinformatics. BMC Bioinformatics, vol. 9, p. 82.

Fukunaga K, Hostetler L (1975) The estimation of the gradient of a density function, with applications in pattern recognition. IEEE Transactions on Information Theory, 21(1):32–40.

Garcia V, Debreuve E, Nielsen F, Barlaud M (2010) K-nearest neighbor search: Fast GPU-based implementations and application to high-dimensional feature matching. IEEE International Conference on Image Processing, Hong Kong, 2010, pp. 3757-3760. doi: 10.1109/ICIP.2010.5654017

Gieseke F, Heinermann J, Oancea C, Igel C (2014) Buffer k-d trees: processing massive nearest neighbor queries on GPUs. 31st International Conference on International Conference on Machine Learning - Volume 32 (ICML'14), Vol. 32. JMLR.org I-172-I-180.

Hariharan B, Arbelaez P, Girshick R, Malik J (2015) Hypercolumns for object segmentation and fine-grained localization. IEEE Conference on Computer Vision and Pattern Recognition.

Hartley R, Zisserman A (2017) Multiple View Geometry in Computer Vision. Cambridge University Press, second edition.

Hu N, Huang Q, Thibert B, Guibas L (2018) Distributable Consistent Multi-object Matching. IEEE/CVF Conference on Computer Vision and Pattern Recognition, Salt Lake City, UT, 2018, pp. 2463-2471. doi: 10.1109/CVPR.2018.00261

Hu X, Huang J, Qiu M, Chen C, Chu W (2017) PS-DBSCAN: An Efficient Parallel DBSCAN Algorithm Based on Platform Of AI (PAI). arXiv preprint arXiv:1711.01034.

Huang Q, Guibas L (2013) Consistent shape maps via semidefinite programming. Computer Graphics Forum, 32(5):177-186.

Johnson J, Douze M, Jégou H, (2019) Billion-scale similarity search with GPUs. IEEE Transactions on Big Data.

Lowe DG (2004) Distinctive image features from scale-invariant keypoints. International Journal of Computer Vision, 60(2):91–110.

Leonardos S, Zhou X, Daniilidis K (2017) Distributed consistent data association via permutation synchronization. IEEE International Conference on Robotics and Automation (ICRA), Singapore, 2017, pp. 2645-2652.

MacKay DJ (2003) Information theory, inference and learning algorithms. Cambridge university press.

MacQueen J, (1967) Some methods for classification and analysis of multivariate observations. The Fifth Berkeley Symposium on Mathematical Statistics and Probability, Volume 1: Statistics, 281–297, University of California Press, Berkeley, Calif.

Montijano E, Cristofalo E, Zhou D, Schwager M, Sagues C (2016) Vision-based Distributed Formation Control without an External Positioning System. IEEE Transactions on Robotics, vol. 32, no. 2, pp. 339-351.

Muja M, Lowe DG (2014) Scalable Nearest Neighbor Algorithms for High Dimensional Data. IEEE Transactions on Pattern Analysis and Machine Intelligence 36 (11), 2227-40.

Ng AY, Jordan MI, Weiss Y (2002) On spectral clustering: Analysis and an algorithm. Neural Information Processing Systems, 2:849–856.

Novotny D, Larlus D, Vedaldi A (2017) Anchornet: A weakly supervised network to learn geometry-sensitive features for semantic matching. IEEE International Conference on Computer Vision and Pattern Recognition.

Oliveira R, Costeira J, Xavier J (2005) Optimal point correspondence through the use of rank constraints. IEEE Conference on Computer Vision and Pattern Recognition, volume 2, pages 1016–1021.

Pachauri D, Kondor R, Singh V (2013) Solving the multi-way matching problem by permutation synchronization. Twenty-seventh Conference on Neural Information Processing Systems.

Parzen E (1962) On estimation of a probability density function and mode. The annals of mathematical statistics, 33(3):1065– 1076.

Rosenblatt M (1956) Remarks on some nonparametric estimates of a density function. The Annals of Mathematical Statistics, 27(3):832–837.

Rublee E, Rabaud V, Konolige K, Bradski G (2011) Orb: An efficient alternative to sift or surf. IEEE International Conference on Computer Vision, Barcelona, Spain, pp. 2564-2571.

Sankaranarayanan A, Veeraraghavan A, Chellappa R, (2008) Object Detection, Tracking and Recognition for Multiple Smart Cameras, Proceedings of the IEEE, vol. 96, no. 10, pp. 1606-1624, Oct. 10.1109/JPROC.2008.928758

Serlin Z, Sookraj B, Belta C, Tron R (2018) Consistent Multi-Robot Object Matching Via QuickMatch. The International Symposium of Experimental Robotics.

Szeliski R (2010) Computer vision: algorithms and applications. Springer Science & Business Media.

Ting KM (2011) Precision and Recall. Sammut C., Webb G.I. (eds) Encyclopedia of Machine Learning. Springer, Boston, MA.

Tron R, Zhou X, Esteves C, Daniilidis K. (2017) Fast Multi-Image Matching via Density-Based Clustering. The IEEE International Conference on Computer Vision.

Vedaldi A, Fulkerson B (2008) VLFeat: An open and portable library of computer vision algorithms. http://www.vlfeat.org/.

Vedaldi A, Soatto S (2008) Quick shift and kernel methods for mode seeking. IEEE European Conference on Computer Vision, pages 705–718. Springer.

Wang Q, Zhou X, Daniilidis K (2018) Multi-Image Semantic Matching by Mining Consistent Features. IEEE International Conference on Computer Vision and Pattern Recognition.

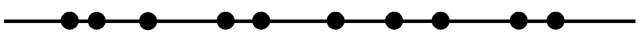
Warn S, Emeneker W, Cothren J, Apon A (2009) Accelerating SIFT on parallel architectures. IEEE International Conference on Cluster Computing and Workshops, New Orleans, LA, 2009, pp. 1-4. doi: 10.1109/CLUSTR.2009.5289155.

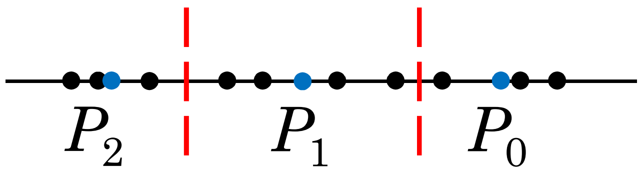
Yan J, Cho M, Zha H, Yang X, Chu S (2015) Multi-graph matching via affinity optimization with graduated consistency regularization. IEEE Transactions on Pattern Analysis and Machine Intelligence.

Yan J, Wang J, Zha H, Yang X, Chu S (2015) Consistency driven alternating optimization for multigraph matching: A unified approach. IEEE Transactions on Image Processing, 24(3):994–1009.

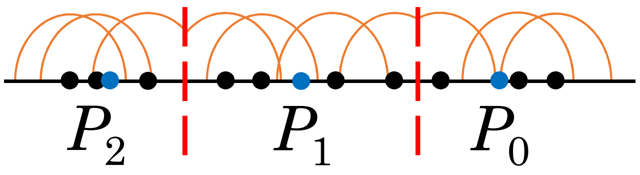
Zhou X, Zhu M, Daniilidis K (2015) Multi-Image Matching via Fast Alternating Minimization. The IEEE International Conference on Computer Vision.

Appendix 1 - NetMatch Example

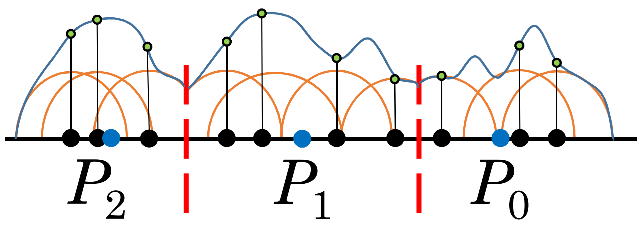

(a) Features in a 1-dimensional feature space



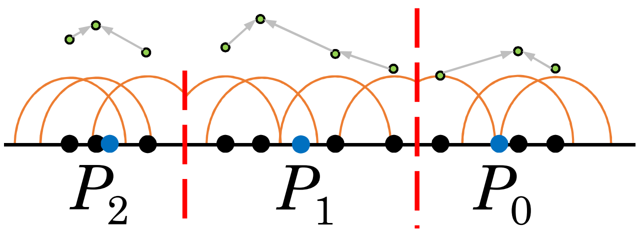
(b) Partitioned feature space for 3 agents (Voronoi seeds are in blue)



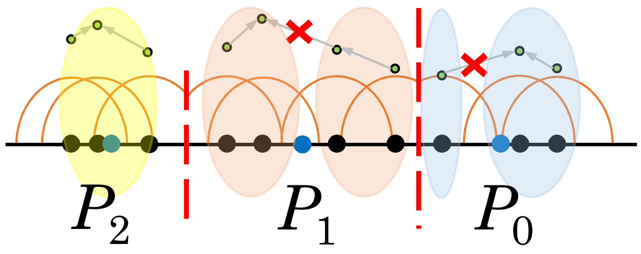
(c) Overlay of finite density kernel for each feature. Partitions are considered individually and density does not extend over the boundary. σ values are set to the minimum distance between any two points in the image of membership.



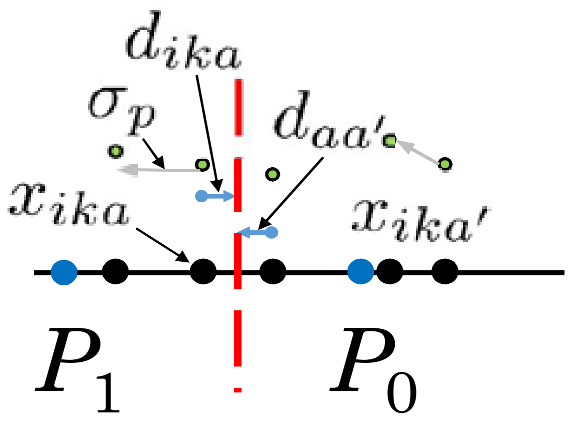
(d) Feature density is calculated for each feature and the sum of densities is shown. Points along the upper curve represent the density value for the feature below it.



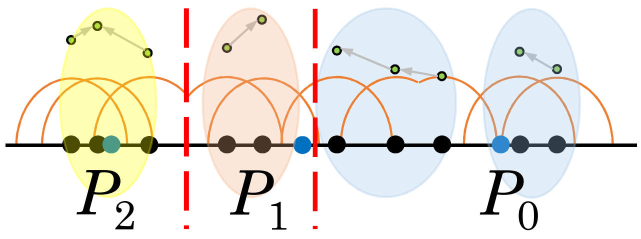
(e) Given these densities, we build a tree in each partition where parent nodes are the nearest node of higher density (from a different image). This structure results in shorter edges between cluster members and longer edges between clusters.



(f) We break each partition tree based on edge length and image membership. This results in the longer edges being broken. The resulting forest of trees represents the initial clustering. Note that the left feature in P_0 should belong to the right cluster in P_1.



(g) We then check for contested clusters with the criterion $(d_{ika} + d_{aa'}) < \sigma_p$. In this case, we check if the feature on the left of the partition could be part of a cluster on the right of the partition. This left feature (x_{ika}) meets the above check, meaning it and its cluster members need to be sent to P_0 for evaluation.



(h) Given the transfer cluster from (g), we rerun the tree building steps from above and find that the new cluster should be matched with the individual feature cluster from before.

Figure 10. (a) Feature space with features. (b) Feature space partition with 3 agents. (c) Finite kernel overlay of features. (d) Cumulative density within partitions. (e) Feature space tree building. (f) Tree breaking to form clusters. (g) Evaluation of contested features. (h) Correct clustering after contested features are swapped.