Article

Autonomous Toy Drone via Coresets for Pose Estimation

Soliman Nasser, Ibrahim Jubran * and Dan Feldman

Robotics & Big Data Labs, University of Haifa, Haifa 3498838, Israel;
snasser1@staff.haifa.ac.il (S.N.); dfeldman@univ.haifa.ac.il (D.F.)

* Correspondence: ijubran@staff.haifa.ac.il; Tel.: +972-54-7410-722

Received: 9 May 2020; Accepted: 24 May 2020; Published: 27 May 2020

Abstract: A coreset of a dataset is a small weighted set, such that querying the coreset provably yields a $(1 + \varepsilon)$ -factor approximation to the original (full) dataset, for a given family of queries. This paper suggests accurate coresets ($\varepsilon = 0$) that are subsets of the input for fundamental optimization problems. These coresets enabled us to implement a “Guardian Angel” system that computes pose-estimation in a rate > 20 frames per second. It tracks a toy quadcopter which guides guests in a supermarket, hospital, mall, airport, and so on. We prove that any set of n matrices in $\mathbb{R}^{d \times d}$ whose sum is a matrix S of rank r , has a coreset whose sum has the same left and right singular vectors as S , and consists of $O(dr) = O(d^2)$ matrices, independent of n . This implies the first (exact, weighted subset) coreset of $O(d^2)$ points to problems such as linear regression, PCA/SVD, and Wahba’s problem, with corresponding streaming, dynamic, and distributed versions. Our main tool is a novel usage of the Caratheodory Theorem for coresets, an algorithm that computes its set in time that is linear in its cardinality. Extensive experimental results on both synthetic and real data, companion video of our system, and open code are provided.

Keywords: pose estimation; localization; indoor navigation and mapping; autonomous sensors for micro drones; coresets; caratheodory

1. Introduction and Motivation

Coresets is a powerful technique for data reduction that was originally used to improve the running time of algorithms in computational geometry (e.g., [1–7]). Later, coresets were designed for obtaining the first PTAS/LTAS (polynomial/linear time approximation schemes) for more classic and graph problems in theoretical computer science [8–11]. More recently, coresets appear in machine learning conferences [12–21] with robotics [12,13,15,16,18,20–24] and image [25–27] applications.

This paper has three goals:

- (i) Introduce coresets to the robotics community and show how their theory can be applied in real-time systems and not only in the context of machine learning or theoretical computer science.
- (ii) Suggest novel coresets for real-time kinematic systems, where the motivation is to improve the running time of an algorithm, by selecting a small subset of the moving points only once and then tracking and processing them (not the entire set) during the movement of the coreset in the next observed frames.
- (iii) Provide a wireless and low-cost tracking system, IoTracker, that is based on mini-computers (“Internet of things”) that run coresets.

To obtain goal (i), we suggest a simple but powerful and generic coreset that approximates the center of mass of a set of points, using a sparse distribution on a small subset of the input points. While this “mean coreset” has many other applications, to obtain goal (ii) we use it to design a novel

coreset for pose-estimation based on the alignment between two paired sets. We then show how this coreset enables us to compute the orientation of a rigid body, in particular a moving robot, which is a fundamental question in Simultaneous Localization And Mapping (SLAM) and computer vision; see references in [28].

For example, we prove that the result of running the classic Kabsch algorithm (which computes an optimal rotation between two sets of points) on the entire input, would yield the same result when applied on the coreset only. This holds even after the input set (including its coreset) is translated and rotated in space, without the need of recomputing the coreset. We prove that the coreset has constant size (independent of the number of input tracked points) for every given input set.

Although we proved the correctness of the coreset for the Kabsch algorithm, by its properties we expect that it would be useful for many other pose-estimation algorithms. As is common in coresets for system applications, even without this proof of correctness, the coreset may be used in practice for many other related pose estimation problems.

To demonstrate goal (iii), we install our tracking system in a university (and soon in a mall) and implement a “Guardian angel” application, that to our knowledge is the first implementation for fictional systems such as Skycall [29]: a safe and low-cost quadcopter leads a guest to its destination room based on preprogrammed routes and based on the walking speed of the human. The main challenge was to control a sensors-less quadcopter in a few dozens of frames per second using weak mini-computers. Unlike existing popular videos (e.g., [29]), in our video the quadcopter is autonomous in the sense that there is no hidden remote controller or another human in the loop, see [30].

This “Guardian angel” was our main motivation and inspiration for designing the coreset in this paper; see Figure 1.

We note that our paper is not about suggesting the best algorithm for pose-estimation, the best tracking system, or about localization of quadcopters. As stated above, our goals are to show the process cycle from deep theorems in computational geometry, as the Caratheorodý Theorem, to real-time and practical systems that use coresets. Nevertheless, we are not aware of similar coresets for pose estimation of kinematic data or low-cost wireless tracking systems that can be used for hovering of a very unstable quadcopter in dozens of frames per second.

Figure 1. Snapshots from the **companion video** in: [30]. **(Left)** “Guardian Angle” system. A safe and low-cost quadcopter autonomously leads a guest to its destination **(Right)** “real-time computation of the Kabsch-Coreset. A set P of $|P| = n$ interest points detected on a planar pattern placed on a drone (red points). A subset (coreset) $C \subseteq P$ of $|C| = 5$ points (green points). It is guaranteed that the pose of the drone when computed using the weighted coreset C will be the same as the pose computed from the whole set P .

2. Related Work and Comparison

In this section we discuss related work on the general Pose Estimation problem and related solutions such as Perspective-n-Point (PnP), Iterative Closest Point (ICP), and other approaches to

solve it. Finally, we suggest how these algorithms can be applied on the coresets of this paper and conclude with summary on related coreset constructions.

Pose Estimation. The pose estimation problem is also called the alignment problem, since given two paired point sets, P and Q , the task is to find the Euclidean motion that brings P into the best possible alignment with Q . We focus on the case where this alignment is the translation μ^* and rotation R of P that minimizes the sum of squared distances to the point of Q . For $|P| = |Q| = n$ points in $\mathbb{R}^d$, the optimal translation μ^* is simply the mean of Q minus the means of P , each can be computed in $O(nd)$ time. Computing the optimal rotation R^* (Wahba's Problem [31]) can be computed independently via the Kabsch algorithm [32] in $O(nd^2)$ time; see Theorem 2.

In the PnP Problem. We are given a set of (known) 3D points and a set of n 2D points (observed points). If we have the camera's internal parameters, a set of n lines in 3D space can be computed from the 2D set of points. The goal is to align the set of 3D points with the set of 3D lines, which makes the problem hard unlike the problems that are discussed in this paper. Indeed, exact solutions for the PnP problem are known only for the case $n \leq 4$, and no provable approximations are known when the data is noisy and $n > 4$, even for the case of sum of squared distances. The Kabsch Coreset in this paper may be used to improve the running time of common PnP heuristics by running them on the coreset. Unlike their usage for Kabsch Algorithm, the theoretical guarantees of the coreset would no longer hold.

A sort of coreset of 4 point for PnP was suggested in [33]. However, unlike our Kabsch coreset, this set is not a subset of the input and provides no optimality guarantees.

ICP. In the previous paragraphs we assumed that the matching between P and Q is given. The standard and popular solution for solving the matching and pose-estimation problems is called Iterative Closest Point (ICP) proposed by Besl and McKay [34]; see [35] and references therein. This algorithm starts with a random matching (mapping) between P and Q , then: (a) runs the Kabsch algorithm on this pair of sets, and (b) rematches each point in P to its nearest point in Q , then returns to step (a). Variations and speed-ups can be found in [36–38].

Faster and Robust Matching Using Coresets. Our Kabsch coreset, similarly to the Kabsch algorithm, assumes that the matching between the points in the registered and observed frame is given. Matching is a much harder problem than, e.g., the Kabsch algorithm (that can be solved in $O(n)$ time) in the sense that we have $n!$ permutations. Nevertheless, the mean coreset that we will present can reduce the running time and increase the robustness of the matching process.

For example, in ICP, each point in $P \subseteq \mathbb{R}^3$ is assigned to its nearest neighbour (NN) in Q which take $O(|P| \cdot |Q|)$ time. Using our Kabsch coreset for P , the running time of the algorithm reduces to $O(|Q|)$. This also implies that NN matching can be replaced in existing applications by a slower but better algorithm (e.g., cost-flow [39]) that will run on the small coreset. This will improve the matching step of ICP, without increasing the existing running times. Such an improvement is relevant even for nonkinematic (single) pair P and Q of points.

Table 1 concludes the time complexity comparison of solving each step of the localization problem with/without using our coresets. The first row of the table represents the case where the matching has already been computed, and what is left to compute is the optimal rotation between the two sets of points. The second row represents step (b) of the localization problem, where the matching needs to be computed given the rotation. In this case, a perfect matching between a set of size k to a set of size m can be achieved, according to [40], in $O(\sqrt{m} + kmk \log(m + k))$ time. Without using a coreset, the size of both sets is n . When using a coreset, the size of P is reduced to rd , although the size of Q remains n . The last row of Table 1 represents a case where we need to compute the matching between two sets of points and the correct alignment is not given. In this case there are $n!$ possible permutations of the original set, each with its own optimal rotation. Using the coreset, the number of permutations reduces to roughly $(rd)!$ since it suffices to match correctly only the coreset points.

Table 1. Time comparison. All the numbers written in the table are in O notation and represent time complexity.

	Without Coreset $ P = n, Q = n$	Using Coreset $ P = rd$
With matching, without rotation	nd^2	$ Q = rd$ d^3r
Without matching, with rotation	$n^{2.5} \log(n)$	$ Q = n$ $n^{1.5}dr \log(n)$
Noisy matching	$nd^2 \cdot (n!)$	$ Q = rd$ $(dr)!$

Relation to Other Coresets. A long line of research is dedicated to the problem of approximating $\|Ax\|^2 = x^T (\sum_{i=1}^n a_i a_i^T) x$ for every $x \in \mathbb{R}^d$, which is related to SVD and linear regression as explained in Section 3.1. A breakthrough with applications to graph sparsification was suggested in [41], via a deterministic coreset construction (weighted subset) of size $O(d/\varepsilon^2)$ that yields a $(1 + \varepsilon)$ approximation for $\|Ax\|$. This result was generalized to low k -rank approximation problem (k -SVD, or k -PCA) using $O(k/\varepsilon^2)$ samples in [42], and for Frobenius norm using $O(k^2/\varepsilon^2)$ samples in [12].

If the coreset is restricted only to be a weighted subset of $\mathbb{R}^d$, and not from the input set, then its cardinality can be reduced to $O(k/\varepsilon)$ points by [43]. More properties may be obtained for approximating other problems (such as k -means) using $O(k/\varepsilon)$ points using [44,45].

However, it is not clear how the approximation error will affect the output rotation matrix that is returned by the Kabsch algorithm via the above coreset. In this paper we focus on exact coresels that have no approximation error ε . This allows us to obtain the optimal solution for the problem. Since our mean coreset does not introduce any error, it can be used in any applications that aim to compute any functions $f(AA^T) = f(\sum_i a_i a_i^T)$, since it preserves the sum $\sum_i a_i a_i^T$.

The only such accurate coreset that we know is $S \in \mathbb{R}^{d \times d}$ for a matrix $A = USx$, where $U \in \mathbb{R}^{n \times d}$ is an arbitrary orthonormal base of the columns of A (e.g., using the SVD $A = UDV^T$ or $U = Q$ from the QR decomposition (Gram–Schmidt) $A = QR$ of A). Hence, $\|Ax\|_2 = \|Sx\|$ for every $x \in \mathbb{R}^d$ and there is no approximation error. However, in this case the rows of S are not a scaled subset of the input rows. Besides numerical and interpretation issues, we cannot use this coreset S for kinematic data since we do not have a subset of points to track over time or between frames.

Coreset for sum of 1-rank positive definite matrices of size $O(d/\varepsilon^2)$ were described, e.g., in [46]; see references therein. Our mean coreset is larger but implies such an exact result and is more general (sum of any $d \times d$ matrices).

3. Warm Up: Mean Coreset

Given a set P of n points (d -dimensional vectors), our basic suggested tool is a small weighted subset of P , that we call *mean coreset*, whose weighted mean is exactly the same as the mean of the original set. In general, we can simply take the mean of P as a coreset of size 1. However, we require that the coreset will be a subset of the input set P . Moreover, we require that the vector of the multiplicative weights will be a sparse distribution over P , i.e., a positive vector with an average entry of 1. There are at least three reasons for using this coreset definition in practice, especially for real-time kinematic/tracking systems:

- (i) **Numerical stability:** Every d linearly independent points in P span their mean. However, this coreset yields huge positive and negative coefficients that canceled each other and resulted in high numerical error. Our requirement that the coreset weights will have positive weights whose average is 1 makes these phenomena disappear in practice.
- (ii) **Efficiency:** A small coreset allows us to compute the mean of a kinematic (moving) set of points faster, by computing the mean of the small coreset in each frame, instead of the complete set of

points. This also reduces the time and probability of failure of other tasks such as matching points between frames. This is explained in Section 1.

- (iii) **Kinematic Tracking:** In the next sections we track the orientation of an object (robot or a set of vectors) by tracking a kinematic representative set (coreset) of markers during many frames. This coreset is computed once for the many following frames. Such tracking is impossible when the coreset is not a subset of the tracked points.

We now formally define this mean coreset.

Definition 1 (Mean coreset). A distribution vector $u = (u_1, \dots, u_n)$ is a vector whose entries are non-negative and sum to one. A weighted set is a pair (P, u) where $P = \{p_1, \dots, p_n\}$ is an ordered set in $\mathbb{R}^d$, and u is a distribution vector of length $|P|$.

A weighted set (S, w) is a mean coreset for the weighted set (P, u) if $S \subseteq P$ and their weighted mean is the same, i.e.,

$$\sum_{i=1}^n u_i p_i = \sum_{j=1}^{|S|} w_j s_j,$$

where $S = \{s_1, \dots, s_{|S|}\}$. The cardinality of the mean coreset (S, w) is $|S|$.

Of course P is a trivial coreset of P . However, the coreset S is efficient if its size $|S| = |\{i \mid w_i > 0\}|$ is much smaller than $|P| = n$. This is related to the Caratheodory Theorem [47] from computational geometry, that states that any convex combination of a set P of points (in particular, its mean) is a convex combination of at most $d + 1$ points in P .

We first suggest an inefficient construction in Algorithm 1 to obtain a mean coreset of only $d + 1$ points, i.e., independent of n , for a set of n points. This is based on the proof of the Caratheodory Theorem which we give for completeness and takes $O(n^2 d^2)$ time, which is impractical for the applications in this paper.

Overview of Algorithm 1 and its correctness. The input is a weighted set (P, u) whose points are denoted by $P = \{p_1, \dots, p_n\}$; see Figure 2 for an illustration. We assume $n > d + 1$, otherwise $(S, w) = (P, u)$ is the desired coreset. Hence, the $n - 1 > d$ points $p_2 - p_1, p_3 - p_1, p_4 - p_1, \dots$ must be linearly dependent. This implies that there are reals $v_2, \dots, v_n$, which are not all zeros, such that

$$\sum_{i=2}^n v_i (p_i - p_1) = 0. \quad (1)$$

These reals are computed in Line 6 by solving the system of linear equations. This step dominates the running time of the algorithm and takes $O(nd^2)$ time using, e.g., SVD. The definition

$$v_1 = -\sum_{i=2}^n v_i \quad (2)$$

in Line 7, guarantees that

$$v_j < 0 \text{ for some } j \in [n], \quad (3)$$

and that

$$\begin{aligned} \sum_{i=1}^n v_i p_i &= v_1 p_1 + \sum_{i=2}^n v_i p_i = \left(-\sum_{i=2}^n v_i \right) p_1 + \sum_{i=2}^n v_i p_i \\ &= \sum_{i=2}^n v_i (p_i - p_1) = 0, \end{aligned} \quad (4)$$

where the second equality is by (2), and the last is by (1). Hence, for every $\alpha \in \mathbb{R}$, the weighted mean of P is

$$\sum_{i=1}^n u_i p_i = \sum_{i=1}^n u_i p_i - \alpha \sum_{i=1}^n v_i p_i = \sum_{i=1}^n (u_i - \alpha v_i) p_i, \quad (5)$$

where the first equality holds since $\sum_{i=1}^n v_i p_i = 0$ by (4). The definition of α in Line 8 guarantees that $\alpha v_{i^*} = u_{i^*}$ for some $i^* \in [n]$ and that $u_i - \alpha v_i \geq 0$ for every $i \in [n]$. Hence, the set S that is defined in Line 10 contains at most $n - 1$ points, and its set of weights $\{u_i - \alpha v_i\}$ is non-negative. Notice that if $\alpha = 0$, we have that $w_k = u_k > 0$ for some $k \in [n]$. Otherwise, by (3), there is $j \in [n]$ such that $w_j = u_j - \alpha v_j > 0$. Hence, $|S| \neq \emptyset$. The sum of the positive weights is thus the total sum of weights,

$$\sum_{p_i \in S} w_i = \sum_{i=1}^n (u_i - \alpha v_i) = \sum_{i=1}^n u_i - \alpha \cdot \sum_{i=1}^n v_i = 1,$$

where the last equality holds by (2) and since u is a distribution vector. This and (5) proves that S is a mean coreset as in Definition 1 of size $n - 1$. In Line 12 we repeat this process recursively until there are at most $d + 1$ points left in S . For $O(n)$ iterations the overall time is thus $O(n^2 d^2)$.

The correctness of the following lemma follows mainly by the Caratheodory Theorem [47] from computational geometry.

Lemma 1. Let $P = \{p_1, \dots, p_n\} \subseteq \mathbb{R}^d$ be a set of $n > d + 1$ points and $u = \{u_1, \dots, u_n\}$ be a distribution. Let (S, w) be the output of a call to MEAN-CORESET(P, u); see Algorithm 1. Then (S, w) is a mean coreset of (P, u) . This takes $O(n^2 d^2)$ time.

We then use the fact that our mean coresets are composable [48–51]: a union of coresets can be merged and reduced again recursively. To reduce the running time of Algorithm 1, we run it only on the first $d + 2$ points of P and reduce the $d + 2$ points to a coreset of $d + 1$ points in $O(d^3)$ time using a single iteration. We then add a new point to the previously compressed $d + 1$ points, compress again, then repeat for each of the remaining points using $n = d + 1$ in Lemma 1 for every point update.

Overview of Algorithm 2 and its correctness. We denote $[n] = \{1, \dots, n\}$ for every integer $n \geq 1$. In Lines 1–3 we respectively set $n = d + 1$, initialize S with the first $d + 1$ points from *stream*, and set the weight of all the points in S to be $\frac{1}{d+1}$. In Line 4 we begin to read the points in the (possibly infinite) input stream of points. In Line 5 we update this counter n , and in Line 6 we read the n th point from the stream. The set P in Line 7 is the union of the coreset for the points read until now with the new n th point p .

In Line 8 we define a distribution vector u such that the weighted set (P, u) has the same mean as the mean of the n points $p_1, \dots, p_n$ that were read until now. The intuition is that the new points represent a fraction of $1/n$ from the n points seen so far, but S (the rest of points in P) represents $(n - 1)/n$ input points. If the i th point in S has a weight w_i , it means that it represents a fraction of w_i from S , i.e., fraction of $w_i(n - 1)/n$ from all the data. Indeed, the mean of the n read points $p_1, \dots, p_n$ and P is the same,

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n p_i &= \frac{1}{n} \sum_{i=1}^{n-1} p_i + \frac{p_n}{n} = \frac{n-1}{n} \sum_{i=1}^{|S|} w_i s_i + \frac{p_n}{n} \\ &= \sum_{i=1}^{|S|} u_i p_i + \frac{p_n}{n} = \sum_{i=1}^{|P|} u_i p_i, \end{aligned} \quad (6)$$

where the second equality holds since $\frac{1}{n-1} \sum_{i=1}^{n-1} p_i = \sum_{i=1}^{|S|} w_i s_i$, and the last equality holds since $p_n = p_{|P|}$. Moreover, u is a distribution vector since

$$\sum_{i=1}^n u_i = \frac{1}{n} + \sum_{i=1}^{|S|} \frac{w_i(n-1)}{n} = \frac{1}{n} + \frac{n-1}{n} = 1,$$

where the second equality is since w is a distribution vector by induction.

In Line 9, we compute a mean coreset (S, w) for (P, u) . Since $|P| = d + 2$, by Lemma 1 this takes $O(d^3)$, and by (6) (S, w) is also the mean coreset for the n points read until now. In Line 10 we output (S, w) and repeat for the next point. The required memory is dominated by the set P of $d + 2$ points. We conclude with the following theorem.

Theorem 1. *Let stream be a procedure that outputs a new point in $\mathbb{R}^d$ after each call. A call to STREAMING-CORESET(stream) outputs a mean coreset of cardinality $d + 1$ for the first n points in stream, for every $n \geq 1$. This takes $O(d^3)$ time for each point update, overall of $O(nd^3)$ time and using at most $d + 2$ points in memory.*

Algorithm 1: MEAN-CORESET(P, u)

Input: A weighted set (P, u) of n points in $\mathbb{R}^d$.

Output: A mean coreset (S, w) for $P = \{p_1, \dots, p_n\}$ of cardinality $|S| \leq d + 1$; see Definition 1.

```

1 if  $n \leq d + 1$  then
2   return  $(P, u)$ 
3 for each  $i \in \{2, \dots, n\}$  do
4   Set  $a_i \leftarrow p_i - p_1$ 
5 Set  $A$  to be the  $d \times (n - 1)$  matrix  $[a_2 | \dots | a_n]$ .
6 Compute  $v = (v_2, \dots, v_n)^T \neq 0$  such that  $Av = 0$ .
7 Set  $v_1 \leftarrow -\sum_{i=2}^n v_i$ 
8 Set  $\alpha \leftarrow \min \left\{ \frac{u_i}{v_i} \mid i \in [n] \text{ and } v_i > 0 \right\}$ 
9 Set  $w_i \leftarrow (u_i - \alpha v_i)$  for every  $i \in [n]$  s.t.  $w_i > 0$ .
10 Set  $S \leftarrow \{p_i \mid w_i > 0 \text{ and } i \in [n]\}$ 
11 if  $|S| > d + 1$  then
12   Set  $(S, w) \leftarrow \text{MEAN-CORESET}(S, w)$ 
13 return  $(S, w)$ 

```

Algorithm 2: STREAMING-CORESET(*stream*)

Input: A (possibly infinite) *stream* of points in $\mathbb{R}^d$.
Output: A mean coreset (S, w) of cardinality $|S| \leq d + 1$
 for the first n points in *stream*, for every $n \geq 1$.

```

1 Set  $n \leftarrow d + 1$ 
2 Set  $S \leftarrow$  union of the first  $d + 1$  points in stream
3 Set  $w_i \leftarrow \frac{1}{d+1}$  for every  $i \in [d + 1]$ 
4 while stream is not empty do
5   Set  $n \leftarrow n + 1$ 
6    $p \leftarrow$  read the  $n$ th point from stream
7   Set  $P \leftarrow S \cup \{p\}$ 
   /*  $S$  is the first  $|S|$  points in  $P$  */
8   Set  $u_i \leftarrow \begin{cases} \frac{w_i(n-1)}{n} & 1 \leq i \leq |S| \\ \frac{1}{n} & i = |S| + 1 \end{cases}$ 
9    $(S, w) \leftarrow \text{MEAN-CORESET}(P, u)$ 
10  Output  $(S, w)$ 

```

3.1. Example Applications

Coreset for 1-mean queries. A coreset for k -mean queries of a set $P \subseteq \mathbb{R}^d$ approximates the sum of squared distances over the points of P to any given set of k centers (points in $\mathbb{R}^d$). There is a long line of research for this type of coresets [11,52–57]. Algorithm 2 yields the first accurate coreset (no approximation error) weighted subset $S \subseteq P$ of size $d + 3$ for the case $k = 1$. The solution also holds for streaming data (and distributed/dynamic data as explained below).

Corollary 1. Let $P' = \{p'_1, \dots, p'_n\}$ be a set of points in $\mathbb{R}^d$. Let *stream* be a corresponding stream whose i th point is $(p' \mid \|p'\|^2 \mid 1) \in \mathbb{R}^{d+2}$ for every $i \geq 1$. Let (S, w) be the n th outputted pair of a call to STREAMING-CORESET(*stream*), and $S' = \{p' \in P' \mid p \in S\} \subseteq P'$. Then $|S| = d + 3$ and for every $x \in \mathbb{R}^d$ we have

$$\sum_{i=1}^n \|p'_i - x\|^2 = \sum_{i=1}^{d+3} w_i \|s_i - x\|^2.$$

Proof. Simple calculations show that

$$\begin{aligned}
 \sum_{i=1}^n \|p'_i - x\|^2 &= \sum_{i=1}^n \|p'_i\|^2 + n \|x\|^2 - 2x^T \sum_{i=1}^n p'_i = \left(\sum_{i=1}^n \|p'_i\|^2, n, \sum_{i=1}^n p'_i \right) \begin{pmatrix} 1 \\ \|x\|^2 \\ -2x^T \end{pmatrix} \\
 &= \left(\sum_{i=1}^n (\|p'_i\|^2, 1, p'_i) \right) \begin{pmatrix} 1 \\ \|x\|^2 \\ -2x^T \end{pmatrix} = \left(\sum_{i=1}^{d+3} w_i s_i \right) \begin{pmatrix} 1 \\ \|x\|^2 \\ -2x^T \end{pmatrix} \\
 &= \left(\sum_{i=1}^{d+3} w_i \|p'_i\|^2 \mid \sum_{i=1}^{d+3} w_i \mid \sum_{i=1}^{d+3} w_i p'_i \right) \begin{pmatrix} 1 \\ \|x\|^2 \\ -2x^T \end{pmatrix} = \sum_{i=1}^{d+3} w_i \|s_i - x\|^2.
 \end{aligned}$$

□

Sum coreset for matrices. Theorem 1 implies that we can compute the sum of n matrices in $\mathbb{R}^{d \times d}$ using a weighted subset of d^2 matrices, simply by concatenating the entries of each matrix to a vector in

$\mathbb{R}^{d^2}$. In Section 4 we reduce this size for the case where we are only interested in the left+right singular vectors of the matrix. This reduction is theoretically small but allowed us to reduce the number of required markers on the object tracked by our system by more than half in the third paragraph of Section 6.2, which was critical to the IR tracking version of our system.

Coreset for SVD. Let $A \in \mathbb{R}^{n \times d}$. Our mean coreset implies that there is a matrix S that consists of $O(d^2)$ scaled rows in A such that for every $x \in \mathbb{R}^d$, $\|Ax\|_2 = \|Sx\|_2$. This is since

$$\|Ax\|_2^2 = (Ax)^T(Ax) = x^T A^T A x = x^T \left(\sum_{i=1}^n a_i a_i^T \right) x.$$

The rightmost term can be computed using a mean coreset for matrices as defined above.

Coreset for Linear Regression. In the case of linear regression, we are also given a vector $b \in \mathbb{R}^n$ and wish to compute a matrix S of $O(d^2)$ weighted rows from A and a vector v of the same size, such that for every $x \in \mathbb{R}^d$ we have

$$\|Ax - b\|_2 = \|Sx - v\|_2.$$

This can be obtained by replacing A with $[A \mid -b]$ in the previous example.

Streaming, Distributed and Dynamic computation. Theorem 1 implies that we can compute the above coresets also for possibly infinite streaming set of row vectors or matrices. Similarly, using m machines the (parallel) running time can be reduced by a factor of m by sending the i th point in the stream to the $(i \bmod m)$ th machine [58] using communication of $O(d^2)$ points of the final coreset to a main machine. Unlike the common usage of binary merge-reduce trees (e.g., [51]), the approximation error, memory, and time do not increase with n , for unbounded stream, due to the fact that the coresets are exact.

Such composable coresets support deletion/insertion of a point in logarithmic update time (but linear space) in the number n of existing points in the set; see details in [51].

4. Application for Kinematic Data: Kabsch Coreset

To track a kinematic set of points (e.g., markers or visual features on a rigid body), we define its initial (zero) positions $\{p_1, \dots, p_n\}$ as the set of n rows of a matrix $P \in \mathbb{R}^{n \times 3}$ which is centered around the origin and compare it to the observed set, i.e., rows $\{q_1, \dots, q_n\}$ of a matrix $Q \in \mathbb{R}^{n \times 3}$ in the current time or frame. The difference (translation and rotation) between P and Q tells us the current position of the set. Using the Maximum Likelihood approach and the common assumption of Gaussian noise (which has physical justification), the optimal solution is the translation and rotation of P that minimize the sum of squared distances to the corresponding points (rows) in Q . Consider the problem of computing the rotation matrix that minimizes the sum of squared distances between the corresponding sets,

$$\text{cost}(P, Q, R) := \sum_{i=1}^n \|p_i - q_i R\|^2,$$

This is known as Wahba's Problem [31]. We denote this minimum by

$$\text{OPT}(P, Q) := \min_R \text{cost}(P, Q, R) = \text{cost}(P, Q, R^*),$$

where the minimum is over every rotation matrix $R \in \mathbb{R}^{d \times d}$.

Tracking translation. Consider the problem of computing the optimal translation, i.e., the translation vector $t^* \in \mathbb{R}^d$ that minimizes

$$\text{cost}(P, Q, t) := \sum_{i=1}^n \|p_i - q_i - t\|^2,$$

over every $t \in \mathbb{R}^d$. Easy calculations show that the optimal translation is the mean (center of mass) of Q . This mean can be maintained by tracking only the small mean coreset of Q over time as defined in Section 3, even without knowing the matching between the points in P and Q .

In this section we thus focus on the more challenging problem of computing the rotation R that minimizes the sum of squared distances between the points of P and QR .

The Kabsch algorithm [32] suggests the following simple but provably optimal solution for Wahba's problem. Let UDV^T be a Singular Value Decomposition (SVD) of the matrix $P^T Q$. That is, $UDV^T = P^T Q$, $U^T U = V^T V = I$, and $D \in \mathbb{R}^{d \times d}$ is a diagonal matrix whose entries are nonincreasing. In addition, assume that $\det(U) \det(V) = 1$, otherwise invert the signs of one of the columns of V . Note that D is unique but there might be more than one such factorization.

Theorem 2 ([32]). *The matrix $R^* = VU^T$ minimizes $\text{cost}(P, Q, R)$ over every rotation matrix R , i.e., $\text{OPT}(P, Q) = \text{cost}(P, Q, R^*)$.*

We now suggest a coreset (sparse distribution) for this problem.

Definition 2 (Kabsch Coreset). *Let $w \in S^n$ be a distribution. Let $\tilde{P}, \tilde{Q} \in \mathbb{R}^{n \times d}$ denote the matrices whose i th row is $\sqrt{w_i} p_i$ and $\sqrt{w_i} q_i$, respectively, for every $i \in \{1, \dots, n\}$. Then w is a Kabsch coreset for the pair (P, Q) if for every pair of rotation matrices $A, B \in \mathbb{R}^{d \times d}$ and every pair of vectors $\mu, \nu \in \mathbb{R}^d$ the following holds: A rotation matrix $\tilde{R}$ that minimizes $\text{cost}(\tilde{P}A + \mu, \tilde{Q}B + \nu, R)$ over every rotation matrix R , is also optimal for $(PA + \mu, QB + \nu)$, i.e.,*

$$\text{OPT}(PA + \mathbf{1}^T \mu, QB + \mathbf{1}^T \nu) = \text{cost}(PA + \mathbf{1}^T \mu, QB + \mathbf{1}^T \nu, \tilde{R}),$$

where $\mathbf{1}^T = (1, \dots, 1) \in \mathbb{R}^n$.

This implies that we can use the same coreset even if the set Q is translated or rotated over time. Such a coreset is efficient if it is also small (i.e., the distribution vector w is sparse).

Recall that UDV^T is the SVD of $P^T Q$, and let r denote the rank of $P^T Q$, i.e., number of nonzero entries in the diagonal of D . Let $D_r \in \mathbb{R}^{d \times d}$ denote the diagonal matrix whose diagonal is 1 in its first r entries, and 0 otherwise.

Lemma 2. *Let $R = GF^T$ be a rotation matrix, such that F and G are orthogonal matrices, and $GD_r F^T = VD_r U^T$. then R is an optimal rotation, i.e.,*

$$\text{OPT}(P, Q) = \text{cost}(P, Q, R).$$

Moreover, the matrix $VD_r U^T$ is unique and independent of the chosen Singular Value Decomposition UDV^T of $P^T Q$.

Proof. It is easy to prove that R is optimal, if

$$\text{Tr}(RP^T Q) = \text{Tr}(D); \quad (7)$$

see [59] for details. Indeed, the trace of the matrix $RP^T Q$ is

$$\begin{aligned} \text{Tr}(RP^T Q) &= \text{Tr}(RUDV^T) = \text{Tr}(GF^T(UDV^T)) \\ &= \text{Tr}(GD_r F^T \cdot UDV^T) \end{aligned} \quad (8)$$

$$+ \text{Tr}(G(I - D_r)F^T \cdot UDV^T). \quad (9)$$

Term (8) equals

$$\begin{aligned}\text{Tr}(GD_r F^T \cdot UDV^T) &= \text{Tr}(VD_r U^T \cdot UDV^T) \\ &= \text{Tr}(VDV^T) = \text{Tr}(DV^T V) = \text{Tr}(D),\end{aligned}\quad (10)$$

where the last equality holds since the trace is invariant under cyclic permutations. Term (9) equals

$$\begin{aligned}\text{Tr}(G(I - D_r)F^T \cdot UDV^T) &= \text{Tr}(G(I - D_r)F^T \cdot (D_r U^T)^T DV^T) \\ &= \text{Tr}(G(I - D_r)F^T \cdot (V^T G D_r F^T)^T DV^T) \\ &= \text{Tr}(G(I - D_r)F^T \cdot F D_r^T G^T V \cdot DV^T) \\ &= \text{Tr}(G \cdot (I - D_r)D_r \cdot G^T V \cdot DV^T) = 0,\end{aligned}\quad (11)$$

where the last equality follows since the matrix $(I - D_r)D_r$ has only zero entries. Plugging the last equality and (10) in (8) yields $\text{Tr}(RP^T Q) = \text{Tr}(D)$. Using this and (7) we have that R is optimal.

For the uniqueness of the matrix $VD_r U^T$, observe that for $N = P^T Q = UDV^T$ we have

$$(N^T N)^{1/2} (N)^+ = (VDV^T)(VD^+ U^T) = VD_r U^T. \quad (12)$$

Here, a squared root $X^{1/2}$ for a matrix X is a matrix such that $(X^{1/2})^2 = X$, and X^+ denote the pseudo inverse of X . Let FEG^T be an SVD of N . Similarly to (12), $(N^T N)^{1/2} (N)^+ = GD_r F^T$.

Since $N^T N = VD^2 V^T$ is a positive-semidefinite matrix, it has a unique square root. Since the pseudo inverse of a matrix is also unique, we conclude that $(N^T N)^{1/2} (N)^+$ is unique, and thus $VD_r U^T = GD_r F^T$. $\square$

Overview of Algorithm 3. The input is a pair (P, Q) of $n \times d$ matrices that represent two paired set of points in $\mathbb{R}^d$. To obtain an object's pose, we need to apply the Kabsch algorithm on the matrix $P^T Q = \sum_i p_i^T q_i$; see Theorem 2. Algorithm 3 outputs a sparse weight vector $w = (w_1, \dots, w_n)$ such that the summation $P^T Q$ equals to the weighted sum $\sum_i w_i p_i^T q_i$ of at most $r(d - 1) + 1$ matrices, where w is a *Kabsch-coreset* as in Definition 2.

This is done by choosing w such that

$$E = U^T \left(\sum_i w_i p_i^T q_i \right) V = \sum_i w_i (U^T p_i^T q_i V) \quad (13)$$

is a diagonal matrix. In this case, the rotation matrix of the pairs $(\sqrt{w_i} p_i, \sqrt{w_i} q_i)_{i=1}^n$ and (P, Q) will be the same by Theorem 2. By letting $m_i = (U^T p_i^T q_i V)$ we need to have the sum $\sum_{i=1}^n m_i$ by a weighted subset of the same sum.

This vector m_i is computed in Line 5. In Line 7 we compute a mean coreset (S', w') using Algorithm 2 for the n vectors $m_1, \dots, m_n$. Since the mean coreset contains only the nonzero weights with their corresponding points, in Line 8 we translate the $|S'| = O(d^2)$ weights in w' to the sparse vector w : if s_i is the i th point in S and $s_i = m_j$, then $w_j = w'_i$. Theorem 1 then guarantees that (13) holds as desired.

We now prove the main theorem of this section.

Theorem 3. Let $P, Q \in \mathbb{R}^{n \times d}$ be a pair of matrices. Let r denote the rank of the matrix P . Then a call to the procedure $\text{KABSCH-CORESET}(P, Q)$ returns a *Kabsch-coreset* w of sparsity at most $r(d - 1) + 1$ for (P, Q) in $O(nd^4)$ time; see Definition 2.

Proof. Since (S, w') is a mean coreset for $m_1, \dots, m_n$ we have that w is a distribution of sparsity at most $r(d-1) + 1$, such that

$$E = U^T \left(\sum_i p_i^T q_i \right) V = U^T \left(\sum_i w_i p_i^T q_i \right) V \quad (14)$$

is diagonal and consists of at most r nonzero entries. Here p_i and q_i are row vectors which represent the i th row of P and Q respectively. Let $\{\sqrt{w_i} p_i \mid w_i > 0\}$ and $\{\sqrt{w_i} q_i \mid w_i > 0\}$ be the rows of $\tilde{P}$ and $\tilde{Q}$ respectively. Let FEG^T be an SVD of $A^T \tilde{P}^T \tilde{Q} B$ such that $\det(F) \det(G) = 1$, and let $\tilde{R} = GF^T$ be an optimal rotation of this pair; see Theorem 2. We need to prove that

$$\text{OPT}(PA + \mu, QB + \nu) = \text{cost}(PA + \mu, QB + \nu, \tilde{R}).$$

We assume without loss of generality that $\mu = \nu = 0$, since translating the pair of matrices does not change the optimal rotation between them [59].

By (14), UEV^T is an SVD of $\tilde{P}^T \tilde{Q}$, and thus $A^T UEV^T B$ is an SVD of $A^T \tilde{P}^T \tilde{Q} B$. Replacing P and Q with $\tilde{P}A$ and $\tilde{Q}B$ respectively in Lemma 2 we have that $GD_r F^T = B^T V D_r U^T A$. Note that since UDV^T is an SVD of $P^T Q$, we have that $A^T UDV^T B$ is an SVD of $A^T P^T Q B$. Using this in Lemma 2 with PA and QB instead of P and Q respectively yields that $\tilde{R} = GF^T$ is an optimal rotation for the pair (PA, QB) as desired, i.e.,

$$\text{OPT}(PA, QB) = \text{cost}(PA, QB, \tilde{R}).$$

□

Algorithm 3: KABSCH-CORESET(P, Q)

Input: A pair of matrices $P, Q \in \mathbb{R}^{n \times d}$.

Output: A sparse Kabsch coreset $w = (w_1, \dots, w_n)$
for (P, Q) ; see Definition 2.

- 1 Set $d' \leftarrow r(d-1)$ where r is the rank of $P^T Q$
 - 2 Set $UDV^T \leftarrow$ an SVD of $P^T Q$
 - 3 **for each** $i \in [n]$ **do**
 - 4 Set p_i and q_i to be the i th row of P and Q , respectively.
 - 5 Set $m_i \in \mathbb{R}^{d'}$ as the entries of $U^T p_i^T q_i V$, excluding its diagonal and last $d-r$ rows.
 - 6 $stream \leftarrow \{m_1, m_2, \dots, m_n\}$
 - 7 $(S, w') \leftarrow \text{STREAMING-CORESET}(stream)$; see Algorithm 2
 - 8 $(w_1, \dots, w_n) \leftarrow$ the weights of w' that correspond to S padded with zero entries.
 - 9 **return** w
-

5. From Theory to Real Time Tracking System

While our coresets are small and optimal, they come with a price: unlike random sampling which takes sublinear time to compute (without going over the markers), computing our coreset takes the same time as solving the pose estimation problem on the same frame. Hence, we use the following pair of parallel threads.

The first thread, which we run at 1 to 3 FPS (frames per second), gets a snapshot (frame) of the currently observed markers Q and computes the coreset for this frame. This includes marker identification, matching problem, and then computing the actual coreset for the original set of markers P and the observed set Q . The second thread, which calculates the object's pose, runs every frame. In our low-cost tracking system (see Section 6) it handles 30 FPS. This is by using the last

computed coreset on the new frames, until the first thread computes a new coreset for a later frame. The assumption of this model is that, for frames that are close to each other in time, the translation and rotation of the observed set of markers will be similar to the translation and rotation of the set Q in the previous frame, up to a small error. Theorem 2 guarantees that the coreset for the first frame will still be the same for the new frame.

6. Experimental Results

We run the following types of experiments:

6.1. Synthetic Data

We constructed a set P of n randomly and uniformly sampled points in $\mathbb{R}^3$, a rotation matrix $R \in \mathbb{R}^{3 \times 3}$, and a translation vector $t \in \mathbb{R}^d$ from a uniform distribution. We defined $Q = P \cdot R + t$ and aimed to reconstruct R and t using the following methods: (i) Calculate the optimal rotation matrix and optimal translation vector from P and Q , as described in Section 4, (ii) Compute the same from the Kabsch-coreset (see Algorithm 3) of size $r \cdot (d - 1) + 1 = 7$ (where $r = d = 3$) and the Mean-coreset (see Algorithm 1) of size $d + 1 = 4$, (iii) Uniform sampling of two sets of corresponding points from P and Q , one of size 7 and the second of size 4, and compute R and t from these sets.

Non-noisy Data. Here we generated data as described above for 100 iterations, where the set $P = \{p_1, p_2, \dots, p_{300}\}$ consisted of 300 randomly sampled points. Each point $p_i \in [0, 3000]^3$, $t \in [0, 3000]^3$ and R was randomly selected among all valid 3D rotation matrices. We then compared methods (i) and (ii) where the coreset was computed in the first iteration only and used throughout all other iterations. The results are shown in Figure 3. As proven in Section 4, the two methods yielded similar results since the data is non-noisy. Surprisingly, the coreset error in most iterations is even lower than the error of the optimal method, probably since the coreset reduces numerical errors; see the beginning of Section 3.

Figure 2. A weighted set (P, u) whose weighted mean is $\sum_{i=1}^4 u_i p_i = 0$ corresponds to four points (in blue) whose sum is the origin (in orange). Algorithm 1 first computes a weighted set (P, v) (red points) whose weighted mean is the origin, and sum of weights is $\sum_{i=1}^4 v_i = 0$. The weights are scaled by $\alpha > 0$ until $\alpha v_i = u_i$ for some i ($i = 1$ in the figure). The resulting weighted set $(P, \alpha v)$ in green is subtracted from the input (P, u) to obtain $(P, u - \alpha v) = (P, w)$, where $w_1 = 0$ so p_1 can be removed. Algorithm 1 then continues iteratively with the remaining points until (P, w) has $|P| = d + 1$ weighted points.

Figure 3. Comparing the results of methods (i) and (ii). The X-axis represents the number of iterations. The Y-axis represents the Mean Squared Errors between the two sets after applying the optimal poses obtained from each of the two methods.

Noisy Data. Here our goal was to test the coresets in the presence of noise. We generated a set $P = \{p_1, p_2, \dots, p_{100}\} \in \mathbb{R}^{100 \times 3}$ of 100 randomly sampled points. Each point $p_i \in [0, 1000]^3$, t is a random vector in $[0, 1000]^3$ and R was randomly selected among all valid 3D rotation matrices. We then computed the set $Q = P \cdot R + t' + m \cdot B$, where $B \in \mathbb{R}^{100 \times 3}$ consists of random and uniform noise in the range $[0, 100]$, m is the magnitude of the noise, and $t' \in \mathbb{R}^{100 \times 3}$ is simply the concatenation of t 100 times. This test compares the error produced by methods (i)–(iii) while increasing the value of m for multiple iterations. The coreset was recomputed every x iterations and the random points were also resampled every x iterations, where x is the calculation cycle. The results are shown in Figure 4; the first graph shows the results for $x = 20$, the second graph shows the results for $x = 300$, and the third graph shows the results for $x = \infty$ (i.e., computed only once). The results show a steady increase in the error of method (iii). Our coreset's error steadily increases until a new coreset is recalculated; at that point the coreset error realigned with the error of method (i) as expected, resulting in stiff decreases that are seen in the graphs. Moreover, the coreset error converges to the error of the random sampling in the third graph (as expected) since the coreset is not recomputed while the noise magnitude becomes larger; in this case the coreset points do not outperform a random sample of the points.

Running Time. To evaluate the running time of our algorithms, we apply them on random data using a laptop with an Intel Core i7-4710HQ CPU @ 2.50GHz processor. We compared the calculation time of the pose estimation on a coreset vs. the full set. This test consists of two cases: (a) Using an increasing number of points while maintaining a constant dimension, (b) Using a constant number of points of different dimensions. The results are shown in Figure 5a,b respectively. The test corresponds to the first row of Table 1. Figure 5a shows that when the coreset size of Algorithm 3 is larger than the number n of points, the computation time is roughly identical, and as n reaches beyond $dr = O(d^2)$, the computation time using the full set of points continues to grow linearly with n ($O(nd^2)$), while the computation time using the coreset, which is dominated by the computation of the optimal rotation, ceases to increase since it is independent of n ($d^3r = O(d^4)$). Figure 5b shows that the coreset indeed yields smaller computation times compared to the full set of points when the dimension $d < \sqrt{n}$, and both yield roughly the same computation time as d reaches $\sqrt{n}$ and beyond.

6.2. IoTracker: A Multicamera Wireless Low-Cost Tracking System

We developed a wireless and low-cost home-made indoor tracking system (<\$100) based on web-cams, IoT mini-computers (hence the name IoTracker), and the algorithms in this paper to compensate the weak hardware; see demonstration video in [30]. The system consists of distributed “client nodes” (one or more) and one “server node”. Each client node contains two components: (A) a mini-computer, Odroid U3 (<\$30) and (B) a pair of standard web-cams (SONY PSEye, <\$5). The server node consists only of a mini-computer. The server node runs the two threads discussed in Section 5.

Figure 4. Comparing the results of methods (i), (ii), and (iii). The X-axis represents the noise magnitude m ; see noise data paragraph in Section 6.1. The Y-axis represents the MSE between the two sets after applying the optimal poses obtained from each of the methods.

Figure 5. Time comparison between calculating the orientation of n points of dimension d given a previously calculated coreset versus using all n points.

Autonomous Quadcopter

We used our tracking system to compute the 6DoF of the quadcopter and send control commands accordingly after reverse engineering its communication protocol. We compared the orientation error of the quadcopter using our coreset as compared to uniform sampling of the IR or visual markers on the quadcopter.

In both tests, the coreset was computed every x frames, the random points were also sampled every x frames, where x is the calculation cycle time. The chosen weighted points were used for the next x frames, and then a new Kabsch-Coreset of size $r(d-1) + 1 = 5$ was computed by Algorithm 3, where $d = 3$ and $r = 2$ as the features on the quadcopter are roughly in a planar configuration. See Section 5 and the video in [30] for demonstrations and results.

Infra-Red (IR) Tracking. Following the common approaches used by the commercial tracking systems, we used IR markers for tracking. We placed 10 Infra-red LEDs on the quadcopter and modified the web-cams' lenses to let only infrared spectrum rays pass, see Figure 6 (left). We could not place more than 10 LEDs on such a microquadcopter because of overweight problem and short battery life. Since the sensorless quadcopter requires a stream of at least 30 control commands per second in order to hover and not crash, we apply the Kabsch algorithm only on a selected a subset of five points.

Our experiments showed that even for such small numbers, choosing the right subset is crucial for a stable system.

Figure 6. (left) 10 IR markers as captured by the web-camera with the IR filter. (right) A Syma X5C sensorless toy microquadcopter. Weight: ~ 100 gr, cost: \$30–\$40.

The system computes the 3D location of each LED using triangulation. Afterwards, it uses Algorithm 3 to compute a Kabsch-Coreset of size $r(d - 1) + 1 = 5$ from the 3D locations, where $d = 3$ and $r = 2$ as the features on the quadcopter are roughly in a planar configuration and samples a random subset (“RANSAC”) of the same size. The ground truth in this test was obtained from the OptiTrack system. The control of the quadcopter based on its positioning was done using a simple PID controller.

For different calculation cycles, we computed the average error throughout the whole test, which consisted of roughly 4500 frames. The results are shown in Figure 7.

Figure 7. IR tracking test: For every calculation cycle (X-axis), we compare between the coresets average error and the uniform random sampling average error. The Y-axis shows the whole test average error for each calculation cycle.

RGB Tracking. To test larger sets of points, we used our tracking system to track visual features (RGB images). We placed a simple planar pattern on a quadcopter; see Figure 1. Due to the time complexity of extracting visual features, we also placed few IR reflective markers and used the OptiTrack motion capture system to perform an autonomous hover with the quadcopter, whilst two other 2D grayscale cameras mounted above the quadcopter collected and tracked visual features from the pattern using

SIFT feature detector; see submitted video. The matching between the SIFT features in both images has some mismatches. This is discussed at the end of Section 1. Given 2D coordinates of the extracted visual features from two cameras, we were able to compute the 3D location of each detected feature using triangulation. As in the IR markers test, a Kabsch-Coreset of size 5 was computed, alongside a random sample of the same size; see Figure 1. The quadcopter's orientation was then estimated by computing the optimal rotation matrix, using the Kabsch algorithm, on both the coreset points and the random sampled points. The ground truth in this test was obtained using the Kabsch algorithm on all the points in the current frame.

For different calculation cycles, we computed the average error throughout the whole test, which consisted of ~ 3000 frames, as shown in Figure 8. The number of detected SIFT features in each frame was 60–100, though most of the features did not last for more than 15 consequent frames; therefore, we tested the coreset with calculation cycles in the range 1 to 15. The average errors were smaller than the average errors in the previous test due to the inaccurate 3D estimation using low-cost hardware in the previous test, e.g., \$5 web-cams as compared to OptiTrack's \$1000 cameras and due to the difference between the ground truth measurements in the two tests.

Figure 8. RGB Tracking test: For every calculation cycle (X-axis), we compare between the coreset average error and the uniform random sampling average error. The Y-axis shows the whole test (3000 frames) average error for each calculation cycle.

7. Conclusions

We demonstrated how coresets that are usually used for solving problems in machine learning or computational geometry can also turn theorems into real-time systems. We suggested new coresets of constant size for kinematic data points in three-dimensional space. This enabled us to compute the Kabsch algorithm in real-time on slow devices by running them on the coreset, while getting provably exactly the same results. In the companion video [30] we demonstrate the first low-cost wireless tracking system that uses coresets and turns a toy quadcopter into a “Guardian Angel” that leads guests to their desired location.

Open problems include extending our coresets for handling outliers, matching between frames, different cost functions and inputs, and multiple rigid bodies.

Author Contributions: Conceptualization, S.N., I.J. and D.F.; software, S.N. and I.J.; validation, S.N. and I.J.; formal analysis, S.N., I.J. and D.F.; writing—original draft preparation, S.N. and I.J.; writing—review and editing, D.F.; visualization, S.N. and I.J.; supervision, D.F. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: We thank Daniela Rus for suggesting the name “Guardian Angel” for our guiding system.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Agarwal, P.K.; Procopiuc, C.M. Approximation algorithms for projective clustering. In Proceedings of the Eleventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), San Francisco, CA, USA, 9–11 January 2000; pp. 538–547.
2. Agarwal, P.K.; Procopiuc, C.M.; Varadarajan, K.R. Approximation Algorithms for k-Line Center. In Proceedings of the 10th Annual European Symposium on Algorithms (ESA), Rome, Italy, 17–21 September 2002; Lecture Notes in Computer Science; Springer: Berlin, Germany 2002; Volume 2461, pp. 54–63.
3. Har-Peled, S. Clustering Motion. *Discrete Comput. Geom.* **2004**, *31*, 545–565. [[CrossRef](#)]
4. Feldman, D.; Monemizadeh, M.; Sohler, C. A PTAS for k -means clustering based on weak coresets. In Proceedings of the 23rd Annual Symposium on Computational Geometry (SoCG '07), Gyeongju, Korea, 6–8 June 2007.
5. Agarwal, P.K.; Har-Peled, S.; Varadarajan, K.R. Geometric Approximations via Coresets. In *Combinatorial and Computational Geometry*; MSRI Publications; Berkeley, CA, USA, 2005; Volume 52, pp. 1–30.
6. Czumaj, A.; Sohler, C. Sublinear-time approximation algorithms for clustering via random sampling. *Random Struct. Algorithms (RSA)* **2007**, *30*, 226–256. [[CrossRef](#)]
7. Phillips, J.M. Coresets and Sketches, Near-Final Version of Chapter 49. In *Handbook on Discrete and Computational Geometry*, 3rd ed.; CRC Press LLC: Boca Raton, FL, USA, 2016.
8. Czumaj, A.; Ergün, F.; Fortnow, L.; Magen, A.; Newman, I.; Rubinfeld, R.; Sohler, C. Approximating the weight of the euclidean minimum spanning tree in sublinear time. *SIAM J. Comput.* **2005**, *35*, 91–109. [[CrossRef](#)]
9. Frahling, G.; Indyk, P.; Sohler, C. Sampling in Dynamic Data Streams and Applications. *Int. J. Comput. Geometry Appl.* **2008**, *18*, 3–28. [[CrossRef](#)]
10. Buriol, L.S.; Frahling, G.; Leonardi, S.; Sohler, C. Estimating Clustering Indexes in Data Streams. In Proceedings of the 15th Annual European Symposium on Algorithms (ESA), Eilat, Israel, 8–10 October 2007; Lecture Notes in Computer Science; Springer: Berlin, Germany 2007; Volume 4698, pp. 618–632.
11. Frahling, G.; Sohler, C. Coresets in dynamic geometric data streams. In Proceedings of the Thirty-Seventh Annual ACM Symposium on Theory of Computing, Baltimore, MD, USA, 22 May 2005; ACM: New York, NY, USA, 2005; pp. 209–217.
12. Feldman, D.; Volkov, M.; Rus, D. Dimensionality Reduction of Massive Sparse Datasets Using Coresets. In *Advances in Neural Information Processing Systems (NIPS)*; MIT Press: Cambridge, MA, USA, 2016.
13. Feldman, D.; Faulkner, M.; Krause, A. Scalable training of mixture models via coresets. In *Advances in Neural Information Processing Systems (NIPS)*; MIT Press: Cambridge, MA, USA, 2011; pp. 2142–2150.
14. Tsang, I.W.; Kwok, J.T.; Cheung, P.M. Core vector machines: Fast SVM training on very large data sets. *J. Mach. Learn. Res.* **2005**, *6*, 363–392.
15. Lucic, M.; Bachem, O.; Krause, A. Strong coresets for hard and soft Bregman clustering with applications to exponential family mixtures. In Proceedings of the 19th International Conference on Artificial Intelligence and Statistics, Cadiz, Spain, 7–11 May 2016; pp. 1–9.
16. Bachem, O.; Lucic, M.; Hassani, S.H.; Krause, A. Approximate k -means++ in sublinear time. In Proceedings of the Conference on Artificial Intelligence (AAAI), Phoenix Convention Center, Phoenix, AZ, USA, 12–17 February 2016.
17. Lucic, M.; Ohannessian, M.I.; Karbasi, A.; Krause, A. Tradeoffs for Space, Time, Data and Risk in Unsupervised Learning. *arXiv* **2015**, arXiv:1605.00529.
18. Bachem, O.; Lucic, M.; Krause, A. Coresets for Nonparametric Estimation—The Case of DP-Means. In Proceedings of the International Conference on Machine Learning (ICML), Lille, France, 6–11 July 2015.

19. Huggins, J.H.; Campbell, T.; Broderick, T. Coresets for Scalable Bayesian Logistic Regression. *arXiv* **2016**, arXiv:1605.06423.
20. Rosman, G.; Volkov, M.; Feldman, D.; Fisher III, J.W.; Rus, D. Coresets for k-segmentation of streaming data. In *Advances in Neural Information Processing Systems (NIPS)*; MIT Press: Cambridge, MA, USA, 2014; pp. 559–567.
21. Reddi, S.J.; Póczos, B.; Smola, A. Communication efficient coresets for empirical loss minimization. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*, Amsterdam, The Netherlands, 12–16 July 2015.
22. Sung, C.; Feldman, D.; Rus, D. Trajectory clustering for motion prediction. In *Proceedings of the 2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, Algarve, Portugal, 7–12 October 2012; IEEE: Piscataway, NJ, USA, 2012; pp. 1547–1552.
23. Feldman, D.; Sugaya, A.; Sung, C.; Rus, D. iDiary: from GPS signals to a text-searchable diary. In *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, Roma, Italy, 11–15 November 2013; ACM: New York, NY, USA, 2013; p. 6.
24. Feldman, D.; Xian, C.; Rus, D. *Private Coresets for High-Dimensional Spaces*; Technical Report; ACM Digital Library: New York, NY, USA, 2016.
25. Feigin, M.; Feldman, D.; Sochen, N. From high definition image to low space optimization. In *Proceedings of the International Conference on Scale Space and Variational Methods in Computer Vision*, Ein-Gedi, Israel, 29 May–2 June 2011; Springer: Berlin/Heidelberg, Germany, 2011; pp. 459–470.
26. Feldman, D.; Feigin, M.; Sochen, N. Learning big (image) data via coresets for dictionaries. *J. Mathem. Imaging Vis.* **2013**, *46*, 276–291. [[CrossRef](#)]
27. Alexandroni, G.; Moreno, G.Z.; Sochen, N.; Greenspan, H. Coresets versus clustering: Comparison of methods for redundancy reduction in very large white matter fiber sets. In *Medical Imaging 2016: Image Processing*; SPIE Medical Imaging; International Society for Optics and Photonics: Bellingham, WA, USA, 2016; p. 97840A.
28. Stanway, M.J.; Kinsey, J.C. Rotation Identification in Geometric Algebra: Theory and Application to the Navigation of Underwater Robots in the Field. *J. Field Robot.* **2015**, *32*, 632–654. [[CrossRef](#)]
29. MIT Senseable City Lab. SkyCall Video, 2016. Available online: <https://www.youtube.com/watch?v=mB9NfEj0ZVs> (accessed on 9 May 2020).
30. Nasser, S.; Jubran, I.; Feldman, D. System Demonstration Video, 2020. Available online: https://drive.google.com/open?id=1HN1iY2Ti_d-akUXKJgckDKh7rmZUyXWG (accessed on 9 May 2020).
31. Wahba, G. A least squares estimate of satellite attitude. *SIAM Rev.* **1965**, *7*, 409. [[CrossRef](#)]
32. Kabsch, W. A solution for the best rotation to relate two sets of vectors. *Acta Crystallogr. Sect. A Cryst. Phys. Diff. Theor. Gen. Crystallogr.* **1976**, *32*, 922–923. [[CrossRef](#)]
33. Lepetit, V.; Moreno-Noguer, F.; Fua, P. Epnp: An accurate O(n) solution to the pnp problem. *Int. J. Comput. Vis.* **2009**, *81*, 155–166. [[CrossRef](#)]
34. Besl, P.J.; McKay, N.D. Method for registration of 3-D shapes. In *Robotics-DL Tentative*; International Society for Optics and Photonics, SPIE Digital Library: Bellingham, WA, USA, 1992; pp. 586–606.
35. Wang, L.; Sun, X. Comparisons of Iterative Closest Point Algorithms. In *Ubiquitous Computing Application and Wireless Sensor*; Springer: Berlin, Germany, 2015; pp. 649–655.
36. Friedman, J.H.; Bentley, J.L.; Finkel, R.A. An algorithm for finding best matches in logarithmic expected time. *ACM Trans. Math. Softw. (TOMS)* **1977**, *3*, 209–226. [[CrossRef](#)]
37. Zhang, Z. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vis.* **1994**, *13*, 119–152. [[CrossRef](#)]
38. Pulli, K.; Shapiro, L.G. Surface reconstruction and display from range and color data. *Graph. Models* **2000**, *62*, 165–201. [[CrossRef](#)]
39. Ahuja, R.K.; Magnanti, T.L.; Orlin, J.B. *Network Flows: Theory, Algorithms, and Applications*; Physica-Verlag: Wurzburg, Germany, 1993.
40. Valencia, C.E.; Vargas, M.C. Optimum matchings in weighted bipartite graphs. In *Boletín de la Sociedad Matemática Mexicana*; Springer: Berlin, Germany, 2015.
41. Batson, J.; Spielman, D.A.; Srivastava, N. Twice-ramanujan sparsifiers. *SIAM J. Comput.* **2012**, *41*, 1704–1721. [[CrossRef](#)]

42. Cohen, M.B.; Nelson, J.; Woodruff, D.P. Optimal approximate matrix product in terms of stable rank. *arXiv* **2015**, arXiv:1507.02268.
43. Ghashami, M.; Liberty, E.; Phillips, J.M.; Woodruff, D.P. Frequent directions: Simple and deterministic matrix sketching. *SIAM J. Comput.* **2016**, *45*, 1762–1792. [[CrossRef](#)]
44. Cohen, M.B.; Elder, S.; Musco, C.; Musco, C.; Persu, M. Dimensionality reduction for k-means clustering and low rank approximation. In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, Portland, OR, USA, 15–17 June 2015; ACM: New York, NY, USA, 2015; pp. 163–172.
45. Feldman, D.; Schmidt, M.; Sohler, C. Turning big data into tiny data: Constant-size coresets for k-means, pca and projective clustering. In Proceedings of the Twenty-Fourth Annual ACM-SIAM Symposium on Discrete Algorithms, New Orleans, LA, USA, 6–8 January 2013; SIAM: Philadelphia, PA, USA, 2013; pp. 1434–1453.
46. De Carli Silva, M.K.; Harvey, N.J.A.; Sato, C.M. Sparse Sums of Positive Semidefinite Matrices. *arXiv* **2011**, arXiv:1107.0088v2
47. Carathéodory, C. Über den Variabilitätsbereich der Fourier'schen Konstanten von positiven harmonischen Funktionen. In *Rendiconti del Circolo Matematico di Palermo (1884–1940)*; Springer: Berlin/Heidelberg, Germany, 1911; Volume 32, pp. 193–217.
48. Indyk, P.; Mahabadi, S.; Mahdian, M.; Mirrokni, V.S. Composable core-sets for diversity and coverage maximization. In Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems, Snowbird, UT, USA, 22–27 July 2014; ACM: New York, NY, USA, 2014; pp. 100–108.
49. Mirrokni, V.; Zadimoghaddam, M. Randomized composable core-sets for distributed submodular maximization. In Proceedings of the Forty-Seventh Annual ACM on Symposium on Theory of Computing, Portland, OR, USA, 15–17 June 2015; ACM: New York, NY, USA, 2015; pp. 153–162.
50. Aghamolaei, S.; Farhadi, M.; Zarrabi-Zadeh, H. Diversity Maximization via Composable Coresets. In Proceedings of the 27th Canadian Conference on Computational Geometry (CCCG 2015), Kingston, ON, Canada, 10–12 August 2015.
51. Agarwal, P.K.; Cormode, G.; Huang, Z.; Phillips, J.M.; Wei, Z.; Yi, K. Mergeable summaries. *ACM Trans. Database Syst. (TODS)* **2013**, *38*, 26. [[CrossRef](#)]
52. Har-Peled, S.; Mazumdar, S. On coresets for k-means and k-median clustering. In Proceedings of the Thirty-Sixth Annual ACM Symposium on Theory of Computing, Chicago, IL, USA, 13–15 June 2004; ACM: New York, NY, USA, 2004; pp. 291–300.
53. Har-Peled, S.; Kushal, A. Smaller coresets for k-median and k-means clustering. *Discret. Comput. Geom.* **2007**, *37*, 3–19. [[CrossRef](#)]
54. Chen, K. On coresets for k-median and k-means clustering in metric and euclidean spaces and their applications. *SIAM J. Comput.* **2009**, *39*, 923–947. [[CrossRef](#)]
55. Langberg, M.; Schulman, L.J. Universal ϵ -approximators for integrals. In Proceedings of the Twenty-First Annual ACM-SIAM Symposium on Discrete Algorithms, Austin, TX, USA, 17–19 January 2010; SIAM: Philadelphia, PA, USA, 2010; pp. 598–607.
56. Feldman, D.; Langberg, M. A Unified Framework for Approximating and Clustering Data. In Proceedings of the 34th Annual ACM Symposium on Theory of Computing (STOC), San Jose, CA, USA, 6–8 June 2011.
57. Barger, A.; Feldman, D. k-Means for Streaming and Distributed Big Sparse Data. In Proceedings of the 2016 SIAM International Conference on Data Mining (SDM'16), Miami, FL, USA, 5–7 May 2016.
58. Feldman, D.; Tassa, T. More constraints, smaller coresets: Constrained matrix approximation of sparse big data. In Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'15), Sydney, Australia, 10–13 August 2015; ACM: New York, NY, USA, 2015; pp. 249–258.
59. Kjer, H.M.; Wilm, J. Evaluation of Surface Registration Algorithms for PET Motion Correction. Ph.D. Thesis, Technical University of Denmark, Lyngby, Denmark, 2010.