Curve Decimation in SE(2) and SE(3)

Generalization of the Ramer-Douglas-Peucker Algorithm to Lie Groups

by Jan Hakenberg, 2019-09-08, ETH Zürich

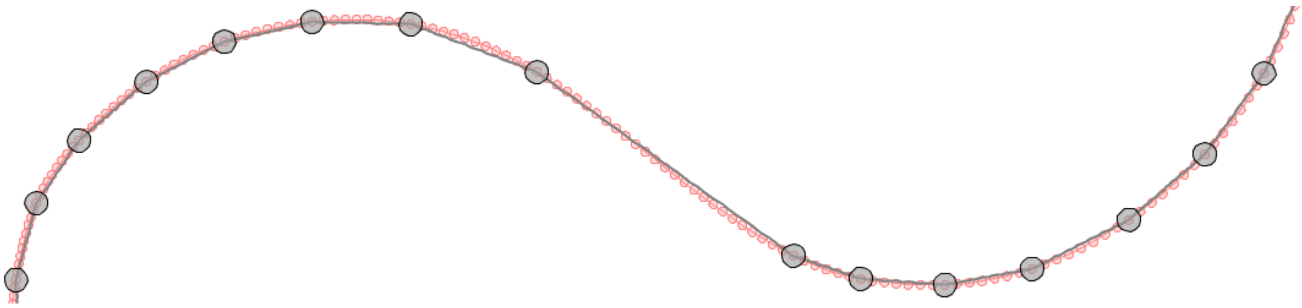


Figure: The original Ramer-Douglas-Peucker algorithm operates on a sequence of points in $\mathbb{R}^2$ (in red). The output is a subset of points from the original sequence (indicated in gray) between which the connecting lines are guaranteed not to deviate more than a given threshold from the input points. ■

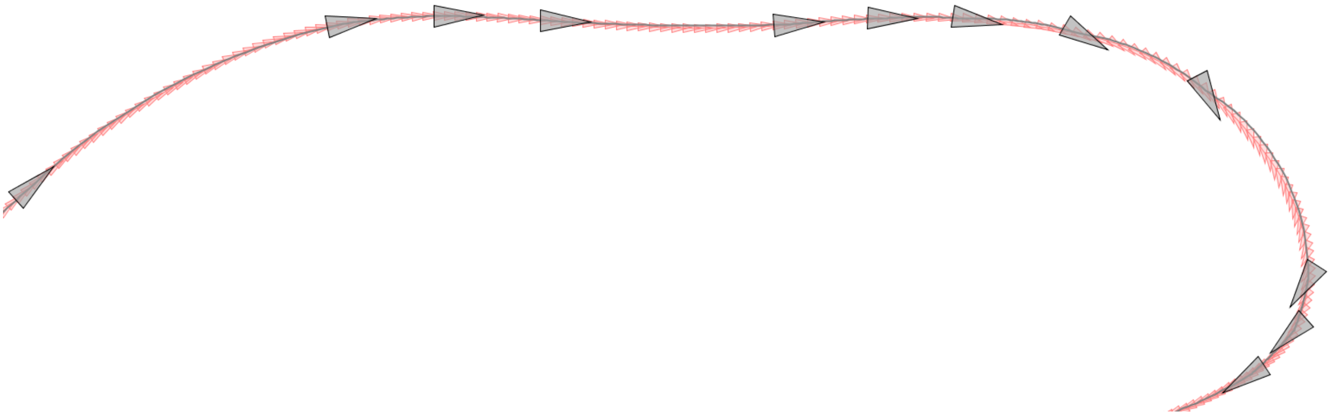


Figure: The curve decimation algorithm presented in the document generalizes the notion of straight lines in $\mathbb{R}^n$ to geodesics in Lie groups. When applied to sequences in SE(2) the error combines the deviation in translation as well as in orientation. ■

Abstract

We generalize the Ramer-Douglas-Peucker algorithm to operate on a sequence of elements from a Lie group. As the original, the new algorithm bounds the approximation error, and has an expected runtime complexity of $O(n \log n)$.

We apply the curve decimation to data recorded from a car-like robot in SE(2), as well as from a drone in SE(3). The results show that many samples of the original sequence can be dropped while maintaining a high-quality approximation to the original trajectory.

Overview

Quote: "*The Ramer-Douglas-Peucker algorithm decimates a curve composed of line segments to a similar curve with fewer points. [...] The algorithm defines 'dissimilar' based on the maximum distance between the original curve and the simplified curve. [...] The expected complexity of this algorithm can be described by the linear recurrence* $T(n) = 2\,T(n/2) + O(n)$*, which has the well-known solution* $O(n \log n)$*. However, the worst-case complexity is* $O(n^2)$." [Wikipedia] ■

For sequences in $\mathbb{R}^n$, the deviation of a point $r \in \mathbb{R}^n$ from the line that connects the points $p, q \in \mathbb{R}^n$ is measured by projecting the point r to the $n-1$-dimensional subspace at p orthogonal to the direction $q - p$. The length of the projected vector is measured using the Euclidean norm.

For sequences in a Lie group G, the deviation of a point $r \in G$ from the geodesic that connects the points $p, q \in G$ is measured by projecting the point r to the $n-1$-dimensional subspace of the tangent space $T_p G$ orthogonal to the direction $\log p^{-1}.q$. The length of the projected vector is measured using a custom scalar product.

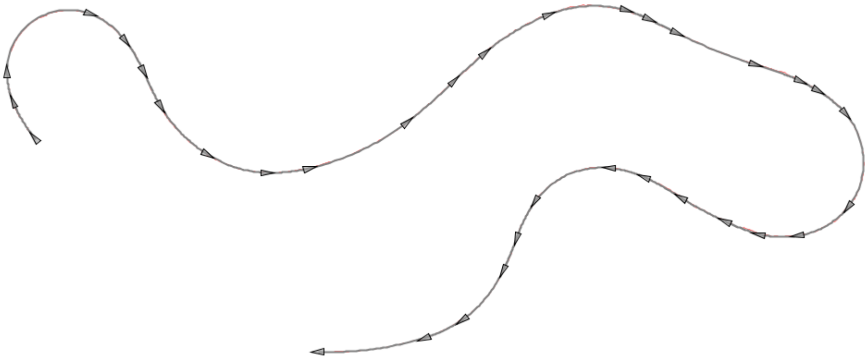
Implementation

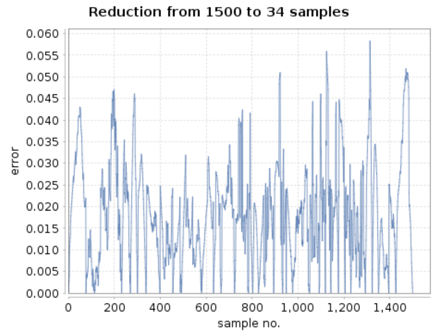
The implementation of the curve decimation for the Lie groups SE(2) and SE(3) is open-source at [IDSC-Frazzoli].

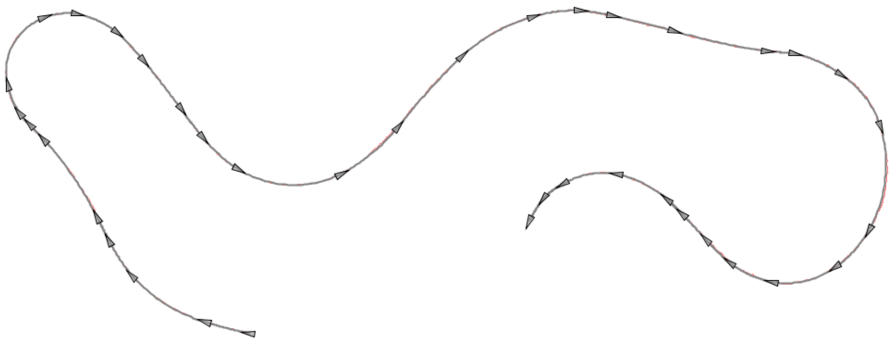
```
/* package */ class LieGroupCurveDecimation implements CurveDecimation {
  private static final TensorUnaryOperator NORMALIZE_UNLESS_ZERO = NormalizeUnlessZero.with(Norm._2);
  // ---
  private final LieGroup lieGroup;
  private final TensorUnaryOperator log;
  private final Scalar epsilon;

  /** @param lieGroup
   * @param log
   * @param epsilon */
  public LieGroupCurveDecimation(LieGroup lieGroup, TensorUnaryOperator log, Scalar epsilon) {
    this.lieGroup = lieGroup;
    this.log = log;
    this.epsilon = epsilon;
  }

  private class LieGroupResult implements Result, Serializable {
    private final Tensor[] tensors;
    private final Scalar[] scalars;
    private final List<Integer> list = new LinkedList<>();

    /** @param tensor of length at least 1 */
    public LieGroupResult(Tensor tensor) {
      tensors = tensor.stream().toArray(Tensor[]::new);
      scalars = new Scalar[tensors.length];
      int end = tensors.length - 1;
      recur(0, end);
      scalars[end] = epsilon.zero();
      if (0 < end)
        list.add(end);
    }

    private void recur(int beg, int end) {
      Scalar max = epsilon.zero();
      scalars[beg] = max;
      if (beg + 1 < end) { // at least one element in between beg and end
        LieGroupElement lieGroupElement = lieGroup.element(tensors[beg]).inverse();
        Tensor normal = NORMALIZE_UNLESS_ZERO.apply(log.apply(lieGroupElement.combine(tensors[end])));
        int mid = -1;
        for (int index = beg + 1; index < end; ++index) {
          Tensor vector = log.apply(lieGroupElement.combine(tensors[index]));
          Scalar dist = Norm._2.ofVector(vector.subtract(vector.dot(normal).pmul(normal)));
          scalars[index] = dist;
          if (Scalars.lessThan(max, dist)) {
            max = dist;
            mid = index;
          }
        }
        if (Scalars.lessThan(epsilon, max)) {
          recur(beg, mid);
          recur(mid, end);
          return;
        }
      }
      list.add(beg);
    }

    @Override // from Result
    public Tensor result() {
      return Tensor.of(list.stream().map(i -> tensors[i]).map(Tensor::copy));
    }

    @Override // from Result
    public Tensor errors() {
      return Tensors.of(scalars);
    }
  }
}
```

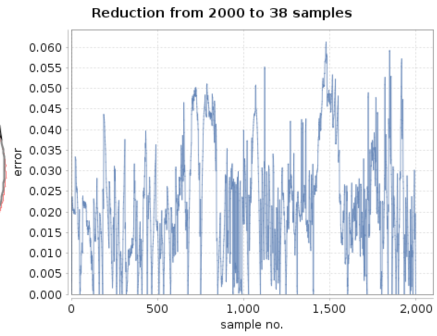
Examples in SE(2) • Go-kart

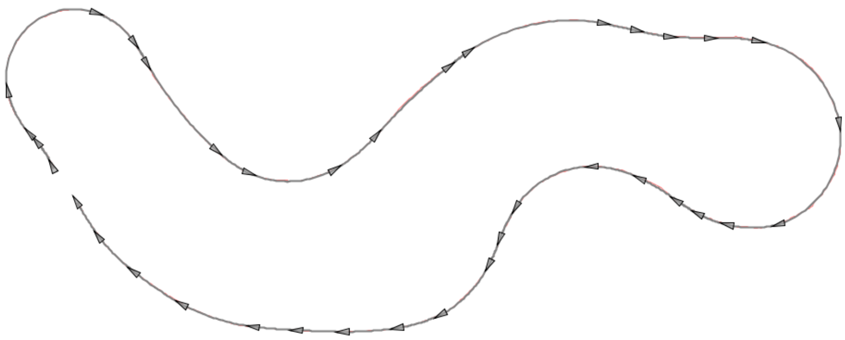
The datasets below were recorded during the operation of the autonomous go-kart of IDSC/ETH Zürich, available at [Ephemeral]. The datasets consist of slow laps as well as fast laps that involve side drift. We apply the new curve decimation algorithm with the approximation error guaranteed to be below **0.064[m] AND 3.667[deg]**.

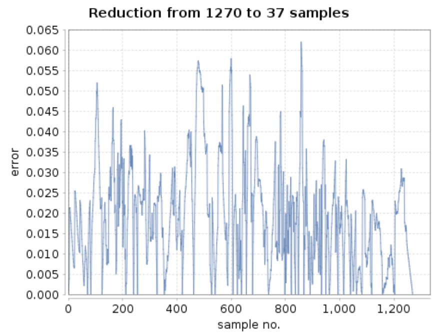


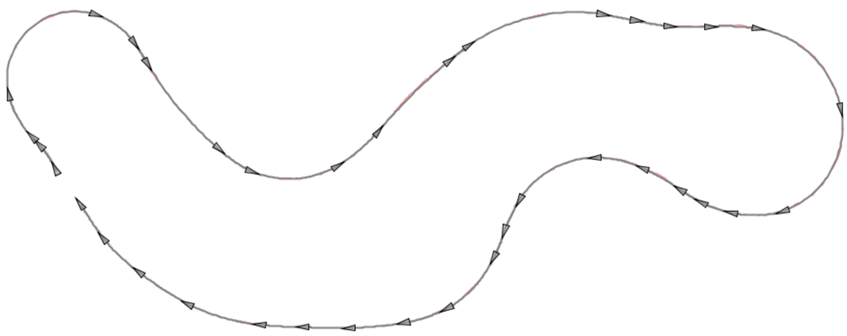


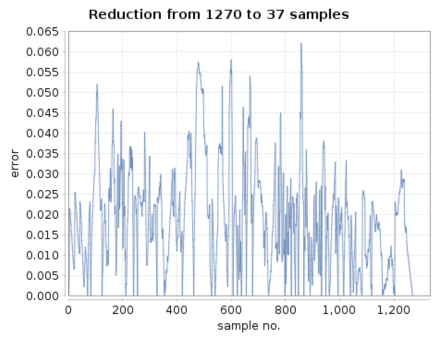


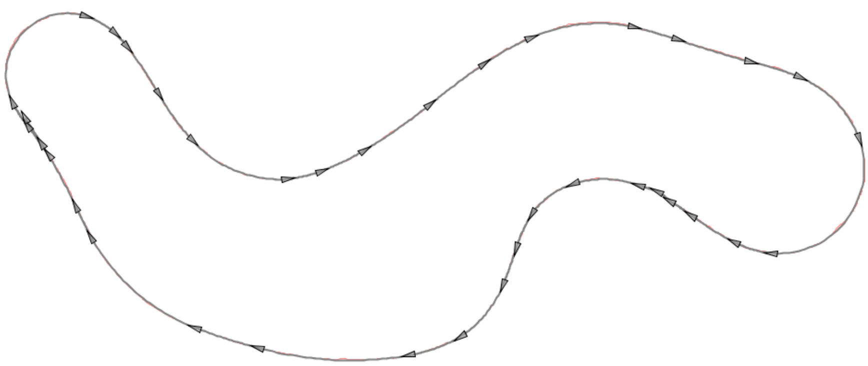


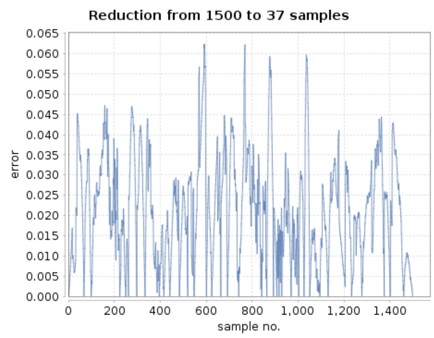


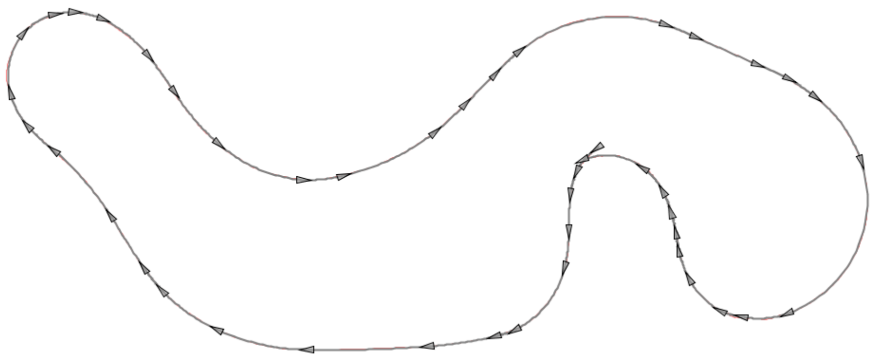


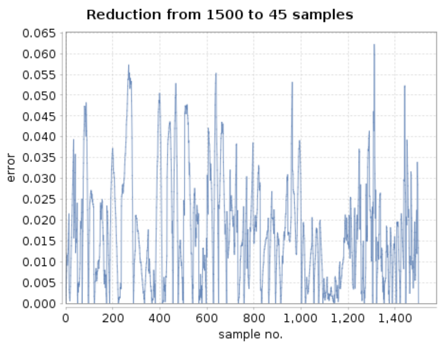


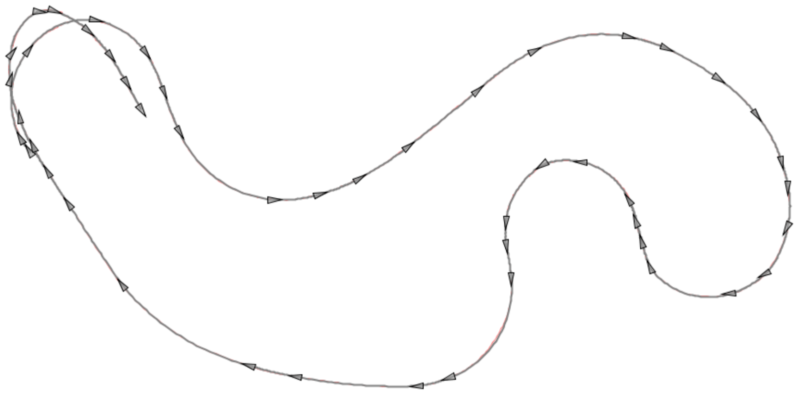


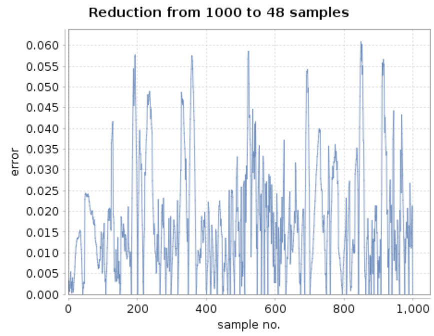


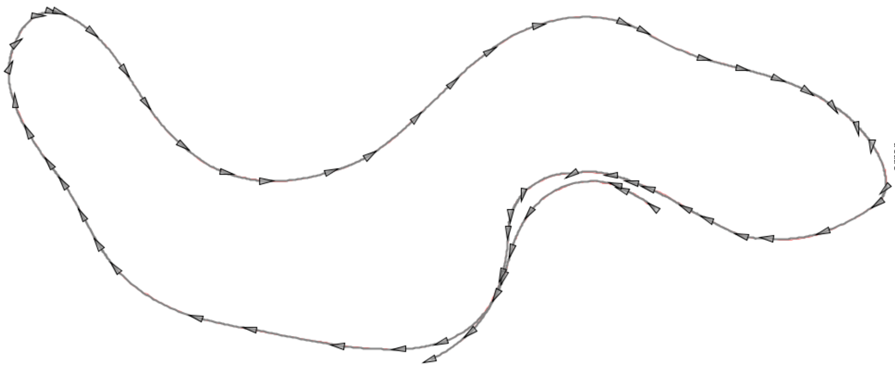


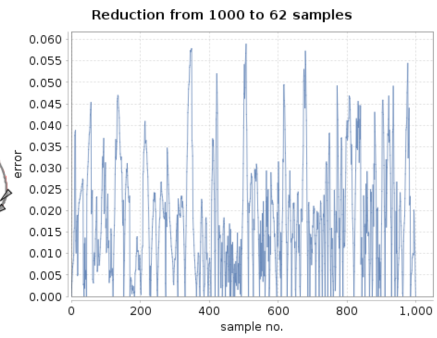


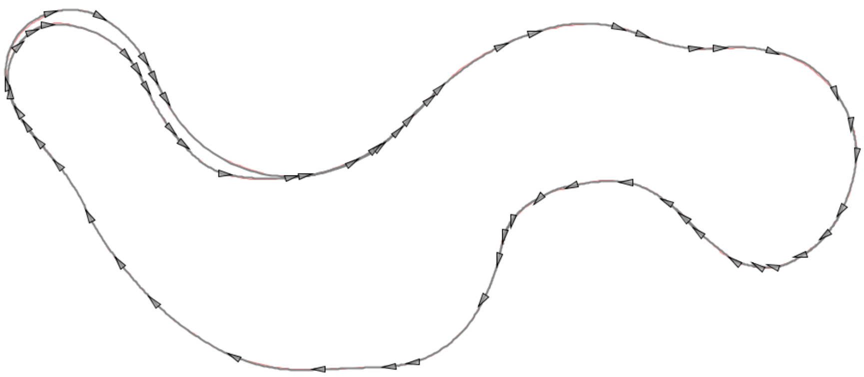

Reduction from 1500 to 45 samples
0.065
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
error
0
200
400
600
800
1,000
1,200
1,400
sample no.

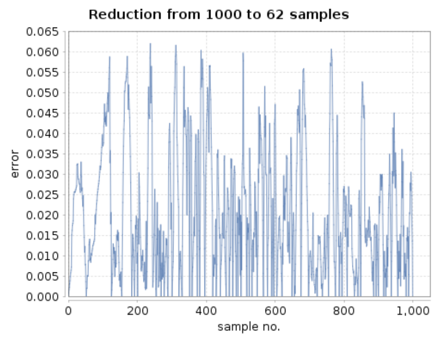


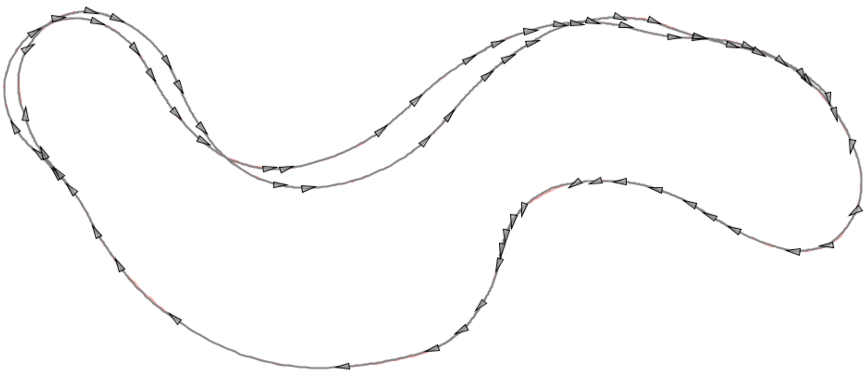

Reduction from 1000 to 48 samples
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
error
0
200
400
600
800
1,000
sample no.

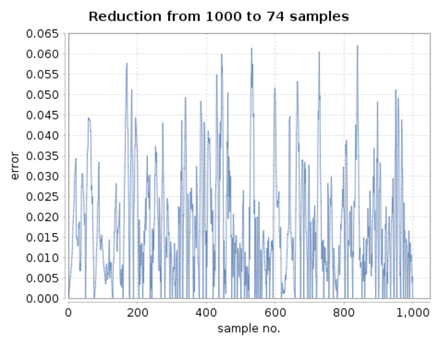


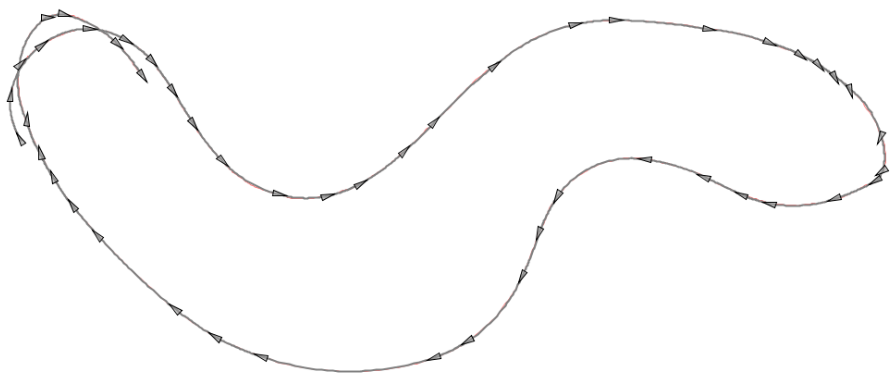

Reduction from 1000 to 62 samples
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
error
0
200
400
600
800
1,000
sample no.

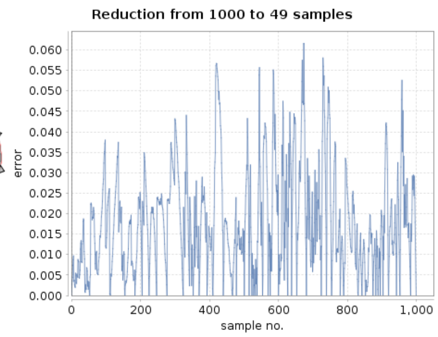



Reduction from 1000 to 62 samples
0.065
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
error
0
200
400
600
800
1,000
sample no.




Reduction from 1000 to 74 samples
0.065
0.060
0.055
0.050
0.045
0.040
0.035
0.030
0.025
0.020
0.015
0.010
0.005
0.000
error
0
200
400
600
800
1,000
sample no.





Examples in SE(3) • Drone Racing Quadrotor

We use the "*UZH-FPV Drone Racing dataset, which is the most aggressive visual-inertial odometry dataset to date" by* [Delmerico, Cieslewski, Rebecq, Faessler, Scaramuzza]. *"Large accelerations, rotations, and apparent motion in vision sensors make aggressive trajectories difficult for state estimation.*"

The (x, y)-coordinates of the trajectories in the datasets range over 20-30[m]. We set the curve decimation to guarantee a deviation below **0.02[m] AND 1.14592[deg]**. The curve decimation results in sequences that consist of **only ~1% of the original data**.

Code

```
frm[m_] := With[{t = m[[{1, 2, 3}, 4]], r = 0.4 m[[{1, 2, 3}, {1, 2, 3}]]}, {
   Red, Line[{t, t + r[[1]]}], Green, Line[{t, t + r[[2]]}], Blue, Line[{t, t + r[[3]]}]}]
```

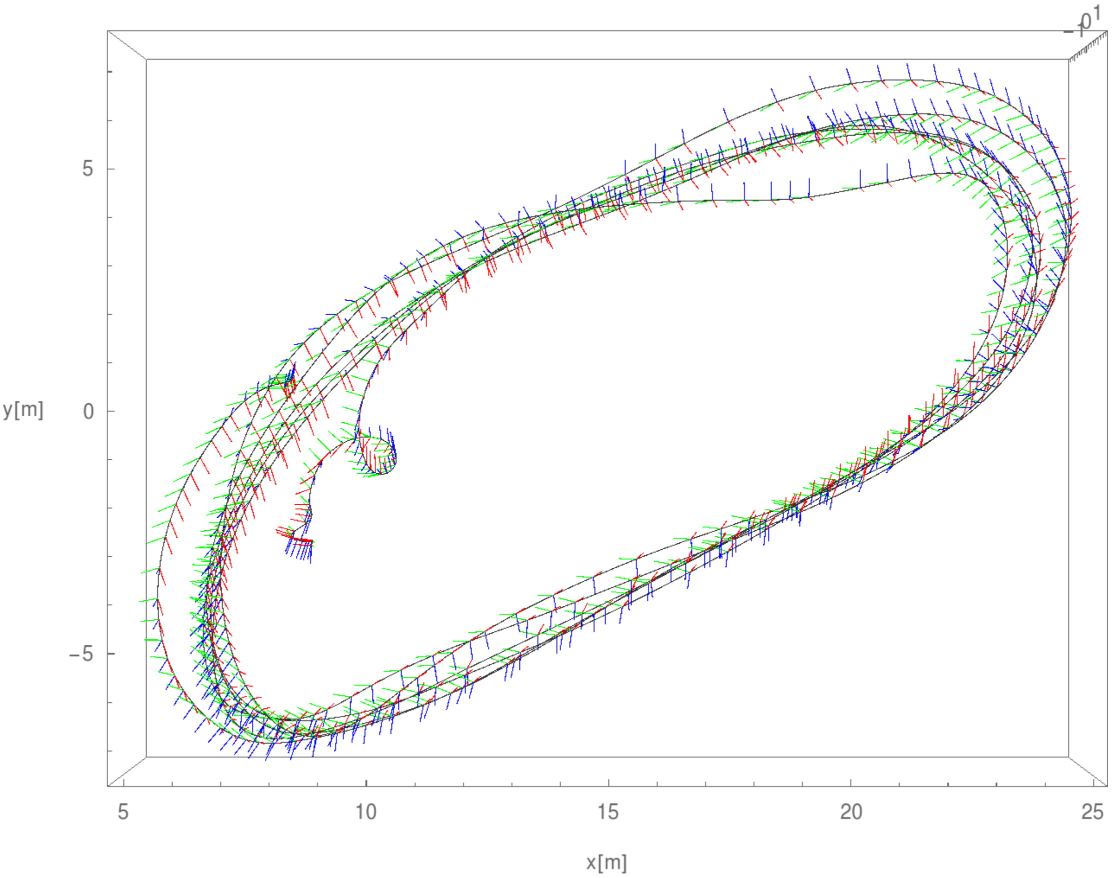
```
shw[name_] := Module[{A = Get[name <> "/poses.file"],
   B = Get[name <> "/decimated.file"], error = Get[name <> "/error.file"]},
  Print[{name, ToString[Length[A]] <> " → " <> ToString[Length[B]] <> " samples"}];
  Print[Rasterize[Graphics3D[{GrayLevel[.2], Line[A[[All, {1, 2, 3}, 4]]], frm /@ B},
     Axes → True, ViewPoint → Above, AxesLabel → {"x[m]", "y[m]"}],
    ImageSize → Large, RasterSize → 1280]];
  Print[Rasterize@ListPlot[error, Joined → True, AxesLabel → {"sample#", "error[m==rad]"},
     PlotStyle → Opacity[.2], PlotRange → All]]]
```

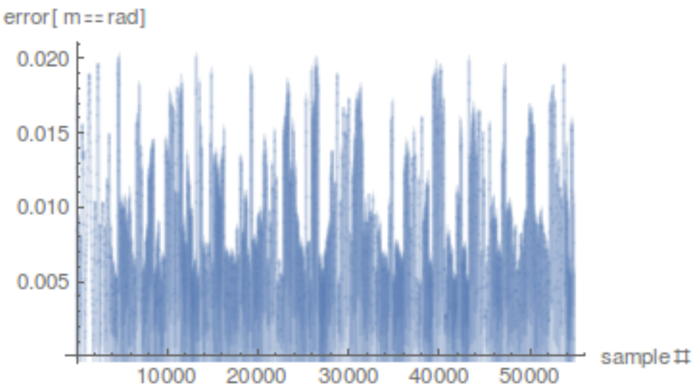
```
fns = FileNames["Documents/uzh/*"];
```

indoor_forward_3_davis

```
shw[fns⟦1⟧]
```

```
{Documents/uzh/indoor_forward_3_davis, 54868 → 568 samples}
```

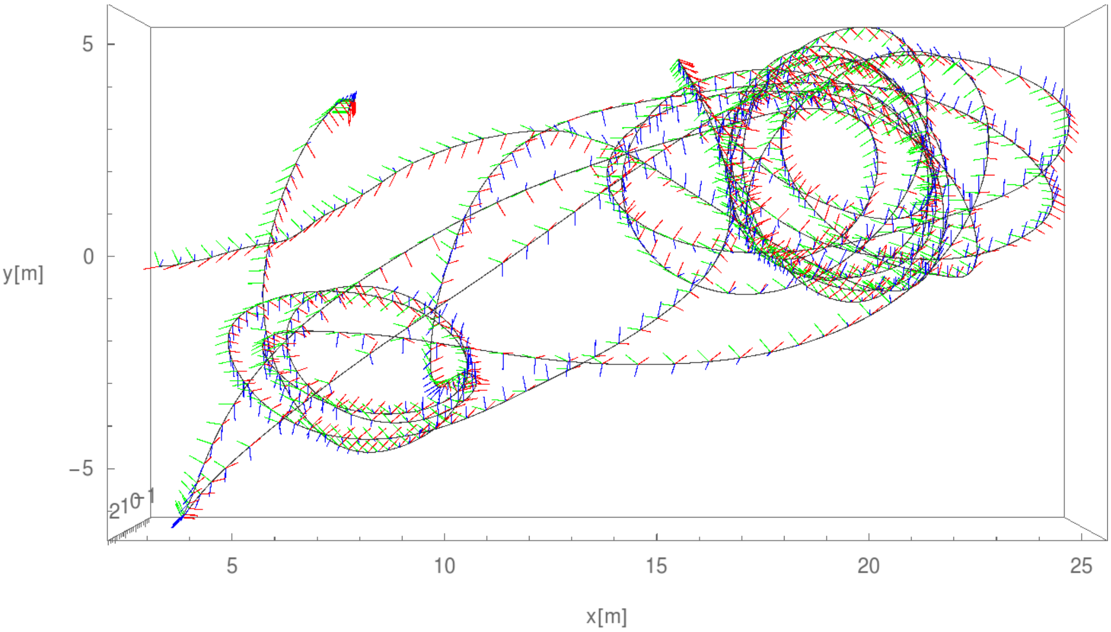


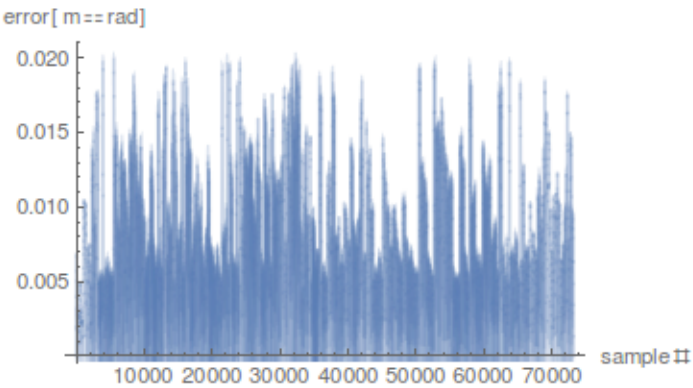


indoor_forward_7_davis

shw[fns⟦4⟧]

```
{Documents/uzh/indoor_forward_7_davis, 73233 → 830 samples}
```

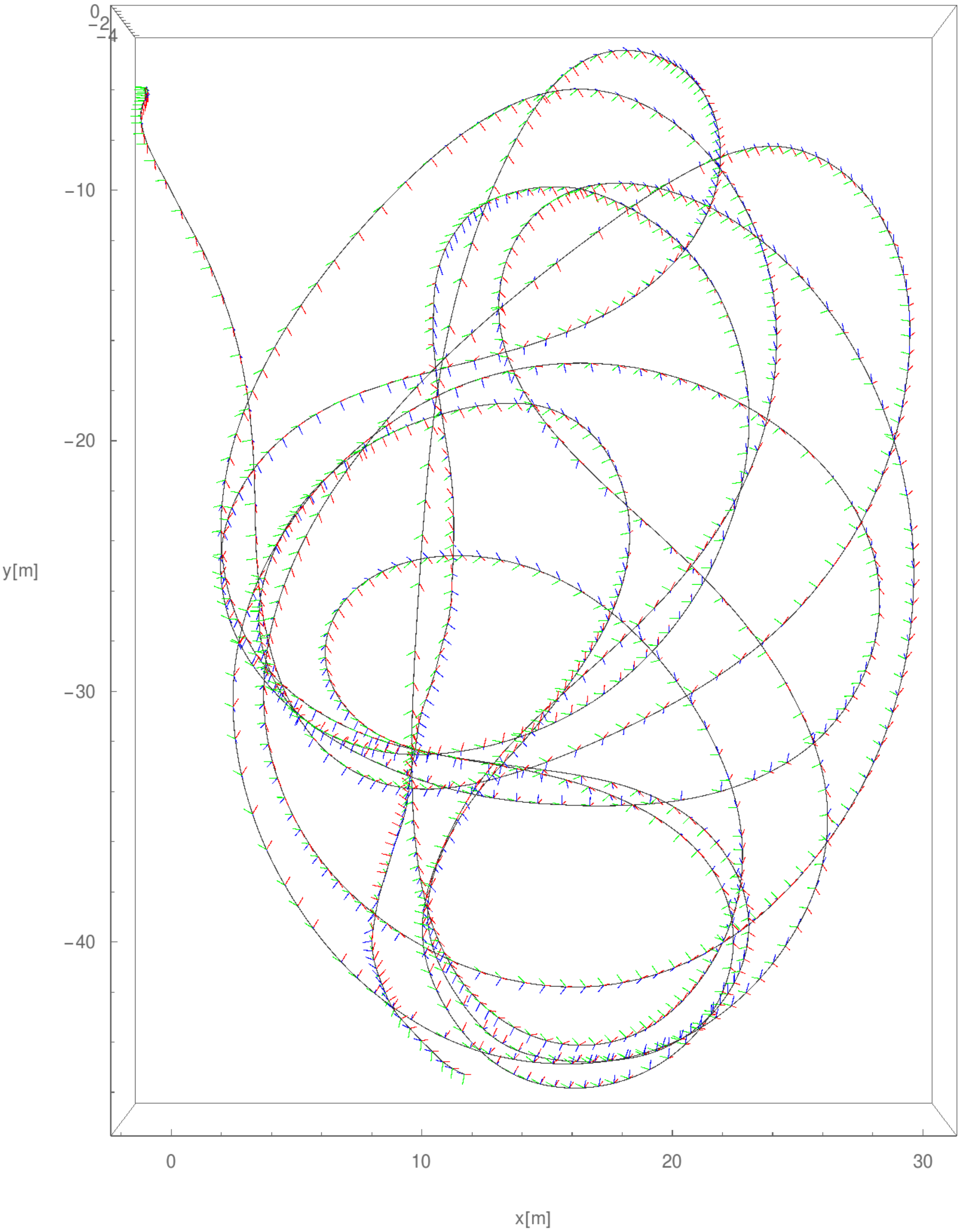


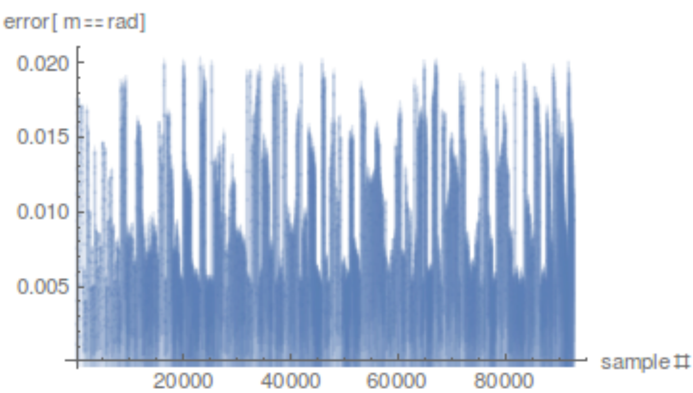


outdoor_forward_3_davis

shw[fns⟦8⟧]

{Documents/uzh/outdoor_forward_3_davis, 92826 → 929 samples}





References

[Wikipedia] *Ramer-Douglas-Peucker algorithm*
https://en.wikipedia.org/wiki/Ramer%E2%80%93Douglas%E2%80%93Peucker_algorithm, 2019

[Delmerico, Cieslewski, Rebecq, Faessler, Scaramuzza]
Are We Ready for Autonomous Drone Racing? The UZH-FPV Drone Racing Dataset, 2019
http://rpg.ifi.uzh.ch/uzh-fpv.html, 2019

[IDSC-Frazzoli] *Library for non-linear geometry computation*, 2019
https://github.com/idsc-frazzoli/owl

[Ephemeral]
https://github.com/idsc-frazzoli/ephemeral