Feature Matching

Course 3, Module 2, Lesson 3 – Part 1

UNIVERSITY OF TORONTO
FACULTY OF APPLIED SCIENCE & ENGINEERING

Learning Objectives

- Learn how to match features based on a predefined distance function
- Learn how to perform **brute force** matching
- List faster alternatives to brute force matching, when the number of features to be matched is high

Image Features: A General Process

Images from CITYSCAPES dataset: https://www.cityscapes-dataset.com/

$$\{f_1, \ldots \ldots \ldots, f_N\}$$

Feature Matching

- **Feature Matching:** Given a **feature and its descriptor**

  in image 1, find the best match in image 2

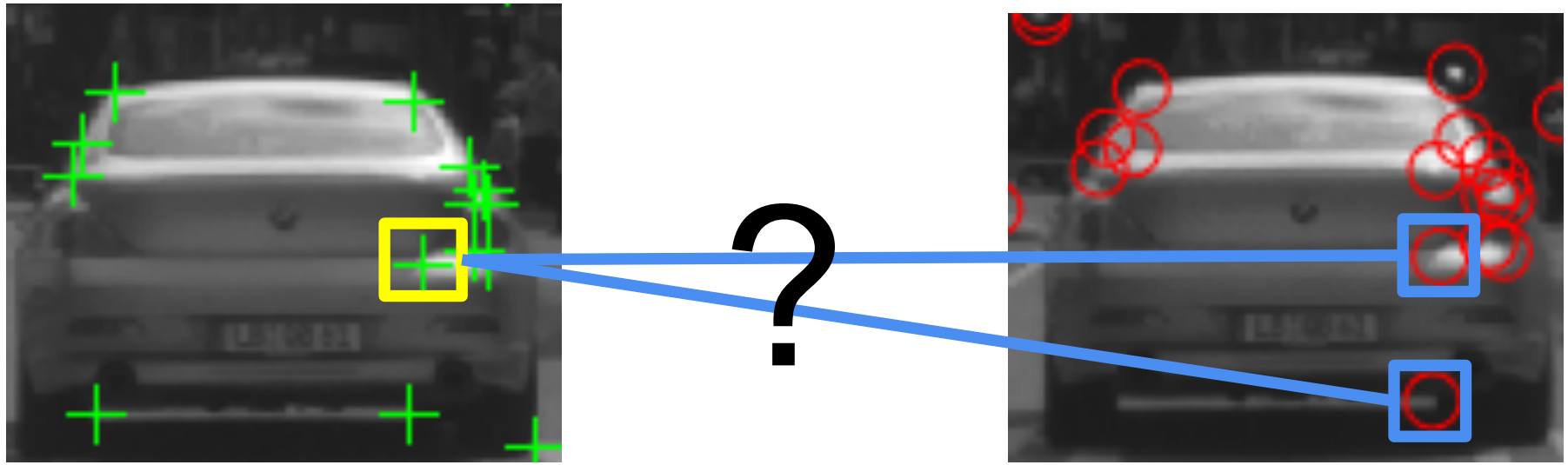


Image 1 Image 2

Brute Force Feature Matching

- Define a **distance function** $d(f_i, f_j)$ that compares the two descriptors
- For every feature f_i in Image 1:
  - Compute $d(f_i, f_j)$ with all features f_j in image 2
  - Find the **closest** match f_c, the match that has the minimum distance

Distance Function

- **Sum of Squared Differences (SSD):**

$$d(f_i, f_j) = \sum_{k=1}^{D} (f_{i,k} - f_{j,k})^2$$

- Other distance functions:
  - **Sum of absolute differences (SAD):**

$$d(f_i, f_j) = \sum_{k=1}^{D} |f_{i,k} - f_{j,k}|$$

  - **Hamming Distance:**

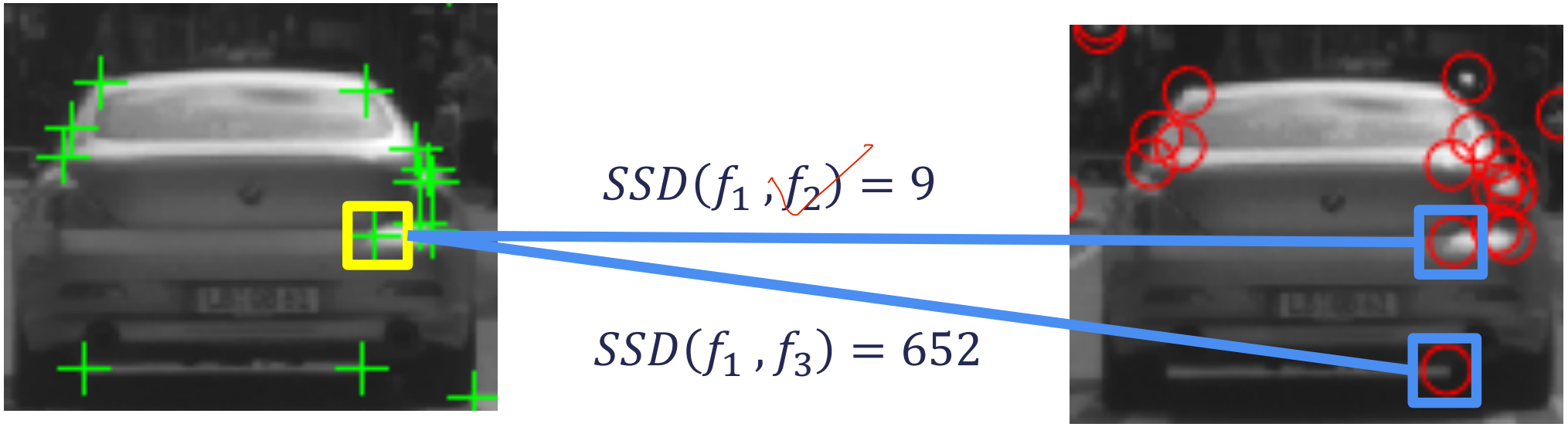
Binary features

$$d(f_i, f_j) = \sum_{k=1}^{D} XOR(f_{i,k}, f_{j,k})$$

Brute Force Feature Matching

$f_1 = [10, 34, 23, 55]$

$f_2 = [10, 37, 23, 55]$

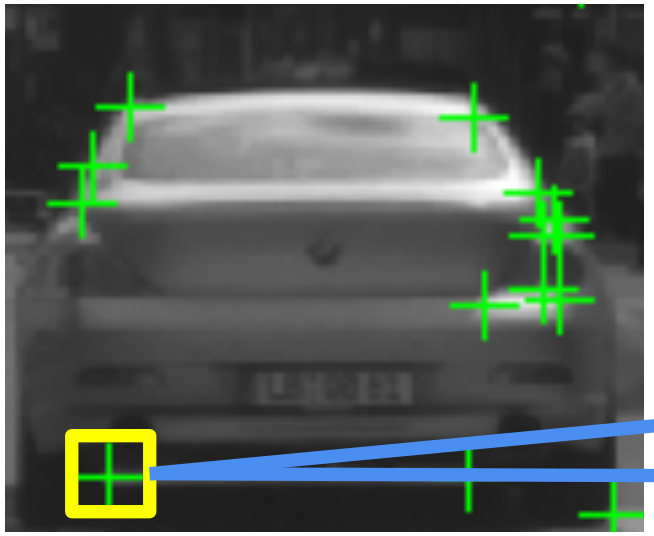


$f_3 = [9, 35, 12, 32]$

Brute Force Feature Matching

$f_1 = [10, 34, 23, 55]$

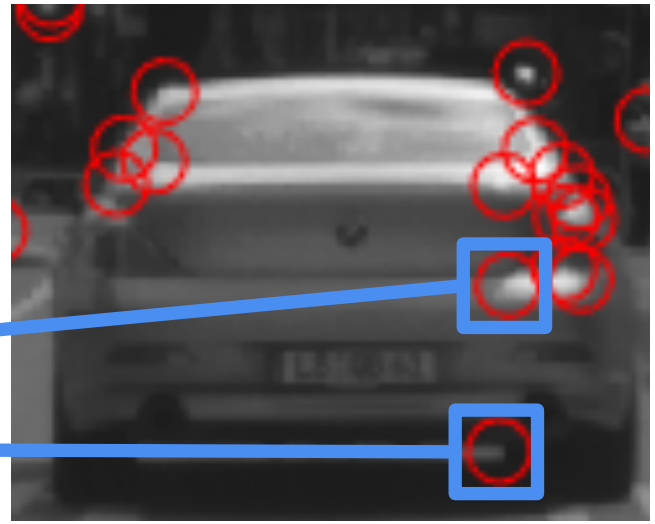
$f_2 = [10, 13, 23, 55]$



$\delta = 20$

$SSD(f_1, f_2) = 441$

$SSD(f_1, f_3) = 652$



$f_3 = [9, 35, 12, 32]$

Brute Force Feature Matching

- Define a distance function $d(f_i, f_j)$ that compares the two descriptors
- Define distance threshold $\boldsymbol{\delta}$
- For every feature f_i in Image 1:
  - Compute $d(f_i, f_j)$ with all features fj in image 2
  - Find the **closest** match f_c, the match that has the minimum distance
  - Keep this match only if $d(f_i, f_j)$ is bellow threshold $\boldsymbol{\delta}$

Feature Matching

- **Brute force** feature matching might not be fast enough for extremely large amounts of features
- Use a multidimensional search tree, usually a **k-d tree** to speed the search by constraining it spatially
- Both of these matchers are implemented in OpenCV as:
  - **cv2.BFMatcher()**: Brute force matcher
  - **cv2.FlannBasedMatcher()**: K-D tree based approximate nearest neighbor matcher
  - Link : https://docs.opencv.org/3.4.3/dc/dc3/tutorial_py_matcher.html