Laboratory 4: Image Mosaicing
(Including Final Projects Intro)

Computer Vision 2022

P. Zanuttigh

CV 2022: HW / Projects

1. Introduction to C++ / OpenCV

No HW

2. **Histogram and Filtering**
3. Road Line Detection

- Select 1 among 2 and 3
- First «simple» HW: 3 pts
- Provide a very short report or comments in the source
- On/off mark

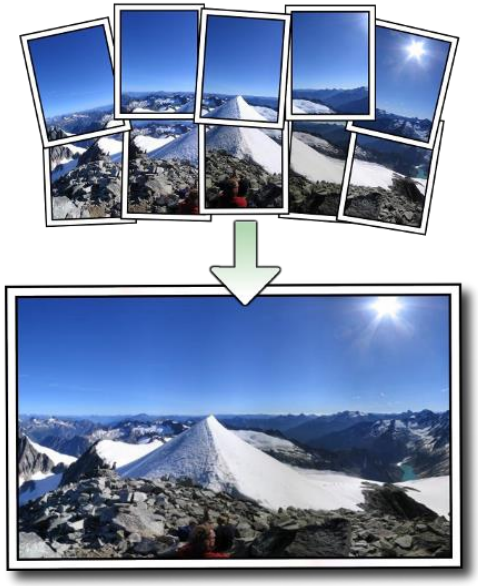
4. **Image Mosaicing**
5. **Parking Spaces Det.**

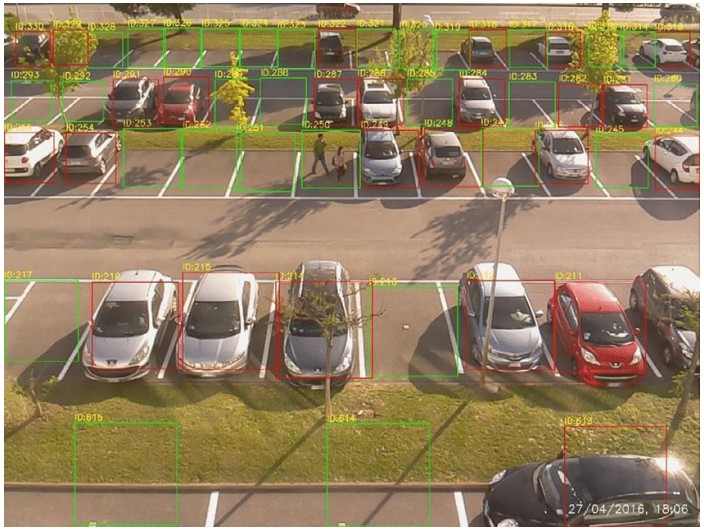
- Select 1 among 4 and 5
- Second «advanced» HW: 6 points
- More detailed report with results
- Mark based on solution quality

Final Exam (written in classroom): ~23 points

Final mark: 3 + 6 + 23 = max 32 points

CV 2022: Final Projects

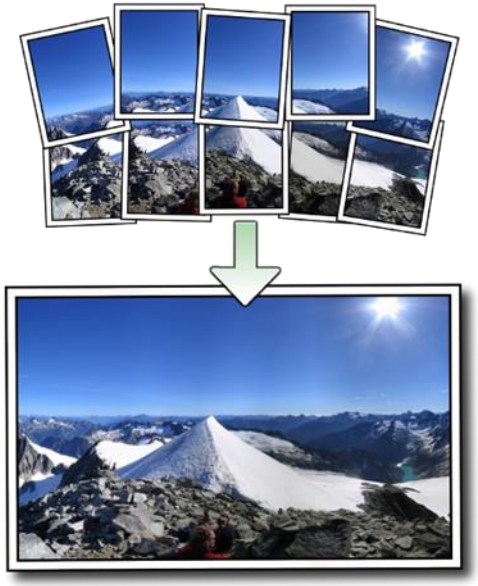


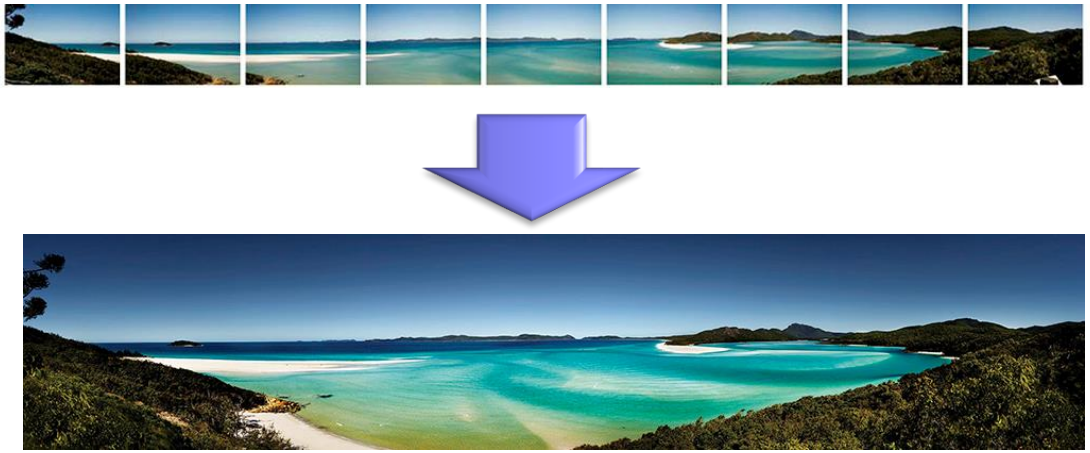


Two options:

1. *Image mosaicing with feature descriptors (LAB4)*
2. Detection of free parking spaces (*LAB5*)

Project 1: Image Mosaicing

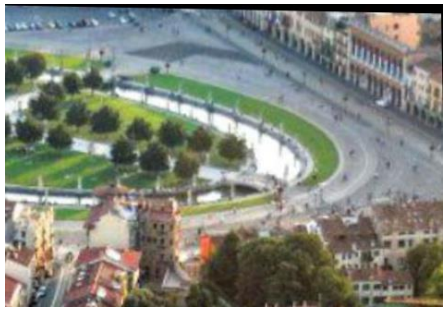




- Join multiple images to create a single bigger image
- *Stitching*: process of joining the various images together
- The task can be performed by finding feature points and matching them across the images

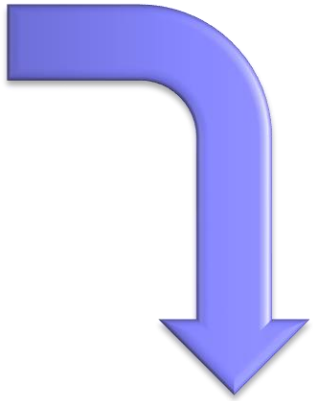
Merge 3x3 Tiles into Mosaic










Algorithm to be Developed

1. Load a set of 9 images roughly arranged in a 3x3 grid
   - You can use one of the provided sets
   - They have been obtained by splitting a larger image and applying some transformations
   - Letters underline the applied transforms (Translation, Scaling, Rotation, Noise, Light changes)
   - Different sets have different level of complexity, no need to solve all of them
2. Extract ORB or SIFT features from the images (SIFT features requires OpenCV>4.4)
3. For each couple of images
   - Compute the match between the different features extracted in step 2
   - You can use the `cv::BFMatcher` class (use L2 for SIFT and Hamming distance for ORB)
   - Refine the found matches by selecting the matches with distance less than *ratio * min_distance*, where *ratio* is a user-defined threshold and *min_distance* is the minimum distance found among the matches
4. You can assume the images are linked together by an affine transform
   - Using the refined matches find the transformation between the images
   - You can use the RANSAC algorithm implemented into the CV *findHomography()* function

This is the baseline assignment, see the next slide for additional suggestions for extra features improving your mark

Extra Ideas

1. Manually implement RANSAC and affine transform estimation
2. Try other algorithms/strategies
3. Try also panoramic images besides the mosaic
4. Acquire your own images (you can take 9 images moving the camera or acquire a single image, split in 9 and try to recombine)
5. Work with color images instead of grayscale
6. Try different feature descriptors
7. Use some blending/mixing techniques for better results
8. Equalize the images to avoid color jumps (test on the “L” images)
9. Try to automatically guess which images are linked to which

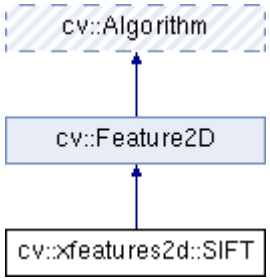
Differently from HW1, this final project lab has a mark,
a few of these or any other «extra» idea could improve your mark

OpenCV: Feature 2D Class

```
virtual void
cv::Feature2D::     (   InputArray                  image,              Input image
detectAndCompute
                        InputArray                  mask,               Compute KP only in regions
                                                                        where mask is not 0
                        std::vector< KeyPoint > &   keypoints,          Output keypoints (location,
                                                                        orientation, scale)
                        OutputArray                 descriptors,        KP descriptors
                        bool                        useProvidedKeypoints
                                                    = false             Set to false
                    )
```

- Base class for feature extractor and descriptors
- *detect* (feature extraction), *compute* (feature description) and *detectAndCompute* (both stages) methods
- Constructor depends on the employed subclass

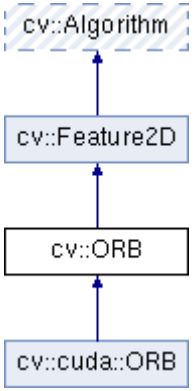
SIFT in OpenCV




```
static Ptr<SIFT>
cv::SIFT::create   ( int      nfeatures = 0,                  # of feature points to extract
                    int      nOctaveLayers = 3,              #  of layers in each octave
                    double   contrastThreshold = 0.04,       Threshold on D(x̂)
                    double   edgeThreshold = 10,             Threshold on eigenvalue ratio
                    double   sigma = 1.6                     Smoothing of the 1st img 1st octave
                  )
```

Usually better performances than ORB

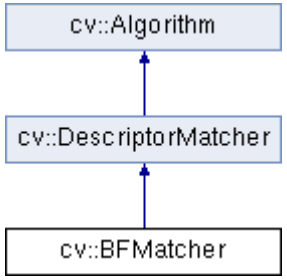
ORB in OpenCV




```
static Ptr<ORB>
cv::ORB::create  (   int    nfeatures = 500,                  Max # of features to extract
                    float  scaleFactor = 1.2f,               Scale step between different pyramid levels
                    int    nlevels = 8,                      # of pyramid levels (multi-scale)
                    int    edgeThreshold = 31,               Avoid computing features close to image boundaries
                    int    firstLevel = 0,                   Set to 0
                    int    WTA_K = 2,                        Set to 2 for comparison between couples of points as in the theory
                    int    scoreType = ORB::HARRIS_SCORE,    Rank extracted corners with Harris criteria
                    int    patchSize = 31,                   Size of patch for feature computation
                    int    fastThreshold = 20                Threshold in FAST algorithm
                 )
```

Feature Matching

```
static Ptr<BFMatcher>
cv::BFMatcher::create ( int       normType = NORM_L2,
                        bool      crossCheck = false
                      )
```



- Brute-Force matching
- Select the type of distance function
  - Use *NORM_L2* for SIFT and *NORM_HAMMING* for ORB
- *crossCheck* : if *true* forces that if A matches B then B must match A

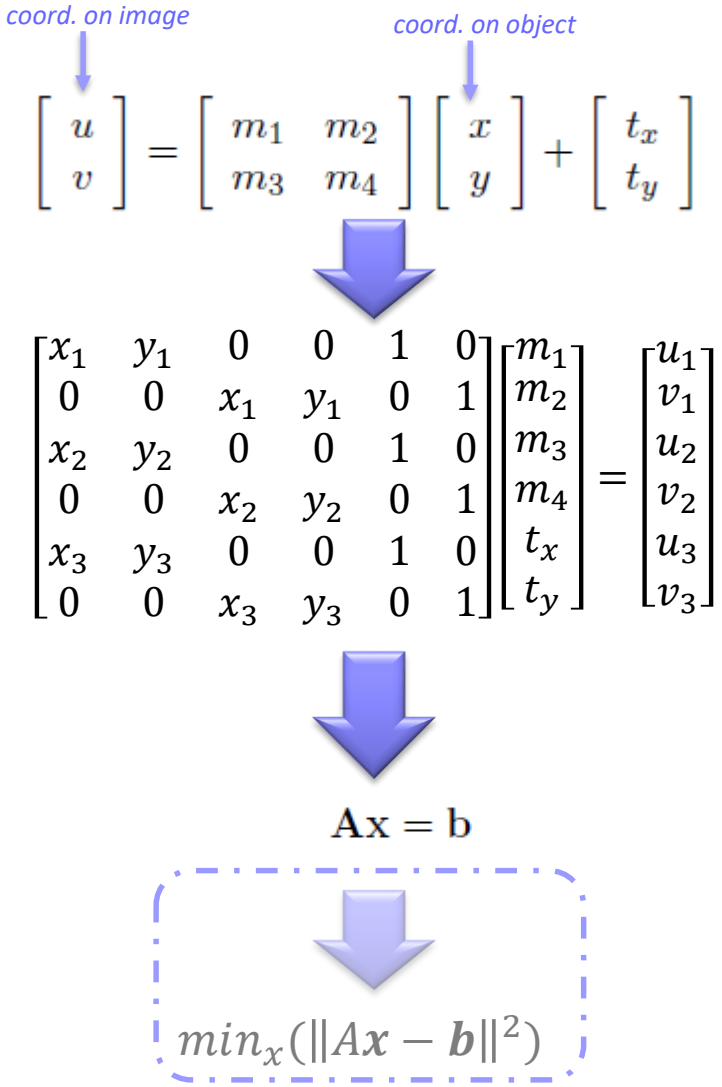
cv::findHomography

Mat cv::findHomography (	InputArray	srcPoints,	Coordinates of the points in the 1st image, a matrix of type CV_32FC2 or a vector<Point2f>
	InputArray	dstPoints,	Coordinates in the 2° image
	OutputArray	mask = noArray(),	Not needed
	int	method = 0,	Use cv::RANSAC
	double	ransacReprojThreshold = 3,	Threshold on reprojection error for a point to be considered an inlier
)			

- Finds the homography between object and its location in the image
- Returns 3x3 matrix with the computed homography

$$s_i \begin{bmatrix} x'_i \\ y'_i \\ 1 \end{bmatrix} \sim H \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}$$

Estimate the Affine Transform



- *Use the OpenCV findHomography() function*
  - *Estimates a more general transformation then the affine one*
  - *8 parameters (DoF) instead of 6*
  - *Includes the RANSAC algorithm*
- *Otherwise (advanced, optional) implement manually a simplified RANSAC estimator*
  1. Try different random sets of 3 points
  2. For each set of 3 KP estimate the corresponding affine transform
  3. Find the one with the largest consensus set
- Simpler (but slower) than the Hough Transform approach seen in the theory, but enough for simple cases

RANSAC (*simplified*)

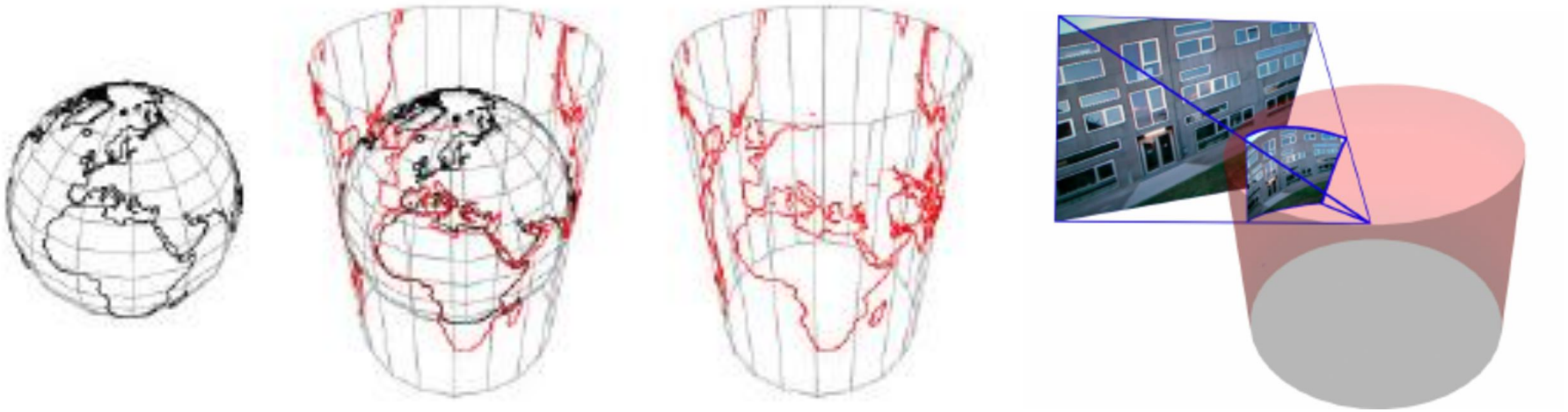
1. Select a random set of 3 correspondences
2. Estimate the affine transform
3. Count how many correspondences are consistent with the selected one. The criteria can be a threshold on the difference between the projections of the keypoint and their locations, $|\Delta u_{max}|+|\Delta v_{max}|<T$ *(e.g., T=3)*
4. Iterate *n* times (e.g., *n*=100) and keep the correspondence with the largest compatible set

Included inside openCV findHomography()

Panoramic Images

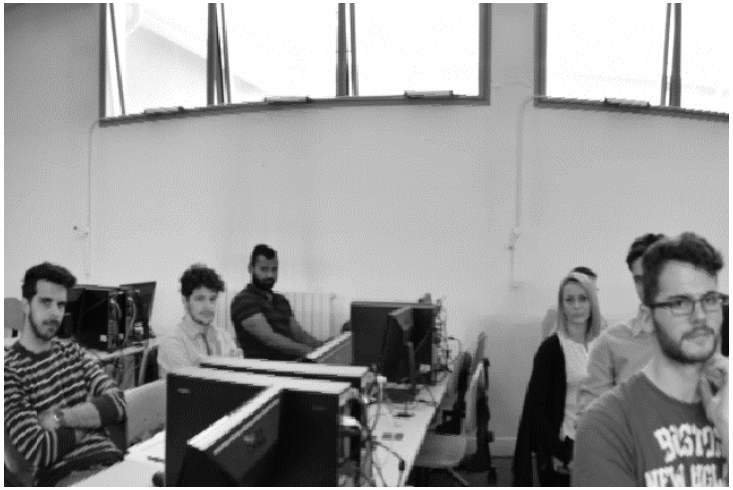
- Pictures covering a 360° field of view in the horizontal direction
- Panoramic images can be built from a set of pictures taken with a rotating camera from a single viewpoint

Cylindrical Projection

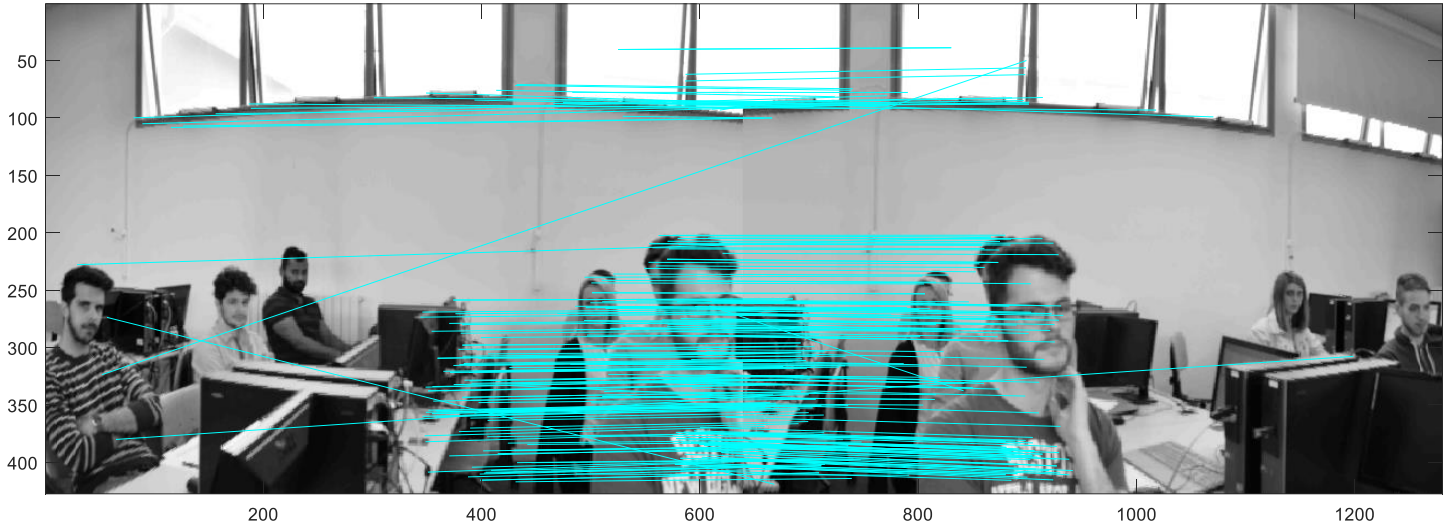


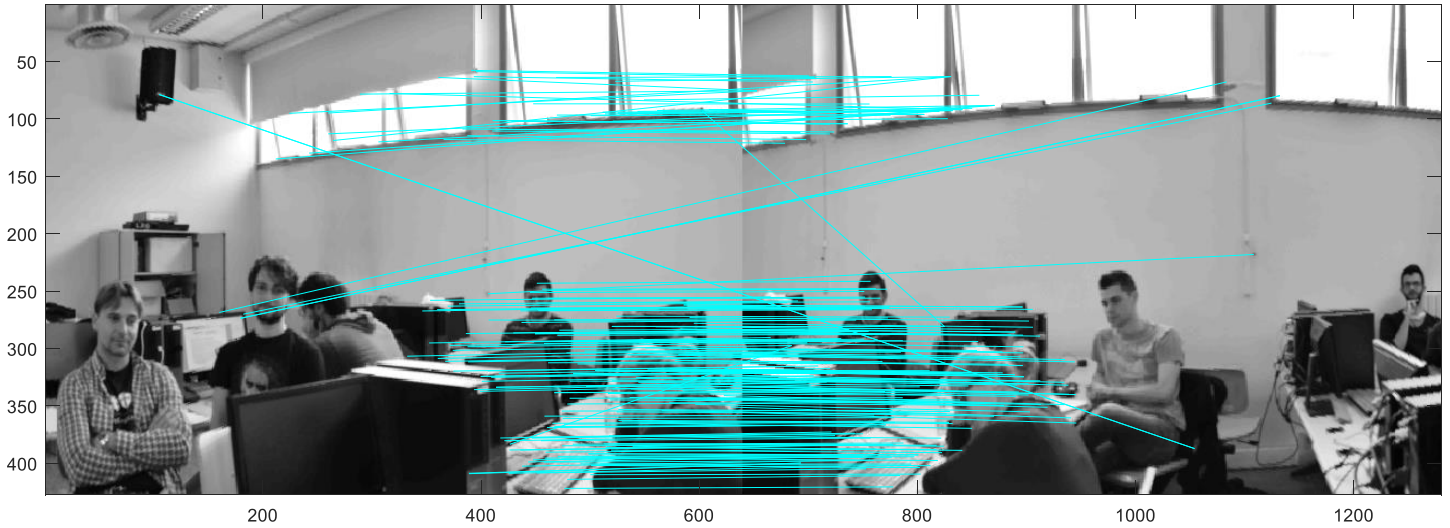
- The photos are projected on a cylinder
- After the cylindrical mapping the transformation between the various pictures becomes a simple *translation*
- See the file *'cylindrical_projection.pdf'* for the theory and equations of the projection
- The code for this task is already provided in the *"panoramic_utils.h"* and *"panoramic_utils.cpp"* files

Example (Cylindrical Projection)



Example (Matching)





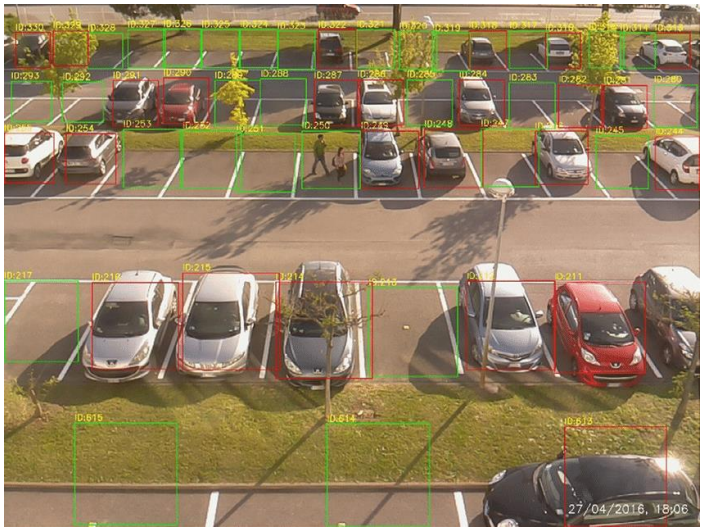
Examples (Panoramic Image)

Matlab+Lowe's toolbox

c++ - OpenCV - SIFT

c++ - OpenCV - ORB

Project 2: Parking Slot Detector



- Find free parking slots in the provided images
- A training set and a test set are provided to allow you to use machine learning approaches
- *More details later during the course*

Parking Slot Detector

Have a look at this two datasets

- https://web.inf.ufpr.br/vri/databases/parking-lot-database/
- http://cnrpark.it/

More detail later in the course, general ideas:

1. Download the data and extract the information you need for the task
2. The datasets are divided into train and test sets
   - The training set is needed only if you use ML-based approaches
3. *Task:* find which parking slots are free and which are occupied
4. Possible approaches:
   - Run edge/corner/feature detector into the slot and analyse found features
   - Analyse image statistics into the slot
   - Extract features (e.g. SIFT, ORB) , then use ML classifiers, e.g., SVM or Random Forests
   - Deep learning CNN binary classifier
   - Deep learning object detector

Support for the Final Project

Come to the labs, ask at the end of the lecture or contact by email the teaching assistant assigned to you

Surname	Teaching Assistant	email
Surname A to G	Mazen Mel	mazen.mel@phd.unipd.it
Surname H to R	Adriano Simonetto	adriano.simonetto@phd.unipd.it
Surname S to Z	Marco Toldo	marco.toldo.3@phd.unipd.it