Réalité augmentée à partir d'images

Christophe Vestri

Le mardi 13 octobre 2015

ARTmobilis

Introduction

Christophe Vestri

vestri@3Dvtech.com

3DVTech

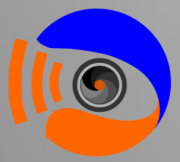
- Bureau d’étude
- Développement
- Consulting

www.3DVTech.com

Objectif du projet

Un parcours urbain en réalité augmentée

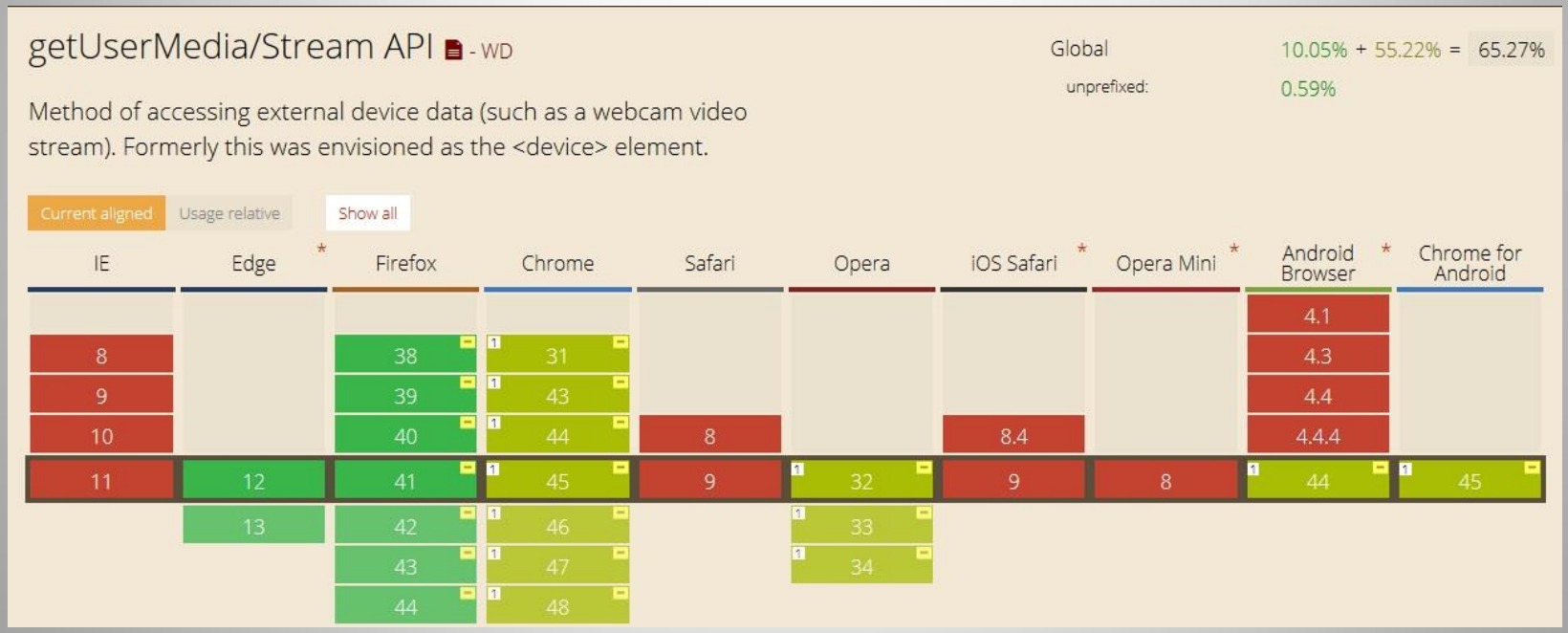
- Géolocalisation des points d'intérêts
- Tracking de la localisation des contenus augmentés
- Support mobile (android, IOS, tablettes)
- OpenSource: https://github.com/artmobilis/
- LabMobilis1:
  - Implémentation orientée Web pour adaptabilité
  - Application HTML5, CSS3 et JavaScript



ARTmobilis

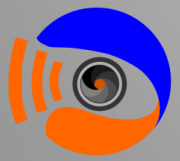
Navigateurs compatibles

- **Caniuse**: 65% des navigateurs
- Compatible avec Firefox/chrome/AndroidBrowser/Edge



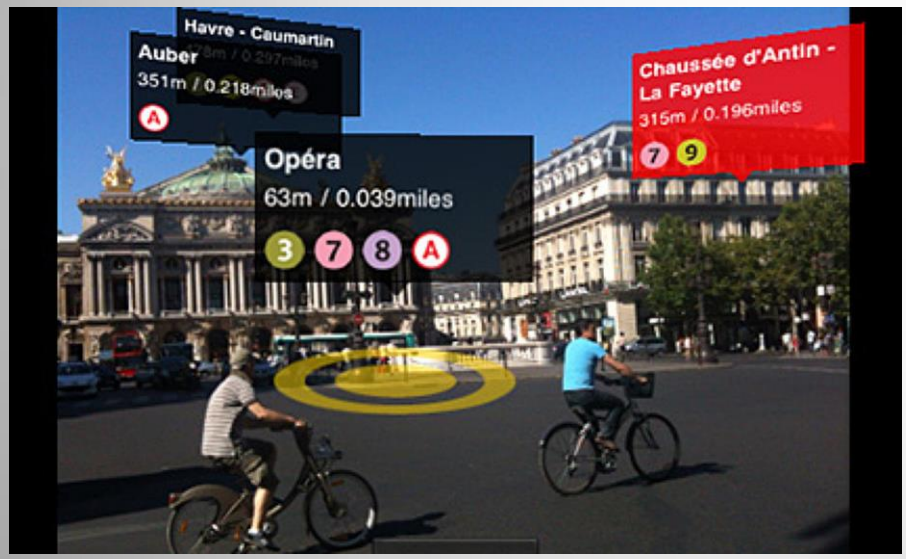
Résumé

- Réalité augmentée
- Librairies Javascript opensource
  - Jsfeat
  - Aruco
  - Threejs
- Les technologies utilisées
- Prototype développé
- Actions futures



Réalité augmentée

- Combiner le monde réel et des données virtuelles en temps réel (RAPro)



Réalité augmentée

Utilisation de marqueurs

- Choix marqueur
- Détection
- Transformation 2D-3D
- Affichage 3D

Réalité augmentée

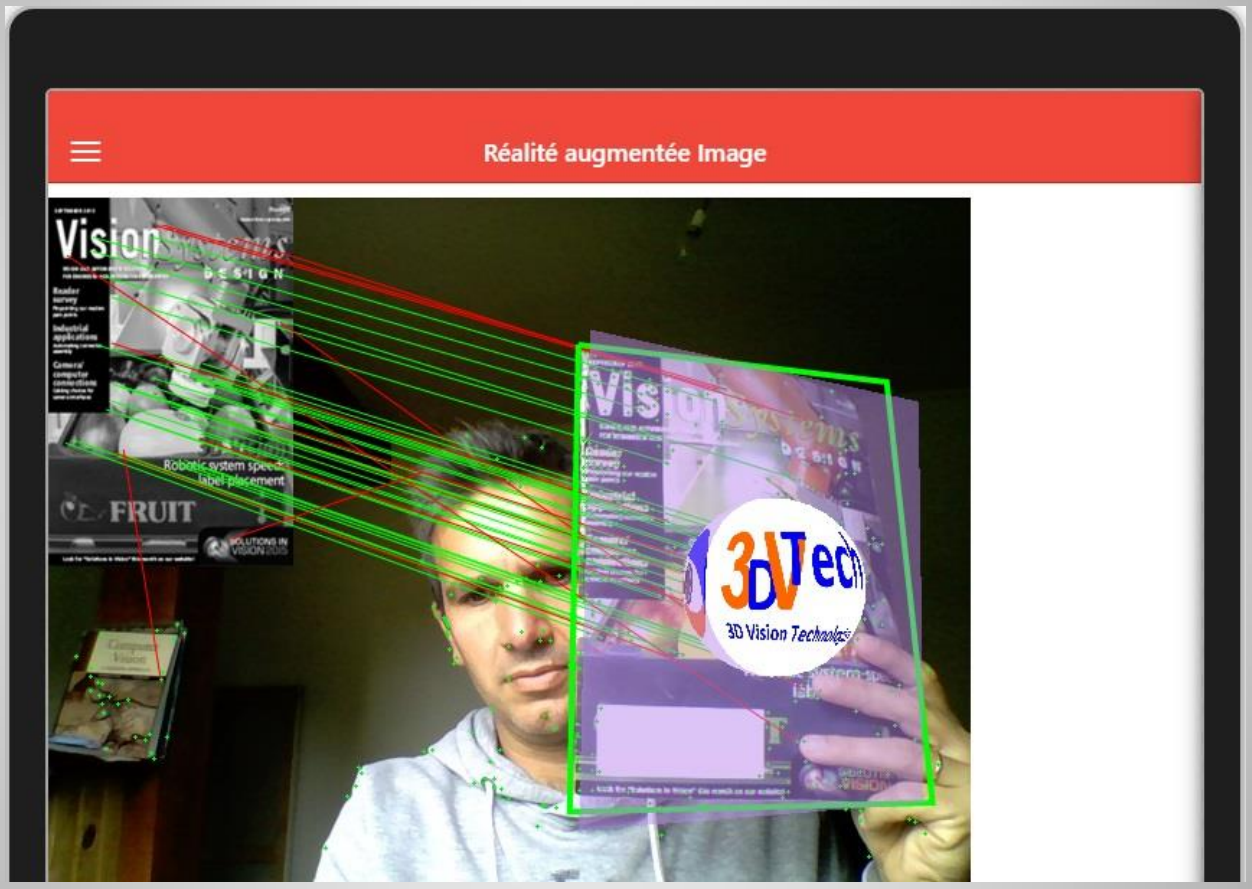
Utiliser les objets naturels

- Apprentissage
- Reconnaissance
- Transformation 2D-3D
- Affichage 3D

MOMO urban art on the Williamsburg Art & Design Building in Brooklyn.

ARTmobilis

Demo preview



Technologies nécessaires

Références

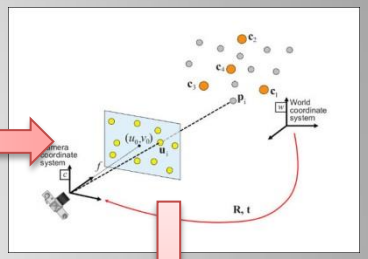
Acquisition vidéo

Détection coins et descripteurs

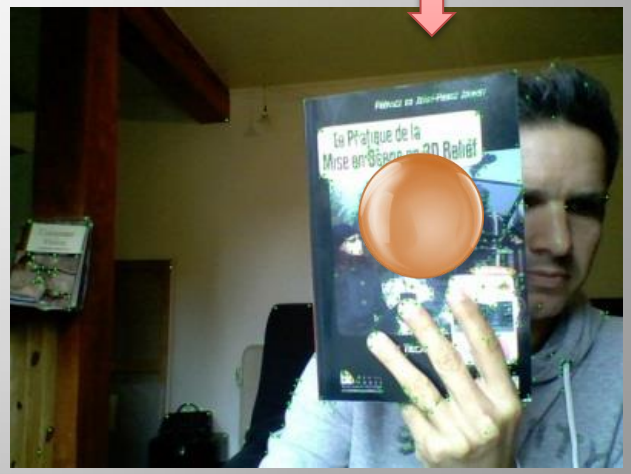
Matching

Localisation du Pattern

Calcul de la pose caméra



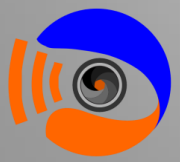
Affichage 3D



ARTmobilis

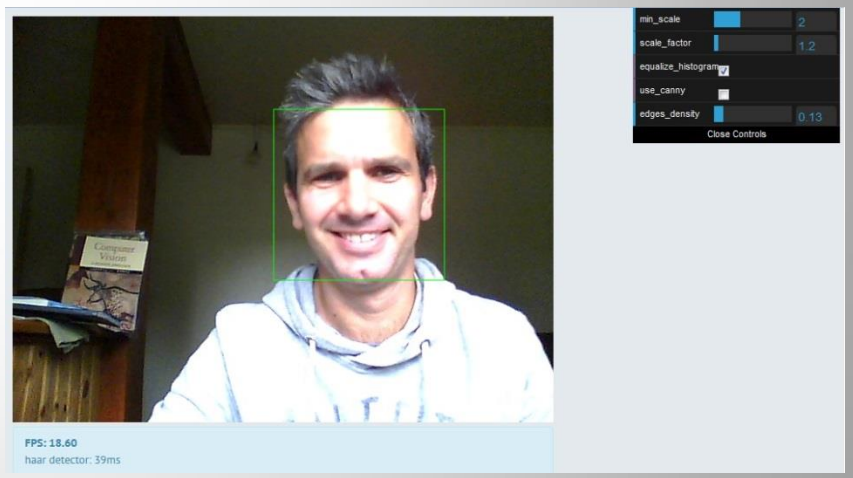
Librairies Javascript utilisées

- **Framework**:
  - Angularjs
  - Ionic
  - Cordova
- **AR Image demo:**
  - jsfeat : https://github.com/inspirit/jsfeat
  - Js-ArUco: https://github.com/jcmellado/js-aruco
  - three.js : https://github.com/mrdoob/three.js



Jsfeat

- Jsfeat: JavaScript Computer Vision library
- Algorithmes modernes de vision pour Html5
  - Custom data structures
  - Basic image processing
  - Linear Algebra and Multiview
  - Feature 2D
  - Optical flow
  - Object detection

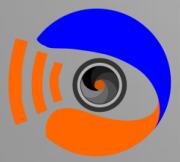


Jsfeat Exercice

- Code dans testhtml/ImageProcessingJSfeat
- Mettre l'image en **noir et blanc**

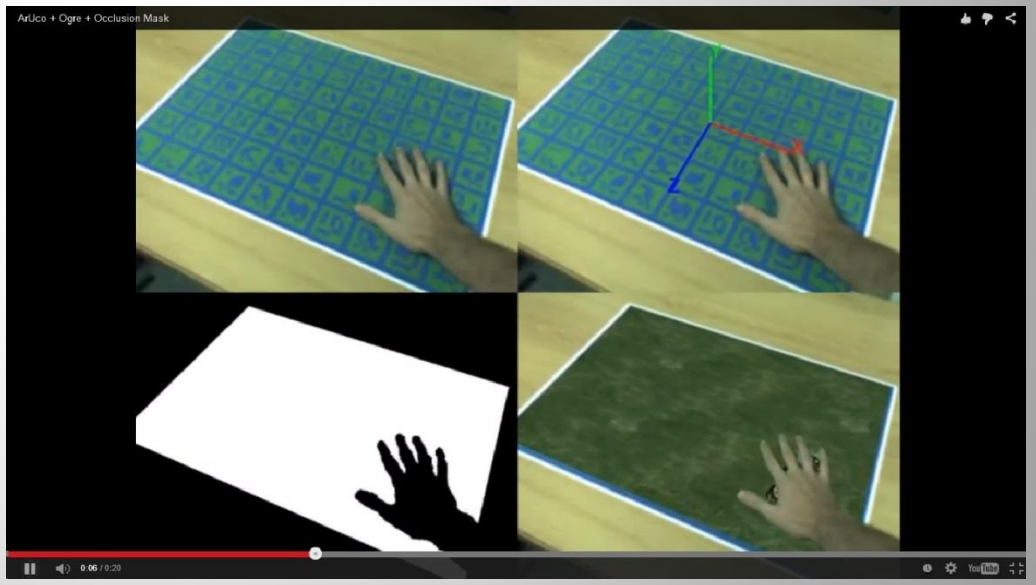
Jsfeat Solution

```
// process each acquired image
function tick() {
    compatibility.requestAnimationFrame(tick);
    stat.new_frame();
    if (video.readyState === video.HAVE_ENOUGH_DATA) {
        ctx.drawImage(video, 0, 0, 640, 480);
        var imageData = ctx.getImageData(0, 0, 640, 480);

        // greyscale conversion
        stat.start("grayscale");
        // I should put my code here
        jsfeat.imgproc.grayscale(imageData.data, 640, 480, img_u8);
        stat.stop("grayscale");

        // render result back to canvas (Warning: format is RGBA)
        stat.start("rewrite");
        // I should put my code here
        var data_u32 = new Uint32Array(imageData.data.buffer);
        var alpha = (0xff << 24); // opacity=1
        var i = img_u8.cols * img_u8.rows, pix = 0;
        while (--i >= 0) {
            pix = img_u8.data[i];
            // write 4 channels: RGBA with GreyGreyGreyAlpha
            data_u32[i] = alpha | (pix << 16) | (pix << 8) | pix;
        }
        stat.stop("rewrite");

        ctx.putImageData(imageData, 0, 0);
        log.innerHTML = stat.log();
    }
}
```



Aruco

- ArUco est une librairie minimale pour la Réalité Augmentée à base de marqueurs (basée OpenCV)
- js-aruco est le portage en JavaScript d'ArUco
  - Image processing
  - Contours
  - Detection marqueurs
  - Calcul de pose



ArUco Exercice

- Code dans testhtml/ImageProcessingAruco
- Faire **tourner la sphère**

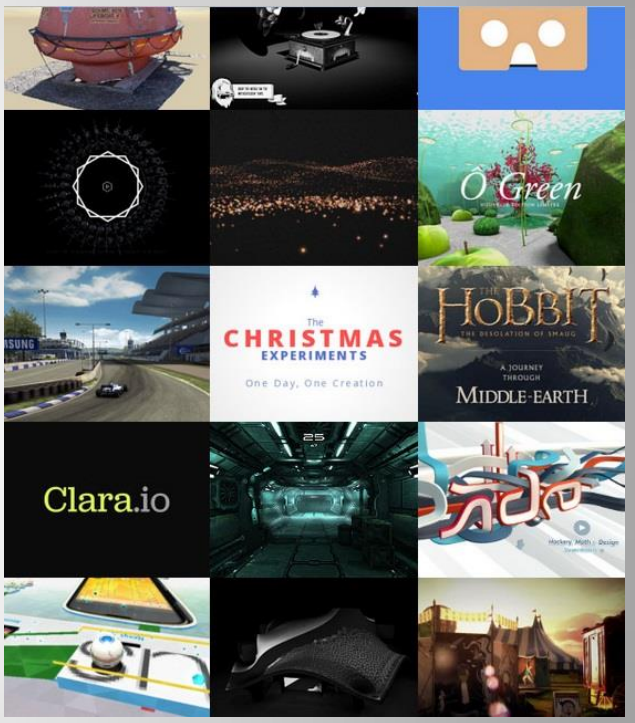
ArUco Solution

```
        stat.start("Posit");
        pose = posit.pose(corners);
        stat.stop("Posit");

        stat.start("Update");
        updateObject(model, pose.bestRotation, pose.bestTranslation);
        stat.stop("Update");

        step += 0.025;
        model.rotation.z -= step;
    }
};
```

Three.js

Three.js simplifie l'utilisation de WebGL

- Renderers: WebGL, <canvas>, <svg>…
- Scenes, Cameras, Geometry, Lights, Materials, Shaders, Particles, Animation, Math Utilities
- Loaders: Json compatible Blender, 3D max, Wavefront OBJ



ARTmobilis

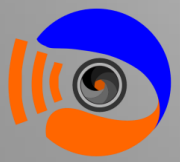
Three.js Exercice

- Code dans testhtml/ImageProcessingThreeJS
- Combiner **l'image et la 3D**
- Indice: Faire des Layers

Three.js Solution

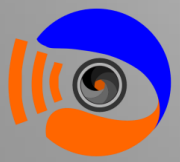
```
<title>JSFeat-ThreeJS - Exercice: Canvas 2D 3D</title>
<h1>JSFeat-ThreeJs - Exercice: Canvas 2D 3D</h1>

<video id="webcam" style="display:none;" height="480" width="640"></video>
<div style=" width:640px;height:480px;margin: 10px auto;">
    <canvas id="canvas2d" width="640" height="480" class="overlapcanvas"></canvas>
    <canvas id="canvas3d" width="640" height="480" class="overlapcanvas"></canvas>
    <div id="log"></div>
</div>
```

```
function createRenderersScene() {
    renderer3d = new THREE.WebGLRenderer({ canvas: canvas3D, alpha: true });
    renderer3d.setClearColor(0xffffff, 0);
    renderer3d.setSize(canvas2d.width, canvas2d.height);

    // for 3d projection
    scene = new THREE.Scene();
    camera = new THREE.PerspectiveCamera(40, canvas2d.width / canvas2d.height, 1, 1000);
    scene.add(camera);

    model = createModel();
    scene.add(model);

    camera.position.z = 5;
};
```



Autres librairies intéressantes

- Computer Vision:
  - tracking.js: https://github.com/eduardolundgren/tracking.js
  - js-objectdetect: https://github.com/mtschirs/js-objectdetect
  - Convnetjs: https://github.com/karpathy/convnetjs
  - sgdSlam: https://github.com/odestcj/sgd-slam
- 3D:
  - Babylon.js: https://github.com/BabylonJS/Babylon.js



ARTmobilis

Technologies nécessaires

Références

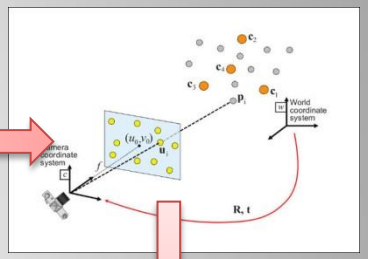
Acquisition vidéo

Détection coins et descripteurs

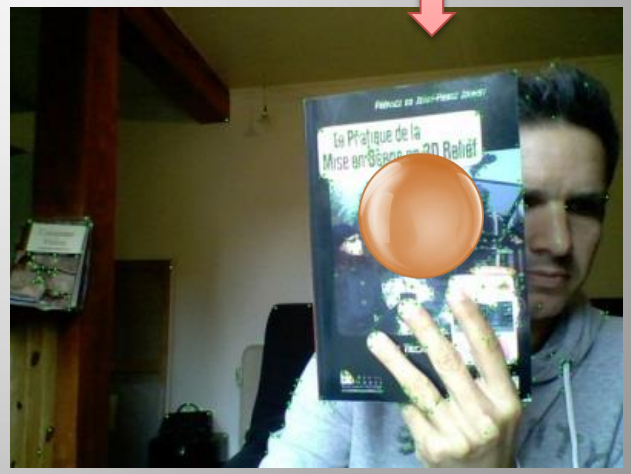
Matching

Localisation du Pattern

Calcul de la pose caméra



Affichage 3D



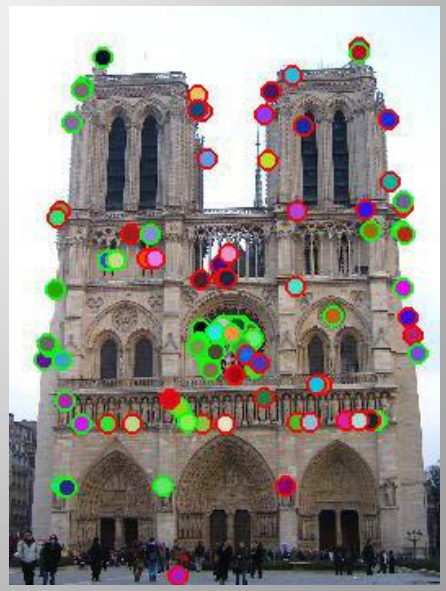
ARTmobilis

Detection de coins

Critères de qualité:

- Caractérisables: distinctif, particularité, reconnaissable, précision
- Répétabilité et invariance: échelle, rotation, illumination, point de vue, bruit

2 opérations distinctes: **détection** et **description**



ARTmobilis

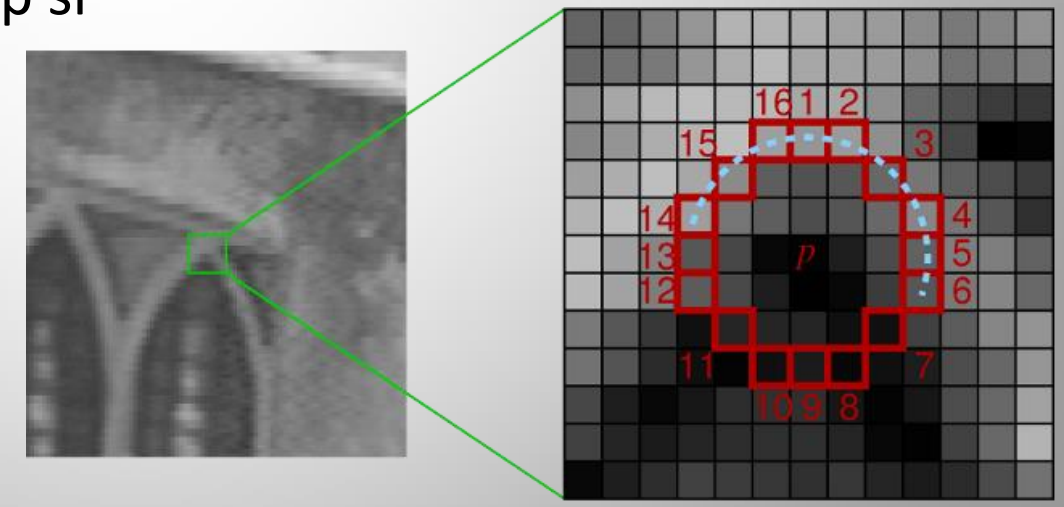
Detection de coins

ORB (Oriented FAST and Rotated BRIEF)

- FAST: Features from Accelerated Segment Test
  http://www.edwardrosten.com/work/fast.html
- Cercle Bresenham 16 pixels autour du point analysé
- On détecte un coin en p si

  l'intensité de N pixels

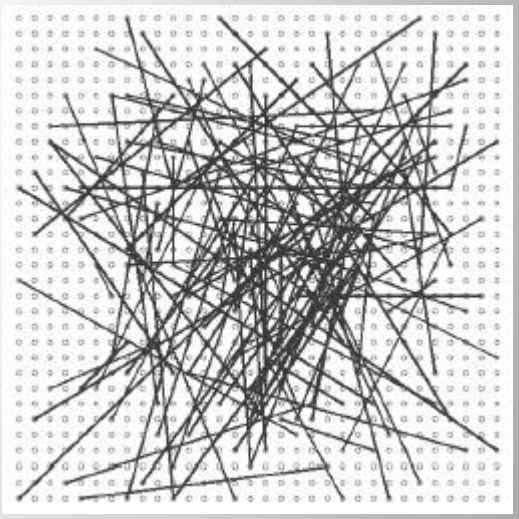
  est > ou < de X% à I_p
- Rapide et robuste



Descripteur de points

BRIEF : Binary robust independent elementary features
http://cvlab.epfl.ch/research/detect/brief

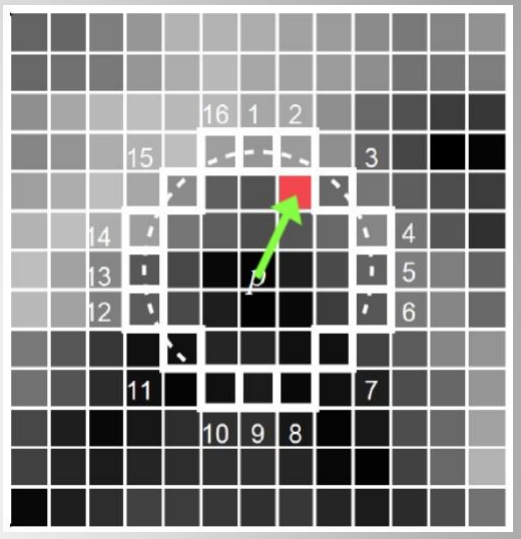
- N paires de points sur un patch
- Comparaison pour chaque paire
  - Si I1<I2 alors c=1
  - Sinon c=0
- Descripteur=100101001…
- Rapide et robuste

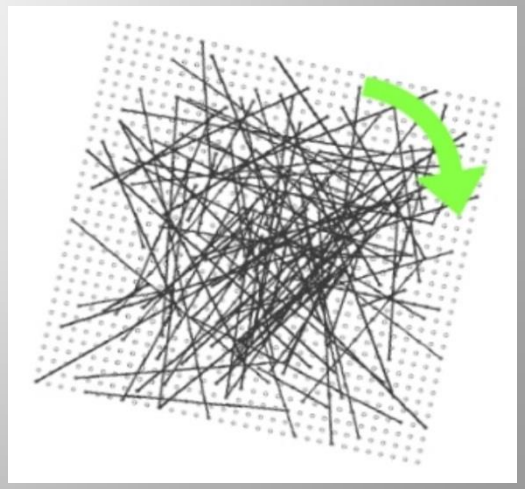


Descripteur de points

ORB (Oriented FAST and Rotated BRIEF)
http://docs.opencv.org/.../py_feature2d/py_orb/py_orb.html

- Prise en compte rotation pour robustesse
- Direction=pixel avec variation la plus forte
- Rotated BRIEF pour aligner les descripteurs lors du matching





Reconnaissance

Appariement des coins

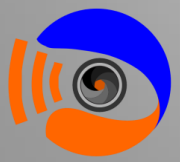
- Brute force matching, on teste toutes les paires
- Similarité= Distance de Hamming (nombre de bits différents)

A = 1 0 1 1 0 0 1 0 0 1 0 0

B = 1 0 0 1 0 0 0 0 1 1 1 1

Distance de Hamming = 3

- Si on a un nombre de coins appariées suffisants, l'objet est retrouvé

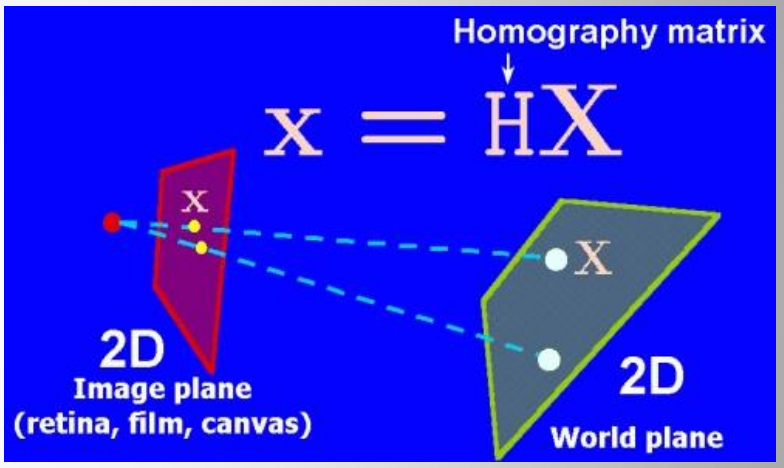


ARTmobilis

Relocalisation

Calcul de l'homographie du plan

- Système d'équation linéaire
- Estimation robuste (RANSAC)
- Filtrage des outliers
- Décomposition en VP

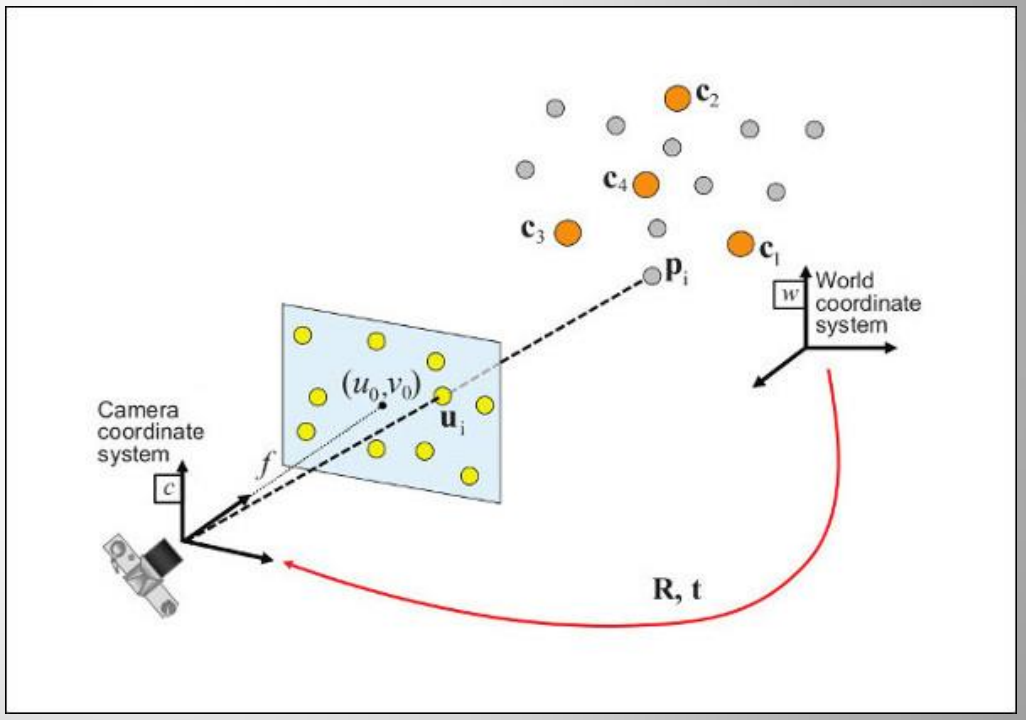


$$\lambda \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \underbrace{\begin{pmatrix} h_{11} & h_{12} & h_{13} \\ h_{21} & h_{22} & h_{23} \\ h_{31} & h_{32} & h_{33} \end{pmatrix}}_{homography\ \boldsymbol{H}} \begin{pmatrix} X \\ Y \\ 1 \end{pmatrix}.$$

Pose 3D

Calcul de la pose de la caméra par rapport à un objet 3D

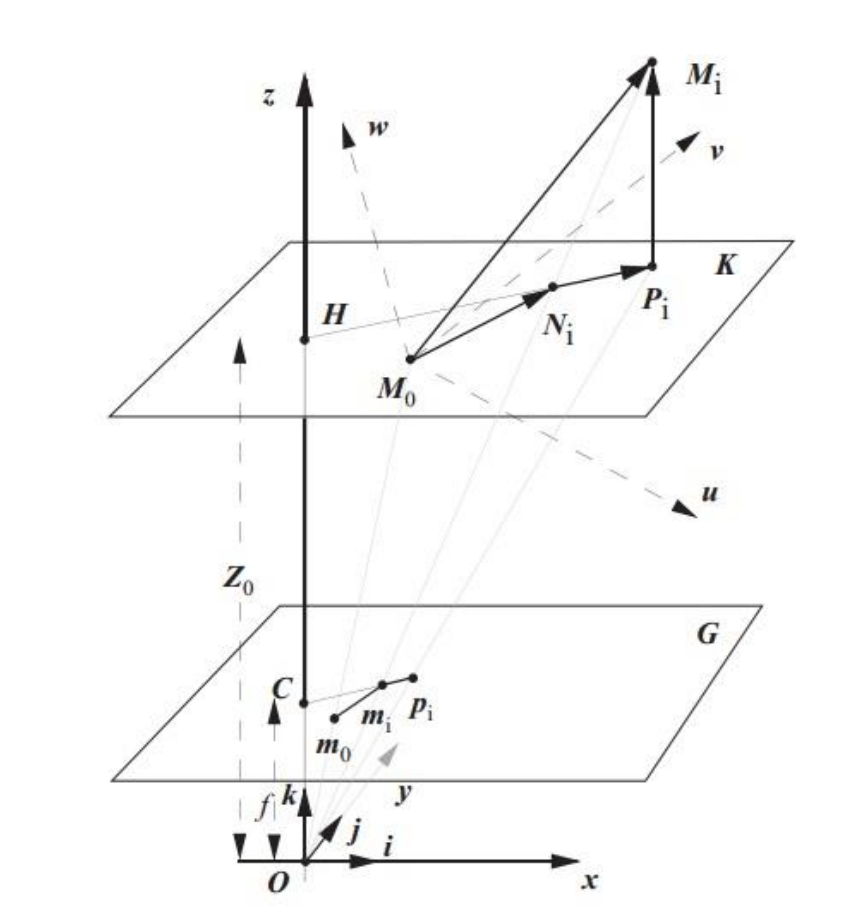
- General case:
  - 6DoF
  - Projection model
- Simplification
  - Calibration connue
    - Perspective-n-Point
  - Projection ortho
    - POSIT



POSIT

POSIT: Pose from Orthography and Scaling with ITerations

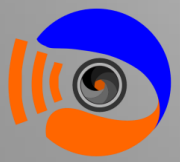
- Algorithme itératif pour résoudre PnP non coplanaires
- 4 points coplanaires: Coplanar POSIT



More on Pose 3D

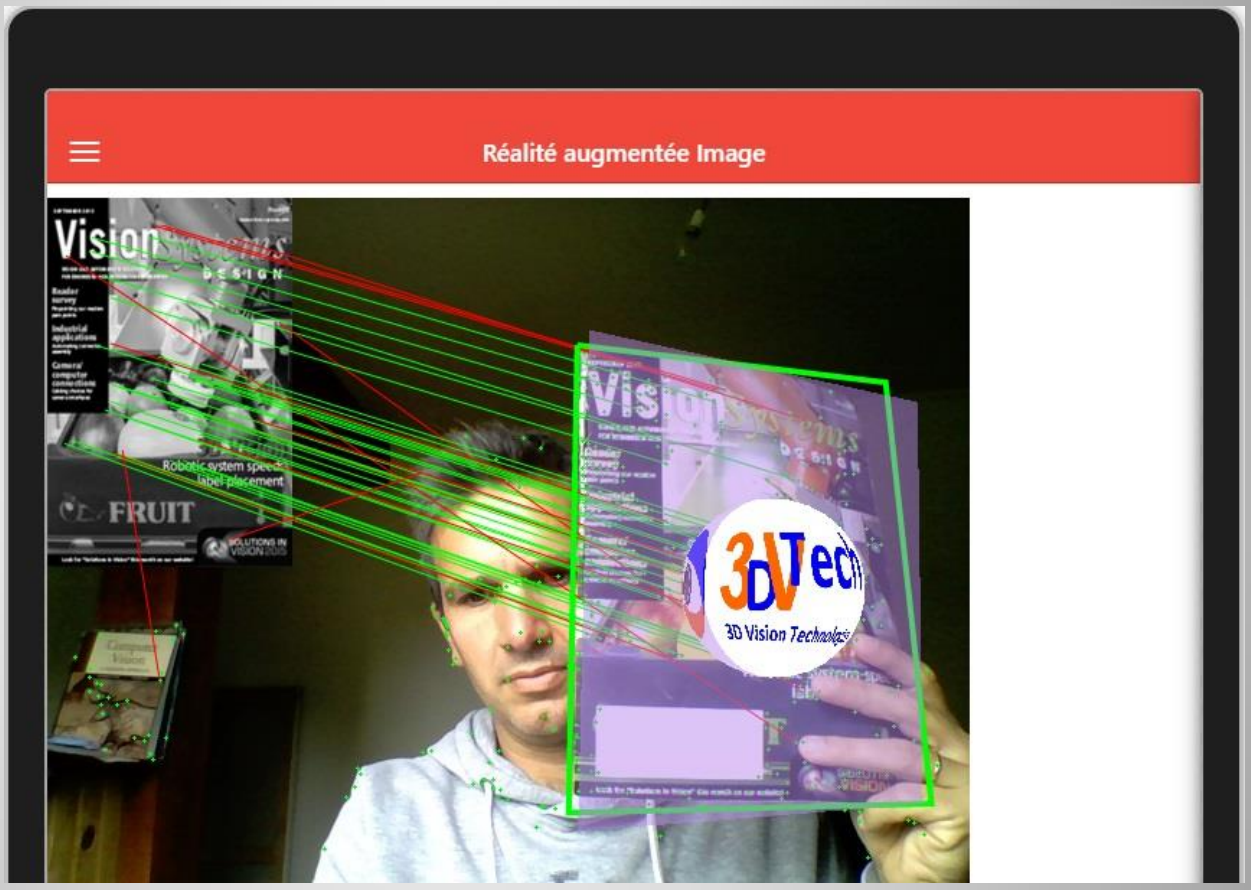
Calcul de la pose de la caméra par rapport à un objet 3D

- POSIT: original publications, 3D pose estimation
- Real Time pose estimation : OpenCV tutorial, C++
- Caméra calibration: OpenCV tutorial, C++
- posest: C++ opensource
- Minimal problems in Computer Vision: many links
- Moving camera = Kalman/SLAM



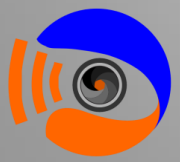
Prototype développé

- Demo
- Code



Next steps

- Javascript ou pas?
- Robustifier toutes les étapes
- Taille des images à apprendre
- Echelle des objets à apprendre (multi-echelle)
- Pose relative -> taille des objets 3D
- Temps réel sur smartphone
- Taille mémoire: combien d'objets? Gps+gyroscope
- Outils de debug et analyse suffisant pour application?
- Download/Lire/écrire images et objets et json



ARTmobilis

Merci

- https://github.com/artmobilis/
- vestri@3DVTech.com