ENSIIE2 – Option IRA (Réalité Augmentée)
TP2 – Recalage basé modèle

Objectifs :
- Construire un modèle *a priori* d'une scène ;
- Exploiter le modèle en vue de calculer la pose.

Pré-requis :
- Programmation javascript et C.

Configuration requise :
- Navigateur web supportant WebGL et WebRTC (Mozilla Firefox / Google chrome) ;
- Accélération matérielle supportée au niveau du système d'exploitation.

Présentation du matériel de TP

Dans ce TP, nous allons mettre en oeuvre un recalage basé modèle. Un bonne partie de votre séance sera consacrée à réaliser une méthode de détection et de recalage pour un objet bien particulier.

Du point de vue technique, l'application comporte les dépendances aux bibliothèques suivantes :
- ARUCO.js est un port javascript de la bibliothèque de reconnaissance et de localisation de cibles codées ARUCO :
  `https://github.com/jcmellado/js-aruco`
- ARCS.js, framework expérimental de programmation orienté composant assurera l'infrastructure logicielle (ARCS = *Augmented Reality Component System*) :
  `http://arcs.ibisc.univ-evry.fr`
- Three.js servira à manipuler les graphes de scène. Sa documentation se trouve à l'adresse suivante :
  `http://threejs.org/docs/`
- La programmation se faisant en Javascript, ceux qui ne se trouvent pas à l'aise avec ce langage peuvent s'appuyer sur les ressources suivantes :
  `http://fr.eloquentjavascript.net/contents.html`
  `http://www.w3schools.com/js/`

Le mode de récupération de l'archive servant de base au TP vous sera communiqué en début de séance. Votre premier travail consistera à décompresser l'archive `ra_tp2.zip`. Cette archive vous propose un squelette d'application que vous aurez à compléter au fur et à mesure.

Les répertoires et fichiers principaux de l'application sont donnés à la table 1. Ils contiennent le code source des composants sur lesquels vous allez travailler. La page à lancer dans le navigateur est intitulée `dicedetector.html`. Cette page charge une description d'application (fichier `arcsapp.json`), instancie les composants nécessaire et exécute l'application.

Nom	Contenu
`build`	Scripts nécessaires au moteur d'ARCS.js
`components`	Répertoire des composants de l'application
`deps`	Dépendances (scripts) requis par les composants
`docs`	Documentation (API moteur et composants)
`tests/dicedetector`	Répertoire de l'application
`tests/dicedetector/dicedetector.html`	Page HTML lançant l'application
`tests/dicedetector/arcsapp.json`	Description/configuration de l'application

TABLE 1 – Principaux répertoires et fichiers de l'archive

Nom	Type	Fonction
animator	Animator	Génération d'impulsions pour rythmer l'animation
detector	DiceDetector	Détection du modèle et calcul de la pose
objectLoader	OBJLoader	Chargement de ressources 3D
objectTransform	ObjectTransform	Application d'une transformation sur un objet
video	LiveSource	Acquisition du flux vidéo provenant de la webcam
viewer	ARViewer	Composition d'images réelles et d'une scène 3D
windowresize	WindowEvent	Détection du redimensionnement du navigateur
ts	TokenSender	Synchronisation avec l'automate paramétrant le flux de données (lié au moteur d'ARCS)
sm	StateMachine	Automate de pilotage du flux de données (lié au moteur d'ARCS)

TABLE 2 – Composants de l'application et fonctionnalités associées

Les composants emploient le mécanisme signal/slot pour communiquer entre eux. Les slots correspondent à des entrées et les signaux à des sorties auxquelles on peut attacher des valeurs ou des objets. Les signaux des composants sont connectés aux slots d'autres pour former une chaîne de traitement. La liste des composants est indiquée dans la table 2 et le flux principal de données est représenté à la figure 1.

L'objet que nous allons chercher à détecter est un dé à 6 faces, chacune étant associé à une image différente (le patron en est joint avec l'énoncé du TP). Vous obtiendrez donc un solide en 3 dimensions sur lequel nous allons faire le recalage d'un seul et même modèle (l'avantage est que l'on peut visualiser le modèle en question sous tous les angles). Une esquisse de l'objet à détecter est donnée à la figure 2. Y sont inclus les repères de l'objet en lui-même (situé au centre du cube) et le repère associé à l'image d'une des faces (l'origine des images est située à leur coin supérieur gauche).

Travail à réaliser

0. Atelier découpage/collage

Réaliser le dé à partir du patron joint à cet énoncé.

Remarque : Le dé n'est pas nécessaire immédiatement. Vous pouvez faire les premières questions sans réaliser le dé.

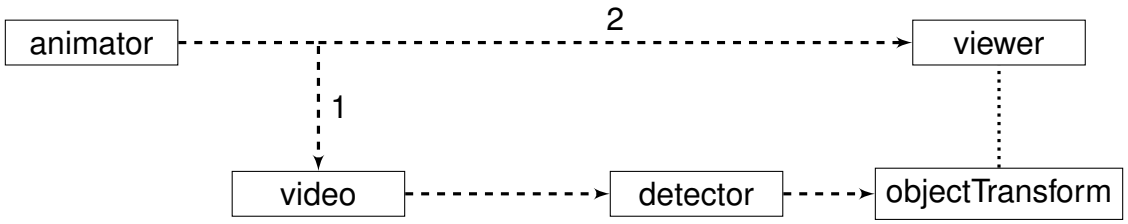

FIGURE 1 – Flux de données de l'application

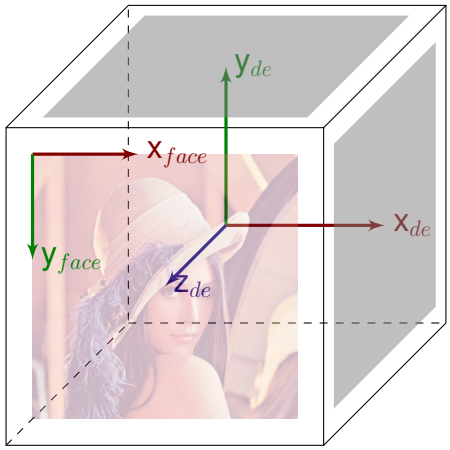

FIGURE 2 – L'objet à détecter et les repères associés au dé et à une de ses faces

1. Associer des descripteurs SURF au modèle

`diedetector.js` est le fichier que vous aurez à modifier. Il contient les déclarations et l'implémentation d'une structure de données et d'un composant :

- `DiceFace` : structure qui définit les faces du dé ;
- `DiceDetector` : classe qui va permettre de détecter le dé et de calculer la pose de la caméra par rapport à ce dernier.

Dans un premier temps, afin que le système reconnaisse l'objet, il va falloir le lui faire apprendre. A cette fin, vous allez devoir modifier le constructeur de `DiceFace` de manière à ce que la structure stocke les points d'intérêt et les descripteurs SURF associés. Au passage, vous stockerez également l'image sur laquelle vous allez calculer les descripteurs (*TODO #1*).

2. Associer des descripteurs SURF à l'image acquise par la caméra

Dans la classe `DiceDetector`, il existe une méthode `detect()` qui est appelée chaque fois qu'une image est acquise par la caméra. C'est cette méthode qui doit lancer la détection d'une ou plusieurs faces du dé. La première étape est de lancer l'extraction des points d'intérêt (*TODO #2*).

3. Retrouver les correspondances entre images apprises et images acquises

Dans la classe `DiceDetector`, il y a une boucle qui permet de trouver les correspondances entre l'image acquise par la caméra et les images apprises. Il vous faut donc compléter cette boucle pour trouver les correspondances (*TODO #3*). A des fins de débogage, vous écrirez le code de la fonction `debugImage()` de la classe `DiceFace` (*TODO #4*).

Quelles sont les faces du dé les mieux détectées ?

Remarque : Vous pourrez regarder la documentation de l'API liée à la manipulation des éléments `canvas` en HTML5 :

— https://developer.mozilla.org/en-US/docs/Web/API/HTMLCanvasElement/getContext
— https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D

4. Afficher les correspondances sur l'image acquise

Dans la classe `DiceDetector`, compléter la méthode `detect()` de manière à afficher les correspondances détectées sur l'image acquise. Dans le principe, plusieurs faces du dé peuvent être visibles et reconnues dans l'image aussi il faut stocker l'ensemble des correspondances dans un vecteur pendant que l'on itère sur toutes les faces du dé, puis on dessine sur l'image, après la boucle, les correspondances trouvées (*TODO #5* et *TODO #6*).

Les correspondances trouvées sont elles toutes valides ?

Remarque : Pensez à modifier la valeur du seuil si vous avez trop de correspondances indues (augmenter le seuil diminue le nombre de correspondances). Le seuil est manipulable avec la fonction `surfDetDes()` ou en passant par la variable globale `SURF.THRESHOLD`.

5. Préparation des jeux de données pour l'estimation de la pose

Dans la classe `DiceFace`, compléter la méthode `get3DCounterPart(...)` qui doit calculer les coordonnées d'un point d'intérêt dans l'espace 3D du cube (*TODO #7*). Les coordonnées du point d'intérêt sont passées en paramètre et sont exprimées dans le repère de l'image sur la face du dé. Il vous faudra donc réfléchir à la conversion du plan 2D vers les points 3D en fonction :

— De la taille de l'image en pixels ;
— De la taille réelle de l'image en mètres ;
— De la position du centre de l'image ;
— Des directions principales ($\vec{x}$ et $\vec{y}$) du repère de l'image.

Remarque : Attention, cette fonction fera appel à vos connaissances en géométrie, en particulier sur ce qu'est fondamentalement un système de coordonnées.

6. Estimer la pose

Une fois les correspondances 2D/3D établies, nous allons lancer un algorithme de calcul de la pose de la caméra par rapport au dé. Nous utiliserons pour cela la méthode `pose` de classe `Posit`. Il ne faudra pas oublier de convertir ensuite le résultat dans un type compatible avec le type `Marker`. Compléter la classe `DiceDetector` et plus particulièrement la méthode `detect(...)` (*TODO #8*).

Le personnage est il recalé correctement par rapport au dé ? Quels sont les défaut apparents du système ?