Introduction to Visual Computing

Assignment# 13

Pose Estimation of the Lego Board

May 23, 2017

Description

The last step to complete our tangible game is to compute the orientation of the board in front of the webcam, so that we can accordingly control the virtual board.

Objectives

Estimate the 3D rotation of the Lego board as seen by the webcam, and use these angles to control in live the board in the game.

Specific Challenges

This week's assignment requires you to be comfortable with matrices manipulation. You will have to re-use what you have learned about 3D projection during the *Introduction to 3D rendering* lecture.

Preliminary steps

If needed, finish last week assignment: you need to have a good estimate of the corners of the board to complete today's assignment.

Part I

Pose estimation

Step 1 – A slice of mathematics

During the *Introduction to 3D rendering* lecture, we formalized a projection of a 3D point $[X, Y, Z]$ (belonging to an object *obj*) onto a 2D point $[x, y]$ as:

$$\begin{bmatrix} x \\ y \\ f \\ 1 \end{bmatrix} \cong \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} & & & T_{obj} \\ & R_{obj} & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{1}$$

Here, f is the focal length of the camera, and the point $[x, y]$ belongs to a plane, orthogonal to the camera axis $\vec{z}$ and laying at a distance f from the camera center. R_{obj} and T_{obj} are the rotation and translation of the object.

> **Note**
>
> In this equation, $\cong$ means that the points are *equivalent in homogeneous coordinates*:
>
> $$\mathbf{x} \cong \mathbf{y} \Leftrightarrow \exists \lambda / \mathbf{x} = \lambda \mathbf{y}$$

The aim of today's assignment is to compute R_{obj}: the rotation matrix of the physical board. Once we have it, we can apply the same rotation to the virtual board in our game, and hence control the game from the physical, tangible board.

Let's note $\mathbf{x} = \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$ the homogeneous coordinates of the projected point, and $\mathbf{X} = \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$ the homogeneous coordinates of the 3D point. Discarding the (fixed) z coordinate of the projected point, we can rewrite the equation (1) as:

$$\mathbf{x} \cong \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1/f & 0 \end{bmatrix} \begin{bmatrix} & & & T_{obj} \\ & R_{obj} & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{X} \tag{2}$$

Note

If you take a ruler and measure it, the Lego board is 255x255 mm. Assuming an origin at the center of the board, the coordinates of the four corners are hence:

$$\mathbf{A} = \begin{bmatrix} -128. \\ -128. \\ 0 \\ 1 \end{bmatrix}, \mathbf{B} = \begin{bmatrix} 128. \\ -128. \\ 0 \\ 1 \end{bmatrix}, \mathbf{C} = \begin{bmatrix} 128. \\ 128. \\ 0 \\ 1 \end{bmatrix}, \mathbf{D} = \begin{bmatrix} -128. \\ 128. \\ 0 \\ 1 \end{bmatrix}$$

Then, we rewrite the projection matrix as a 3×3 matrix multiplied by $[I|0]$:

$$\mathbf{x} \cong \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f \end{bmatrix} \left[\begin{array}{c|c} I & 0 \end{array} \right] \begin{bmatrix} & & & T_{obj} \\ & R_{obj} & & \\ & & & \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{X} \tag{3}$$

Note

$[I|0]$ represents a matrix divided up into a 3×3 block (the identity matrix) plus a column vector, here the zero vector:

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

This then becomes:

$$\mathbf{x} \cong \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1/f \end{bmatrix} \left[\begin{array}{c|c} R_{obj} & T_{obj} \end{array} \right] \mathbf{X} \tag{4}$$

Since we work in homogeneous coordinates, we can multiply by f and equation (4) is then equivalent to:

$$\mathbf{x} \cong \begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix} \left[\begin{array}{c|c} R_{obj} & T_{obj} \end{array} \right] \mathbf{X} \tag{5}$$

We note K the matrix $\begin{bmatrix} f & 0 & 0 \\ 0 & f & 0 \\ 0 & 0 & 1 \end{bmatrix}$. It is called the *intrinsic matrix* of the camera (*intrinsic* because it only depends on internal parameters of the camera – for now only the focal length).

The final form of the equation is hence:

$$\mathbf{x} \cong K \left[\begin{array}{c|c} R & T \end{array} \right] \mathbf{X} \tag{6}$$

Note

$K \left[\begin{array}{c|c} R & T \end{array} \right]$ is often noted P (for camera *p*rojection matrix) in the literature.

If we expend $\left[\begin{array}{c|c} R & T \end{array} \right]$, and assuming we know the focal length f (hence, K is known), we can observe that we have *12 unknowns to determine*:

$$\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \cong K \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{7}$$

How to solve such an equation?

Two points that are equivalent in homogeneous coordinates have a cross-product equal to zero:

$$\mathbf{x} \cong \mathbf{y} \Leftrightarrow \mathbf{x} \times \mathbf{y} = 0$$

So equation (7) can be written as:

$$K^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

where $\left(\begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \right)$ represents one 2D $\leftrightarrow$ 3D correspondence between one corner extracted from our image and one physical corner of the board.

Due to a linear dependency, solving this equation would actually require *6 2D-3D correspondences*. However, we only know 4 of them (one per corner).

Since the 3D points of the board are *coplanar*, we can however further simplify the equation (steps not presented here), and we are finally left solving:

$$K^{-1} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \times \begin{bmatrix} r_{11} & r_{12} & t_1 \\ r_{21} & r_{22} & t_2 \\ r_{31} & r_{32} & 1 \end{bmatrix} \begin{bmatrix} X \\ Y \\ 1 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \tag{9}$$

We solve this set of linear equations using the SVD algorithm from the Processing's OpenCV library.

Step 2 – Install the OpenCV library

Install the library OpenCV from Processing. Go to *Sketch Menu ->Import Library...->Add Library...* and search and install OpenCV. In order to use the library, go to the main sketch and

- Add at the top `import gab.opencv.*;`
- Add `OpenCV opencv;` as global variable
- In the setup() function add the line `opencv = new OpenCV(this,100,100);`

This will initialize the library so that it can be used in the project. For example,

```
.
.
.
import gab.opencv.*;

OpenCV opencv;
.
.
.
void setup(){
    opencv = new OpenCV(this, 100, 100);
.
.
.
}
```

Step 3 – Get the pose

Download the class `TwoDThreeD.pde` from Moodle and add it to your project.

The implementation is complete, you only need to provide the corners. The class constructor takes as input the size of the image and an estimation of your application frame rate (frames per second, set it to 0 for still images). Such estimation is used to smooth the pose using a low passing filter and quaternion interpolation. Rotations can be computed by calling the function `get3DRotations` with the quad corners as argument.

Display the resulting rotations r_x, r_y, r_z for each of the quads you found in the image.

Note

Make sure to transfer quad corners to homogeneous coordinates before passing to `get3DRotations()`.

Note

`get3DRotations()` returns angles in radians.

Download four sample images provided on Moodle and apply your code to compute board rotations in each image. You should find the angle values close to:

`board1.jpg:` $r_x = -9^\circ, r_y = 12^\circ, r_z = 4^\circ$

`board2.jpg:` $r_x = -9^\circ, r_y = -4^\circ, r_z = 14^\circ$

`board3.jpg:` $r_x = 21^\circ, r_y = 4^\circ, r_z = 0^\circ$

`board4.jpg:` $r_x = 17^\circ, r_y = -47^\circ, r_z = -5^\circ$

In the next section you will use the the computed angles to control the virtual board in your game.

Part II

Integration with the game

The last part of the project consists in using the computed angles to control the virtual board of your game.

Integrate the extraction of corners and computation of the board orientation with your game (using r_x and r_y to tilt the virtual board).

In order to show both the game window and the image processing window, you will need to do some refactoring of your code.

Integrate your image processing code in the game that you developed during the first phase of your project until week 6. If you want to have multiple windows, one for tuning the image processing parameters and one for the game, you could copy the code of your main process (i.e. containing the `setup()`, `draw()` methods) within an `ImageProcessing` class that extends `PApplet`:

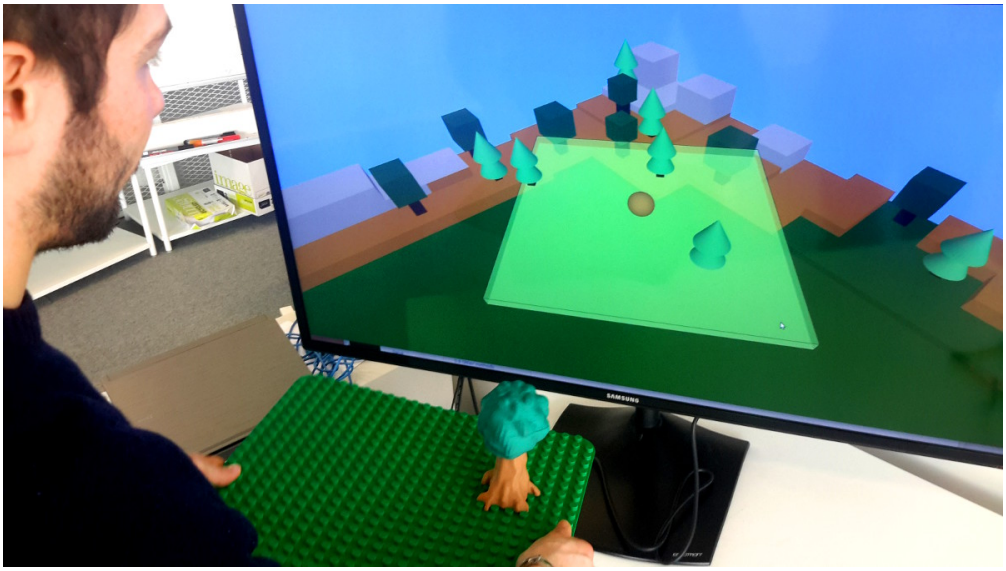
```
class ImageProcessing extends PApplet {
        Capture cam;
   //...
        void settings(){...}
   void setup(){...}
   void draw(){...}
   //...
}
```

You can then create an object `ImageProcessing imgproc;`. You can refer to it and start a new window process by using the following code in your main game process:

```
void setup(){
    //...

    imgproc = new ImageProcessing();
    String []args = {"Image processing window"};
    PApplet.runSketch(args, imgproc);

    //...
    }
void draw(){
    //...

    PVector rot = imgproc.getRotation();
    // where getRotation could be a getter for the rotation angles you computed previously

    //..

    }
}
```

You can also choose to display the image processing as a `PGraphics` of the game process.

Test first with the sample images, and then with your webcam.



Congratulations! You just completed your first tangible game!