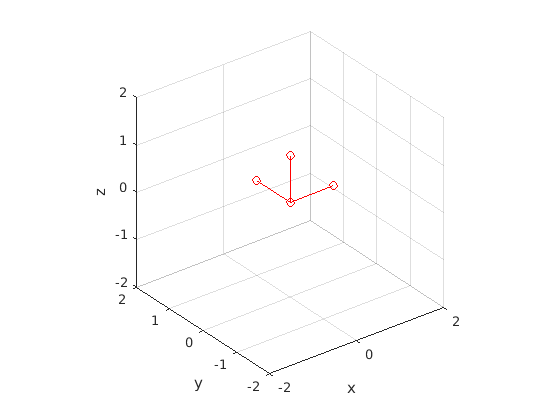
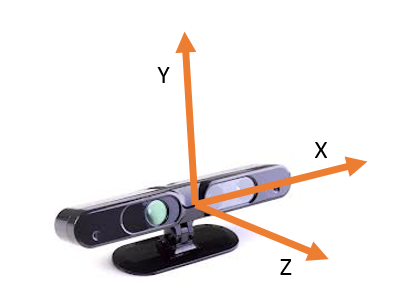
# VISIÓN POR COMPUTADOR

**Exercise 9: Camera model.**

Concepts: Perspective projection matrix, spatial transformation.

1. **Homogeneous transformations:** Given a camera located at the origin of coordinates, which the following reference frame:



Obtain the transformation matrices (4 x 4) to:

|  |  |
| --- | --- |
| **Transformation** | **Equivalent in videogame** |
| Move the camera 0.5 meters forward. | The player moves 0.5 meters forward, i.e. he/she is walking in straight line. |
| Rotate the camera 35º to look to the left (yaw rotation). *Note: the angles must be introduced in radians. You can use the helper function* ***deg2rad(deg).*** | The player rotates 35º to the left in place. |
| Move the camera 0.5 meters upwards and rotate it downwards. | The player jumps and looks down, for example, to observe the same object from a higher perspective, to jump on a box, etc. |

Recall that a transformation matrix has the following form:

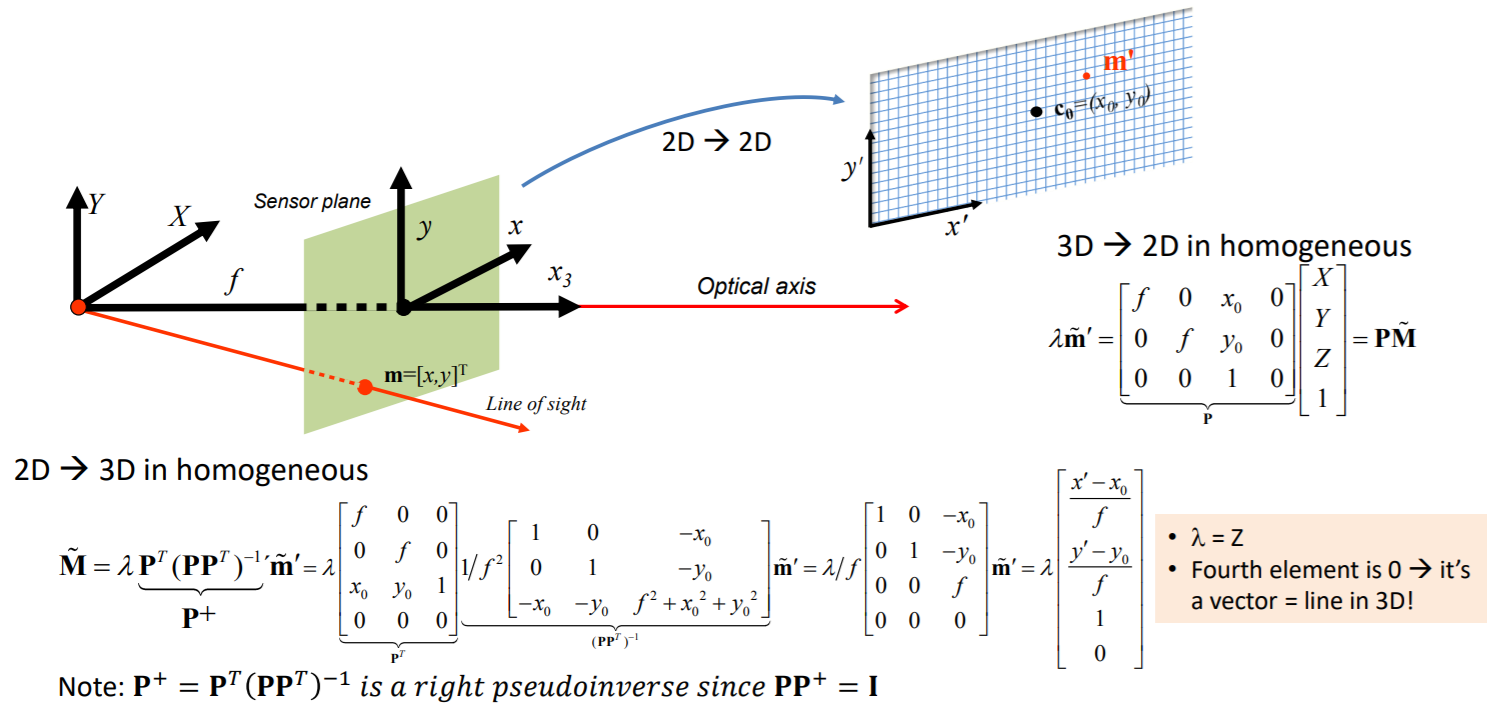
Show the original and the transformed reference frames with ***showTransformation(T)***, where *T* is the transformation matrix of each exercise part.

1. **Camera-to-world transformation**: In this exercise we will work with images provided with an RGB-D camera. These cameras provide standard RGB images together with depth images which measure how far is the point observed by each pixel.

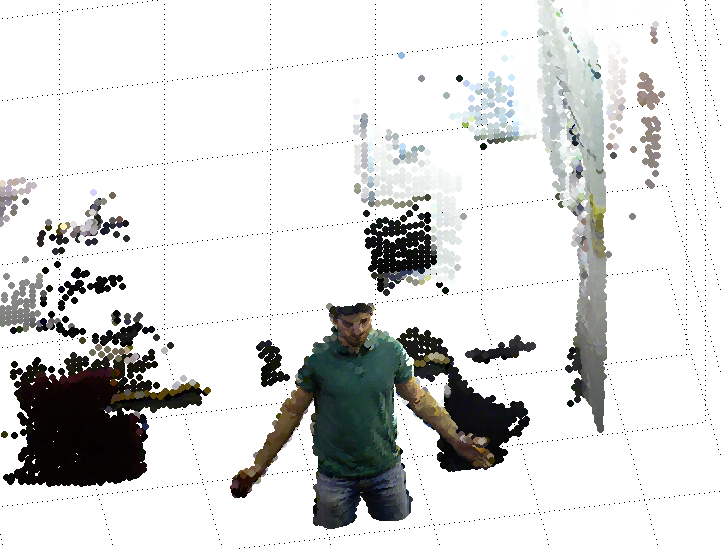
 

Thus, if the intrinsic parameters are known (matrix K), we can compute the 3D coordinates of the observed points. To do that, you have to:

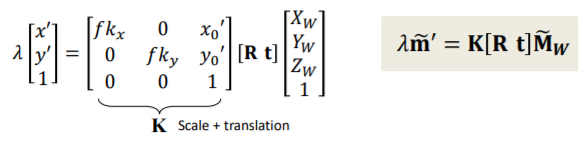
* Read the RGB and depth images provided and show them (you will have to re-scale the depth image to visualize it properly). To read the depth image use ***imread( )***.
* Obtain the 3D coordinates of the observed points (relative to the reference frame of the camera), assuming that and . Recall the following slide for obtaining the 3D coordinates of the line passing through a point in the image. Notice that in this formulation the reference system of the image is placed at the bottom left part of the image, and in Matlab it is placed at the top left part.



* Plot the 3D points with their corresponding color with ***plot3DScene(****X,Y,Z,image\_RGB, downsample****)***. An example of execution of such function:

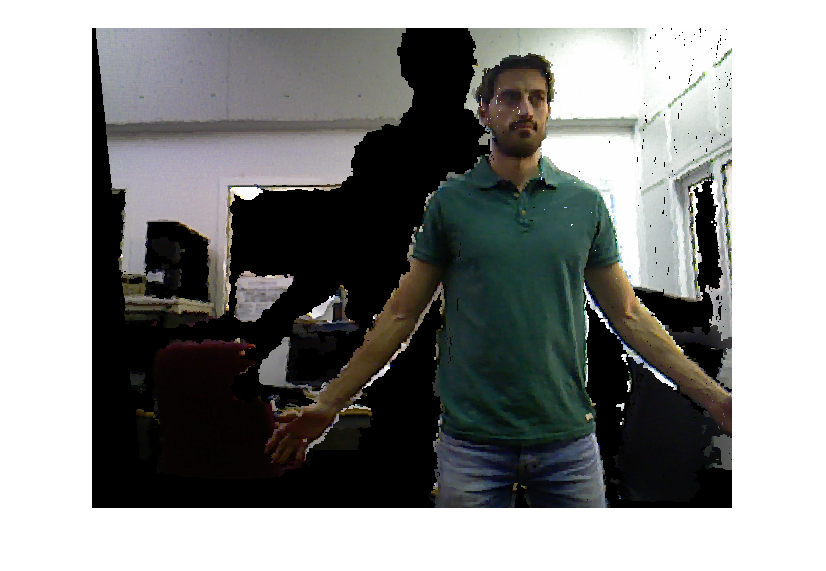
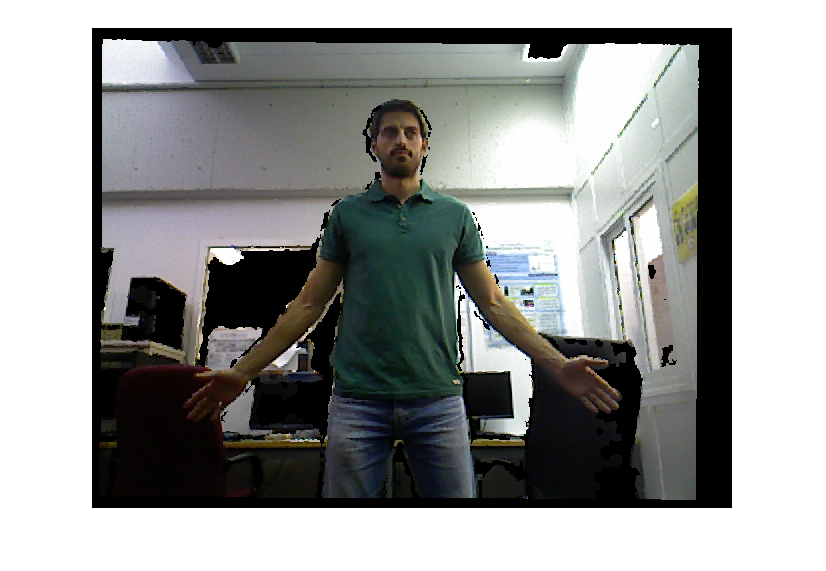


1. **World-to-camera transformation:** Now we will simulate views of the same scene from different perspectives. To that end, you have to define the homogeneous transformation [R t] associated to these new perspectives and use it, together with K, to create the new images. From the last lesson:



Generate at least two images from different perspectives moving the camera around the observed points. Take into account that these images will only look “reasonable” if the camera does not get very far from its original location.

Use the function ***renderNewImage(****x\_proj, y\_proj, depth\_transformed, image\_RGB****)***.



*Left, initial image. Right, example of an image taken from a different perspective.*

**Commands:**

|  |  |
| --- | --- |
| **showTransformation(T)** | Creates a visualization of a ref. frame at the origin (blue) and transformed (red) according to T (4x4). |
| **plot3DScene(X,Y,Z,image\_RGB,downsample)** | Shows a coloured 3D point cloud. X,Y,Z are matrices with the spatial coordinates of the points, “image RGB” is the colour image and “downsample” is used to reduce the number of points to draw (use downsample ~= 4). |
| **renderNewImage(x\_proj, y\_proj, depth\_transformed, image\_RGB)** | Creates a new image by rendering the scene from a different perspective. “x\_proj” and “y\_proj” are the pixel coordinates of the scene points after transforming and projecting them, “depth\_transformed” is the depth after transforming them and “image\_RGB” is the original colour image. |

**Appendix: Exercise code**

close all

clear variables

% 1: Homogeneous transformations

% 2: Camera-to-world transformation

% 3: World-to-camera transformation

execute **=** 1**;**

% -------------------------------------------------------------------------

% 1: Homogeneous transformations

%

**if** execute **==** 1

% Initial translations

tx **=** 0**;** ty **=** 0**;** tz **=** 0**;**

% Initial reference frame

T **=** **[**1 0 0 tx

0 1 0 ty

0 0 1 tz

0 0 0 1**];**

showTransformation**(**T**);**

% Move the camera 0.5 meters forward

tx **=** 000000000**;** ty **=** 000000000**;** tz **=** 000000000**;**

T **=** **[**1 0 0 tx

0 1 0 ty

0 0 1 tz

0 0 0 1**];**

showTransformation**(**T**);**

% Rotate the camera 35degrees to look to the left (yaw rotation)

tx **=** 000000000**;** ty **=** 000000000**;** tz **=** 000000000**;**

yaw **=** deg2rad**(**35**);**

T **=** **[**000000000**];**

showTransformation**(**T**);**

% Move the camera 0.5 meters upwards and rotate it downwards

tx **=** 000000000**;** ty **=** 000000000**;** tz **=** 000000000**;**

pitch 000000000**;**

T **=** **[**000000000**];**

showTransformation**(**T**);**

% -------------------------------------------------------------------------

% 2: Camera-to-world transformation

%

**elseif** execute **==** 2 **||** execute **==** 3

% Read the rgb and depth images images

im\_rgb **=** 000000000**;**

im\_depth **=** 000000000**;**

% Show them

figure

imshow**(**im\_rgb**)**

figure

imshow**(**im\_depth**)**

% Camera intrinsic parameters

f **=** 000000000**;**

x0 **=** 000000000**;**

y0 **=** 000000000**;**

% Initialization of useful vectors

**[**rows**,**cols**]** **=** size**(**im\_depth**);**

points\_x **=** zeros**(**rows**,**cols**);**

points\_y **=** zeros**(**rows**,**cols**);**

points\_z **=** zeros**(**rows**,**cols**);**

% Depth scale and max depth considered

scale **=** 0.0002**;**

max\_depth **=** 5**;**

% Obtain the 3D coordinates of each pixel

**for** y**=**1**:**rows

**for** x**=**1**:**cols

points\_z**(**y**,**x**)** **=** double**(**im\_depth**(**y**,**x**))\***scale**;**

**if** points\_z**(**y**,**x**)** **>** max\_depth % check max depth

points\_z**(**y**,**x**)** **=** 0**;**

**end**

points\_x**(**y**,**x**)** **=** **-**1**\***000000000**;**

points\_y**(**y**,**x**)** **=** **-**1**\***000000000**;**

**end**

**end**

downsample **=** 4**;** % increment this downsampling depending on your

% pc's computational power

**if** execute **==** 2

plot3DScene**(**points\_x**,**points\_y**,**points\_z**,**im\_rgb**,**downsample**);**

**end**

**end**

% -------------------------------------------------------------------------

% 3: World-to-camera transformation

%

**if** execute **==** 3

% Modify this transformation

transform **=** **[-**0.3**;** **-**0.3**;** **-**0.3**;** 0**;** **-**10**;** 0**];** % -z, -x, -y, z, yaw, pitch, roll

T **=** TMatrixFromValues**(**transform**);**

% Useful variables

proj\_x **=** zeros**(**rows**,**cols**);**

proj\_y **=** zeros**(**rows**,**cols**);**

z\_t **=** zeros**(**rows**,**cols**);**

% Coompute the coordinates (pixels) of each 3D point

**for** x**=**1**:**cols

**for** y**=**1**:**rows

% Apply the transformation T to the point

% The coordinates are reorder because of the Matlab coordinates

% system: x towards (our z), y left (our x), z upwards (our y)

p **=** **[**points\_z**(**y**,**x**);** points\_x**(**y**,**x**);** points\_y**(**y**,**x**);** 1**];**

z\_t**(**y**,**x**)** **=** T**(**1**,:)\***p**;**

x\_t **=** T**(**2**,:)\***p**;**

y\_t **=** T**(**3**,:)\***p**;**

% Now, use the K matrix to retrieve proj\_x (x') and proj\_y

% (y'). These are the coordinates (x',y') of the 3D point in

% the image!

proj\_x**(**y**,**x**)** **=** **-**1\*000000000**;**

proj\_y**(**y**,**x**)** **=** **-**1\*000000000**;**

**end**

**end**

renderNewImage**(**000000000**);**

**end**