Aston University
BIRMINGHAM UK

CS4790 RAS

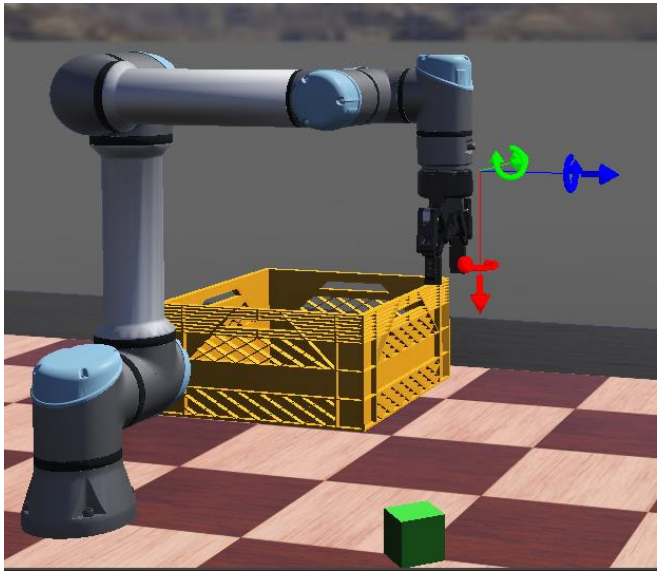
Tutorial
Camera Characteristics and Transformations

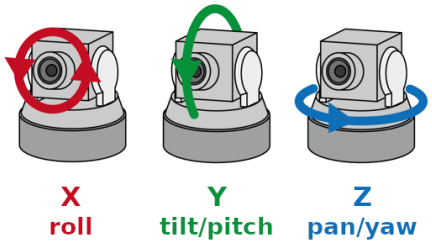
Martin Rudorfer | m.rudorfer@aston.ac.uk

UNIVERSITY OF THE YEAR 2020 The Guardian

Camera Characteristics

- Our robot has a depth camera
- Both are in the same pose (see picture on the right)
- They have a resolution of 128 x 64 and a FOV of 1.0





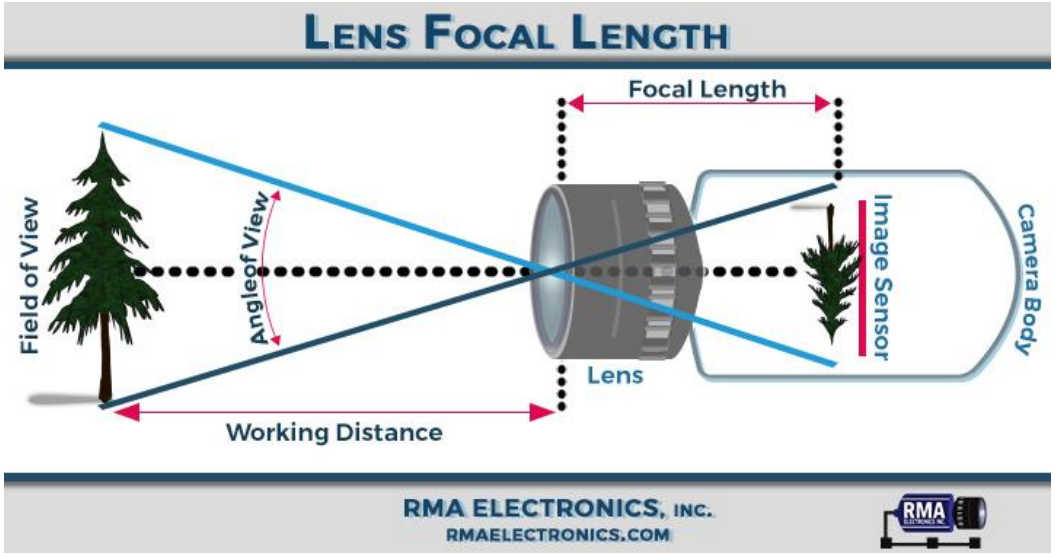
Where are the cubes?

General approach:

- Get the depth image from the robot
-
-
-
-
-

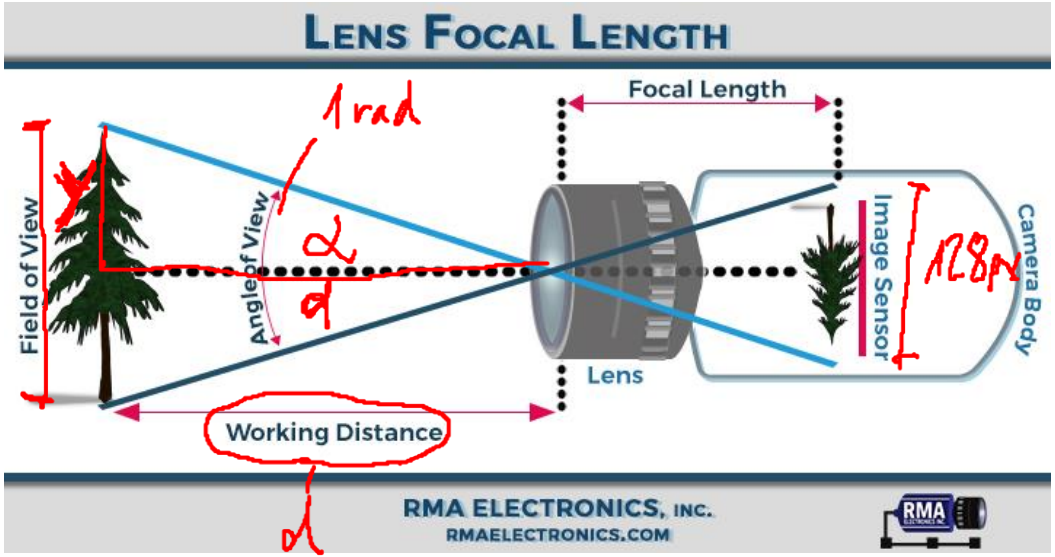
Camera Characteristics

- Using field of view (FOV) and sensor resolution to calculate $\rho[m/px]$



Camera Characteristics

- Using field of view (FOV) and sensor resolution to calculate $\rho[m/px]$



$$\rho = \frac{X}{\omega} = \frac{Y}{\omega/2}$$

$$\tan\alpha = \frac{Y}{d}$$

$$Y = d \cdot \tan\alpha$$

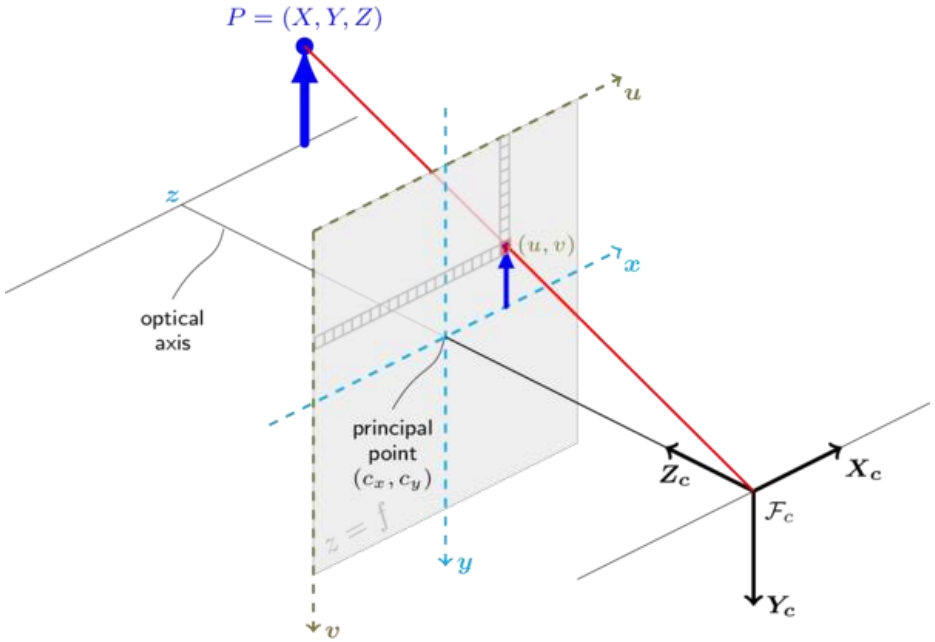
$$\rho = \frac{d \cdot \tan\alpha}{\omega/2}$$

$$\rho = \frac{0.3m \cdot \tan(0.5)}{64\,px} \approx 0.00256\ \frac{m}{px}$$

Camera Characteristics

Principal Point

- based on pixel coordinates u, v we can calculate the distance from the desired point



Camera Characteristics

Principal Point

- based on pixel coordinates u, v we can calculate the distance from the desired point

$c_x = w/2$

$c_y = h/2$

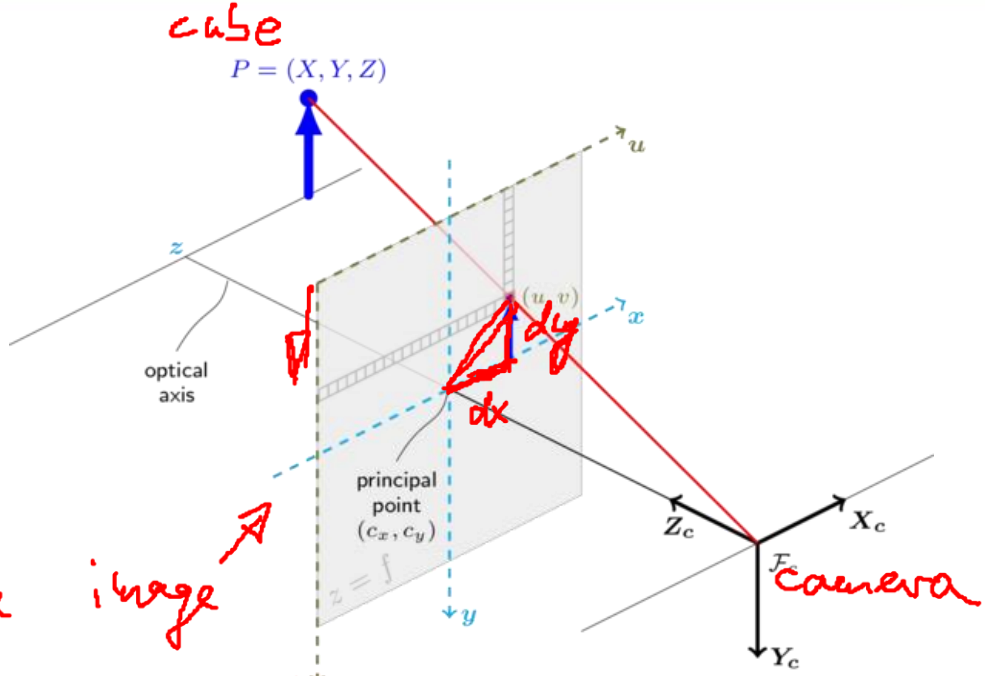
$dx = u - c_x$

$dy = c_y - v$

→ pixel difference image

real difference [m]: $s \cdot dx = x$

$s \cdot dy = y$



Understanding Transformations

In the first robot manipulation tutorial:

- we learned about the inverse and forward kinematics
- we learned how to move the robot to a (x, y, z) position in space, while keeping the same rotation

Let’s figure out how we can adjust the rotation as well!

- learn about different representations of rotation
- learn how to use them in Python

Understanding Transformations

Rotations in 2D

- This is similar to our mobile robots, which live on a plane and only have one angle
- 1 rotational DOF, can be described by 1 value

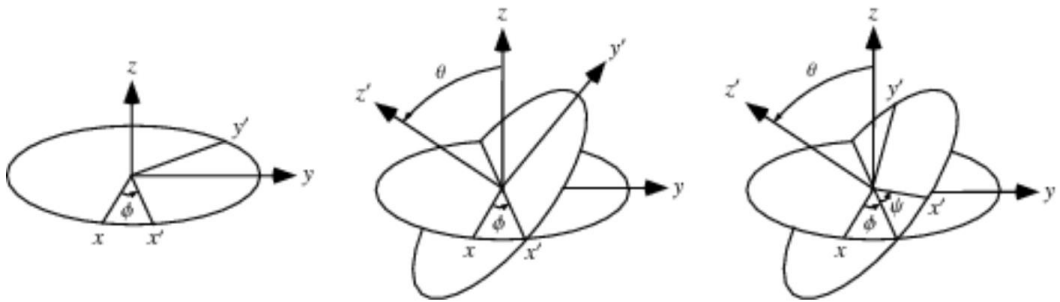
Rotations in 3D

- As applied to an end-effector of a manipulator arm
- 3 rotational DOF, can be described by 3 values
- there are multiple representations, most of which use more than 3 values

Understanding Transformations

Euler Angles

- rotation in 3D is defined by a set of three angles (θ, ϕ, ψ) and a rotation sequence
- rotation sequence defines the sequence of rotation axis and whether they are applied to static or rotating frame



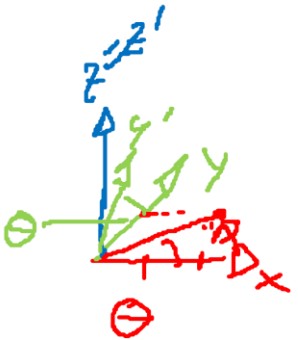
- in the picture: rotate around z, then around new x', then around new z'
- you have to be careful which sequence is being used (roll-pitch-yaw, ZXZ, …)
- it is difficult to perform calculations with them (addition, interpolation, etc.)

Understanding Transformations

Rotation Matrices

- defined by a 3x3 orthonormal matrix
- describing the axes of the new coordinate system relative to the old one – still quite intuitive
- e.g., rotation around z-axis:

$$Rz(\theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



- can be multiplied to chain transformations

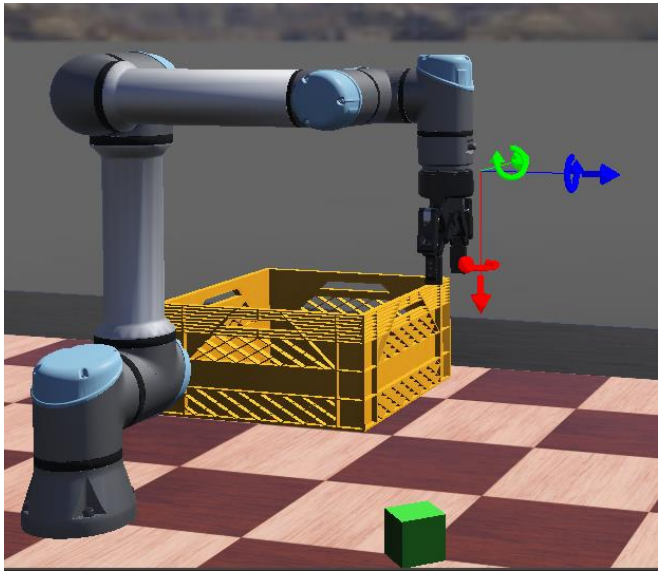
Understanding Transformations

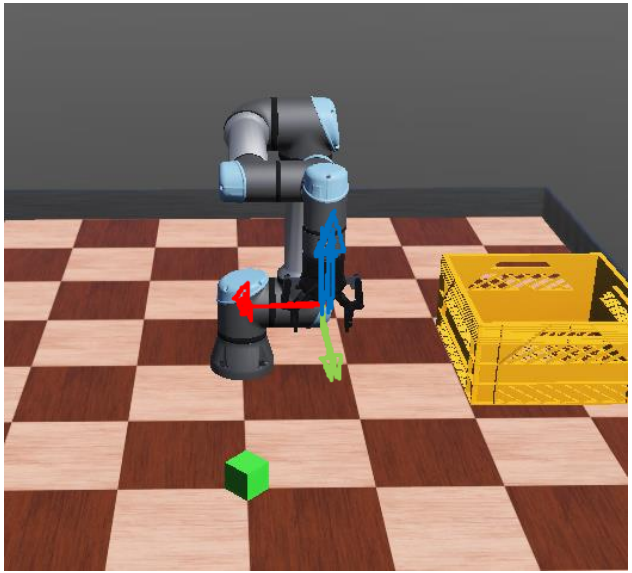
Quaternions

- defined by 4 values which are normalized to magnitude of 1 → “unit quaternion”
- not very intuitive… see for yourself, e.g. here: https://www.youtube.com/watch?v=zjMuIxRvygQ
- however, very good for computations, and less memory required than rotation matrix

Camera/TCP Transformation

- TCP: Tool Centre Point
- Note that camera and TCP have different coordinate systems!





Using Transformations!

How to use in our Webots project?

- check the source!
  - kinpy: https://github.com/neka-nat/kinpy/blob/master/kinpy/transform.py
  - transformations package: https://github.com/cgohlke/transformations/blob/master/transformations/transformations.py
- kinpy.Transform supports all three of the aforementioned representations!
- Let’s take a look at an example…