Sensing & Perception

Lab 3 – Computer Vision

Pôle

Numérique

Author	: Guillaume GIBERT
Version	: 1 R 0
Promotion	: 2026
Program	: EENG3
Date	: May 21st 2024

1. Introduction

The goals of this lab are:

- To calibrate a camera using a chessboard to estimate the intrinsic and distortion parameters;
- To learn how to detect/track a ball based on different colour representations;
- To servo a 2-link planar robot to follow the ball movements;
- To practise coding in C++;
- To practise updating a makefile;
- To control the versioning of your software using git/gitflow.

To do so, the following equipment is provided:

- A laptop running a Linux OS;
- A Poppy Ergo Jr Robot in the 2-link planar robot configuration;
- A webcam.

2. Setup

For this lab, you will be using PC running a Linux OS (Ubuntu). However, the same code should compile on Windows or Mac OS as you will be using multi-platform libraries (OpenCV, Dynamixel SDK).

The Dynamixel SDK (https://github.com/ROBOTIS-GIT/DynamixelSDK) and the OpenCV (https://github.com/opencv/opencv) libraries are already installed on the provided laptop. The full description of the installation process can be found in the appendix section 5 if you want to install it on your own laptop.

Please create a directory eeng3 in the home directory: /home/ros/eeng3 if it is not already the case. Please write your code in this directory.

3. Exercises

The goal of this lab session is to build a robotic system composed of a camera, a laptop and 2-link planar robot capable of servoing the position of the end-effector to the position of the ball. The pipeline of processing (represented in Figure 1) starts with grabbing an image that should be undistorted thanks to the intrinsic and distortion parameters of the camera obtained after camera calibration. Then, the ball position is detected based on its colour, the output of this function is a position in pixels (u, v) that should be converted to a position in the real world (x, y) in mm. The Inverse Kinematics function converts this position to joint values and the robot moves accordingly.

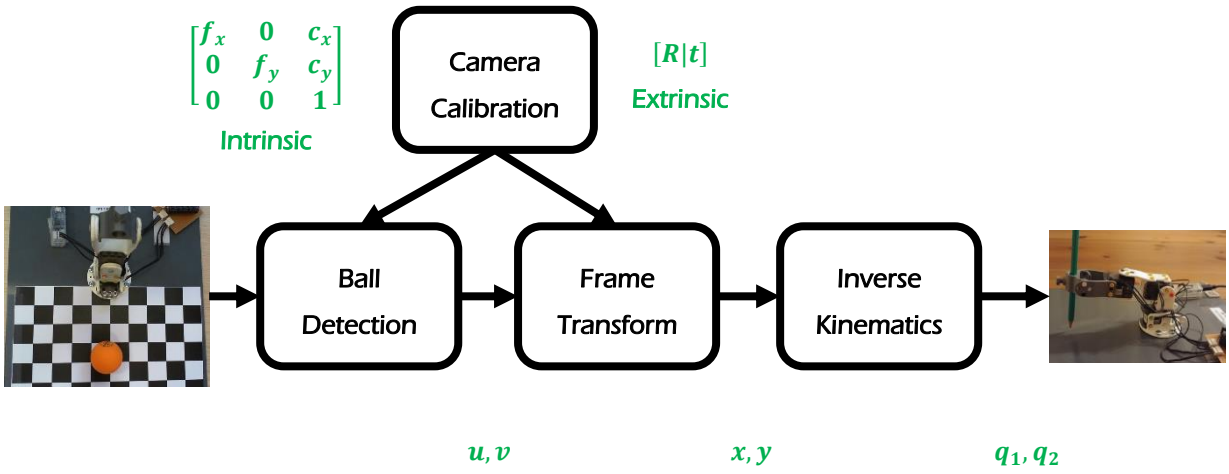

Figure 1 : Schematic representation of the pipeline of processing: an image is grabbed and is undistorted thanks to the intrinsic parameters of the camera, then the ball position is detected based on its colour leading to a position in pixels (u, v) which is transformed in a position in mm (x, y); this position is used by the inverse kinematic module of the robot to estimate the corresponding joint values that are sent to the robot.

3.1. Camera calibration

3.1.1. Git repository

Open a web browser at this address: https://gitarero.ecam.fr/user/login

Log in with your Ecam credentials

Create a new git repository entitled SensingLab3 (keep the default configuration)

Open a Terminal

Go to the target directory using the following command:

$ cd /home/ros/eeng3

Type the following command to clone the git repository you have just created:

$ git clone https://gitarero.ecam.fr/[username]/SensingLab3.git

Update the description of the project and create a commit

Create a develop branch

3.1.2. Camera calibration

Nowadays, cameras use complex lenses and can grab high quality images. However, some distortions may appear at the image edges. To reduce these distortions, camera calibration can be used (see https://docs.opencv.org/3.4/d4/d94/tutorial_camera_calibration.html). Following the pinhole camera model, the projection of a 3D point (X_w, Y_W, Z_w) of the real world onto the image plane is a pixel at the position (u, v). The equation that relates the coordinates of these two points is:

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} [R|t] \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

The first matrix is composed of the intrinsic parameters of the camera (focal length and optical centre) whereas the second one corresponds to the extrinsic parameters (rotation and translation of the camera reference frame with respect to the word reference frame).

The calibration procedure uses several images of a chessboard taken with different viewpoints to estimate the intrinsic and the distortion parameters of the camera. Knowing these parameters, images can be undistorted.

3.1.3. Code

Create a src, a include, a data, a lib and a bin folder

Copy the file CameraCalibration.cpp in the src folder

Copy the file default.xml in the data folder

Copy the makefile

Compile the code using the following command line:

$ make

Launch the program using the following command line:

$ ./bin/CameraCalibration data/default.xml

Press g to start grabbing images, move the chessboard to capture images with different viewpoints

Press u to display the distorted and undistorted images alternatively

Once the calibration procedure is over, a file is created to store the intrinsic and distortion parameters. Open this file and locate the intrinsic and distortion parameters.

Create a commit

Merge your code into the main branch

Push it to the remote repository

3.2. Colour detection

3.2.1. Colour representation

A colour image is composed of 3 channels (sometimes even 4 channels for transparency). By default, a colour image is represented by **B**lue, **G**reen and **R**ed (BGR) channels on OpenCV. This type of representation is efficient for computer but difficult to understand for humans. Indeed, humans prefer using the **H**ue, **S**aturation, **V**alue (HSV) representation as represented in Figure 2. The **H**ue channel corresponds to the colour, the **S**aturation channel corresponds to the "intensity" of the colour and the **V**alue channel corresponds to the "brightness". To find a given colour in an image, usually you look up for the range of **H** and **S** using the colormap and leave the **V** channel to the range [20, 255]. For example, to find the yellow colour using the colormap provided in Figure 2, the **H** channel is in the range [25, 35], the **S** channel is in the range [100, 255] and the **V** channel is in the range [20, 255].

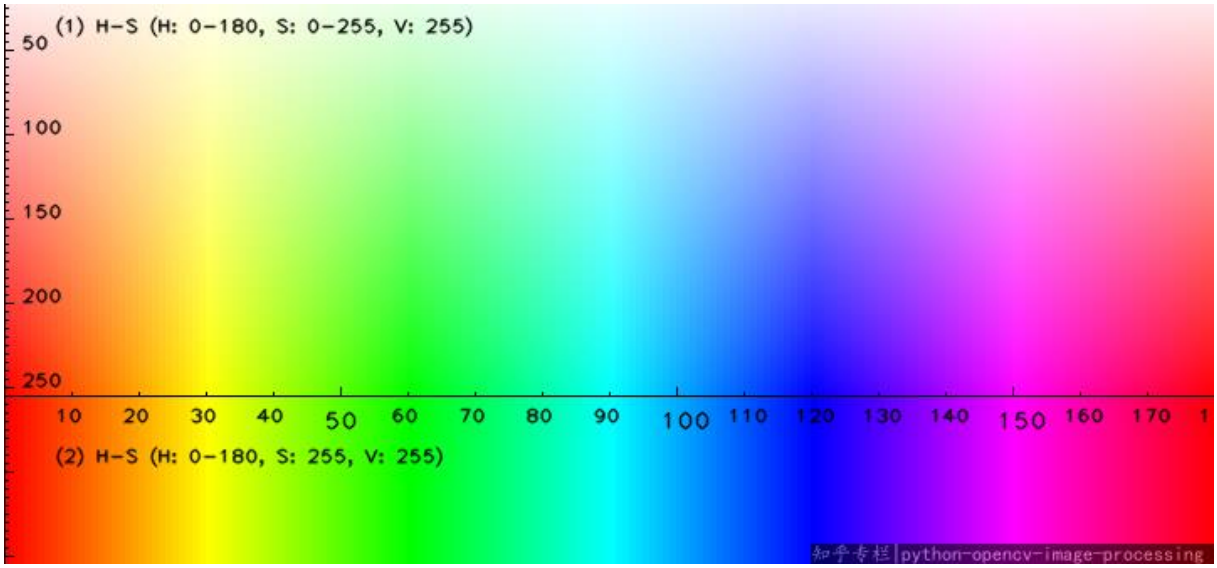

Figure 2: Colormap of the HSV representation in OpenCV: on the x-axis, variation of the Hue channel corresponding to colour while on the y-axis, variation of the Saturation channel corresponding to the "intensity" of the colour.

3.2.2. Code

Move to the develop branch

Copy the file RedBallDetection.cpp in the src folder

Update the makefile to compile and link this new cpp file

While the STL is known by the system and does not need further settings from your part, you need to provide more information to the compiler/linker so it can access the OpenCV files:

- During the **compilation** phase, you need to provide the directory where to find the h files of the OpenCV library by adding the following argument: **-I**<path to the include files> .
- During the **linking** phase, you need to provide the directory where to find the object/library files of the OpenCV library by adding the following argument: **-L**<path to the object/library files> and also the object/library you want to link by adding the following argument: **-l**<name of the object/library> .

For your information, the tool *pkg-config* retrieves information about installed libraries in the system.

In general, you can use the following line to retrieve the necessary information for compilation/linking against a given library:

$ pkg-config –cflags –libs <name of library>

Compile the code using the following command line:

$ make

Launch the program using the following command line

$ ./bin/RedBallDetection -o <color_param_filename> -f <fps> -i <camera_param_filename> -s <structural_elem_size>

This program has several options: change the output filename (-o), change the framerate (-f), change the camera parameters filename (-i), change the size of structural elements (-s). Default values for all these options are provided in the code.

Use the trackbars to select the ranges of the H, S and V channels.

Once the ranges are selected, press s to save them in a file.

Create a commit

Update the code to select the ranges either in the HSV or in the BGR representation.

Create a commit

Merge your code into the main branch

Push it to the remote repository

Q. What is the impact of the size of the structural elements on the ball detection?

3.3. Color tracking

Move to the develop branch

Copy the file RedBallTracking.cpp in the src folder

Update the makefile to compile and link this new cpp file

Compile the code using the following command line:

$ make

Launch the program using the following command line

$ ./bin/RedBallTracking -c <color_param_filename> -f <fps> -a <area_threshold> -s <structural_elem_size> -i <camera_param_filename>

This program has several options: change the colour params filename (-c), change the framerate (-f), change the area threshold (-a), change the size of structural elements (-s), change the camera parameters filename (-i). Default values for all these options are provided in the code.

Create a commit

Q. What is the impact of the size of the structural elements on the ball tracking?

Q. What is the impact of the area threshold on the ball tracking?

(**Optional**) Implement the code that can make the ball disappear when you press 'm'

Q. How would you implement it?

Create a commit

Merge your code into the main branch

Push it to the remote repository

3.4. Reference frame transform

The camera is placed on a pole at a height of around 35 cm. It grabs top view pictures of the scene. Therefore, the image plane is parallel to the horizontal plane. In this constrained environment, the {camera} reference frame and the {robot}one are only separated by a pure translation as represented in Figure 3.

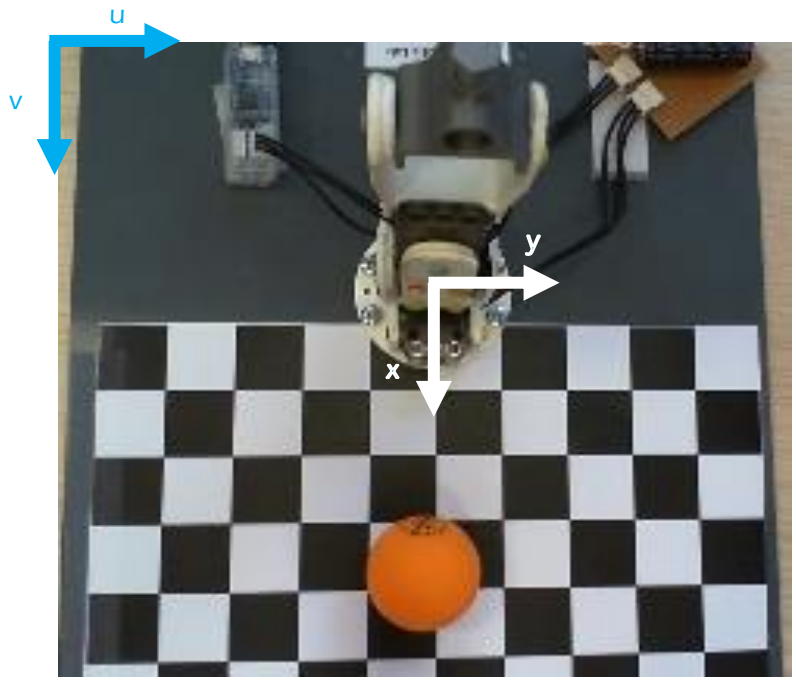

Figure 3: Top view of the robotic system with the two reference frames: {camera} in blue and {robot} in white.

The images grabbed with OpenCV have a resolution is 640 x 480 pixels.

Using a ruler, determine the ratio cm by pixel in the x and y directions

Deduce the horizontal and vertical Fields Of View (FOV)

Q. Can you confirm these values with the technical specifications of this camera?

Centre the (u, v) coordinates by subtracting half of the image width and height

Apply the ratio cm/pixel to convert this pixel coordinates to cm:

$$\begin{cases} x = \left(v - \frac{im_{height}}{2}\right) * ratio_{cm/pix_{height}} + \\ y = \left(u - \frac{im_{width}}{2}\right) * ratio_{cm/pix_{width}} \end{cases}$$

N.B.: An offset may be added along the x-axis depending on the position of the centre of the camera.

N.B.: These relations stand only if the {camera} and the {robot} reference frames share the same horizontal plane. If the camera was slightly tilted, the relationship would be more complicated. In this case, the extrinsic parameters should be estimated to obtain the transformation matrix (rotation + translation) that gives the correspondence between the two reference frames (see Figure 4).

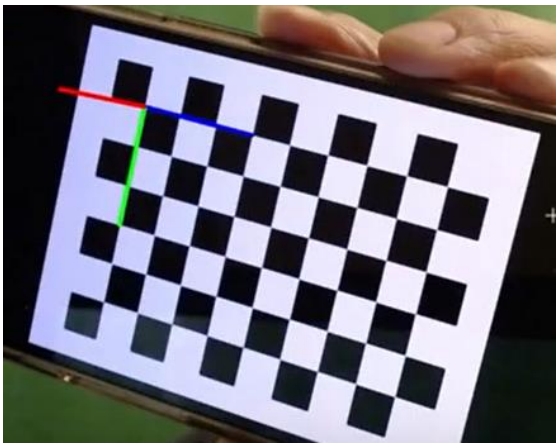

Figure 4: Extrinsic parameters of the camera corresponds to the transformation between 3D points in the real world and 2D points in the image plane. Once determined, the world reference frame can be projected on the image plane.

3.5. Robot servoing

Move to the develop branch

Copy the file RobotServoing.cpp in the src folder

Update the makefile to compile and link this new cpp file

Compile the code using the following command line:

$ make

Launch the program using the following command line

$ ./bin/RobotServoing -c <color_param_filename> -f <fps> -a <area_threshold> -s <structural_elem_size> -i <camera_param_filename>

This program has several options: change the colour params filename (-c), change the framerate (-f), change the area threshold (-a), change the size of structural elements (-s), change the camera parameters filename (-i).

Create a commit

Play with the different parameters and conclude on their impact

Merge your code into the main branch

Push it to the remote repository

Check the graph of the branches using the following command:

$git log –graph

4. Links

https://opencv.org/

https://docs.opencv.org/

https://github.com/opencv/opencv

https://learnopencv.com/find-center-of-blob-centroid-using-opencv-cpp-python/

https://nghiaho.com/?page_id=671

5. Appendix

5.1. Installation of Dynamixel SDK

5.1.1. Get sources

By default, the Dynamixel SDK is not installed on the laptop. Therefore, the first step before using it is to install it. Please follow the instructions below:

Move to the eeng3 directory using the following command:

$ cd eeng3

Clone the Dynamixel SDK git repository on the laptop using the following command:

$ git clone https://github.com/ROBOTIS-GIT/DynamixelSDK.git

At this point, the latest source code of the SDK is copied on your laptop. You need to compile/link the code to create object files/libraries (i.e., binary code) that you can use in your own programs later on.

Go into the DynamixelSDK/c++/ directory using:

$ cd DynamixelSDK/c++

Go into the build directory using:

$ cd build

Go into the linux64 directory using:

$ cd linux64

5.1.2. Compile/Link

In this directory, there is a makefile that describes the rules to compile/link all the source code of the DynamixelSDK.

Compile/Link the Dynamixel SDK using the following command:

$ make

Display the content of the directory again

5.1.3. Install

Install the libraries and include files of the SDK into the OS using the following command:

```
$ make install
```

5.1.4. Upgrade USB access

Please add account id to dialout group in order to access the USB port. Replace the <your_account_id> in the command below to your actual user id:

```
$ sudo usermod -aG dialout <your_account_id>
```

5.2. Installation of OpenCV library

To setup a PC running a Linux OS, you shoud install the OpenCV library using the following command:

```
$ sudo apt-get install libopencv*
```

To setup a PC running a Windows OS, you may follow the instructions described here: https://medium.com/csmadeeasy/opencv-c-installation-on-windows-with-mingw-c0fc1499f39