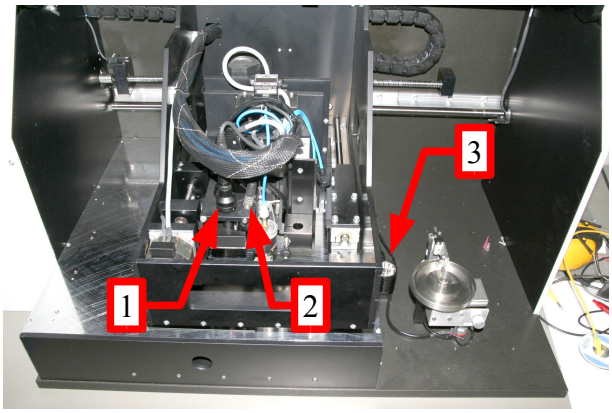
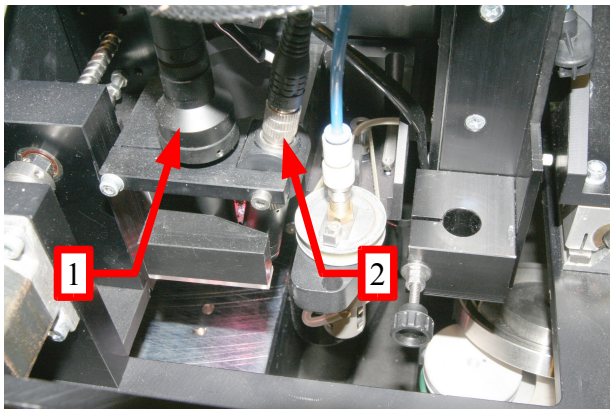
Camera API

Introduction

The pick and place machine has three cameras. Two of them are placed on the moving head of the machine facing downwards. The “detail camera” (1) and the “overview camera” (2). The “lookup camera” (3) is placed beside the working area, facing upwards.



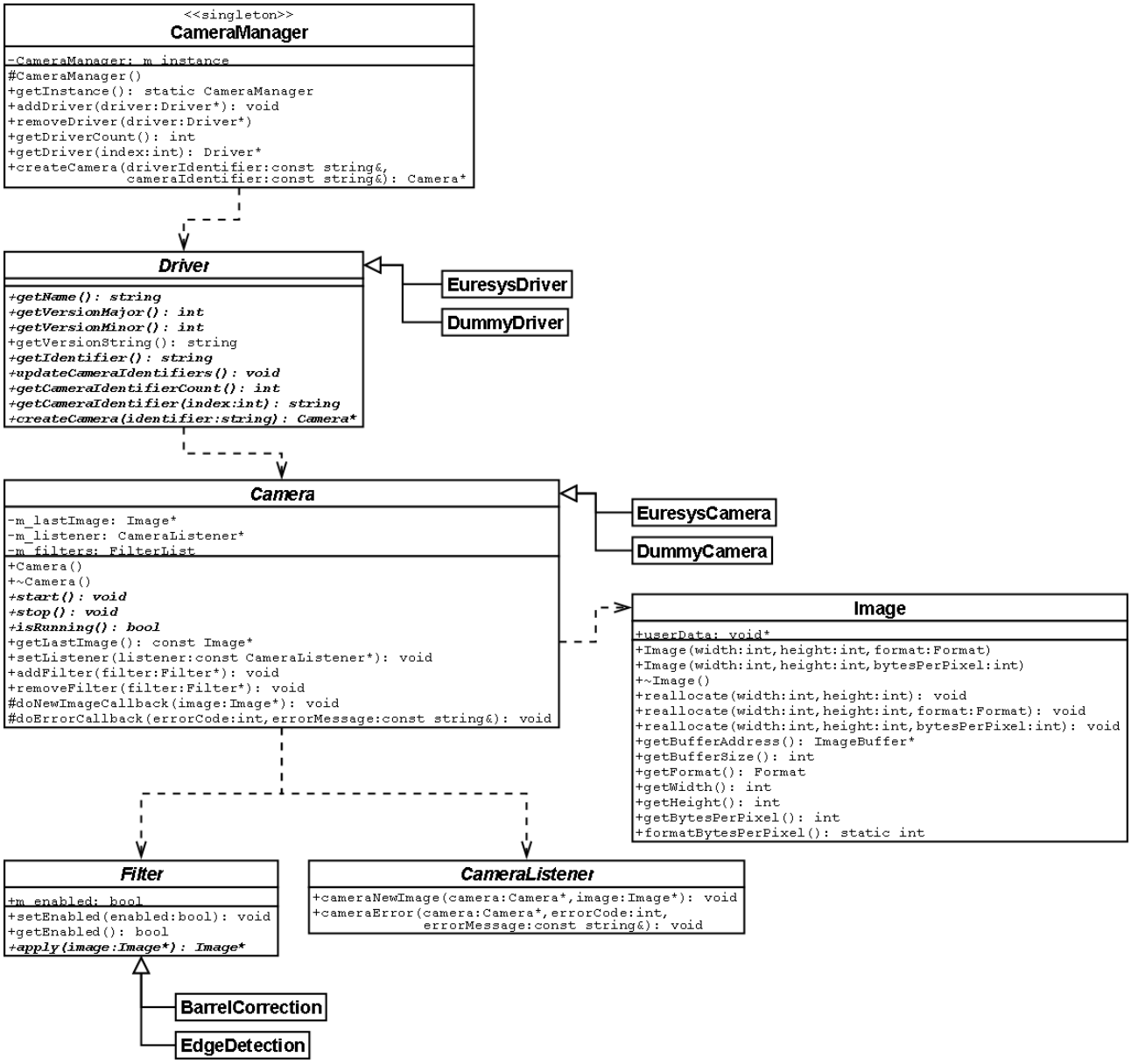


These cameras are connected to a frame grabber card (Euresys Picolo Pro 2) in the computer. To be able to get the image from the cameras into the program without having to write driver specific low level code, we developed the camera API.

The goals when designing the camera API was:

- It should be easy to add support for new frame grabber cards.
- It should be possible to apply filters for doing image enhancement and modification.
- The allocation and deallocation of image buffers should be managed by the API.
- The API should be easy to understand and use.
- All classes in the API should be well documented using doxygen.

Design



Camera manager

The CameraManager class is used to gather all drivers in one place and simplifies the creation of Camera instances. The CameraManager implements the singleton design pattern to make all drivers available anywhere in the application.

Driver

A driver is used to list and create cameras for a specific frame grabber card. At the moment there are two implementations of the Driver class. A dummy driver that can be used for testing on a computer without any frame grabber card, and a Euresys driver that adds support for the Euresys Picolo Pro 2 (with driver version 3.x). To add support for another frame grabber card a version of the driver class for that card can be implemented.

Camera

The camera, like the driver, only supports a specific frame grabber card. The camera is

used to start and stop the stream of images.

Filter

Filters are used to modify images coming from the camera before they are passed on to the camera listener. Many image filters (like brightness, saturation, inversion, etc.) can be done directly on the incoming image (in-place). The apply method then returns a pointer to the input image. Other filters (like edge detection, blur, resizing, etc.) must have a separate result image that is returned.

Image

The image class is used to simplify the memory allocation and handling. It also holds some properties for the allocated image buffer like width, height and pixel format. The pixel format describes how the information for each pixel is encoded.

Barrel distortion

Because the overview camera has a wide angle lense and is tilted to capture the same area that the detail camera, the image from it is distorted. The wide angle lense will give an barrel distortion while the tilting will give a perspective distortion. To be able to pick coordinates from the image and draw graphical feedback of the commands the machine should execute, the image needed to be corrected.

A barrel correction filter was developed that corrected both the barrel distortion and the perspective distortion. The filter calculates the mapping from the correct coordinates (x, y) to the distorted coordinates $(\hat{x}, \hat{y})$ and then copies the pixel color from that pixel to the corrected image.

$$CorrectedImage(x, y) = DistortedImage(\hat{x}, \hat{y})$$

The distorted coordinates is calculated

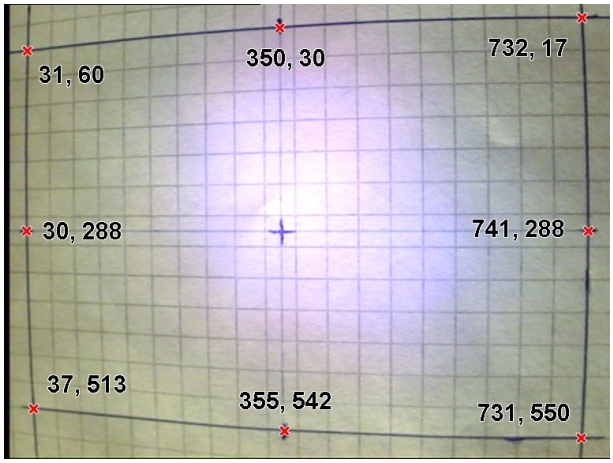
$$\hat{x} = a_1 x^2 y + a_2 x y^2 + a_3 x^2 + a_4 y^2 + a_5 x y + a_6 x + a_7 + y + a_8$$
$$\hat{y} = b_1 x^2 y + b_2 x y^2 + b_3 x^2 + b_4 y^2 + b_5 x y + b_6 x + b_7 + y + b_8$$

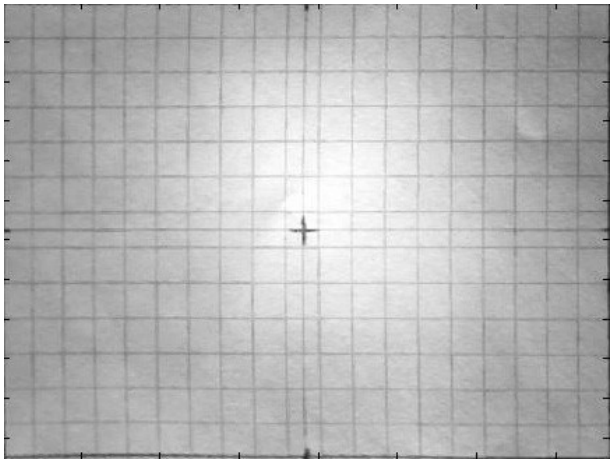
The $a_{1...8}$ and $b_{1...8}$ is calculated by solving the linear equations $\hat{x} = M * a, \hat{y} = M * b$ where

$$M = \begin{bmatrix} (x_1^2 y_1) & (x_1 y_1^2) & (x_1^2) & (y_1^2) & (x_1 y_1) & (x_1) & (y_1) & 1 \\ \dots & \dots & \dots & \dots & \dots & \dots & \dots & \dots \\ (x_8^2 y_8) & (x_8 y_8^2) & (x_8^2) & (y_8^2) & (x_8 y_8) & (x_8) & (y_8) & 1 \end{bmatrix}$$

By picking distorted coordinates that represents a distorted rectangle and then use the coordinates for the corners and the the center of each edge when calculating the mapping, the distorted rectangle will correspond to the border pixels of the corrected image.

Here is an example of an image from the overview camera that has been corrected using the barrel correction filter.





Code example

The following example shows how to create a camera and start the acquiring.

```
using namespace camera;

// Get the camera manager instance
CameraManager *cameraManager = CameraManager::getInstance();

// Create a dummy driver and add it to the camera manager
DummyDriver *dummyDriver = new DummyDriver();
dummyDriver->setImageSize(768, 576);
cameraManager->addDriver(dummyDriver);

// Print all drivers and the available cameras for each driver
for(int i = 0; i < cameraManager->getDriverCount(); i++) {
      printf("Driver #%d: %s", i, cameraManager->getDriver(i)->getVersionString());
      for(int j = 0; j < cameraManager->getDriver(i)->getCameraIdentifierCount(); j++) {
            printf("       Camera identifier #%d: %s", j, cameraManager->getDriver(i)-
>getCameraIdentifier(j));
      }
}

// Create the first camera of the first driver
std::string driverIdentifier = cameraManager->getDriver(0)->getIdentifier();
std::string cameraIdentifier = cameraManager->getDriver(0)->getCameraIdentifier(0);
Camera *camera = cameraManger->createCamera(driverIdentifier, cameraIdentifier);

// Add the barrel correction filter to the camera
int distortedRectangle[8][2] = {{31, 60}, {350, 30}, {732, 17}, {30, 288}, {741, 288},
{37, 513}, {355, 542}, {731, 550}};
BarrelCorrection *barrelCorrection = new BarrelCorrection(distortedRectangle);
camera->AddFilter(barrelCorrection);

// Set the listener and start the camera
camera->setListener(myCameraListener);
camera->start();

//Delete the camera and the filter
delete camera;
delete barrelCorrection;
```