

RT Systems Lab

**Weekly Report number (1)**



Contact Us

Julian Ewaied 215543497 Namir Ballan 326165156

Contents

[This Week's Tasks 3](#_Toc141218558)

[Tasks Done 3](#_Toc141218559)

[Understanding Camera Matrix 3](#_Toc141218560)

[Installing OpenCV and Eigne3 libraries for C++ 5](#_Toc141218561)

[Implementing xy recovering process 6](#_Toc141218562)

# This Week's Tasks

* Understanding the camera matrix.
* Installing OpenCV and Eigne3 libraries for C++
* Summarizing important library functions
* Implementing xy recovering process.
* Implementing z-coordinate recovering

# Tasks Done

## Understanding Camera Matrix

The camera is a physical device which takes a 3D scene, and converts it into pixels. To be more precise, it takes all points and converts them into points , where all the points are of depth 1. Since this is not a bijection, we need more than one picture to recover the z-coordinate. The camera matrix is a matrix that recovers the points with normalized z coordinate . Let let the camera matrix, then:

Where is the center of the image in pixels and are the focal length of the image.

With simple calculations one can see that:

If is the depth of the point, then

Now let's look at two frames with vertical translation upwards as described in the following figure.

Figure 1: read frame is frame k-1, and black frame is frame k. we look at the y coordinate of the same point in two different frames, and calculate the depth using the distance

We can see that

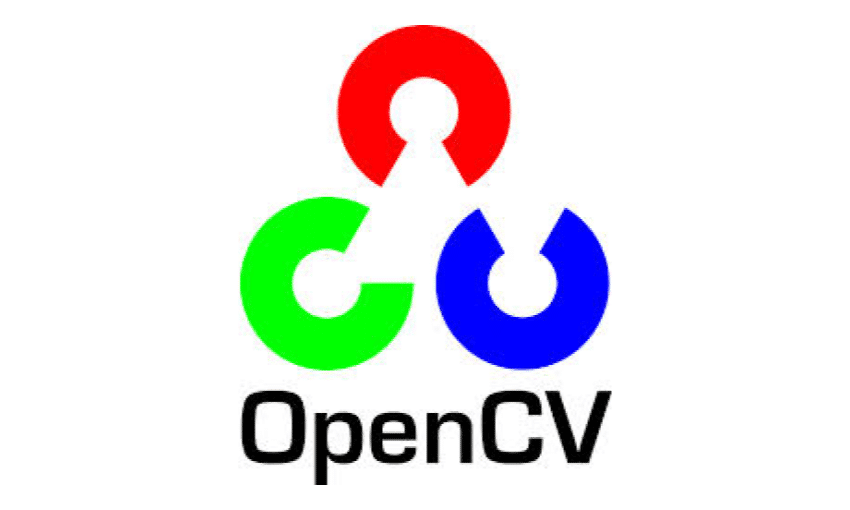
And therefore, we can conclude that:

And hence,

Therefore, we can conclude that:

## Installing OpenCV and Eigen3 libraries for C++

All the packages we needed are OpenCV for image files handling, and Eigen3 for doing the calculations as fast as possible. The installation was very complex at the beginning, but then we realized we can simplify it using visual studio. After downloading the code files of each of the libraries, all we need to do is to add the path to the library code to the properties. The following videos explain perfectly how to do it.



Download OpenCV



OpenCV tutorial



Download Eigen



Eigen tutorial

Note! OpenCV automatic installer can't download to a privileged folder, be careful.

We ran a sample OpenCV code (and also Eigen3 code, but it's less interesting):

#include <iostream>

#include <opencv2/core.hpp>

#include <opencv2/imgcodecs.hpp>

#include <opencv2/highgui.hpp>

using namespace cv;

int Run()

{

cv::Mat img = cv::imread("C:/Users/WIN10PRO/Pictures/fish.png");

namedWindow("Test", WINDOW\_AUTOSIZE);

imshow("Test", img);

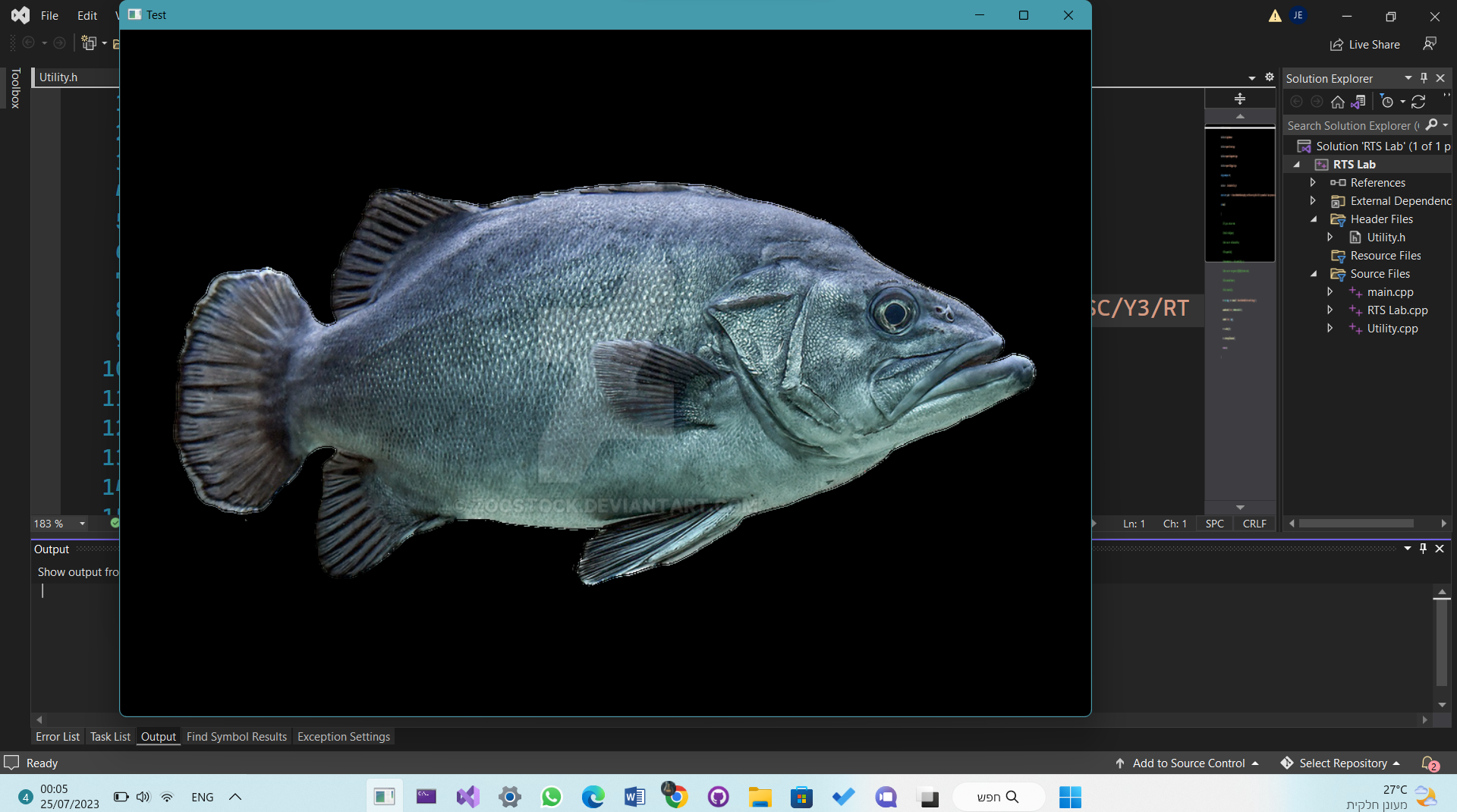
cv::waitKey(0);

cv::destroyAllWindows();

return 0;

}

Here is the result:



We also fixed the CSV class to make it return a matrix of Eigen::Vector3i objects instead of tuples for later use, and tested it for a simple csv file with the following driver code.

#include <iostream>

#include <Eigen/Dense>

#include "../include/Utility.h"

const char\* path = "C:/Users/WIN10PRO/Desktop//test.csv";

int Run()

{

CSVFile file(path);

file.openFile();

vector<matrix> v = file.readFile(0, 1);

std::cout << v[0][0][1] << std::endl;

file.setPath("None");

file.closeFile();

}

## Implementing xy recovering process

We are currently implementing the calculation . For that we built a class called Analyzer, which takes as an input the camera information, processes the data. The first step was to build the camera matrix inverse, which we did in two steps: first building, then retrieving the matrix. This way, we won't calculate the inverse camera matrix each time we need it, but rather only the first time. Notice that there is no need for us to change the camera parameters in the running stage, so we simply kept them private with no setting methods. We ran a sample example and got the following inverse camera matrix:

We also had a static function that given a rotation axis and a rotation angle in degrees, it returns the rotation matrix. Here's a sample code that we used to do a sanity-check.

int Run()

{

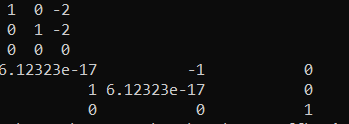
CSVFile file(path);

Analyzer a(1, 1, 2, 2);

a.buildCameraMatrix();

cout << a.getCameraMatrix() << endl;

cout << Analyzer::getRotationMatrix(90, Eigen::Vector3d(0, 0, 1));

}

Next, we will need to get a vector of 2D integer points, and convert them all into the normalized 3D form, which will be done with the function project3D.