Exercise 2: Computer Vision[*]

EENG350: Systems Exploration, Engineering, and Design Laboratory

Tyrone Vincent

Department of Electrical Engineering
Colorado School of Mines

Fall 2022

1 Introduction

The objective of this module is to introduce you to OpenCV. This will help give the robot the gift of vision, enabling it to detect beacons and react accordingly. In real time, the robot will need to:

- Take a picture
- Identify the specific beacon(s) detected.
- Calculate the relative position and orientation from itself to the beacon(s). This will help the robot understand its current position and where it needs to move to next.

This tutorial is meant to get you to the point where you are able to detect different colors and shapes in an image, and perform some processing that removes background noise and cleans up the image.

Exercise Objectives:

- Work in the Open CV environment on the Raspberry Pi
- Capture and store and image on the Raspberry Pi
- Use Open CV to process images, detect markers, and determine the angle and distance to markers

2 Setting up the Camera

Shut down the Pi and connect the camera module to the Pi. You are using a 5 MP (MegaPixel) camera. The specifications for this camera can be found here.

You will use a python implementation of OpenCV to process the images from the camera. OpenCV has been pre-installed on your Rasberry Pi. If you would like to re-install it, or install it on another Pi, use the instructions in the Appendix.

To run your code at the command line, type `python3 <filename.py>`. You should also be able to use the IDEs for Python 3.

3 Pi Camera and OpenCV Tutorial

Use the Open CV documentation as a reference. You may need to select a different version of the documentation depending on what was loaded on your Pi. At the python command line, type `import cv2` and then

[*]Developed and edited by Tyrone Vincent and Vibhuti Dave with assistance from Darren McSweeney. This work is licensed under the Creative Commons Attribution-NonCommercial-ShareAlike 3.0 Unported License. To view a copy of this license, visit http://creativecommons.org/licenses/by-nc-sa/3.0/ or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

`print(cv2.__version__)` to find the version that was installed. Note that there are two underscores on the left and right of the word `version`.

Here is a beginner's tutorial to capturing an image, recording video, and applying some image effects to the pictures you take. This tutorial uses the PiCamera module to take images. For real-time images, it is a good idea to change the auto white balance of the camera. The white balance is set to auto by default. You may wish to fix the white balance after an initial calibration ensure consistent images (scroll down to section 3.5). In the tutorial, the images are saved to a file, but you will also want to be able to capture to an OpenCV object. Later on, you might also be interested in reading section 4.9: Rapid capture and streaming

Here is a set of tutorials to use OpenCV with Python. **Right now** go through **at least** the tutorials on "GUI features for Open CV" , "Core Operations", and "Image Processing in OpenCV" (OpenCV should already be installed, so you don't need to go through the "Install" tutorial) . You should also bookmark these pages and go back to it as necessary as you need more features from picamera and OpenCV. In this tutorial, you use the OpenCV `VideoCapture` function to capture images. This can be useful to be able to quickly get images into memory, but is a little trickier to use when trying to set the camera speed, white balance, etc.

In the tutorial, you will have noticed that they use the Numpy module. Numpy is an optimized library for fast array calculations. Refer to this Python-Numpy tutorial for any questions related to Numpy.

Aruco markers are a very useful tool within OpenCV. Here is the Aruco detection tutorial. Read through and try the example code through Marker Detection. This tutorial uses C++ functions, but you can figure out the Python calling format by comparing to the examples here and here.

4 Camera and OpenCV Exercises

The following are exercises to demonstrate to a TA or instructor. You can do each in a separate python file, or have each exercise in the same file but within a function, for example

```
def cv_exercise1():  ...
```

(1) Write a Python script that does the following:

  a. asks the user for a filename to store the image

  b. captures an image. Note: For real-time images, it is a good idea to change the auto white balance of the camera. The white balance is set to auto by default. You may wish to fix the white balance after an initial calibration ensure consistent images.

  c. stores the captured image using the filename provided by the user. Your program and image will be in the same directory.

  d. displays the image on the screen

  (Hint: There is an example code called take `picture.py` on Canvas. You can use this as a starting point.)

(2) Write a Python function that takes an image save directly to a OpenCV object, resizes the image to half its size without changing the aspect ratio (Hint: Use `cv2.resize`). Refer to geometric transformations of images to get details on scaling an image. Example: a 1280 X 1118 should be resized to 640 X 559. Show the resized image and have the function return the resized image.

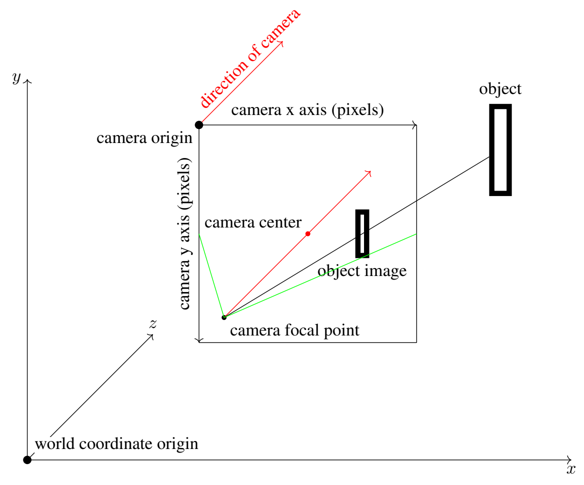
(3) Write a Python function that takes an image and converts it to a grayscale image using cv2.cvtColor. Show the new image.

(4) Using the examples from the Aruco tutorial, create image files of two Aruco markers, and then print these images (you can move the image file to another computer using a USB disk, the Canvas file function or a cloud service like Google Drive). Then use your image-to-file program from the previous section to take several images with one or two markers in the image, and an image without any markers. Write a Python function that loads an image, converts it to grayscale, detects if the image has an aruco markers, and prints the IDs of the aruco markers in the

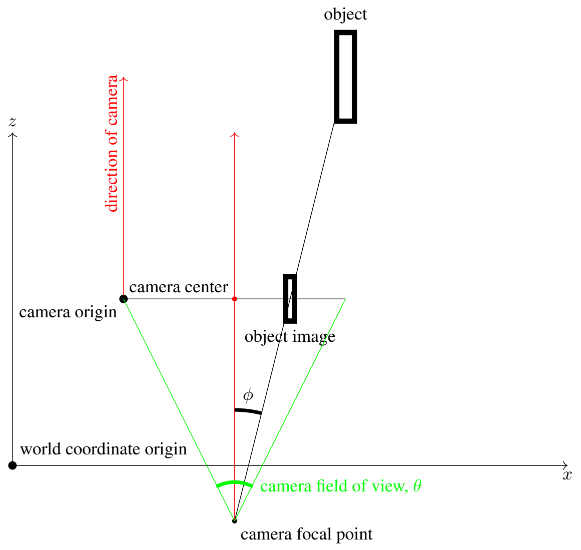
image. If there are no aruco markers, the function should print "No markers found". Use the Aruco tutorial and the aruco.detectMarkers() function to do the detection.

(5) Write a Python script that continuously takes a picture using the camera (using the method described in Getting Started with Videos in the openCV tutorial), converts it to grayscale, and runs the Aruco detection. Have the code print the ID of the marker if one is found, otherwise print "No markers found". Print out at least 2 Aruco markers and demonstrate that the code works when the camera sees one marker or multiple markers. (Having trouble taking multiple images? Read the PiRGBArray class documentation carefully)

(6) If the location of a marker in the image is known, the position of the marker relative to the camera can be determined. For this exercise, you will find the horizontal angle between the camera center and the object, which is shown as ϕ in the figure (b) below. The angle ϕ describes the angle that the camera would have to turn in order to face the object. The angle from the camera to the object can be determined using the pinhole camera model, which assumes movement in the image horizontally or vertically is proportional to movement in the world in a plane perpendicular to the camera. The angle from the camera to the object is thus the same as the angle from the image center to the image of the object, and if you know the camera field of view (shown in green in the figure below). Note that the field of view is the angle from one edge of the image to the other edge, and half the field of view is the angle from the center of the image to the edge of the image. The angle of the object ϕ can be determined by multiplying half the field of view by the following ratio: The number of pixels from the center of the image to the pixel in the middle of the object, divided by the number of pixels from the center of the image to the edge of the image. Modify your code so that the angle to the yellow hexagon from the camera can be determined.



(a) In the figure on the left, the camera plane is shown in perspective in the world coordinate system. An object is shown in the world, and the object image is displayed on the camera image. Each pixel in the object image is on a straight line between the camera focal point and the corresponding point on the object



(b) In the figure on the right, the same scene is shown, but looking down from above. The camera field of view is noted, and the angle of the object relative to the camera ϕ, is also shown.

Note: There is a more complicated method for determining pose discussed in Aruco detection tutorial, but only use that if you are familiar with or ready to learn about the camera matrix and camera calibration, and you understand how to interpret the resulting pose transformation matrix. This method can be more accurate if the marker is not facing the camera, but is not necessary if the vertical plane of the marker is parallel to the camera y axis (such as would occur if the marker is on a stand the same height as the robot), and the vertical distance is used to determine the object size.

5 Documentation

Create a document that includes your well documented code and **results** (i.e., image files), along with text that allows the reader to understand the purpose of the code and the significance of the results for each of the exercises above. In addition, discuss the following:

1. How is an OpenCV image stored in memory?
2. What does `image.shape` return? Here image is the variable name that stores the image you are referring to.
3. What do the parameters, `fx` and `fy` refer to in `cv2.resize`?
4. What would happened if, `cols/2`, and `rows/2` in the following function were changed to `cols/4` and `rows/4`?

   ```
   M = cv2.getRotationMatrix2D((cols/2,rows/2),90,1)
   ```
5. What function must be used after `cv2.getRotationsMatrix2D` to actually perform the rotation? What parameters do you pass to this function?
6. How does Aruco marker detection work? What are the steps taken by `aruco.detectMarkers()` to detect an Aurco marker?
7. What is the dimension of the arrays `ids` and `corners` from the output of `detectMarkers()` when it sees 2 aruco markers?
8. `detectMarkers()` has adjustable parameters. From aruco.detectMarkers() you can click on this parameter to get more detail. What parameter determines the minimum perimeter for a marker contour to be detected? How does this affect the size of markers that can be detected? I would encourage you to try to change this parameter on an example image to verify your answer. Note that in python, the code `parameters = aruco.DetectorParameters_create()` creates the parameter object, and elements of the object can be changed using the dot notation, e.g. `parameters.adaptiveThreshWinSizeMin = 1`. You can get a list of all attributes of an object via `dir(parameters).`

Upload a copy of this document to the assignment link in Canvas.

6 Demonstration

Be prepared to demonstrate your results, explain your code, and modify your code to achieve specific goals.

7 Appendix

If you need to re-install OpenCV on Python 3 for any reason, use the following steps:

- Open a command prompt, and run the following commands:
  - `sudo apt-get update`
  - `sudo apt-get install python3-pip`
  - `pip3 install opencv-python`
  - `pip3 install opencv-contrib-python`
  - `sudo apt-get install -y libhdf5-100 libcblas-dev libatlas-base-dev libjasper-dev`
  - `sudo apt-get install -y libqtgui4 libqt4-test`
  - `sudo apt-get install python3-picamera`
  - `pip3 install "picamera[array]"`
- Go to /home/pi/.local/lib/python3.5/site-packages and remove the numpy folders. (Several of these packages install a version of numpy that does not work - removing these folders allows the default numpy installation the Pi to be used.)