Introduction

Welcome to class ECE 180 DA/DB - the two quarter design course that guides you in your projects in computer vision, speech processing, communications and system design. An outline of the course will be presented by the instructor in the first lecture.

The first few lab sections will be more structured to teach you tools that previous teams before you have found useful. Today's lab section is intended to align on the following requirements. The links lead to web pages of these tools. You will not need your H/W for this lab (just computer and attached webcam). **Note**: there is always a GUI to implement the SAME actions described in CMDs: however GUIs are a premium support. Remote servers may not be friendly to GUIs.

Preliminaries

Much of the details provided in this specification are command-line tips and tricks and it is very difficult to build a one-size-fits all line-by line tutorial. It is common to get stuck in this class, but the internet, peers and TAs are all @ your service. The same tasks CAN be achieved through all the GUIs (Anaconda, SourceTree, Github etc..). If Windows users would like to use a terminal interface, please consider: MobaXTerm. This course assumes basic experience in coding (CS 31, 31.5).

Outline

In today's class, we intend to align on the following tools and libraries to create an inclusive platform on which your projects can be built. In particular we will be covering the following topics:

1. Git, GitHub / Bitbucket, Command-Line Shortcuts (required)

2. Package Managers - (mini-)conda, pip

3. Virtual Environments (highly recommended)

4. OpenCV Installation and First Python**3** Script

5. Open-Ended Project using OpenCV (and NumPy ...)

   (a) Anything interesting :) (TA-approved) **OR**
   (b) Face-tracking and coordinate reading (see section for more details)

Figure 1: What You Should Know about Bad Software Development by Unknown Source

1   Git, GitHub / Bitbucket, CMD...

Git is a **version control software** that will give you confidence that, if you completely ruin a project on your local machine, you can always find a version that works in a "Google Drive" somewhere. **You MUST make your REPO public so TAs and other students can view**. More points to those teams who can **claim** their code being used among their peers and using their peers' code. Please cite and cite clearly.

GitHub and **Bitbucket** are services that offer the functionality of Git maintenance and make the remote-side storage look beautiful (you could technically design your own GitHub).
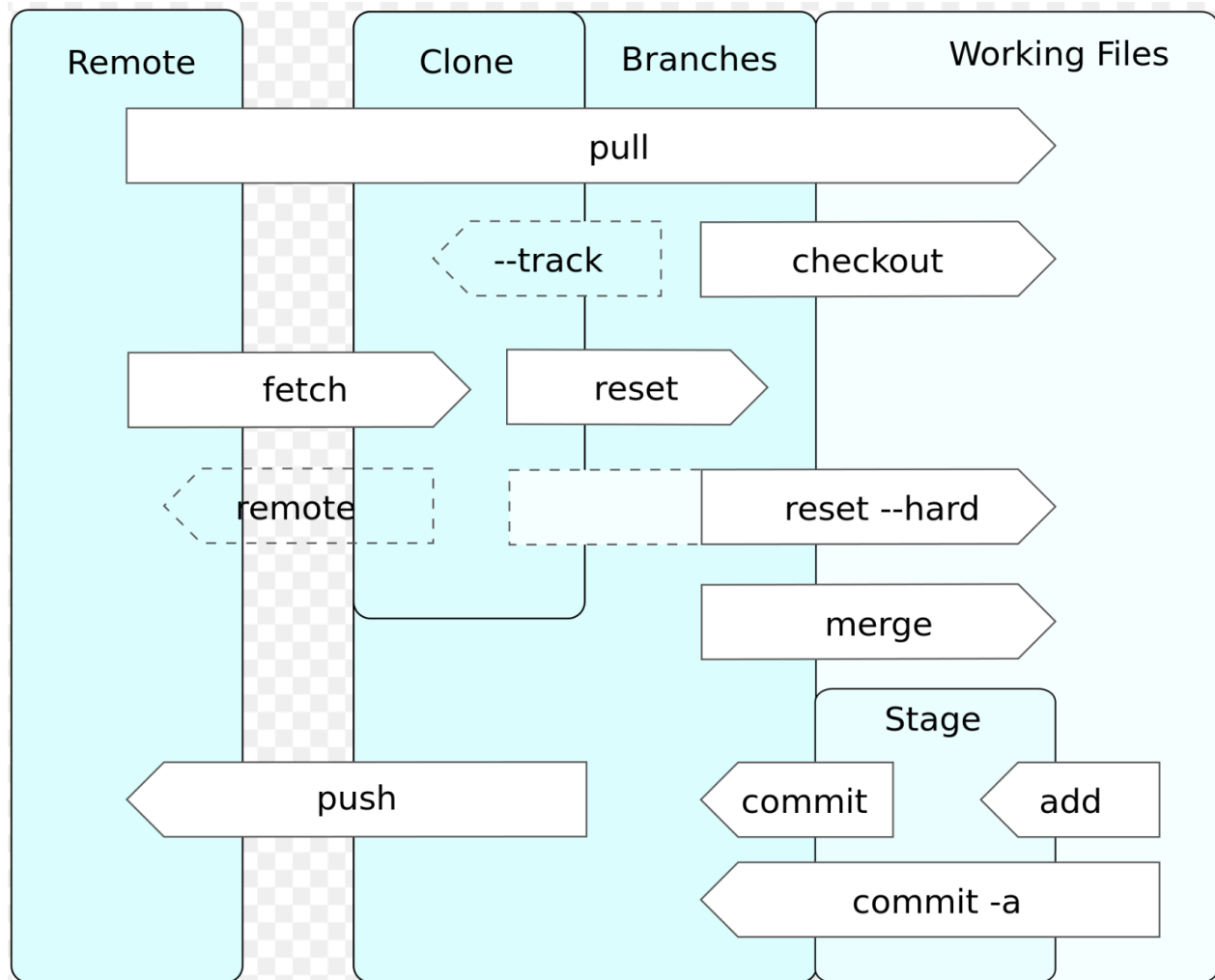
Figure 2:   An Outline of Git By Daniel Kinzler - Own work, CC BY 3.0, https://commons.wikimedia.org/w/index.php?curid=25223536

SourceTree is an application that is a client-side GUI that can be used to see which files are changed in your local machine (in something more illuminating than the black command line).

For students new to version control, please use the GUI in github.com and the GUI in the SourceTree application in case the command line is intimidating (it can be!). Even otherwise, seeing files in a human-readable layout could help massively.

Task 1: Open a Git-maintained public repository "180DA-WarmUp" in your personal Github account on Github and add your name, Github ID in the correct teamID that you want to be a part of (for the capstone project) here. Your repo for this session should be visible to

me at www.github.com/your github id/180DA-WarmUp/.

Play around with SourceTree (optional). **Watch out for and accept** an invite after class to join the 180D Github organization which will be the main hub for your capstone project, NOT your tutorial code - that will be in your own local Github account.

2 Package Managers

Anaconda is one of the major package managers dealing with Python. A *package manager* is one of the ways that allows software developers to download packages while ensuring that all the dependencies that are required for packages are also installed (and not installed multiple times for all sorts of linking problems). One of the beautiful things of using Anaconda is that it installs another virtual environment containing a fresh Python over your system, so that any dependencies that you install are installed only for your Anaconda version of Python and will not affect any other Pythons your system may contain (for example, this is commonly an issue with the Mac built-in Python). Thus, when installing your version of Anaconda onto your system, you are essentially installing a new Python on top of it.

As for the Anaconda and Miniconda, here are the major differences. For your computer, feel free to install whichever version you would like, based on your preferences. Anaconda is ultimately larger in size, but contains more user-friendly tools. Miniconda is the barebones version and only exists in the command-line. When we move to the Raspberry Pi, it is recommended that everyone use the Miniconda installation though (this is just a preview).

The benefit of this option is that it can be done completely independently of your operating system and your group's operating systems. The installation will be the same across the board and will aid in ensuring that everyone is working in the same version of OpenCV.

If you decide you don't want to use the Conda line, there are other options, but whatever you end up doing, as long as you will be able to access OpenCV, this is fine.

For more information downloading Conda, please see this guide here.

1. Download Anaconda or Miniconda.

   For Miniconda, the Mac and Linux versions will install in your Terminal as a bash installer, while the Windows installer is an exe installer that will install only the Anaconda prompt, by which you can open Python, install packages such as NumPy, etc.

   We recommend that you use Python 3, as Python 2 has been deprecated.

2. Install conda:

   - **Windows**: Run the installer. Use default settings unless you have any slight concerns. From here on out, you will be using the installed *Anaconda Prompt* that was installed to do commands.

- **Mac/Linux**: *Miniconda (bash):* Use terminal and install using

  ```
  $ (sudo) bash [name of file you downloaded]
  ```

  If absolutely necessary, try using sudo in front of this line to install. When you are accepting the license conditions, press 'q' to acknowledge that you want to quit out of the license before typing yes.

  Anything else: Double click the .pkg file and follow instructions on that.

Just to double check for a working installation, also try both just typing in

```
$ conda --version
```

and

```
$ conda update conda
```

For anything else in the tutorial that you may want to use, but you may not have (e.g. Numpy), you just need to install it

```
$ conda install numpy
```

Typically, try installing with conda first and if it's not present, use pip.

3 Virtual Environments

Note: Make sure you have finished sections 1,2 before this section Provided all steps have been followed, you should now have:

1. One Git-maintained repository hosted on a service

2. conda installed

Now we can create a virtual environment. Please look at this link for more clarity on setting up a virtual environment. The steps for your purpose is given below. Please check out this link for more clarity on installing packages (Python3 comes with Conda)

1. `$ cd [PATH to your Git-repo]`

2. Execute the following commands:

```
$ conda -V
$ conda update conda
$ conda create -n yourenvname python anaconda
$ conda activate yourenvname
$ conda install -n yourenvname pip
$ conda install -n yourenvname numpy matplotlib pandas
$ conda deactivate
```

Creating the environment is a one-time command, while activating has to be done in each shell session. If you use a coding-designed text-editors (e.g. Sublime Text and VSCode), they will likely have the ability to deal with environments automatically.

Task 2: In the local repo, please finish section 3 and within the virtual environment, open a new file test.txt and save some sample text. Add, commit and push to remote repo and verify that it shows up in your GUIs (Github, SourceTree).

4   OpenCV Installation and Python3 Script

Congratulations on your first contribution to an open-source software. Now we can start with OpenCV and your first Python3 script. Please install OpenCV as described here. Another way would be: (in project path)

1. `$ cd [PATH to your Git-repo]`

2. Execute the following commands:

```
$ conda activate yourenvname
$ conda install -c conda-forge opencv
$ conda deactivate
```

To verify that OpenCV is correctly installed, please do the following:

1. `$ cd [PATH to your Git-repo]`

2. Execute the following commands to access the Python3 interpreter and import cv2 (it should give no errors):

```
$ conda activate yourenvname
$ python
>> import cv2
>> import numpy as np
>> CTRL - D
$ conda deactivate
```

Note that you only need to activate the virtual environment when you actually want to run a file (to access the libraries that you call in your script). Now we can create our first Python3 script (my favorite editor is Vim, but any text editor from Nano to a development environment like PyCharm/XCode works).

Task 3: The following code is written in SublimeText (the color codes are fantastic in this editor IMO). Please type it into your editor and save it as test.py IN the local repo. Run (AFTER activating the VIRT. ENV) as: $ python3 filename.py

```python
if __name__ == '__main__':
    x = "ECE_180_DA_DB"
    if x == "EE_180_DA_DB":
        print("You are living in 2017")
    else:
        # this is a comment
        x = x + " - Best class ever"
        print(x)
```

Figure 3: Intro to Python - Pavan's work on 09/24!

To finish off, please add, commit and push changes to your remote repo. Now you should have 2 files in the remote location (please verify)

5   Image Processing

Of course, the main reason we installed OpenCV is that it is used all over the industry for image processing. From this point forward, this tutorial is meant mainly to guide you rather than aid you in how things work. Begin reading some of the earlier tutorials to understand how OpenCV works, but what we really want is to get to the good stuff - actual image processing. You can find a whole variety of tutorials of how to use OpenCV online. For these tutorials we will be focusing on the Python version, but there are always corresponding C++ versions.

5.1   Playing Around with Static Images

From here on out, you can start playing around with images. Just choose a random image from Google (literally can be anything you want with at least some color contrast across the image). Alternatively, you can capture an image on the webcam to use as an example image. A lot of the time, testing how to work with your video cameras work best when you are working with static images.

There are many things that you can do with images. You can convert your images to grayscale, highlight specific colors, merge two pictures, etc. Look through some of the tutorials and find some of the potential functions you might find useful.

Some important functions that may be useful to you for your project going forward are:

- Converting to different color schemes (Grayscale, RGB, HSV). You will find that RGB is actually one of the less useful schemes to use, despite the fact that it is the scheme that is always output by cameras.

- Filtering based on color (Thresholding).

- Edge Detection.

- Template Matching.

- Convex Hull / Bounding box contours.

In this link, you can find a whole list of short applications that are feasible for this section.

5.2   Video Cameras

Now we get to the actual juicier parts of how to use these cameras, with actual live feeds. In actuality, the idea is about the same. Here is a tutorial that makes it capable of using your webcam. What the code actually does is that it reads images frame by frame. As such, you can manipulate the video camera flow using the same imaging techniques that you have used in the previous section.

Using your webcam, we would like to challenge you to implement something basic using one of the tutorials that you have completed previously.

A couple of ideas are:

1. Track some object (printout of 2D point locations)

2. Object detection (printout of yes/no as to if an object is in frame).

3. Gesture detection (printout of times when gestures have been completed).

However, we would love that you could create a mixture of these or anything else that is non-trivial and cool.

6   Camera Exercises

The most common use of the camera in the project is for *localization* - learning the position
of an object with relation to something else. The easiest way to do this in projects is to use
color to track objects in the camera. Today, the initial exercises will be based on how to do
this - and some of the problems with it (especially since everyone will be working remotely
this year).

Task 4:

Do this task individually. You can share your results with your teammates to see if their
results are similar to yours.

1. Choose something that is relatively monochromatic with a color fairly different from
   your background surroundings (a water bottle, a piece of clothing). Try to create
   a video stream where you track this object with a bounding box surrounding it by
   thresholding HSV or RGB values. Is HSV or RGB typically better? How large is the
   threshold range that you need to track the object?

2. Now change the lighting condition (turn on or off the lights or turn on your phone
   flashlight on the object). Is there a major difference in the tracking ability of your
   object?

3. Now navigate to a Color Picker on your phone (Zoom into the color zone so that it
   covers a good portion of your phone screen). Since you can pick your color with the
   website, see if that is the color (with a small range) that you can pick up with your
   camera. Does changing your phone brightness help or hurt with how your code is able
   to track the color?

4. Create a new piece of code that can determine the "dominant" color in a designated
   (central) rectangle in your video feed (Use K-Means, see a tutorial to find an image's
   dominant colors). Use your non-phone object and change the brightness of its sur-
   roundings. Note the change of the color. Do the same with your phone. Is one or the
   other more robust to brightness?

You CAN use any of the BASELINE scripts linked above. CITE the references AND provide
any improvements to this baseline that you may have accomplished (IN a header comment in
the .py script). Please push your contributions to your public repo, INCLUDING a screenshot
of the results.

If you talk to your labmates about this, you'll find that the results may vary. The main
reason is there are differences in the cameras, lighting, etc, especially since everyone is doing
this remotely. Thus, when you think about how to use your camera, you will need to limit
the effect of these inconsistencies in your remote interactions. Therefore, you will probably
need to either collect more data (to choose a better threshold) or to calibrate with the camera
(recommended).

7   Advanced Topics and Examples

Here are some more advanced topics that we want everyone to consider, but are too difficult for the scope of today's lab. However, we want everyone to at least read into them so that they understand what is possible with the OpenCV (and other computer vision) technology for their own projects.

For some of these more advanced topics, though, there are some other physical constraints that might make them difficult to implement in your projects (or you will have to find workarounds). For example, some of them are very computationally expensive, making it hard to feasibly use them in real time.

1. Stereo Vision and 3D Positioning.

   A single camera is able to capture a 2D image. You can imagine that every pixel corresponds to a single ray of light extending from the center of the camera. However, it loses the information as to where on the ray the point is when converting to 2D. This is great for memories, but it makes it somewhat difficult to work with cameras to locate something. Two (stereo) cameras, capturing images of the same object from different positions, should be able to determine the object's 3D positioning, given knowledge of its relative positioning among each other (or a certain calibration to determine its positioning). This works by trying to find the intersection of the two rays of light for two corresponding pixels. As such, the goal of stereo vision is to improve the ability to understand depth from images.

   One powerful technique that you can read more about is the process of calibrating with a checkerboard. Once you have fixed the architecture that you want to work with for your projects, you can undergo a calibration that attempts to both give you the relative rotation and translation matrix between the two cameras, and the ability to get a pretty accurate estimate of the depth of objects when captured on both cameras.

2. Optical Flow.

   For those of you more "controls" oriented people, the optical flow is somewhat similar to the "derivative" of a video in each of the two spatial dimensions. In some situations, this may be important for your application because you want to track a more complex, non-uniform motion gesture. This will not be necessary in any simplified project concepts (where we choose a very distinct object that is easy to track), but this provides another solution for students who may want to try being more creative in what kinds of objects they want to track in videos.

   This typically will take a long time and it may be difficult to implement in real-time, but has a lot of possible applications, if you end up getting it working. The linked algorithm is not the only possible optical flow algorithm, and you can potentially look up less computationally expensive algorithms.

3. Neural Networks in Gesture and Position Detection

Neural Networks have exploded in the last 10 years, largely due to its usefulness in image processing. These days it is increasingly easy to find pre-trained (or architectures to train) neural networks that can accomplish some of the tasks that you would like to do. These are generally computationally expensive and (likely) difficult to implement on embedded hardware, but there are a lot of neural networks to explore. These are typically outside the scope of OpenCV, but oftentimes you can use OpenCV to do some of these implementations.

Some previously implemented examples:

(a) Gesture (roughly rock/paper/scissors) detection. (See post.py uploaded with this document)

(b) Student: American Sign Language Gesture Detection.

(c) OpenPose. There is also a Student Installation and Implementation Here.

(d) I3D. Not exactly necessarily useful for this class, but showcases some of the powerful video neural networks out there today.

7.1 Previous Example Code

Here are some previous projects for some example code:

(a) Red Lego Bounding Box Detection (C++).

(b) Bike Hand Signal Detection.

(c) Straight Line Walking Detection.