Developer’s Guide

This document will help you get up and developing for UCSD’s Global TIES Digital Vision Screening project.

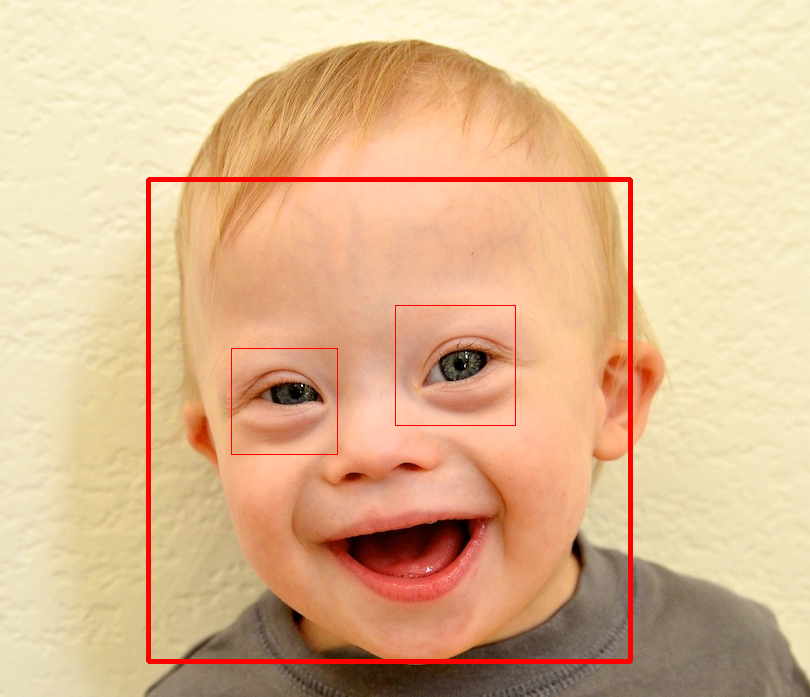


Table of Contents

[Project Overview 3](#_Toc358820355)

[Summary 3](#_Toc358820356)

[Ultimate Goal 3](#_Toc358820357)

[Subject Matter 3](#_Toc358820358)

[Get Coding! 4](#_Toc358820359)

[Technologies Overview 4](#_Toc358820360)

[Front End 4](#_Toc358820361)

[Back End 4](#_Toc358820362)

[Environment Setup 4](#_Toc358820363)

[Windows 5](#_Toc358820364)

[Linux 5](#_Toc358820365)

[Mac 5](#_Toc358820366)

[Code Structure 5](#_Toc358820367)

[Front End 5](#_Toc358820368)

[Backend 5](#_Toc358820369)

[Github 101 6](#_Toc358820370)

[Resources 8](#_Toc358820371)

[Documents 8](#_Toc358820372)

[User Stories 8](#_Toc358820373)

[Requirements 8](#_Toc358820374)

[Class Diagrams 8](#_Toc358820375)

[Links 8](#_Toc358820376)

[Technologies 8](#_Toc358820377)

[Editors 8](#_Toc358820378)

[Git 9](#_Toc358820379)

[Eye Detection 9](#_Toc358820380)

[Pupil and Iris Detection 9](#_Toc358820381)

[Sclera/Crescent Detection 9](#_Toc358820382)

[Software Concepts 9](#_Toc358820383)

[Miscellaneous 9](#_Toc358820384)

# Project Overview

## Summary

The DVS Image Analysis team works on developing algorithms that perform computational image analysis on a patient's image to determine the presence of three types of vision disorders: amblyopia, strabismus, and cataracts. Amblyopia, or more commonly referred to as "lazy eye," is the loss of or reduced vision in one eye. It is the most common cause of visual impairment in children. Strabismus, more commonly known as "crossed eyes", refers to the misalignment of both eyes. This prevents the eyes from working together, resulting in vision impairment, faulty depth perception, double vision, and, if left untreated, amblyopia. Cataracts is a clouding of the eye that can impede vision.

For more information about the three types of vision disorders we are working to identify, the following links are useful resources.

Amblyopia: <http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0002009/>

Strabismus: <http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0001999/>

Cataracts: link

The Image Analysis team works on one of the essential parts of the Image Analysis application; primarily the detection of the iris in an image of the patient's eye and from there assessing the amount of the reflection in the eye caused by the Bruckner Reflex test (in other words, crescent detection) to ultimately determine whether the patient has amblyopia. The Bruckner Reflex test is highly dependent on capturing images with a strong red-eye effect.

## Ultimate Goal

The setup we’ll be working with is a laptop connected to a camera. This camera will take two pictures, one horizontal and one vertical (this is to maximize the crescent effect of red eye). The program will then detect the child’s eyes, allow the user to confirm this detection, detect several other features including the pupil, crescent, and sclera. These features will also be confirmed by the user. The program will then move on to analyze these eye features using machine learning and notify the user of the application which, if any, eye disease the child may have.

## Subject Matter

I’m going to be lazy with this section and refer you to the EyeDiseaseDescriptions doc on github. It reviews the eye-related subject matter that we need to be familiar with. Refer there if you’re confused.

# Get Coding!

The following sections will outline the details of what you need to do to get set up but here’s the short list:

1. Get yourself added to the UCSD-TIES github organization
2. Clone the DVS-Python repository
3. Set up your environment
4. Ensure that the code you cloned works as intended on your local machine
5. Make a change to a file (add a comment or something)
6. Save the change, commit it, and push to the github repository (to make sure you can)
7. Take a look at the repository’s /doc folder, read documentations for the tech we’re using and *talk to your teammates* to get yourself oriented with the existing project and what you should be doing.
8. Get coding!

## Technologies Overview

We use Python as our coding language. The backend uses PIL and OpenCV libraries for image analysis. The Front End uses wxPython to develop a local GUI. We use github for version control. We don’t have a particular editor or IDE we work in. Some use IDLE, some Pydev, or whatever works and you can get to be set up with OpenCV. Take a look at sublime if you’re looking for a solid and simple editor (you can even edit multiple lines at the same time, it’s pretty neat).

### Front End

* wxPython
  + Python 2.7.4

### Back End

*dependent technologies are indented*

* OpenCV 2.4.5
* NumPy (latest revision as of 05/01/13)
  + Python 2.7.4
* PIL 1.1.7
  + Python 2.7.4
* IDLE Python (or other python editor)
  + Python 2.7.4

## Environment Setup

See the github wiki for this. Expect a fair amount of craziness trying to get your setup working right. And if you get stuck or kinda stuck ask a teammate! Someone’s probably seen that error before.

### Windows

See the github wiki.

1. Install Python
2. Instal NumPy
3. Install OpenCV
4. Copy a file – I believe this is copying opencv into the appropriate python path directory or something.
5. Install the Python Imaging Library
6. Set Path

### Linux

See the github wiki.

### Mac

See the github wiki.

## Code Structure

Generally the project tries to implement a model-view-controller architecture. The backend team works on the model portion of the architecture, an object-oriented designed set of classes meant to mirror the structures in the eye. In general each structure can detect the structures within it. For example the eye class can detect a pupil (these detections are usually done in the \_\_init\_\_ method). The controller is implemented by a module called, surprisingly, Controller.py. This provides methods that the front end team use to interface with the model.

The front end team structures their work into panels, each representing a page of the user interface. Generally the panels are named sequentially. So the first panel is panel 1, the next panel the user sees is called panel 2, etc.

Commenting conventions have been established in /doc/DocstringExample.py. Please follow these as much as is reasonable.

### Front End

See /doc/wxPython documentation.docx to get started with wxPython and the front end code.

### Backend

The Backend’s team files are all located in the /src folder of the team’s github repository. The model files are the following(from topmost in architecture to bottommost): Patient.py, FacePhoto.py, HorizontalPhoto.py,VerticalPhoto.py, and Pupil.py. Controller.py provides the interface for the front end team to interface with the backend without needing to know implementation details. You’ll notice some other files, eyeDetection.py, and eyeRemove.py. Generally in our development we’ve found it easier to write a script to perform CV detections and then, once we have that script working, to refactor it into the Model structure. You may safely ignore these files without missing out on anything.

Controller.py’s methods are fairly simple. They just call the relevant setters and getters in the model structure and do some error checking. There’s a section at the bottom called testing that mimics calls the front end team will make to make it easier to debug the backend’s code without dealing with front end code. You may comment this out if you’re testing with the real GUI.

Now, onto the model structure, the heart of the backend. Before you proceed you will want to take a look at the class diagram in the /doc folder so you can get yourself oriented. The structure is fairly simple: A Patient has a HorizontalPhoto and a VerticalPhoto both of which are FacePhotos. Each FacePhoto can detect its own Eyes. Each eye can detect its own Pupil and sclera. And a Pupil can detect its own crescent. You’ll notice that “region” is referenced a lot in the documentation and in the comments. This is not a python or openCV type. It’s just a placeholder we’ve left there to indicate that some sort of region should be stored. This may vary. For example eye regions are stored in a rectangle format, as a tuple of four ints representing (x1,y1,x2,y2) where (x1,y1) and (x2,y2) are the opposite corners of the rectangle. (Note: OpenCV and PIL deal with rectangles differently. PIL does not use opposite corners of the rectangle but instead (x,y,w,h). (x,y) is the upper left corner of the rectangle. w is width. h is height.) Of course Pupil and sclera don’t have defined region types since the code we have now can’t detect them. We’ll probably want a circle with a center for Pupil and an OpenCV ROI for the sclera.

To avoid ridiculous amounts of rogue print statements in the code most modules have a module-level variable “DEBUG” at the top. If you want to see print output just set it to true and run again. If you’re putting in any print statemtents make sure to wrap them in an if DEBUG. Be a bit careful though, occasionally an if DEBUG will wander before an else statement, stealing it from its original parent if and messing up the control flow.

## Github 101

Our repository is here. You’ll need to be added as a UCSD-TIES member before you can edit the repo. You can do that by asking your team leader to add you. They’ll either have the authority to do so or know who does. Here’s a nice tutorial on github basics. To start off clone the repository and make sure it’s working.

Some general tips and best practices:

* Make your commit messages informative
* Pull, and pull often to avoid conflicts
  + That means every time you sign on to get working with the project, even if you haven’t made any changes
* Pull before you push
* Push as often as possible so others can see your code
  + But be judicious, if your code is still being debugged and breaks others’ code keep it local or isolate it so team members can see it but it doesn’t break the build
* If you get a conflict you have to go into the file and manually edit it to make sure it’s what you want then commit that

# Resources

## Documents

### User Stories

These are short descriptions of application functionality from the user’s perspective. They’re used in the Agile method of software engineering and are helpful to keep track of development progress and communicate with the client.

Location on github:

Description of User Stories:

### Requirements

These are short descriptions of application functionality that the system needs to provide. They help narrow down user stories into actual coding/implementation strategies.

Location on github:

Description of Requirements:

### Class Diagrams

This object-oriented diagram lists how classes interact, outlining inheritance and composition relationships as well as listing the attributes and methods a class contains. If you get lost in the backend’s model, you should refer to this. Our diagram is written in standard UML format for a class diagram.

Location on github:

Description of class diagram:

## Links

### Technologies

Python Docs

OpenCV Docs

Short python/opencv talk

PIL Docs

wxPython Docs

### Editors

IDLE

VIM

PyDev

Sublime

### Git

UCSD-TIES Organization Page

Our repository

Explanation of Git

Overview of Git Commands

### Eye Detection

Source of the algorithm we’re currently using

### Pupil and Iris Detection

Source of the algorithm we’re currently using

Other algorithm

Other algorithm

### Sclera/Crescent Detection

Color detection in openCV

Blob detection in openCV

### Software Concepts

Model View Controller (MVC)

Agile

Scrum

Refactoring

User Stories

Requirements

Class Diagram

### Miscellaneous

Eye Structures

Sclera

Crescent

ROI