

I. Database and server information

- a. SensorBase:
- b. Server


II. Files you need:

- a. snake (folder) --- incomplete
- b. medFilter
- c. findXY (file)
- d. gvfmmain (file)
- e. checkGrad
- f. calibrate
- g. orient
- h. findRegionDimensions (file)
- i. findMode (file)
- j. checkRes (file)
- k. brightness-intensityGradients-checkGrad-calibrate.py (file) --- incomplete
- l. getPollution (file)
- m. matchBright (file) --- incomplete
- n. Any SMS files
- o. Any slogging code
- p. Code that uploads results
- q. imageGrayscale

III. Implementation

- a. Architectural overview, and notes on the status of individual components
 - i. Photo taken on phone and sent to SensorBase
 - 1. Automatically uploaded to SensorBase if there is internet access
 - 2. If there is no internet access: We have adapted Saro's SMS code so that results can be sent via SMS, however, we still need a program on the phone that will call SMS only when internet isn't found
 - ii. The server queries SensorBase for new images every 3 hours
 - 1. The three hour period is adjustable
 - 2. The server accepts no more than 15 images during each query period—see "Methods and Design", delta
 - iii. Images are checked for quality
 - 1. Checks to make sure the image meets the minimum resolution, see "methods and design" and "evaluation" for current value
 - 2. Images converted to grayscale
 - 3. Median filter applied
 - 4. The image is segmented
 - a. We have not been able to successfully segment our test images. One problem is not iterating the GVF and snake deformation enough. Resolution may also have an effect, but we didn't officially test this. We're also unsure of how to parse the output from the segmentation to find which points in the output correspond to which objects within the image (see comments in "snakeinterp1.py"). We were looking into using a segmentation method that is similar to the one we are currently trying to use but consistently works with multiple images with precedent

est., we think ☺, on how to parse the output. This new method is detailed at: <http://www.i3s.unice.fr/~debreuve/acontour.htm>

- i. You might want to just use the matlab code.
Unfortunately, Matlab code takes a ridiculous amount of memory to run the segmentation code (we frequently got the error “System out of memory”).
There was a project a few years ago with dietsense that integrated matlab and python. This might be useful for writing the segmentation code and also have the added bonus of being easily integrated with our python code.
E-mail Nicolai Munk Petersen at nmp@webcore.eu for more info. He worked on the python-matlab integration project
***Once we have the output from the segmentation figured out, we should be able to fill in the gaps in the matchBright program we have right now

5. Light intensity gradient check

a. Bad images:

- i. If zero across the entire image, the image is bad, because it's either totally wash-out, totally yellowed, or totally dark
- ii. If not zero within a 50 pixel radius of the filter and color chart (see “Methods and Design”, pre-processing)
- iii. Bad images are not analyzed—we want have a program that sends a notice of the bad image and unsuccessful analysis to sensorbase and back to the phone; ideally, this program will also tell about the type of error, so that Surya and the people taking the pictures will know what is being done wrong, but we haven't added this feature yet.

b. Good images:

- i. If zero with a 50 pixel radius of the filter and the color chart

6. Good images are calibrated (see “Methods and Design”, pre-processing)

iv. The images are matched to the color chart

1. The color chart is oriented in the frame(see “methods and design”)

- a. We have a program to do this right now but it assumes that the color chart is to the left of the filter. We would like a more flexible program that doesn't assume this. I would like to work on this part if you like (oh, and I'm also cool with helping with any segmentation stuff....like I told Nithya, I can commit a max of 4 hours a week to work on CENS stuff until I see how busy everything else will be, but I will have some time to help you if you want/need, and am still very interested in continuing work on the project)

2. The filter is matched to a bar on the color chart

3. Results are uploaded on SensorBase and sent to the phone with less detail via SMS. More tests are needed to determine how well SMS works
- b. Order to run the files in—already called in order on server:
 - i. Slogging code
 - ii. checkRes
 - iii. imageGrayscale
 - iv. medFilter
 - v. GVFmain (make sure you've imported all the files from the 'snake' folder, since this program calls all those files)
 - vi. checkGrad (calls 'calibrate' if images are good)
 - vii. orient
 - viii. matchBright
 - ix. getPollution
 - x. upload result code (name?)
- c. libraries you need to download for all the code to run:
 - i. pylab (this is optional once the program is totally working because it just displays different things, but while we were coding, esp. when coding the snake, we found it really useful to display the result along the way)
 - ii. python imaging library (PIL)
 - iii. scipy
 - iv. numpy