SPECTREYE

Using AI to extract spectrometer angles from images

•••

https://github.com/ws-kj/Spectreye
Will Savage - H-B Woodlawn, Arlington, VA
Dr. David Lawrence, Mr. Nathan Brei, Dr. Brad Sawatzky

Jefferson Lab

Background - JLAB spectrometers

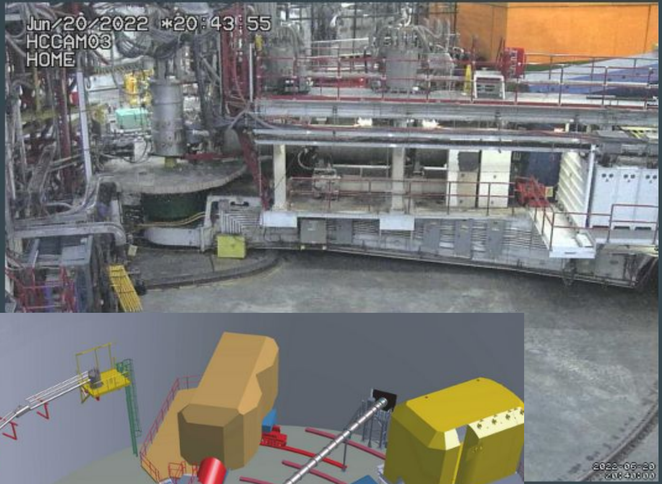
Hall C contains two train-sized devices:

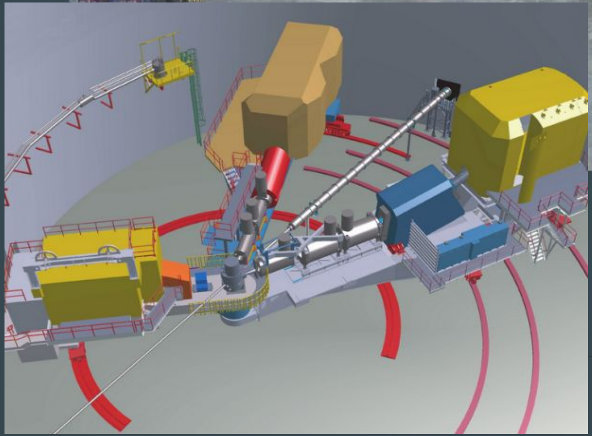
the **High Momentum Spectrometer** (HMS)

and the **Super High Momentum Spectrometer** (SHMS)

The spectrometers sit at variable angles from the beamline, and record the momentum of the charged particles perpendicular to the beamline.

A Hall C spectrometer.





A render of Hall C. SHMS center, HMS right.

Background - project goal

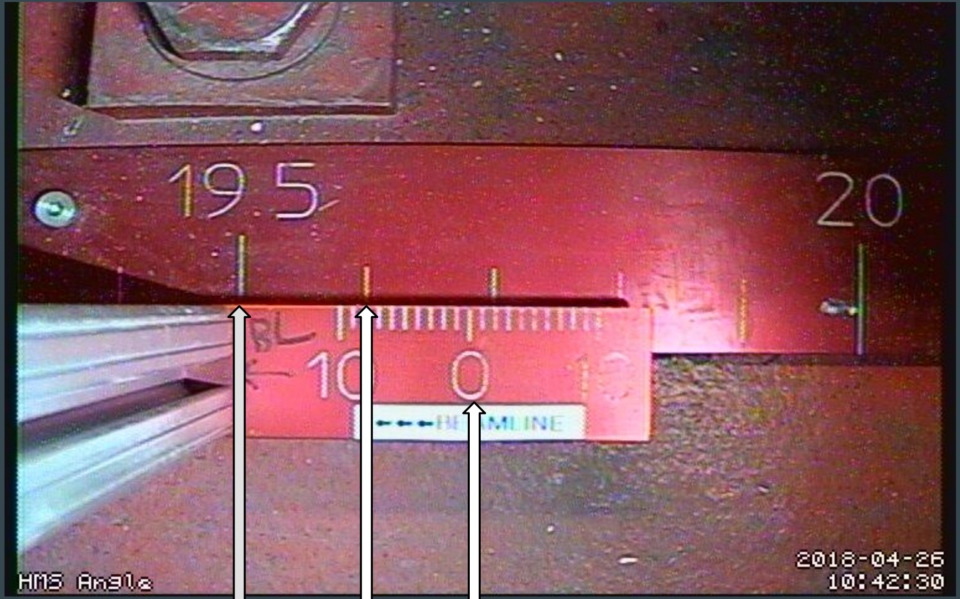
PROBLEM: Each Spectrometer contains an ‘encoder’ device, which records the current angle.

However, **the encoders tend to drift and report inaccurate angles**.

GOAL: The Spectreye program should be able to take a scale image, and **return the true angle accurate to 0.01°**.

The tool will ideally supplement the encoder and notice potential inaccurate readings.

Background - Reading a vernier scale



19.5 + .1 + .08 = 19.68°

(Note: HMS angles increase left to right, SHMS angles increase right to left)

Project Design - development stack

Languages / Environment

Initial prototype - **Python** and **Bash**

Final build - **C++11** and **CMake**

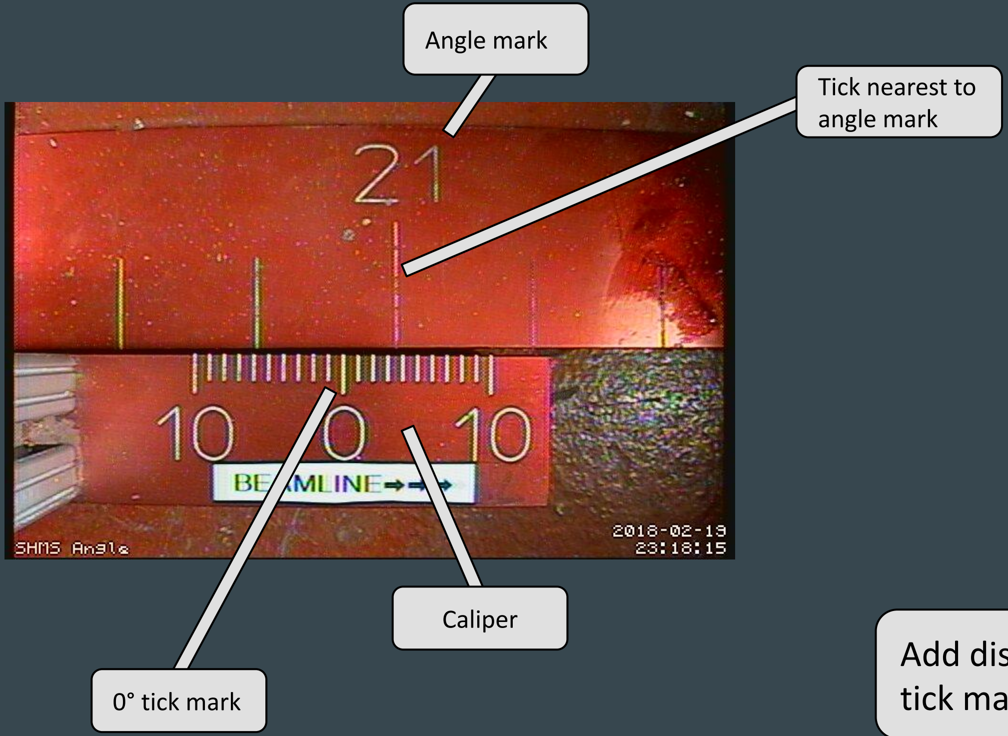
Libraries and models

Image manipulation - **OpenCV**

OCR - **EAST (DNN Model)** and **Tesseract**

Written and optimized for **x64 Linux**

Project Design - ideal angle process



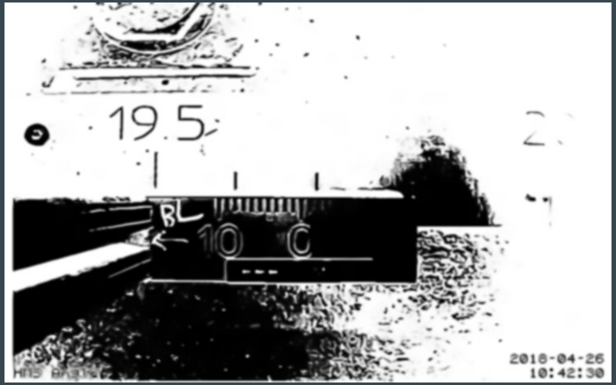
1. Locate angle mark text
2. Read angle mark
3. Locate tick nearest to angle mark
4. Locate 0° tick mark
5. Add distance between angle mark and 0° tick mark to angle reading

Project Design - OCR steps

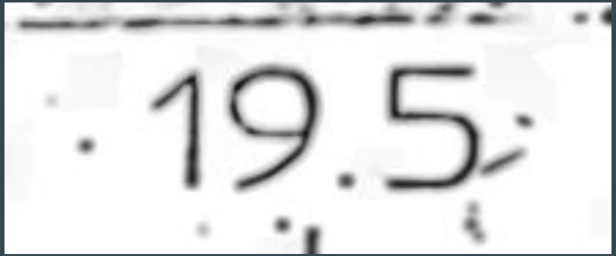
1. Filter image for Optical Character Recognition (OCR)

2. Attempt to read selected bounding box using Tesseract

3. Read encoder angle mark if Tesseract fails



Filtered image for OCR bounding box detection



Isolated mark for stage 2 OCR

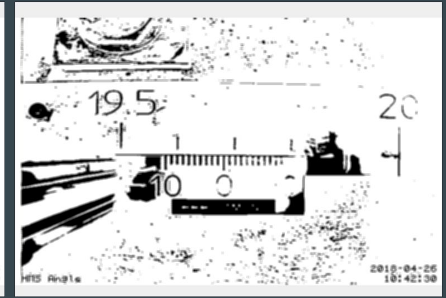
Project Design - maximizing OCR success

Bounding Box Detection

Alternative image filtering algorithms are used if the primary one fails

If EAST fails, Tesseract is reconfigured for bounding box detection

Primary threshold filter



Color mask filter

Angle Mark Reading

Tesseract settings are automatically tweaked if no number is able to be read

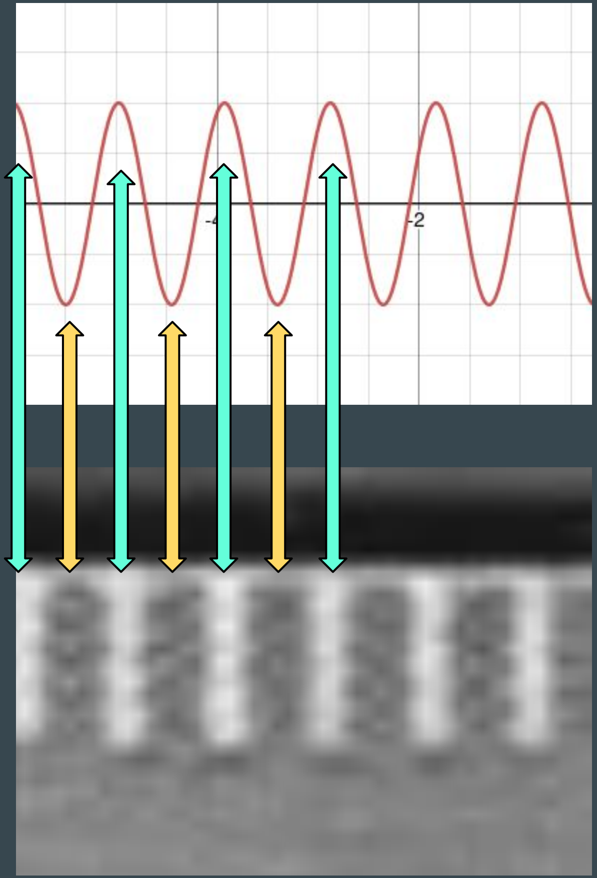
If encoder data is available and no angle mark can be read, a ‘composite’ guess is created

Composite guess combines nearest angle mark to encoder angle with Spectreye-detected ticks

Project Design - Locating tick positions

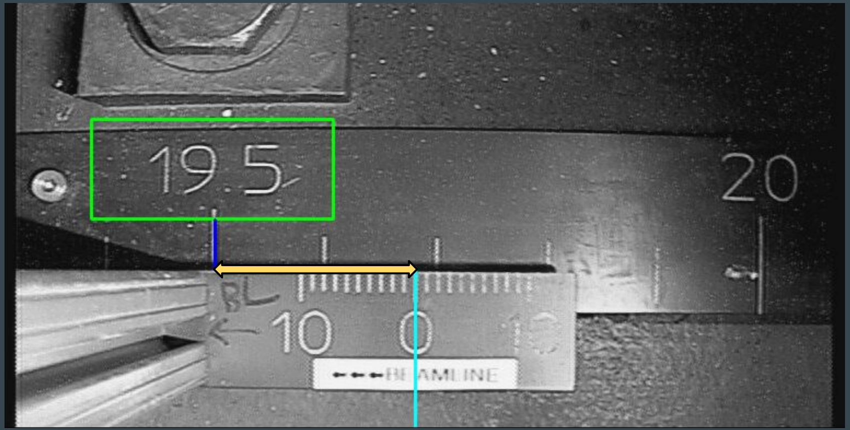
When an angle image is grayed and filtered, the color value of a row containing ticks can be thought of as a wave.

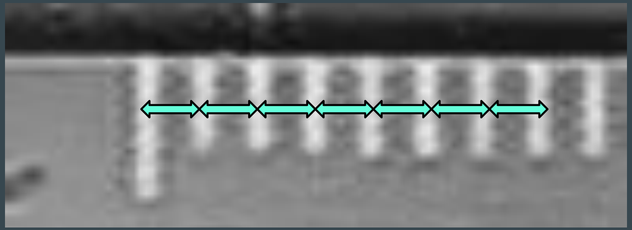
By finding the position of each crest in the 'wave' of color values, the center of each tick can be determined quickly and accurately



Project Design - Find tick width and center

1. Determine pixel width between 0.01 degree ticks on caliper
2. Locate center tick
3. Determine distance in pixels between angle marking and center tick





Pixel width of 0.01 degrees

/ 100 = # of degrees to add to angle mark

Project Design - Choosing final guess

At the end of the algorithm, there are usually two guesses to choose from:

the OCR guess and **the composite guess**

OCR guess uses mark reading from Tesseract

Composite guess uses nearest mark to encoder angle

Composite chosen automatically if Tesseract read fails

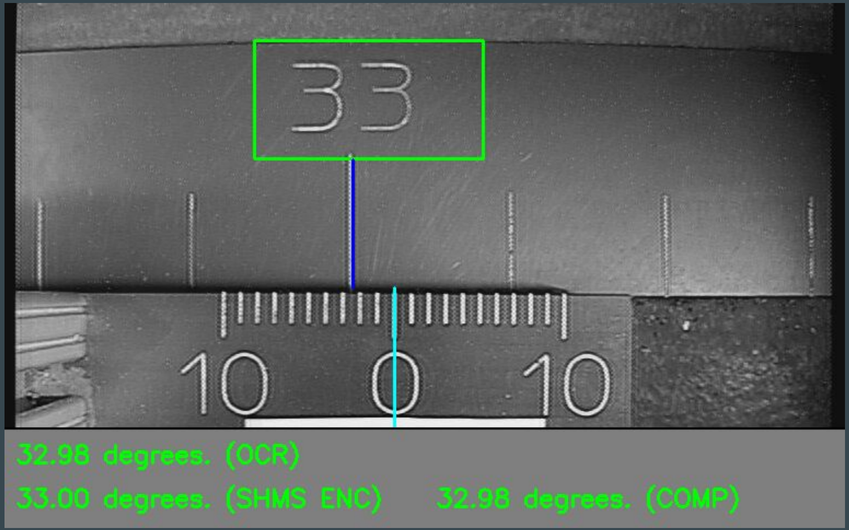
Composite chosen if OCR reading > 1° off from encoder angle

Composite chosen if OCR reading is not within the possible angles for each spectrometer

Otherwise, the OCR guess is preferred

```
Θ = mark + (((center - mark)/tick_width) * 0.01 )
```

Results - success example



Visual output - contains identified mark, middle tick, OCR reading, encoder reading, and composite reading.

```
[15:06 will] (cpp) -> ./spectreye ../../images/qtest/SHMS_0.jpg 33.0

  Spectreye reading for /home/will/src/Spectreye/images/qtest/SHMS_0.jpg

  SHMS - SUCCESS - 32.98 deg
  --  Timestamp:  2018-03-21 16:58:30
  --  OCR guess:  32.98 deg
  --  Comp guess: 32.98 deg
  --  Angle mark: 33.00 deg
  --  Tick count: -0.02 deg
```

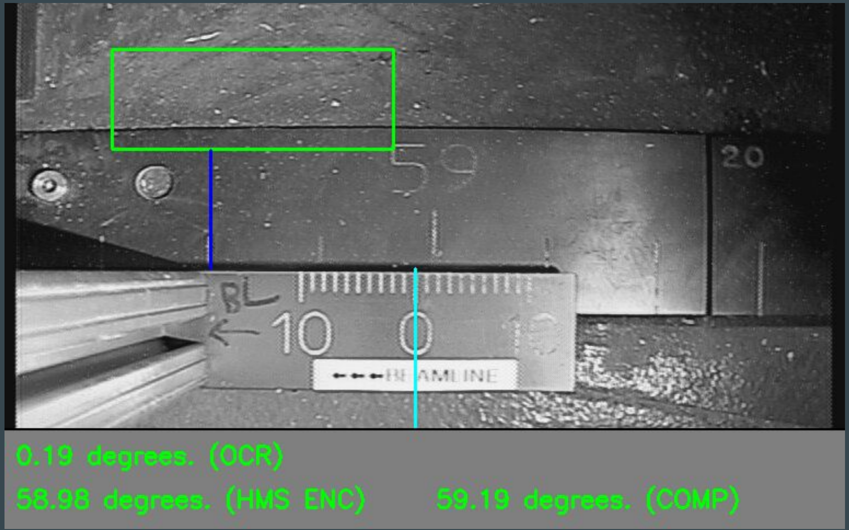
Text output - returned as struct when called from external C++ program.

Timestamp - Date extracted from image file, useful for comparison with encoder data.

OCR guess - Angle predicted using Tesseract read of angle mark.

Comp guess - Angle predicted using encoder angle mark combined with self-detected ticks.

Results - critical failure example (undetected)



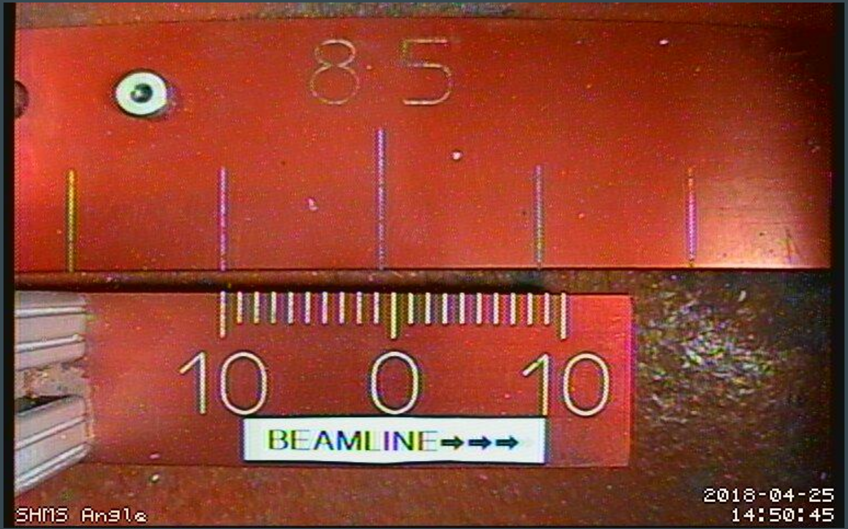

```
[18:39 will] (cpp) -> ./spectreye ../../images/singles/HMS_angle_01645.jpg 58.98
rebuild
Estimating resolution as 458

   Spectreye reading for /home/will/src/Spectreye/images/singles/HMS_angle_01645.jpg

   HMS - NOREAD - 59.19 deg
   --  Timestamp:   2018-02-23 11:52:00
   --  OCR guess:   0.000 deg
   --  Comp guess:  59.19 deg
   --  Angle mark:  0.000 deg
   --  Tick count:  0.187 deg
```

The system doesn’t know that the bounding box is wrong, and returns a noread instead of a failure.

This is a dangerous issue because lab operations may be performed on an incorrect number, which could jeopardize data.

This issue can essentially only be solved by improving OCR accuracy.

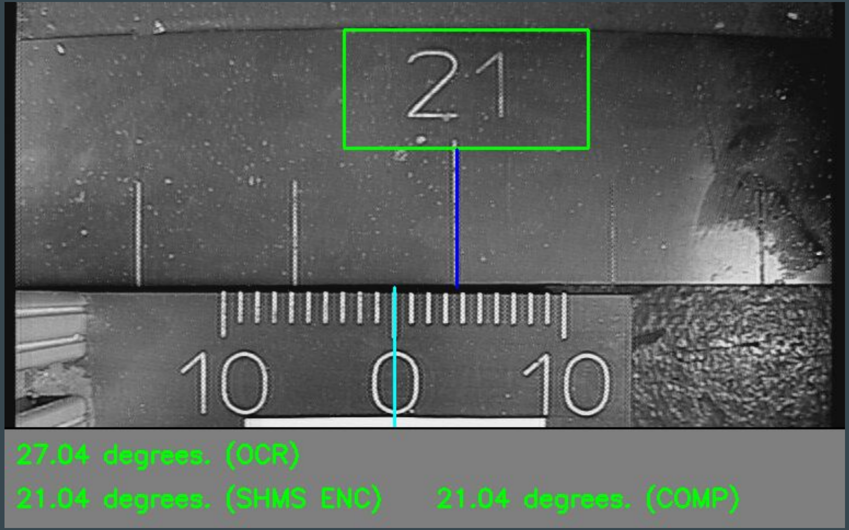
Results - critical failure example (detected)




```
[22:20 will] (cpp) -> ./spectreye ../../images/qtest/SHMS_3.jpg

  Spectreye reading for /home/will/src/Spectreye/images/qtest/SHMS_3.jpg

  SHMS - FAILURE - 0.00 deg
  --  Timestamp:  2018-04-25 14:50:45
  --  OCR guess:  0.000 deg
  --  Comp guess: 0.000 deg
  --  Angle mark: 0.000 deg
  --  Tick count: 0.00 deg
```

The program was completely unable to locate any angle marks, so it returned a complete failure status.

Results - “exceed” example

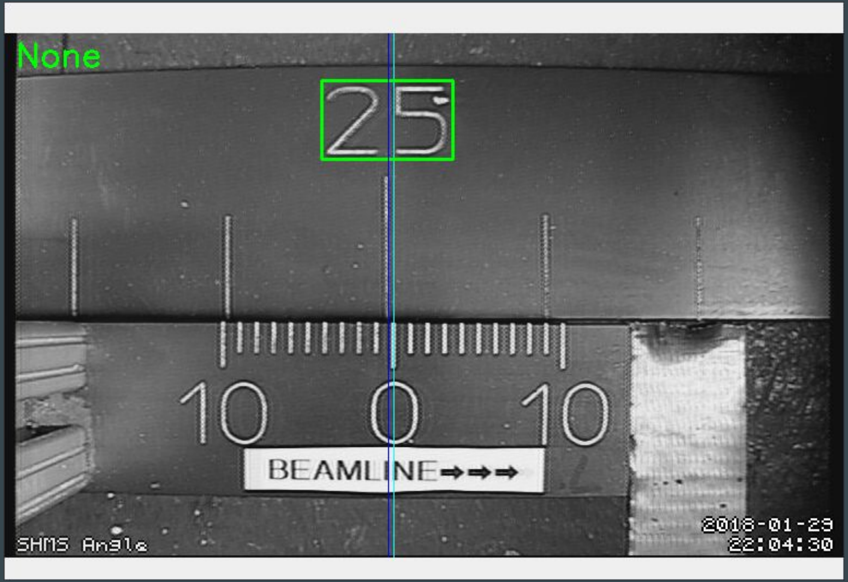



```
[18:28 will] (cpp) -> ./spectreye ../../images/qtest/SHMS_1.jpg 21.04

  Spectreye reading for /home/will/src/Spectreye/images/qtest/SHMS_1.jpg

  SHMS - EXCEED - 21.04 deg
  --  Timestamp:  2018-02-19 23:18:15
  --  OCR guess:  27.04 deg
  --  Comp guess: 21.04 deg
  --  Angle mark: 27.00 deg
  --  Tick count: 0.036 deg
```

For this image, Tesseract read “21” as “27,” and thus the program deems the OCR pass successful.

The difference between the OCR and encoder is more than one degree, so the composite guess is returned.

Results - “noread” example




```
calculated angle: None degrees
{
    "status": "NOREAD",
    "name": "/home/will/src/Spectreye/images/singles/SHMS_angle_02231.jpg",
    "angle": null,
    "mark": "5.0",
    "tick": "0.0",
    "runtime": "1.8546",
    "device": "SHMS",
    "timestamp": "2018-01-29 22:04:30"
}
2018-01-29 22:04:30 (SHMS)
encoder angle: 25.0 deg. spectreye angle: None deg.
encoder mark: 25.0 deg. spectreye mark: 5.0 deg.
encoder tick: 0.0 deg. spectreye tick: 0.0 deg.
composite guess: 25.0 deg.
```

The Python prototype generates a “noread” for this image. It locates the angle mark, but is unable to read it.

Encoder data exists for this image, so a composite guess is able to be generated.

Results - codebase status

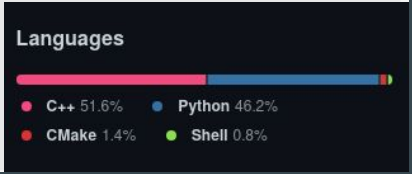
Spectreye can be built as a shared library for use in C++, or as an standalone executable for use in the command line.

The GitHub repository contains around 1200 lines of C++ and 1100 lines of Python.

The project be built and run on any Linux or MacOS system.

Using the library

```cpp
#include <iostream>
#include <spectreye.h>

...
std::string myimage = "SHMS_image.jpg";
double encoder_val = 45.23;

// argument specifies debug mode, true will display image
Spectreye* s = new Spectreye(true);

// struct containing status, guesses, and other information
SpectreyeReading reading = s->GetAngleSHMS(myimage, encoder_val);

// formatted description
std::cout << Spectreye::DescribeReading(reading) << std::endl;

if(reading.result != RC_FAILURE) {
    // do something with angle reading
    std::cout << reading.angle << std::endl;
}

// free allocations
s->Destroy();
delete s;
...
```



Conclusion

Spectreye satisfies the goals outlined at the beginning of the project.

The program is extremely portable, and is ready for use in the lab either as a C++ library or as a standalone executable.

The program generally performs well, but further work is required to boost accuracy when image/scale conditions are not ideal.

Next Steps

The weakest link in the Spectreye algorithm is the OCR.

Alternatives to EAST and Tesseract could be explored to significantly boost reliability, such as a custom trained deep learning model.

Higher OCR accuracy would lower the rates of both detected and undetected failures.

Acknowledgements and thanks

Dr. David Lawrence

Mr. Nathan Brei

Dr. Brad Sawatzky

Chauntèe Pitts

Carol McKisson

Jalyn Dio

Dr. Mihyeon Kim

W&M Center for Gifted Education

JLab Science Education