Machine vision in Python

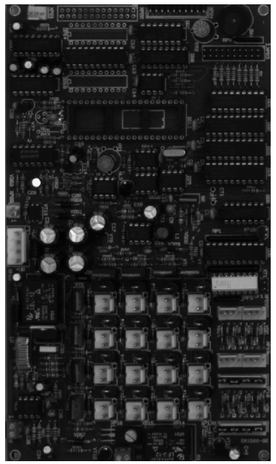
Jan Margeta | jan@kardio.me | @jmargeta

KardioMe

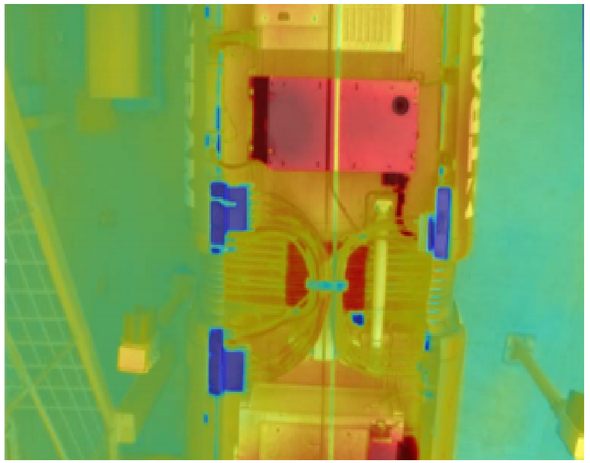
May 4, 2018


KardioMe
... ping me if you wanna know more ...

Machine vision?

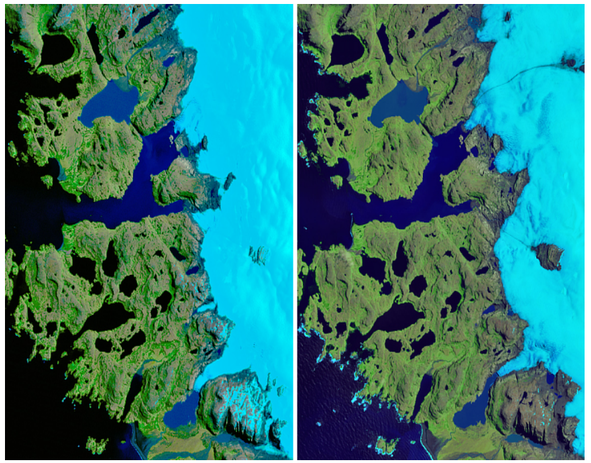


Quality control in waste reduction



Early failure detection

Courtesy of Alstom transport



Climate change monitoring

USGS/NASA Landsat

And much much more...

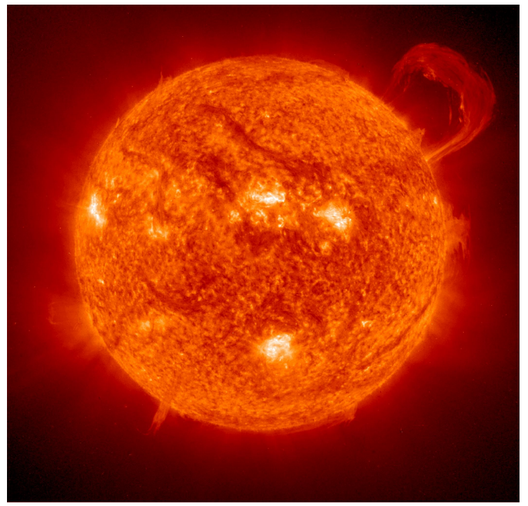
Getting some hardware

Camera

resolution
frame rate
"color"
interface

Optics

field of view
F-number
camera compatible



Lighting

intensity
strobing
color
pattern

The (usual) Flow

Camera software

Aravis

Generic
GigE Vision and USB3
C interface
Python via GI
LGPL

Camera vendor's SDK

Vendor lock in
GigE Vision and USB3
Somewhat more reliable
Full cam functionality
C or C++ interfaces only
proprietary

Let's wrap it!

Options for wrapping C and C++ libraries

- Python extension modules
- Swig
- ctypes
- CFFI
- Cython

Cython I.

Expose C or C++ structures

```
cdef extern from "CameraParams.h":
    cppclass GIGE_DEVICE_INFO:
        unsigned int nIpCfgOption
        unsigned int nIpCfgCurrent
        unsigned int nCurrentIp
        unsigned int nCurrentSubNetMask
        unsigned int nDefultGateWay
        unsigned char chManufacturerName[32]
        unsigned char chModelName[32]
        unsigned char chDeviceVersion[32]
        unsigned char chManufacturerSpecificInfo[48]
        unsigned char chSerialNumber[16]
        unsigned char chUserDefinedName[16]
        unsigned int nNetExport
        unsigned int nReserved[4]
```

Cython II.

Wrap vendor's SDK

```
cdef class GigEDeviceInfo:
    cdef:
        GIGE_DEVICE_INFO c_dev_info

    @staticmethod
    cdef create(GIGE_DEVICE_INFO dev_info):
        instance = GigEDeviceInfo()
        instance.c_dev_info = dev_info
        return instance

    @property
    def current_ip(self):
        return self.c_dev_info.nCurrentIp
    ...
```

Cython III.

Compile

```
from distutils.core import setup
from Cython.Build import cythonize

setup(
    name = "Cythonized app",
    ext_modules = cythonize('camera.pyx'),
)
```

Using the camera

Find connected cameras

```
import pyhikvision
factory = pyhikvision.camera.TlFactory()
devices = factory.enum_devices()
```

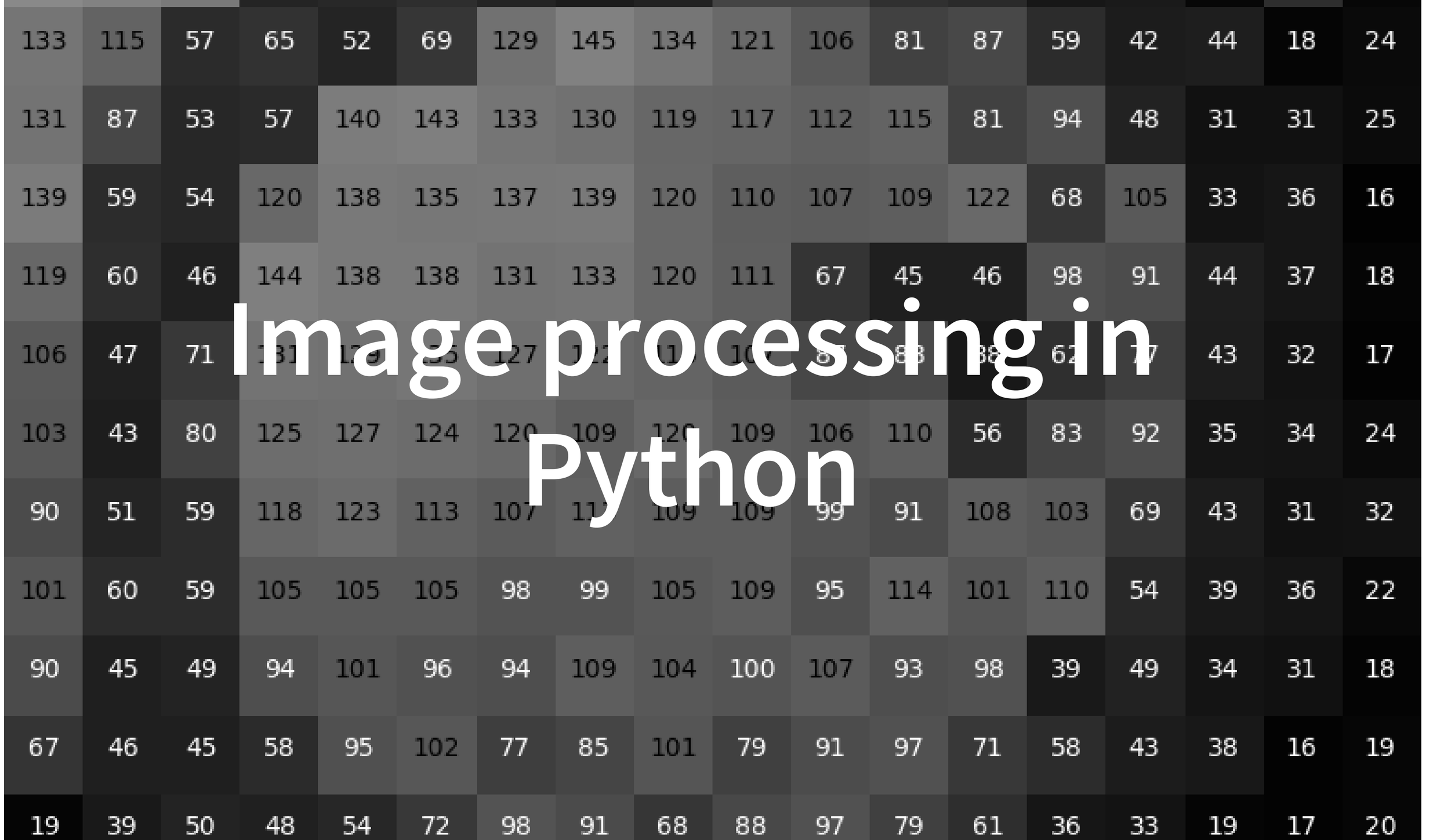
Setting camera parameters

```
camera = factory.create_device(devices[0])
camera.open(1, 0)
camera.width = 1920
camera.height = 1374
camera.pixel_format = 'Mono8'
camera.frame_rate = 5
camera.exposure_auto = 'continuous'
camera.gain_auto = 'continuous'
```

Do image acquisition

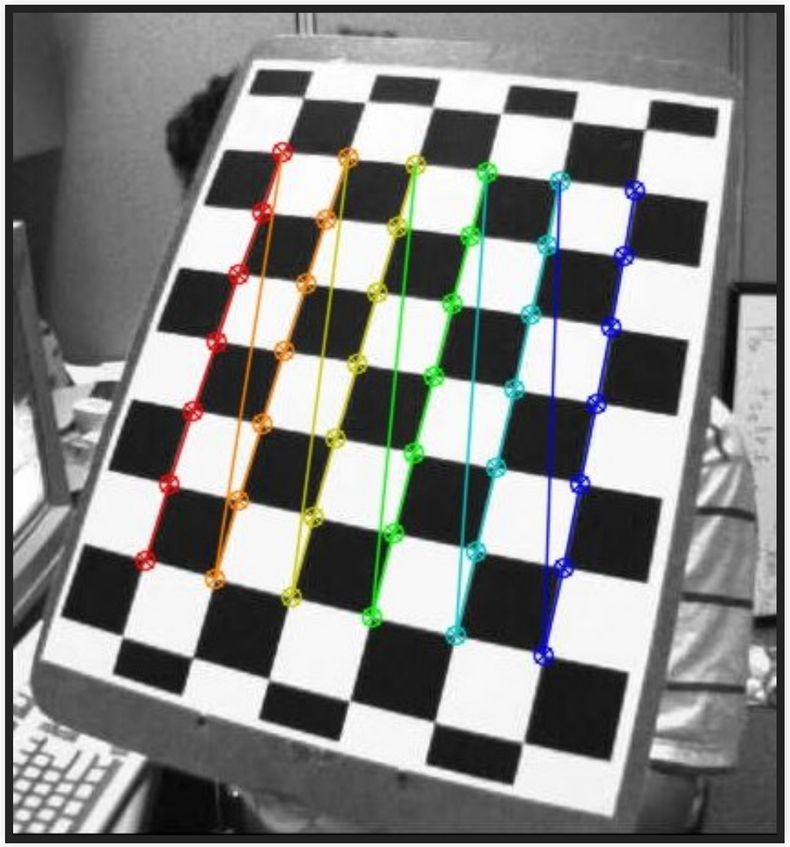
```
# Allow the camera to start grabbing frames
camera.start_grabbing()

# Grab a frame
frame = camera.grab_frame()

# Async is also (kind of) supported
frame = await camera.grab_frame_async()
```

Frames as numpy arrays!


133 115 57 65 52 69 129 145 134 121 106 81 87 59 42 44 18 24
131 87 53 57 140 143 133 130 119 117 112 115 81 94 48 31 31 25
139 59 54 120 138 135 137 139 120 110 107 109 122 68 105 33 36 16
119 60 46 144 138 138 131 133 120 111 67 45 46 98 91 44 37 18
106 47 71 43 32 17
103 43 80 125 127 124 109 109 106 110 56 83 92 35 34 24
90 51 59 118 123 113 109 109 99 91 108 103 69 43 31 32
101 60 59 105 105 105 98 99 105 109 95 114 101 110 54 39 36 22
90 45 49 94 101 96 94 109 104 100 107 93 98 39 49 34 31 18
67 46 45 58 95 102 77 85 101 79 91 97 71 58 43 38 16 19
19 39 50 48 54 72 98 91 68 88 97 79 61 36 33 19 17 20
Image processing in
Python

Camera calibration

Lenses are far from perfect!



Pinhole model

$$K = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix}$$

4 projection parameters
5 distortion parameters

Camera calibration with OpenCV

```
image_points, object_points = [], []
calibration_object_3d = make_checkerboard_3d(checkerboard_size)

for fname in image_fnames:
    im = load_image(fname)

    board_found, corners = cv2.findChessboardCorners(
        im, checkerboard_size, None)
    if board_found:
        image_points.append(corners_refined)
        object_points.append(calibration_object_3d)

_, camera_matrix, distortion_coeffs, _, _ = cv2.calibrateCamera(
    object_points, image_points, im.shape[::-1], None, None)
```

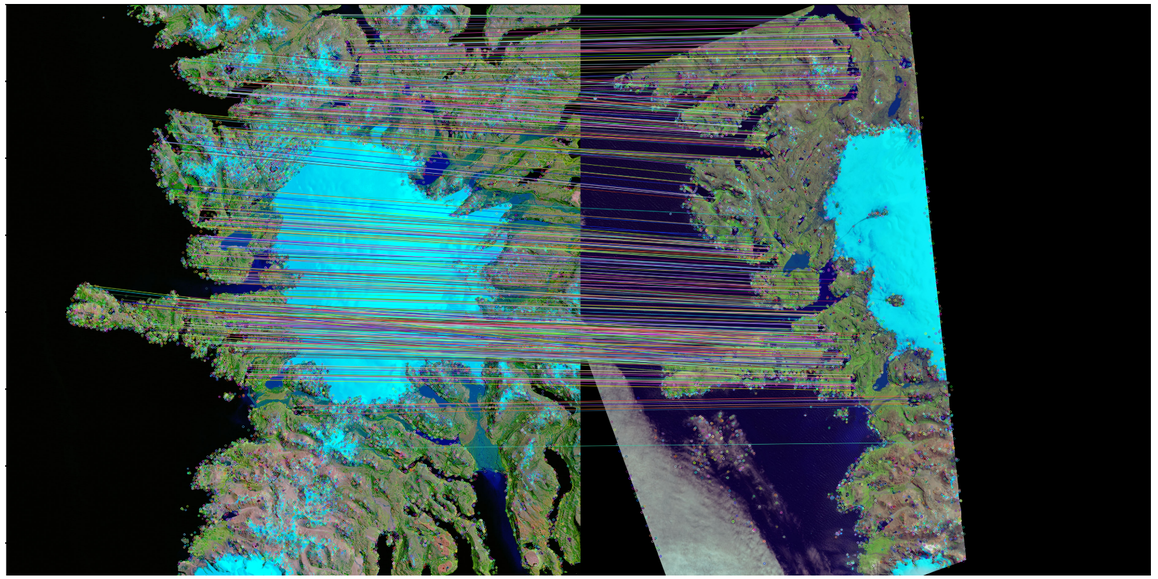
Undistort images

```
im_undist = cv2.undistort(im, camera_matrix, distortion_coeffs)
```

Image alignment

600-1000 m glacial retreat over 16 years - Kerguelen Islands - Indian Ocean

November 2001 - January 2017



NASA Earth Observatory images by Joshua Stevens, using Landsat data from the U.S. Geological Survey

Finding image correspondances

```
import cv2

# find the keypoints and descriptors with ORB
features = cv2.AKAZE_create()
keypoints_ref, descriptors_ref = features.detectAndCompute(im_ref, No
keypoints, descriptors = features.detectAndCompute(im, None)

# create BFMatcher object
bf = cv2.BFMatcher(cv2.NORM_HAMMING, crossCheck=True)

# Match descriptors.
matches = bf.match(descriptors_ref, descriptors)

# Sort them in the order of their distance.
matches = sorted(matches, key=lambda x:x.distance)
```

Robust image alignment - RANSAC

```
from skimage import transform
from skimage.measure import ransac

# transformation of matched keypoints to their coordinates
# (points, points_ref) omitted for brevity

model_robust, inliers = ransac(
    (points, points_ref),
    transform.SimilarityTransform, min_samples=3,
    residual_threshold=2, max_trials=100)
```

Image alignment

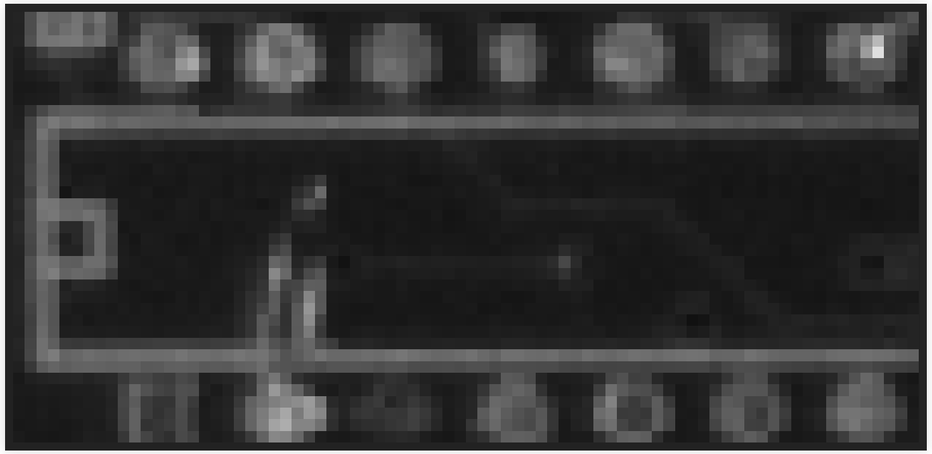
600-1000 m glacial retreat over 16 years - Kerguelen Islands - Indian Ocean

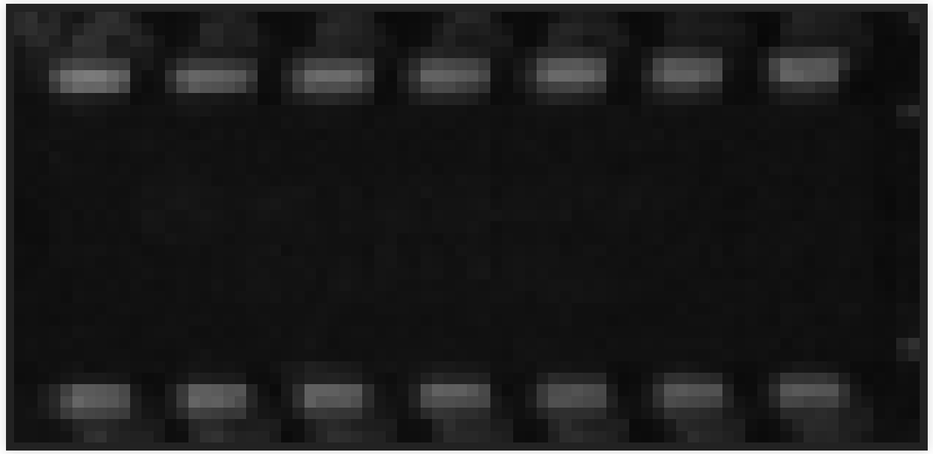
NASA Earth Observatory images by Joshua Stevens, using Landsat data from the U.S. Geological Survey

Mosaics - for tram maintenance

Object detection

Image -> Yes/No





Writing small reusable modules for computer vision in Gluon

Example detection model in Gluon

```
def make_detection_model(num_output_classes=4):
    net = gluon.nn.Sequential()
    net.add(nn.Conv2d(10), activation='relu')
    net.add(nn.GlobalAvgPool2D())
    net.add(nn.Flatten())
    net.add(nn.Dense(2))
    return net
```

Training a module

```
for batch in train_data:
    data = batch.data[0].as_in_context(computation_context)
    label = batch.label[0].as_in_context(computation_context)

    # record the computational graph
    with mx.autograd.record():
        prediction = net(data)

        # compute the loss
        loss = loss_function(prediction, label)

    # compute the gradients
    loss.backward()

    # run one optimization step
    trainer.step(data.shape[0])
```

Easy to switch the training on and off

Async

Acquisition, processing, and real-time updates with Sanic

```
from sanic import Sanic
app = Sanic(__name__)
...
app.add_task(main())
app.run(host="0.0.0.0", port=8000, debug=True)
```

```
async def main():
    camera_manager = app.camera_manager

    acquisition_queue = Queue(maxsize=1)

    app.loop.create_task(image_producer(
        camera_manager=camera_manager,
        queue=acquisition_queue
    ))

    ...
```

Producers and consumers

```
async def image_producer(camera_manager, queue):
    logger.debug(f'Starting image producer')
    while True:
        cameras = camera_manager.cameras
        logger.debug(f'Found {len(cameras)} cameras')

        await asyncio.gather(
            *[grab_and_enqueue(cam, queue) for cam in cameras],
            return_exceptions=True
        )
    await queue.put(None)
```

Real-time GUI updates

```
class StateHub:
    def __init__(self, serializer: Callable):
        self.serializer = serializer
        self.subscribers: set = set()
        self.cache: Any = None

    async def subscribe(self, subscriber, replay_last=True):
        self.subscribers.add(subscriber)
        if replay_last and self.cache is not None:
            await subscriber.send(self.cache)

    def unsubscribe(self, subscriber):
        self.subscribers.remove(subscriber)
```

Publish updates

```
    async def send(self, message) -> None:
        """Publish the message to all subscribers."""
        if self.serializer:
            message = self.serializer(message)

        self.cache = message

        subscribers = self.subscribers.copy()
        for subscriber in subscribers:
            try:
                await subscriber.send(message)
            except ConnectionClosed:
                self.unsubscribe(subscriber)
```

Prepare the state update publisher

```
@app.listener('before_server_start')
async def prepare_info_hub(app, loop):
    """Prepare the hub"""
    app.hub = StateHub(serializer=serializer)
```

Websockets for real-time updates

```
@app.websocket('/feed')
async def feed_camera(request, ws):
    await app.hub.subscribe(ws)
    # prevent the websocket from closing
    while True:
        await asyncio.sleep(0.001)
```

Deployment

- Camera drivers can be nasty
- Docker and friends make things simpler to deploy and restart

Conclusion

- Vision in python is fun and a pleasure to work with
- OpenCV, scikit-image, imageio,...
- Python is fast enough
- 2-3 frames per second (12MPix images)

Machine vision in Python

Jan Margeta | jan@kardio.me | @jmargeta

KardioMe

May 4, 2018