Visual Servoing & Tracking in Python

Samuel Felton

23 May 2024

ViSP + Python: Why?

Mature and modular C++ library

- Control laws, VS primitives
- Tracking algorithms
- Interface with sensors robots

ViSP Visual Servoing Platform

Popular and flexible language

- Easy to learn
- Fast prototyping
- Access to other libraries

Objectives

Accelerate research

- Spend less time on code
  - More time for exploration
  - Experiments
- Access to other resources
  - Deep learning
  - Plotting tools
  - Other research projects

Increase ViSP usage

- Promote Rainbow's research
- More users
  - Feedback
  - Bug reports
  - Potential contributions

Python bindings

Bindings: glue between C++ and Python

By hand? Okay for small libraries, not larger ones

- Boilerplate: tedious
- Constant evolution
  - Changes to C++ API
  - Interaction with Python users may change

- Solution: automatic bindings generation
  - Common: OpenCV, Panda3D
  - But still, library-specific


```
py::class_<Generator>(m, "Generator")
  .def(py::init<const SceneSet&, long, py::dict>())
  .def("generate", &Generator::generate)
  .def("move_from_pose", &Generator::move_from_pose)
  .def("denoising_ae_data", &Generator::denoising_ae_data)
  .def("scene_clustering_sample", &Generator::scene_clustering_sample)
  .def("new_vs_example", &Generator::new_vs_example);
```

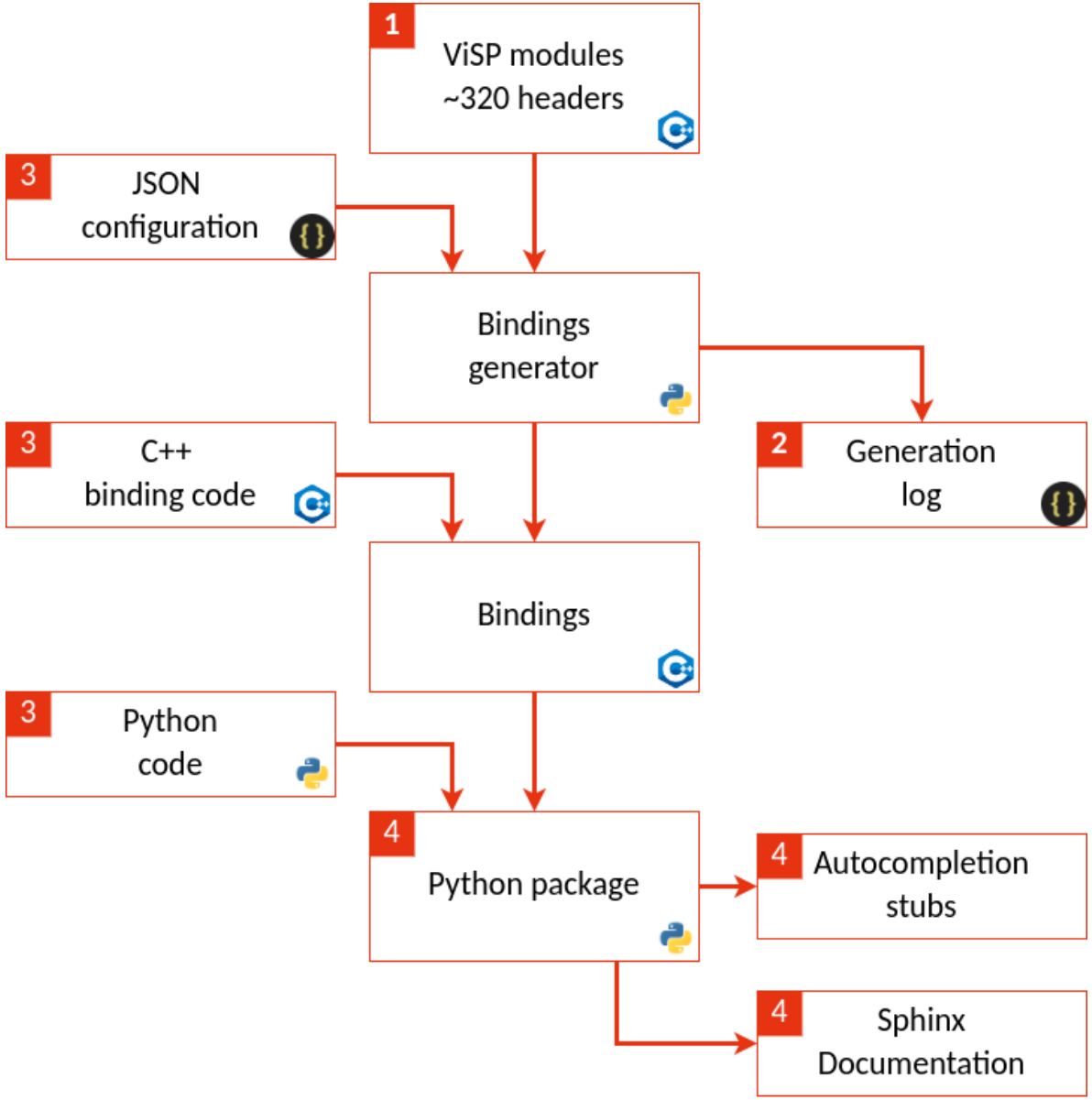
Python bindings : Requirements

1. Unobtrusive
   - Low impact on C++ sources
   - Minimize effort from developers

2. Best effort
   - Generate trivial API without any user input
   - Rely on config for other cases
   - Warn about potential issues

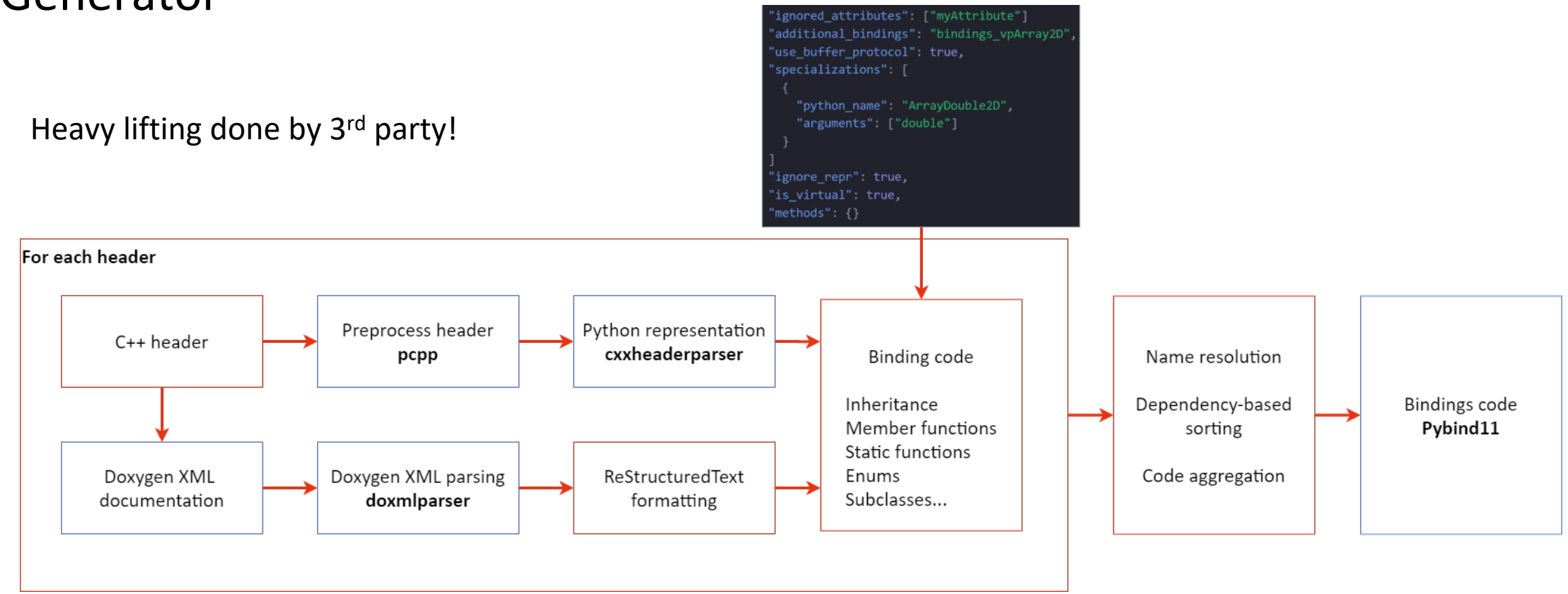
3. Configurable
   - Tweak generation behavior
   - Custom C++/Python code

4. Seamless Python API
   - Documentation, autocompletion
   - Python type support



Generator

Heavy lifting done by 3rd party!



https://github.com/ned14/pcpp
https://cxxheaderparser.readthedocs.io/en/latest/
https://github.com/doxygen/doxygen
https://pybind11.readthedocs.io/en/stable/

Generation results

Statistics:

- 250+ classes
- 100+ enumerations
- 6k+ symbol definitions

```
================================================
Statistics for module mbt:

        Ignored headers: 0
        Ignored classes: 1
        Unacknowledged pointer/ref holders: 15
        Ignored methods: 13
        Methods with default parameter policy: 10
        Methods returning a reference: 16
```

Most of the API is generated

Remaining symbols reported to the developer

```
"vpMbtPolygon* getPolygon(unsigned int)": {
  "reason": "return_type",
  "fix": {
    "static": false,
    "signature": "vpMbtPolygon* getPolygon(unsigned int)",
    "ignore": true
  },
  "class": "vpMbGenericTracker"
},
```

Some bindings are still handwritten

- Interface with NumPy
- Non trivial functions (heavy pointer use)
- Performance-critical methods

Usage

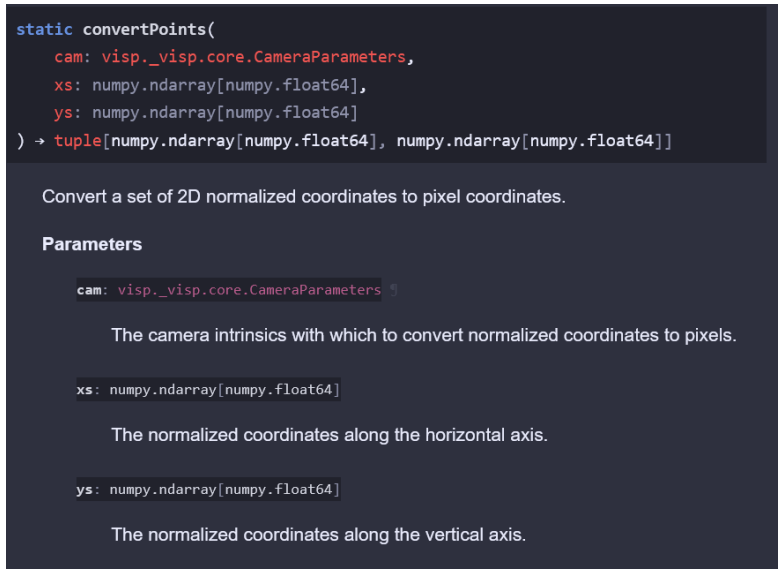
```python
from visp.core import Matrix, CameraParameters
from visp.core import PixelMeterConversion
import numpy as np

np_array = np.random.random((2, 2))
print('Building ViSP matrix from numpy array')

matrix = Matrix(np_array)
print(f'Matrix first row {matrix[0, :]}')
print(f'Matrix first column {matrix[:, 0]}')
print(f'Full ViSP matrix: ')
print(matrix)

h, w = 480, 640
cam = CameraParameters(px=600, py=600, u0=w / 2, v0=h / 2)
print(cam)

print('Converting points from pixel to normalized coordinates')
n = 5
u, v = [np.random.uniform(0, dimension, n) for dimension in (w, h)]
x, y = PixelMeterConversion.convertPoints(cam, u, v)
print(f'xs = {x}\nys = {y}')
```

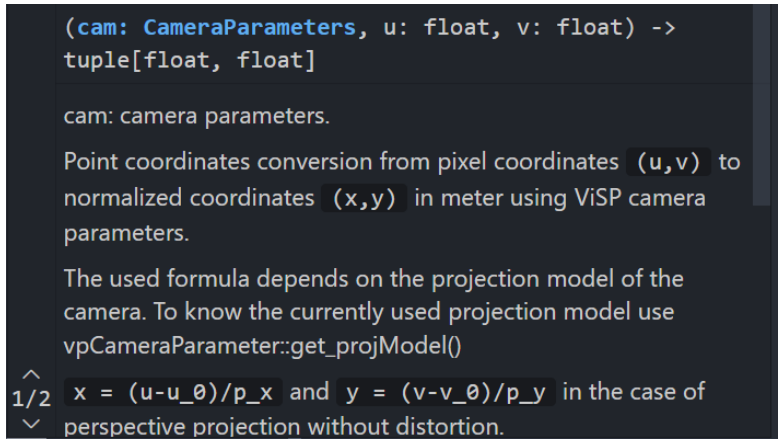
```
Building ViSP matrix from numpy array
Matrix first row [0.19350232 0.11168603]
Matrix first column [0.19350232 0.81346226]
Full ViSP matrix:
0.1935023215  0.1116860314
0.8134622576  0.4946122081
Camera parameters for perspective projection without distortion:
  px = 600        py = 600
  u0 = 320        v0 = 240

Converting points from pixel to normalized coordinates
xs = [ 0.03655584  0.17144516 -0.24142859 -0.28412666  0.42821268]
ys = [ 0.32289244  0.23884591  0.11391857 -0.36693617 -0.22059141]
```

NumPy/buffer representation: bridge to other APIs

- Matplotlib
- Pytorch
- Scikit…
- Realsense wrapper

User friendliness





Documentation: Sphinx

- Available online
- Autogenerated API reference
- Transition from C++ to Python
- Interface with NumPy and others
- Some examples, more coming
  - IBVS
  - PBVS
  - Model-Based Tracker

Autocompletion: Pybind11-stubgen

https://github.com/sizmailov/pybind11-stubgen
https://www.sphinx-doc.org/en/master/

Proof of concept

Centering task

- Look-at a car
- 2 DoFs: Pan-tilt camera
- Consider single feature: 2D point
  - Desired: image center
  - Current: center of car in image

How to get center of object?

- Compute bounding box with YoloV8
- Pretrained network
- Easy to use in Python

Servoing: ViSP modules vs and visual_features

Proof of concept: simulation

Define input scene: ImageSimulator from ViSP

Planar scene: quick prototyping

Could be camera + robot combination

```
def get_simulator(scene_path: str) -> ImageSimulator:
  scene_image = np.asarray(Image.open(scene_path).convert('RGBA'))
  scene_image = ImageRGBa(scene_image)
  simulator = ImageSimulator() # Planar scene from single image
  l, L = 1.5, 1.0
  scene_3d = [
    [-l, -L, 0.0], [l, -L, 0.0],
    [l, L, 0.0], [-l, L, 0.0],      You, now • Uncommitted changes
  ]
  simulator.init(scene_image, list(map(lambda X: Point(X), scene_3d)))
  simulator.setCleanPreviousImage(True, color=Color.black)
  simulator.setInterpolationType(ImageSimulator.InterpolationType.BILINEAR_INTERPOLATION)
  return simulator
```

Proof of concept: task definition

```
# Initialization
simulator = get_simulator(args.scene)
cTw = HomogeneousMatrix(-2.0, 0.5, Z, 0.0, 0.0, 0.0)
I = ImageRGBa(h, w)
Idisp = ImageRGBa(h, w)
simulator.setCameraPosition(cTw)
simulator.getImage(I, cam)

# Define centering task
xd, yd = PixelMeterConversion.convertPoint(cam, w / 2.0, h / 2.0)
sd = FeaturePoint()
sd.buildFrom(xd, yd, Z)

s = FeaturePoint()
s.buildFrom(0.0, 0.0, Z)

task = Servo()
task.addFeature(s, sd)
task.setLambda(0.5)
task.setCameraDoF(ColVector([0, 0, 0, 1, 1, 0]))
task.setServo(Servo.ServoType.EYEINHAND_CAMERA)
task.setInteractionMatrixType(Servo.ServoIteractionMatrixType.CURRENT)
target_class = args.class_id # Car
```

Proof of concept: servoing loop

```
# Build current features
results = detection_model(np.array(I.numpy()[..., 2::-1])) # Run detection
boxes = map(lambda result: result.boxes, results)
boxes = filter(has_class_box, boxes)
boxes = sorted(boxes, key=lambda box: box.conf[0])
bbs = list(map(lambda box: box.xywh[0].cpu().numpy(), boxes))
```

```
if len(bbs) > 0:
  bb = bbs[-1] # Take highest confidence
  u, v = bb[0], bb[1]
  x, y = PixelMeterConversion.convertPoint(cam, u, v)
  s.buildFrom(x, y, Z)
  v = task.computeControlLaw()
```

```
# Move robot/update simulator
cTcn = ExponentialMap.direct(v, time.time() - start)
cTw = cTcn.inverse() * cTw
simulator.setCameraPosition(cTw)
```

Proof of concept: results

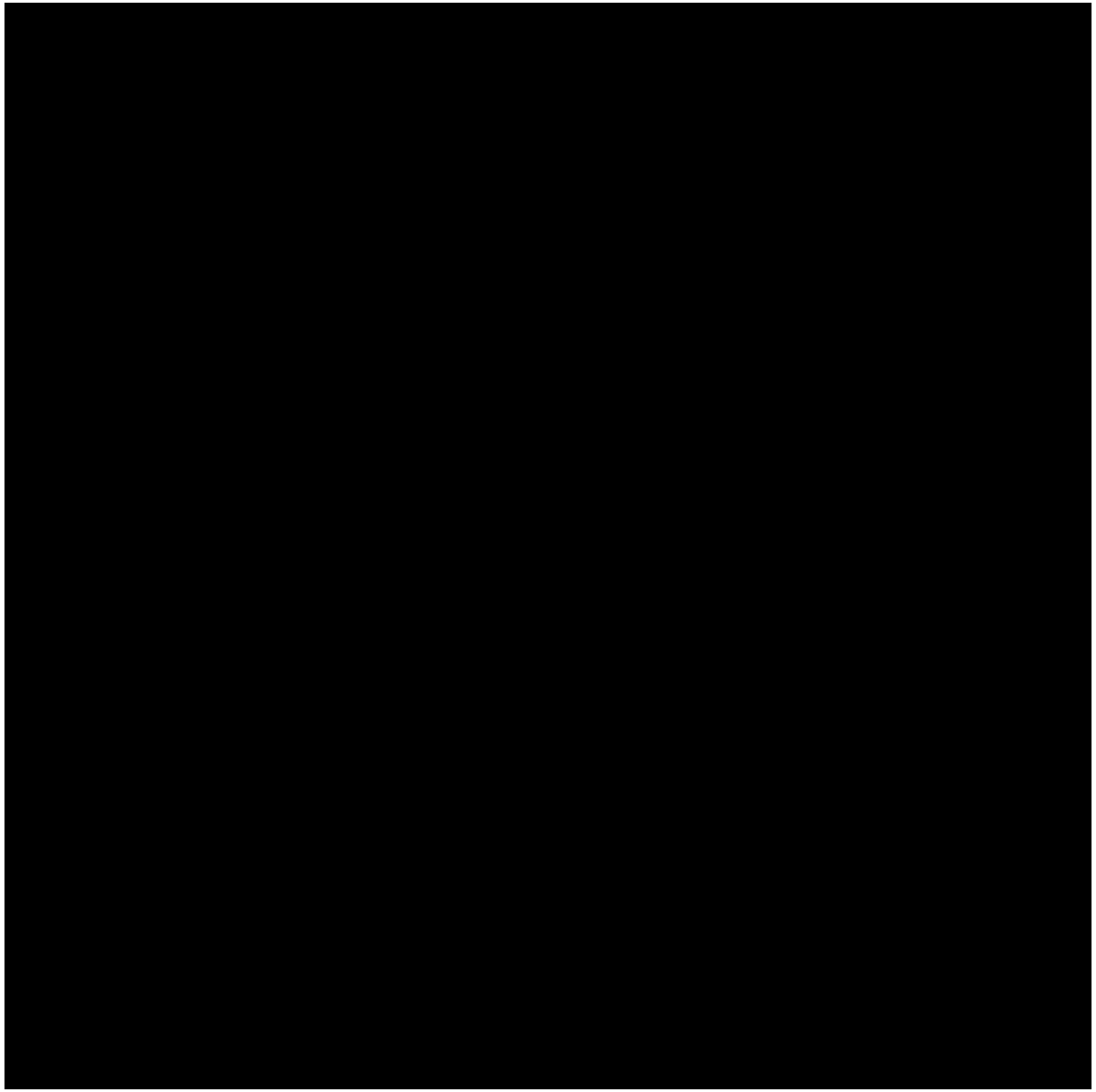
ViSP + machine learning in Python

Leverages python libs

- Numpy
- Pytorch
- Pillow

Simple code: <200 lines

- Control loop: ~50 lines
- Plotting: ~50lines



Next steps

Most of ViSP API available!

Scripts can use ViSP, but no inheritance

Interesting for

- Visual features
- From previous presentation
  - New tracking features
  - New rendering/postprocessing pipeline

More examples, more tutorials

Feedback is more than welcome!

```
class MyCustomFeature(BasicFeature):
  def __init__(self):
    BasicFeature.__init__(self)
    ...

  def init(self):
    '''Init inherited from BasicFeature'''

  def error(self, s_star: ColVector, select = FEATURE_ALL) -> ColVector:
    ...

  def interaction(self, select = FEATURE_ALL) -> Matrix:
    ...

s = MyCustomFeature()
...
sd = MyCustomFeature()
...
task = Servo()
task.addFeature(s, sd)
```

Thank you!

inria
inventors for the digital world