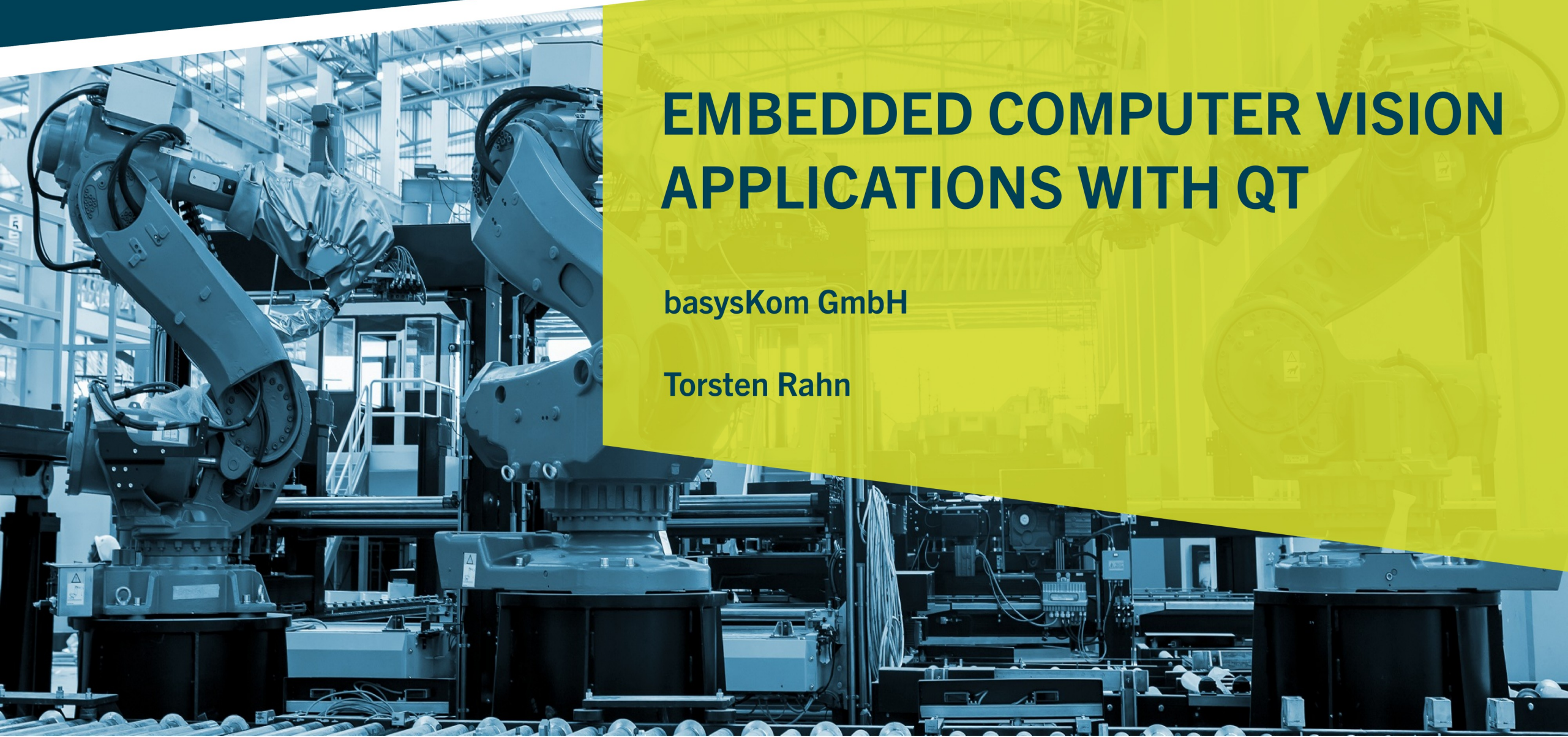
basysKom



EMBEDDED COMPUTER VISION APPLICATIONS WITH QT

basysKom GmbH

Torsten Rahn

CONTENT

- Introduction
- Qt and OpenCV
- Conclusion

INTRODUCTION

Who we are

basysKom

- Located in Darmstadt & Nürnberg
- Software Engineering Services (Consulting, Training, Coaching & Development)

Myself

- Software Engineer
- torsten.rahn@basyskom.com

Our Background

- Software Engineering Services (Consulting, Training, Coaching & Development)
- Focused on industrial applications

A great deal of experience with Application/ HMI development (Qt & HTML5) and connectivity (OPC UA/MQTT/REST)

QT AND OPENCV

OpenCV

Open Source Computer Vision Library

- Website: **opencv.org**
- Current Version: 4.0.1
- Languages: C++ (native), Java, Python
- Platforms: Windows, Linux, Mac OS, iOS and Android
- License: BSD

- 2D and 3D feature toolkits
- Egomotion estimation
- Facial recognition system
- Gesture recognition
- Human–Computer Interaction
- Mobile robotics
- Motion understanding
- Object identification
- Segmentation and recognition
- Structure from motion (SFM)
- Motion tracking
- Augmented Reality

Qt

Cross-platform software development for embedded & desktop

- Website: **qt.io**
- Current Version: 5.12
- Cross Platform
- License: Dual-License – Commercial & Open Source

“Classical” Qt Framework

- language: native C++ / Bindings
- compiled
- imperative

QObject

Qt Quick

- language: QML (Javascript / CSS / JSON)
- interpreted (compiling optional)
- declarative

Qt Quick and OpenCV

Computer Vision meets modern HMI-Development

- **Advantage:**
  - OpenCV's computer vision algorithms
  - fast and fluid Qt Quick User Interface on embedded
- **Challenge:**
  - Qt and OpenCV are using different
    - API patterns
    - data structures

Let's bridge the gap …

Qt and OpenCV: Image Format Conversion

Qt and OpenCV are using different storage data types for images

- OpenCV uses a matrix `cv::Mat` with data stored in GBR channel order in columns and rows

```
cv::Mat cvImage = cv::imread("foo.png",
                             CV_LOAD_IMAGE_COLOR);
QImage img((uchar*) cvImage.data,
           cvImage.cols, cvImage.rows,
           cvImage.step, QImage::Format_RGB888);
img = img.rgbSwapped().copy();
```

- Channel order for `QImage` depends on *Endianess* – except for RGB888 format where it's always RGB
- GBR → RGB via `QImage::rgbSwapped()`

Introduction to QML

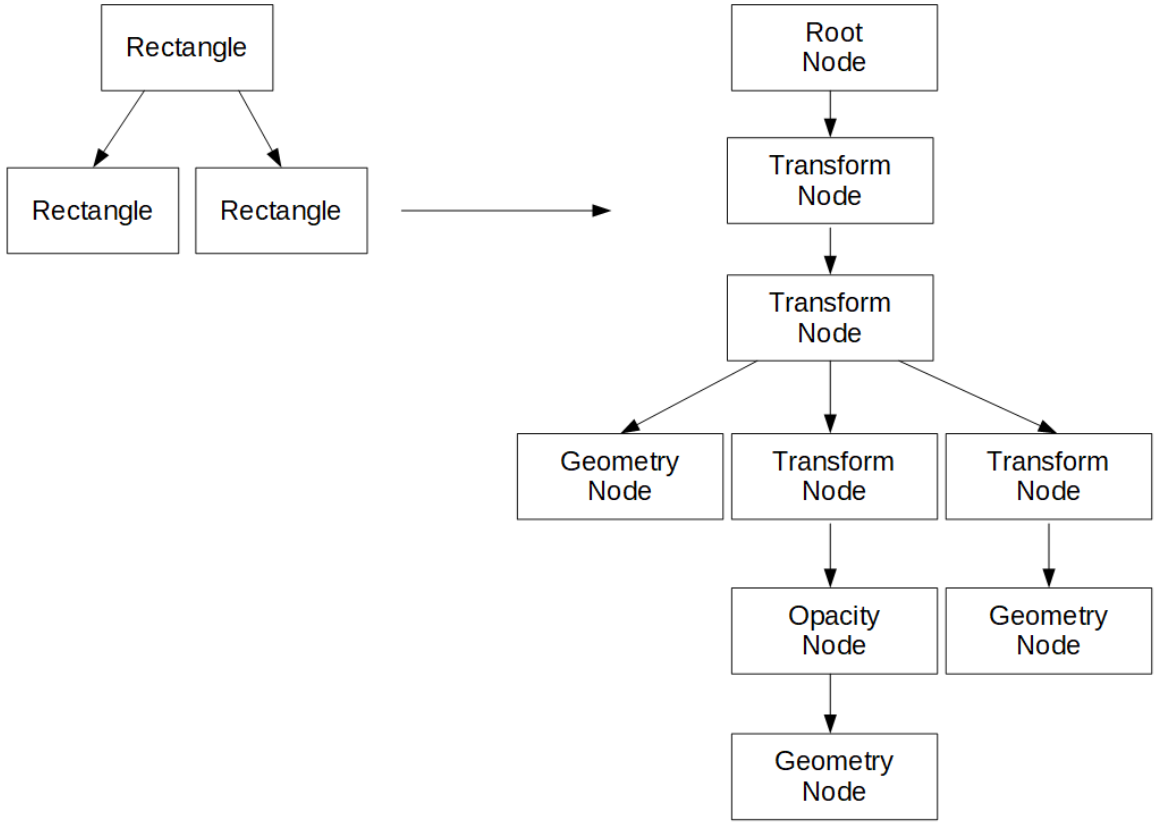
```
import QtQuick 2.0

Rectangle {
    id: root
    width: 300; height: 350
    color: "blue"
    Rectangle {
        anchors {left: parent.left; anchors.top: parent.top}
        width: parent.width/2; height: parent.height/2
        color: "green"
    }
    Rectangle {
        anchors {right: parent.right; bottom: parent.bottom}
        width: parent.width/2; height: parent.height/2
        color: "yellow"
    }
}
```

Concept

- Elements
- ids
- Properties
- Property-Bindings

Qt Quick Scenegraph



Concept

- Optimized for requirements of modern graphics hardware
- Uses OpenGL commands
- Transform nodes encode affine Transformations as 4x4 matrices
- Geometry nodes ...
  - manage geometry buffer
  - define material

Combining OpenCV with QML

```
Item {
    width: 300; height: 300

    CVMat {
        source: "foo.jpg"
        anchors {left: parent.left;
          anchors.top: parent.top; anchors.bottom: parent.bottom}
        width: parent.width / 2
    }
    CVContour {
        input: inputImage
        anchors {right: parent.right;
          anchors.top: parent.top; anchors.bottom: parent.bottom}
        width: parent.width / 2; height: parent.height / 2
    }
}
```

CVMat - Implementing a QQuickItem using OpenCV

Inheritance from QQuickItem

- Don't forget to set the flag `ItemHasContents` in the ctor

```
QSGNode *AbstractMat::updatePaintNode(QSGNode *oldNode, QQuickItem::UpdatePaintNodeData *)
{
    QSGSimpleTextureNode *node = static_cast<QSGSimpleTextureNode *>(oldNode);
    if (!node) {
        node = new QSGSimpleTextureNode();
        QImage img((uchar*) m_cvImage.data, m_cvImage.cols, m_cvImage.rows,
                   m_cvImage.step, QImage::Format_RGB888);
        img = img.rgbSwapped().copy();
        QSGTexture *texture = window()->createTextureFromImage(img);
        node->setTexture(texture);
    }
    node->setRect(boundingRect());
    return node;
}
```

CVContours - Finding Contours with OpenCV

Using findContours()

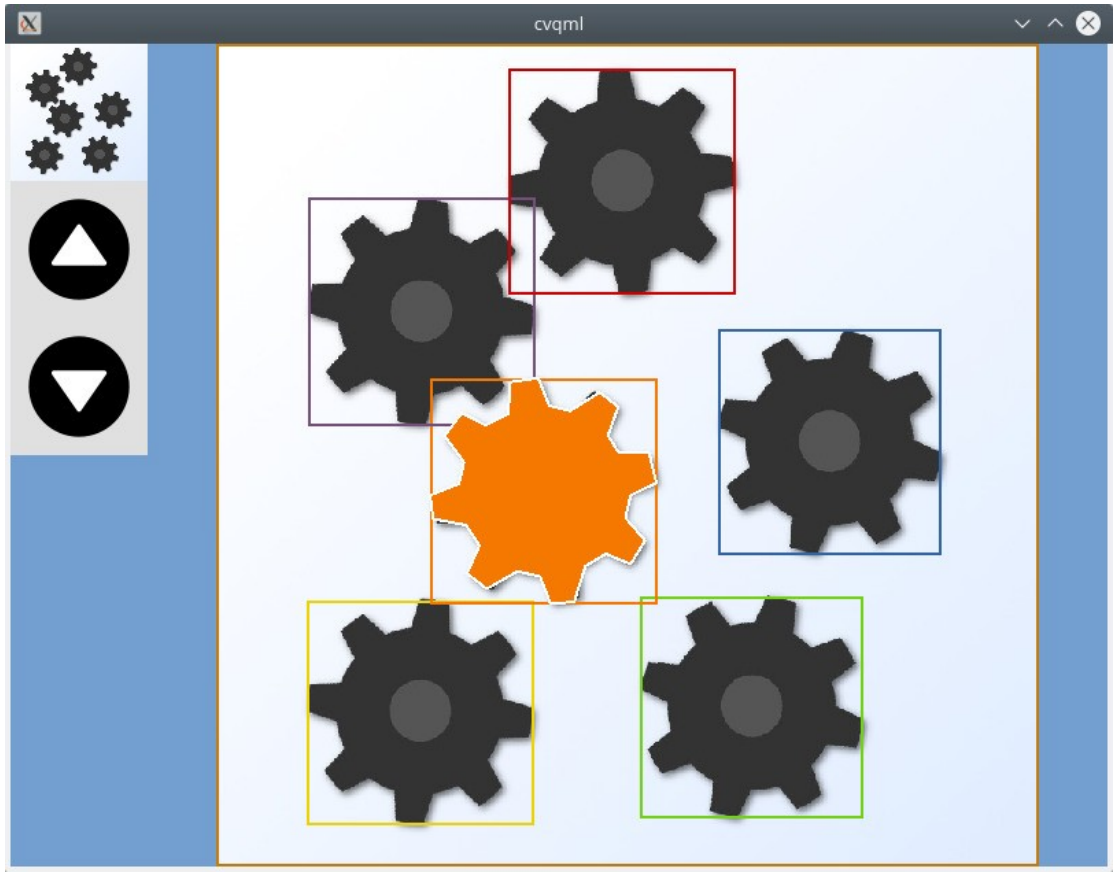
- For preparation we need to convert the image into a binary black-and-white image:

```
Mat gray;
cvtColor(*mat, gray, COLOR_BGR2GRAY);
Mat bw;
threshold(gray, bw, 50, 255, THRESH_BINARY | THRESH_OTSU);
vector<vector<Point>> contours;

findContours(bw, contours, RETR_LIST, CV_CHAIN_APPROX_SIMPLE);
```

- Alternatively we could call `Canny()` instead of `threshold()`
- Afterwards we could call `drawContours()` or ...

Application Example



Live Demo

Our demo application additionally uses

- Qt Quick Controls 2
- Qt Quick Shape API (available since 5.10)

LiveCV http://livecv.dinusv.com/

CONCLUSION

Conclusion

Combination of Qt Quick and OpenCV provides a solid foundation for Computer Vision HMIs

- Differences regarding APIs should be taken into account right from the start
- Qt Quick provides flexible means for interaction and UI rendering
- LiveCV provides a great environment for interactive prototyping

Weblink Recommendations

Tutorials, Documentation and Examples:

- https://www.elektroniknet.de/design-elektronik/embedded/opencv-und-qt-quick-ein-einstieg-161630.html
- https://www.youtube.com/watch?v=2zTY6CFhP_A
- https://doc.qt.io/qt-5.11/qml-QtQuick-shapes-shape.html
- https://opencv.org/
- https://www.basyskom.com/download/cvqml.zip

THANK YOU!
QUESTIONS?

Torsten Rahn
Software Engineer
torsten.rahn@basyskom.com

basysKom Hall 4/400, Hall 4/258 (Qt)
www.basyskom.com