Augmented Reality

Tomáš Fabián

VSB-Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Computer Science

Passive vs. Active

- **Active 3D glasses** use active liquid crystal shutter glasses that close and open their lenses at a very fast rate.
- **Passive 3D glasses** have no moving parts and instead work by incorporating different polarized lenses in the left and right side of the glasses.

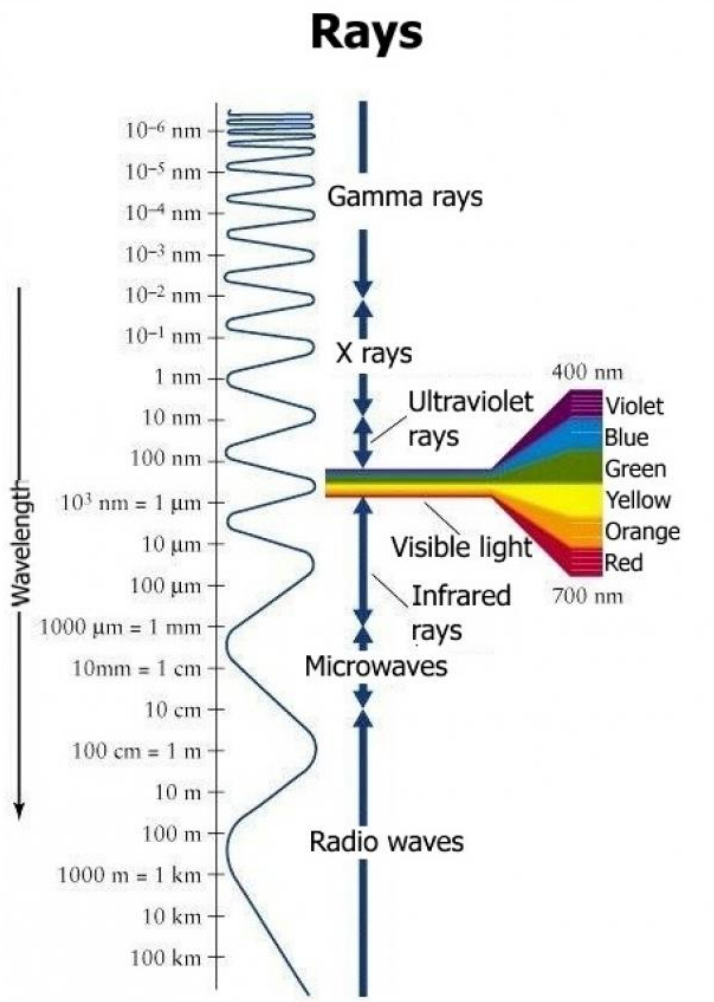
Passive vs. Active

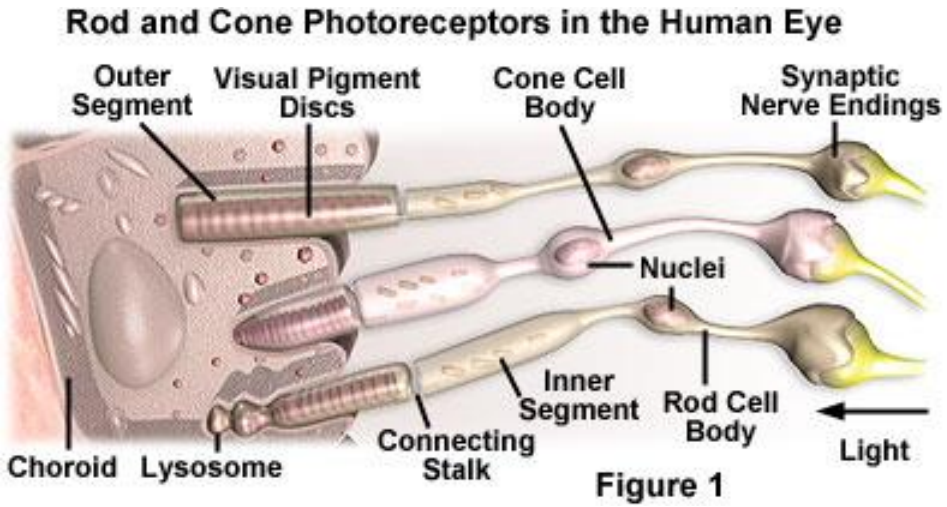
- The primary difference between the two options is the resolution.
- Active 3D TVs offer 1080 horizontal lines, while passive 3D TVs are only able to offer 540 lines.
- Passive 3D TVs cannot offer "true HD," they do offer *perceived* HD (i.e., the viewer's eyes won't necessarily be able to tell the difference).
- Active 3D TVs provide clearer edges without the jagged edge problem that can appear on passive 3D TVs.

Passive vs. Active

- A passive 3D TV requires that a viewer not tilt his or her head very much and remains more or less at the same horizontal level as the television.
- One of the major disadvantages of active 3D TVs is referred to as "crosstalk".
- The TV needs to be perfectly synchronized to the glasses and together they must coordinate at least 60 shutter flickerings each second.

Human Vision System

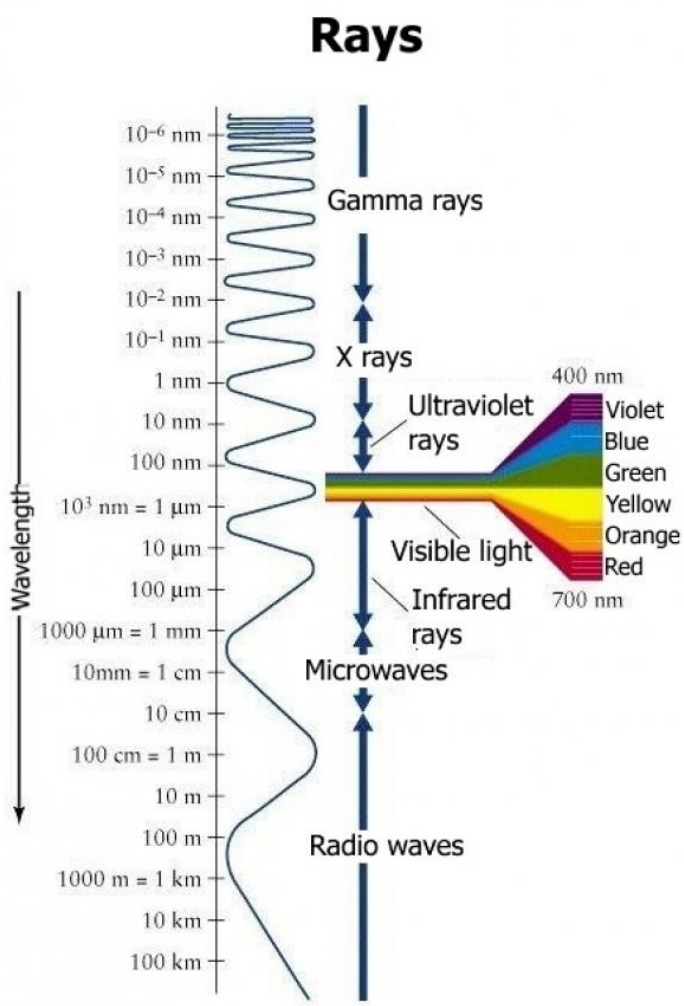


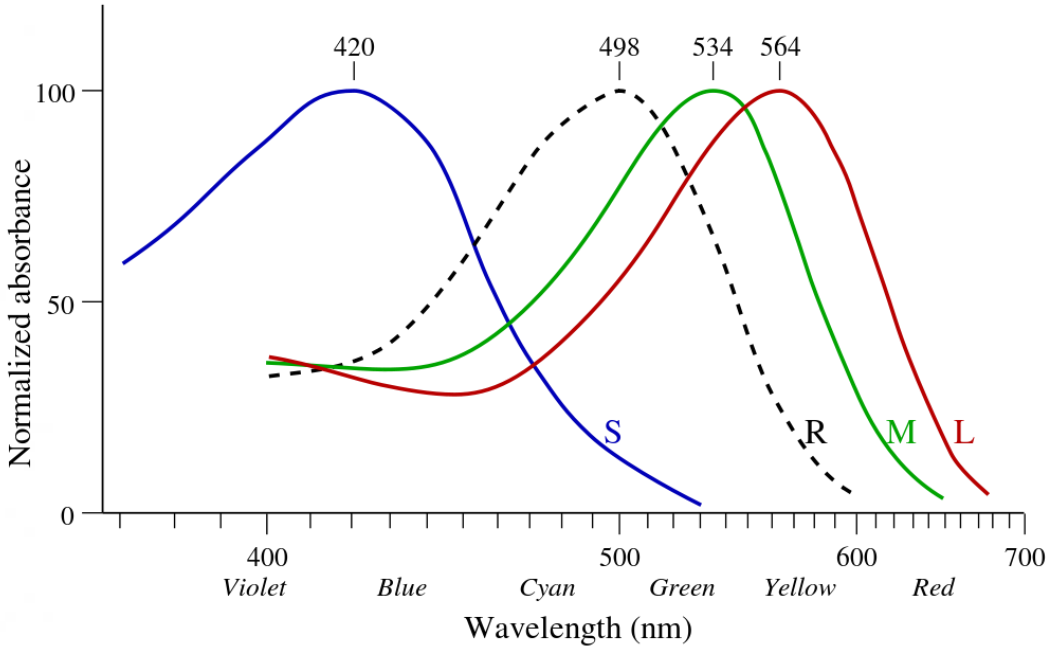


6M čípků (cones):
modrofialová (425 nm)
zelená (530 nm)
oranžová (560 nm)
Trichromatické vidění

120M tyčinek (rods)

Human Vision System





6M čípků (cones): **120M tyčinek (rods)**

modrofialová (425 nm)

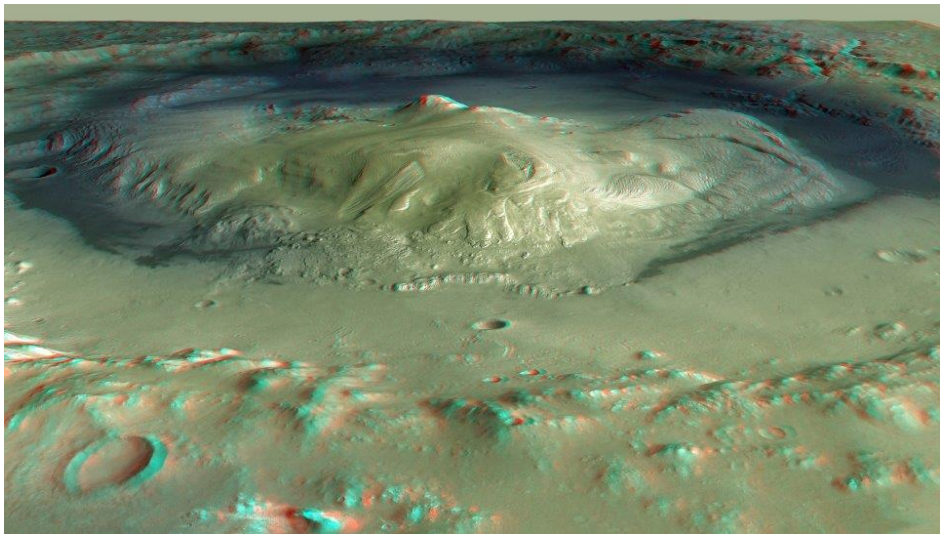
zelená (530 nm)

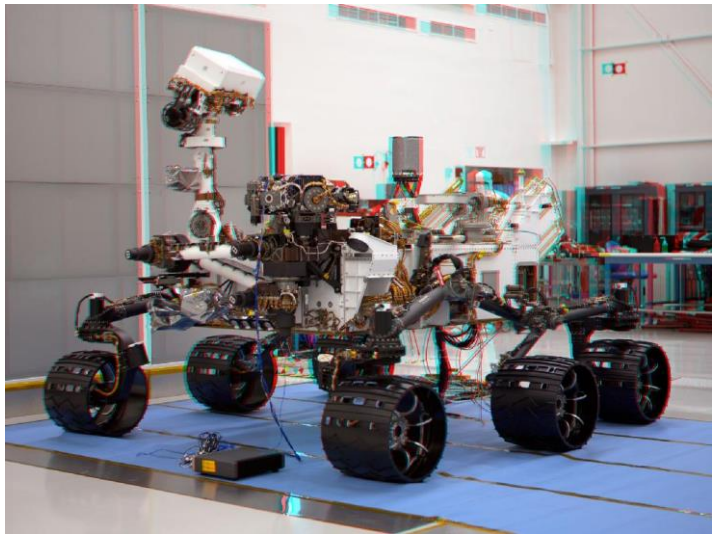
oranžová (560 nm)

Trichromatické vidění

Anaglyph 3D

- Stereoskopický efekt docílen zakódováním levého a pravého obrazu pomocí filtrů (např. červená, azurová)

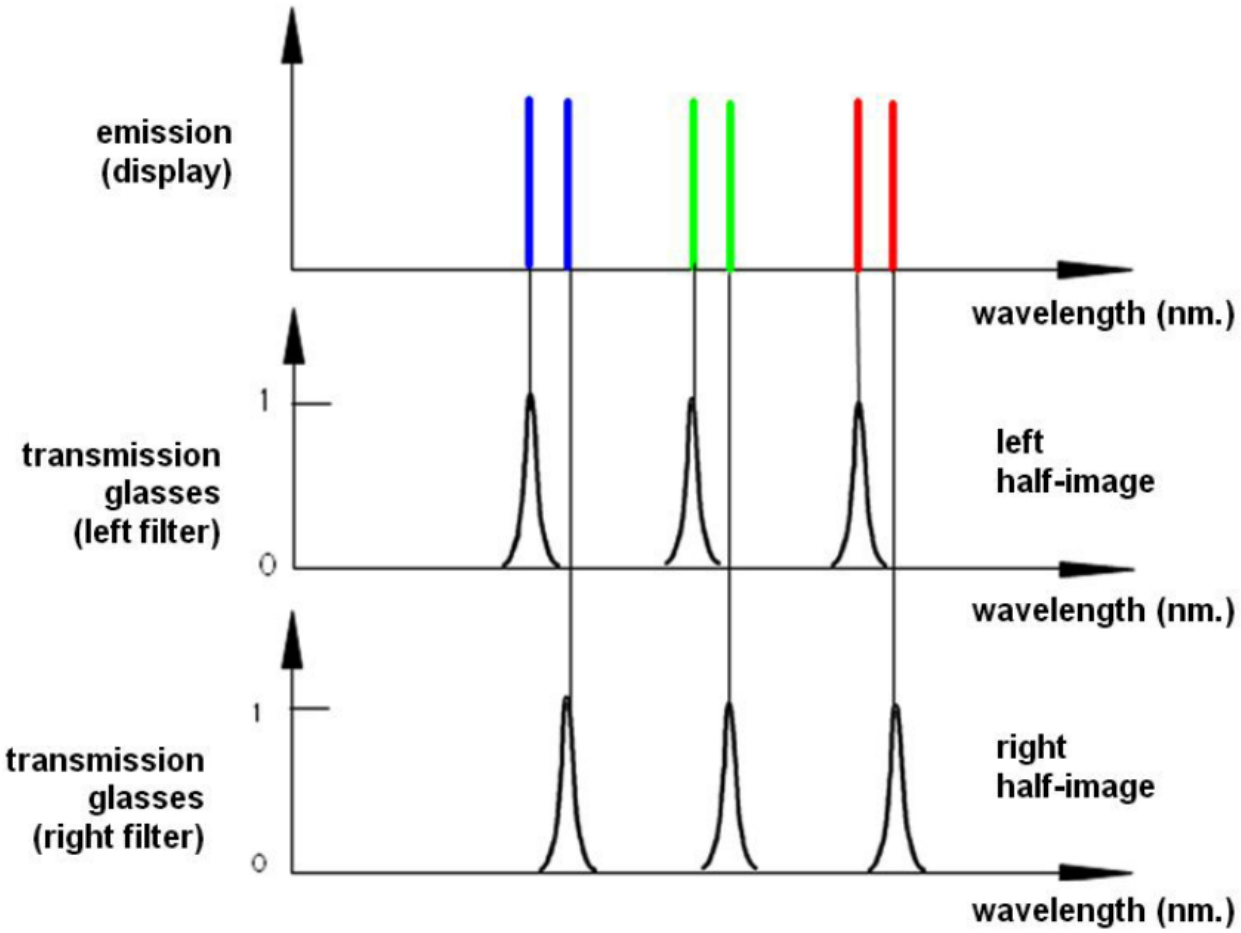




Infitec: Advanced Stereo Projection

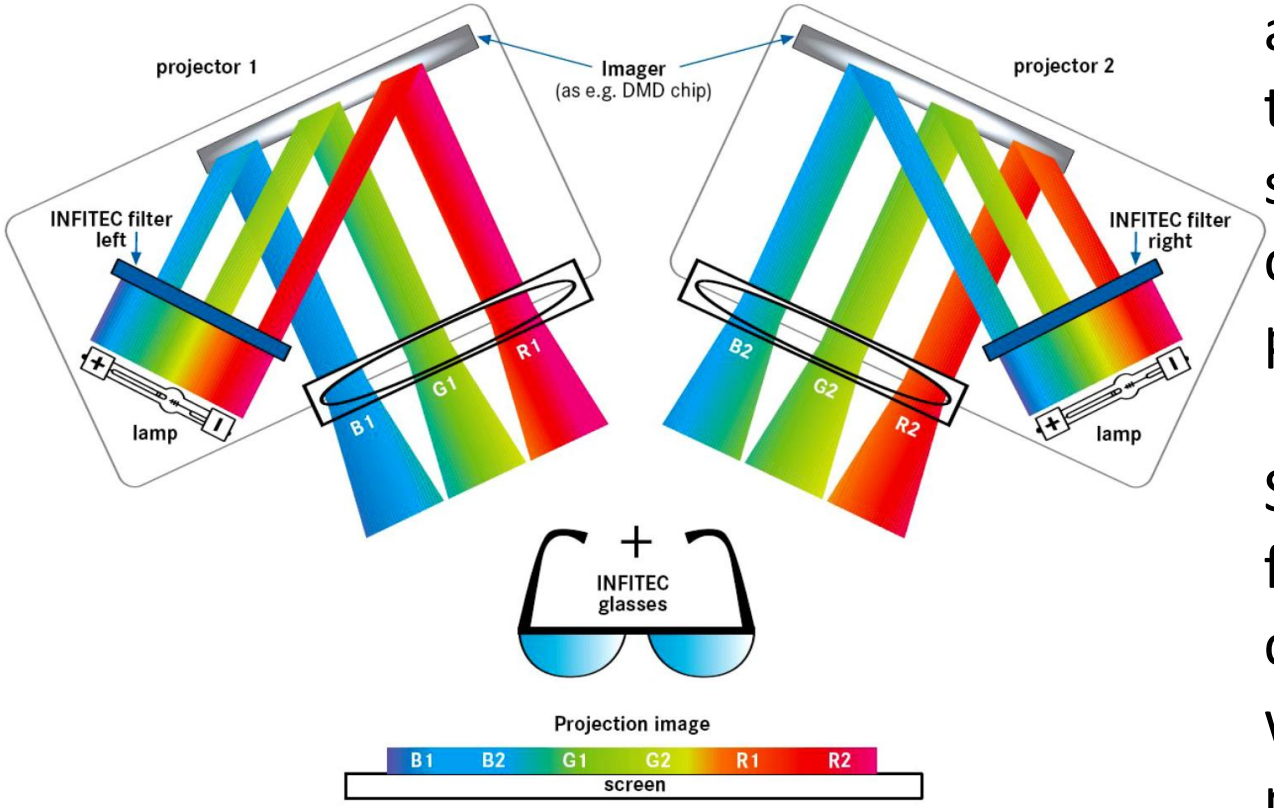
- Besides polarization and active shutter technologies, the wavelength multiplexing approach is an upcoming technology platform.
- Up to 1999 the only known spectral imaging technology was the anaglyph.

Infitec: Advanced Stereo Projection



Stereo-display principle using two wavelength emission triples and two matching complementary spectral-filter sets for the glasses.

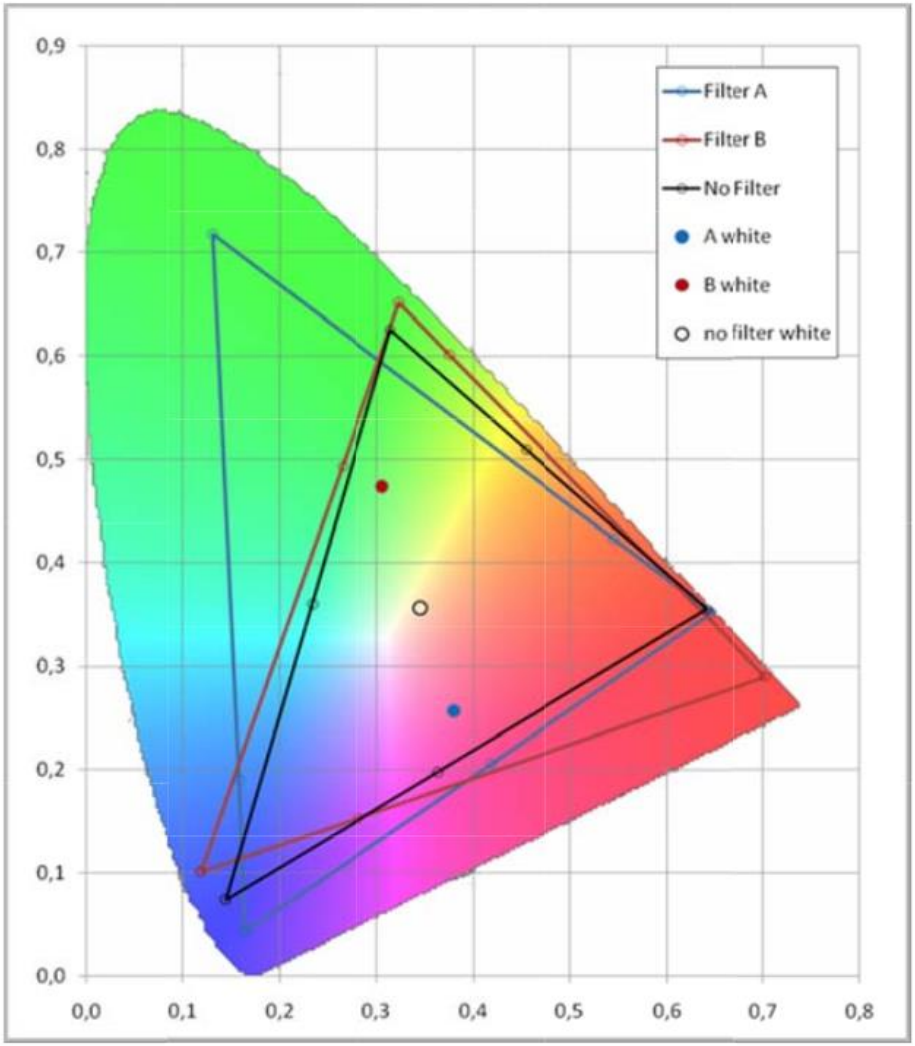
Infitec: Advanced Stereo Projection



Schematic set-up of an interference-filter technique system for stereo imaging using conventional digital projectors.

Stereo interference filter technology is compatible with any digital projection technology (LCD, DLP, and DILA).

Infitec: Advanced Stereo Projection



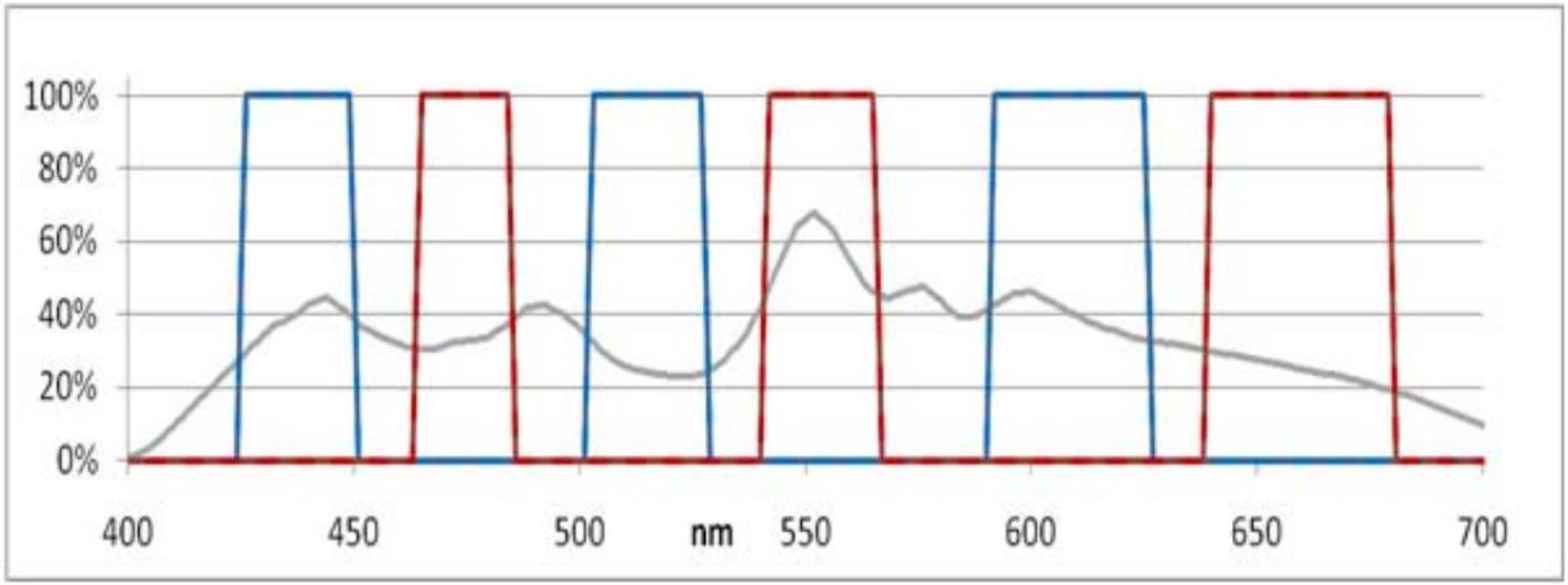
Positions of the individual bands in the spectrum → significant discrepancies in the color triangles.

Balanced colors → corrected by image processing.

Filtering a color correction → 12% of the initial brightness.

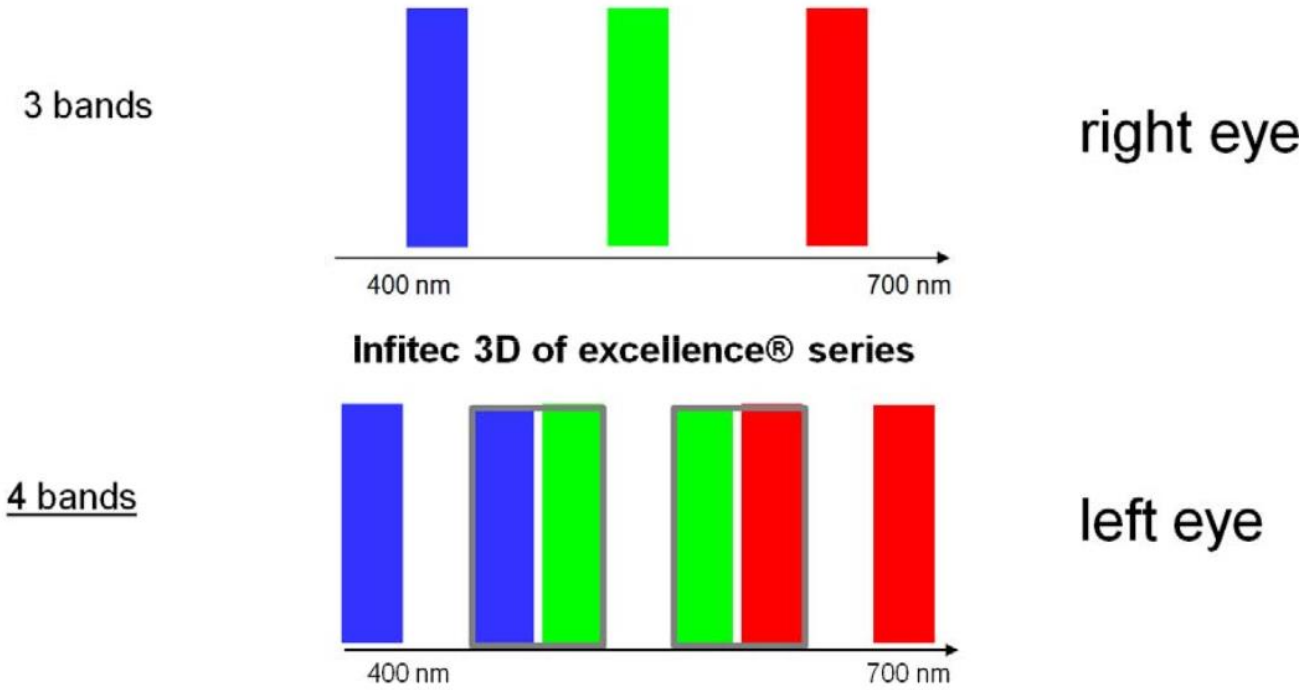
← CIE 1931 xy chromaticity diagram

Infitec: Advanced Stereo Projection



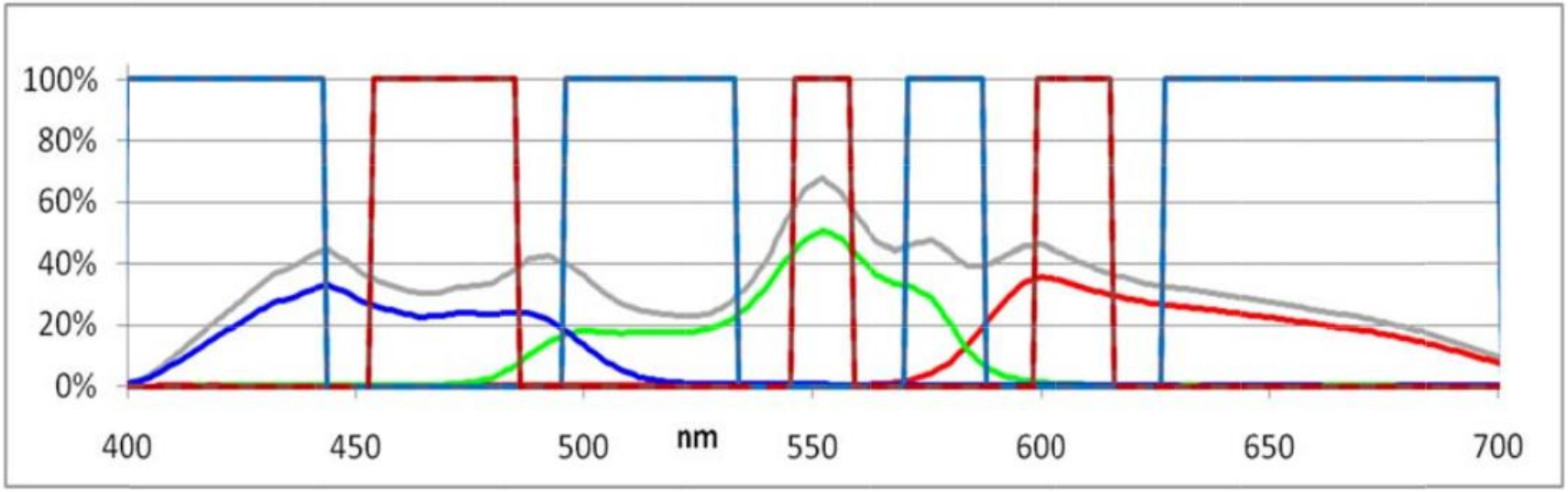
Projector white output spectrum and triple band filter configuration.

Infitec: Advanced Stereo Projection



Infitec 3-4 band filter design concept.

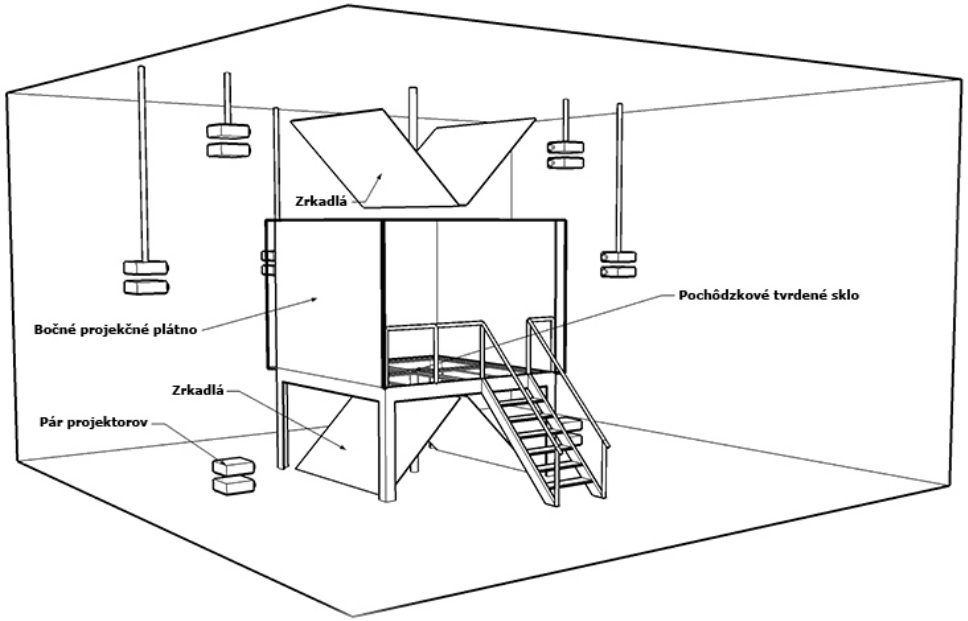
Infitec: Advanced Stereo Projection



Optimized 3-4 band filter for UHP lamp system.

CAVE

- TU Zvolen, SK
- Boční stěny 3 x 2,25m
- Vrchní a spodní stěny 3 x 3m
- Stereoskopický systém INFITEC

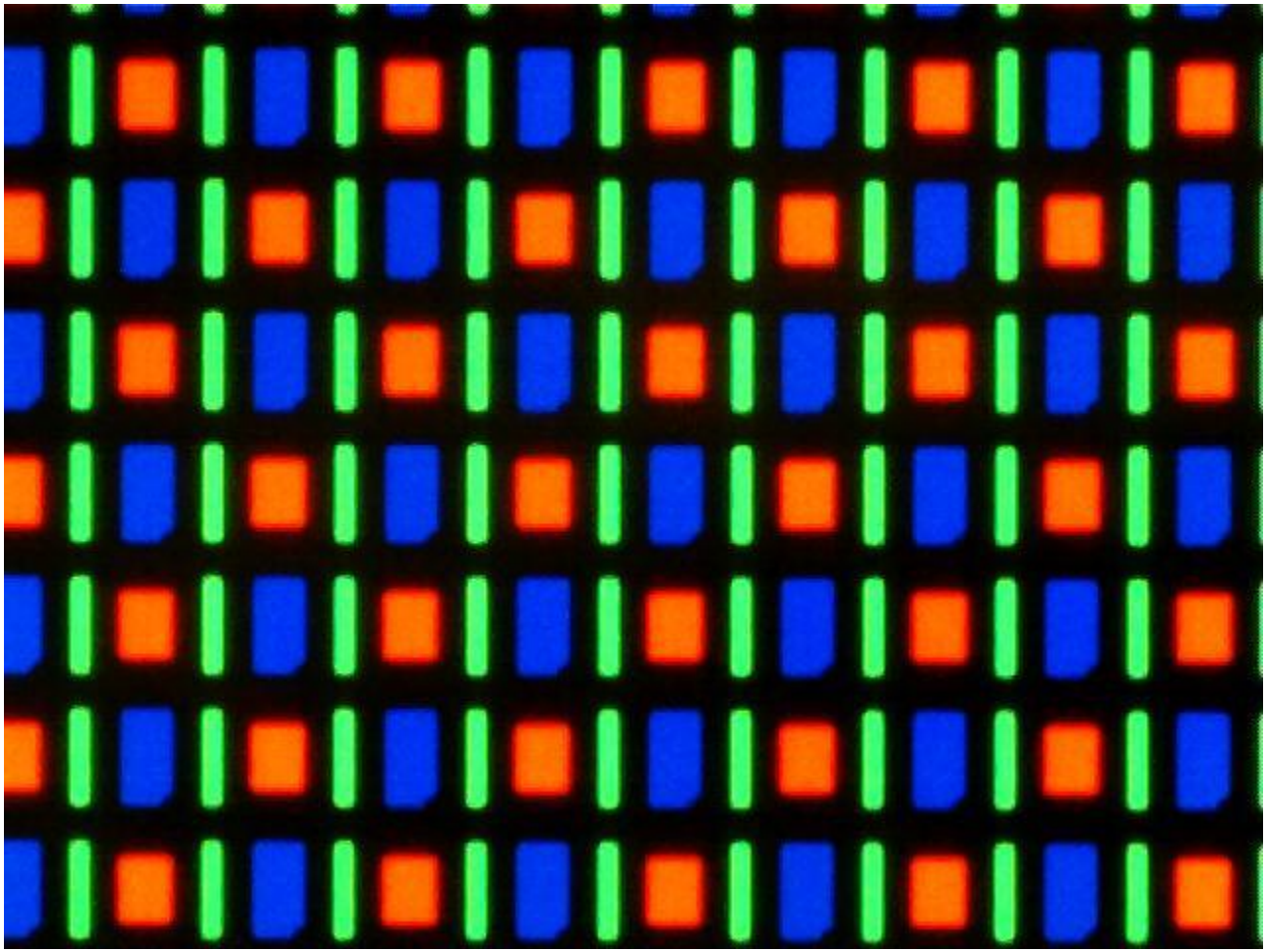


CAVE

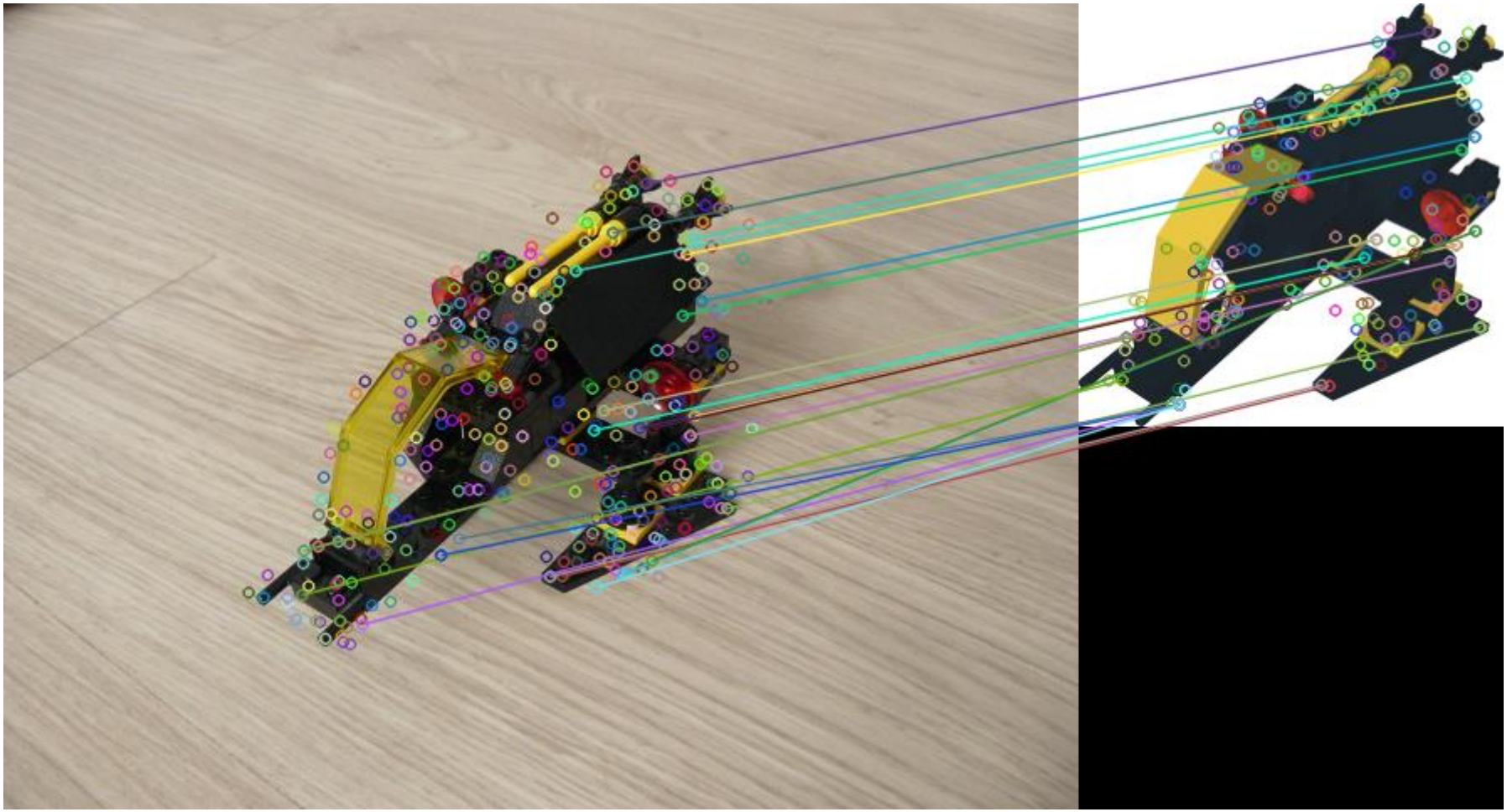
Oculus Rift

- Full HD OLED display s PenTile maticí (960x1080 na oko), tenký zelený G subpixel, větší R a B jsou sdílené
- FOV: 110° h, 90° v
- Čidla:
  - near-infra kamera pro natočení hlavy (až 110°) @ 60Hz
  - gyroskop
  - akcelerometr
  - magnetometr
  - vše @ 100Hz

Oculus Rift



Features Extraction - Example



Features Extraction – SIFT (principles)

- Scale Invariant Feature Transform
- Method to detect **distinctive** and **invariant** image feature points.

- Key points – interesting image regions can be matched between images to perform:
  - Object detection and recognition
  - Compute geometrical transformations between images

Features Extraction – SIFT (principles)

- Scale Invariant Feature Transform
- Method to detect distinctive and **invariant** image feature points.

- SIFT key points are scale and orientation invariant
  - The change in scale would not affect the key points
  - The same applies to scale
  - Partialy invariant to illumination, 3D view point, occlusion, clutter, noise
- Other types of invariance
  - Affine or perspective

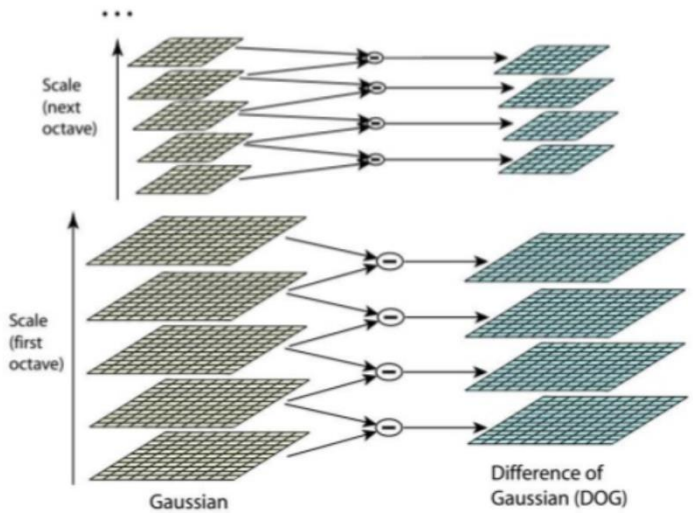
Features Extraction – SIFT (principles)

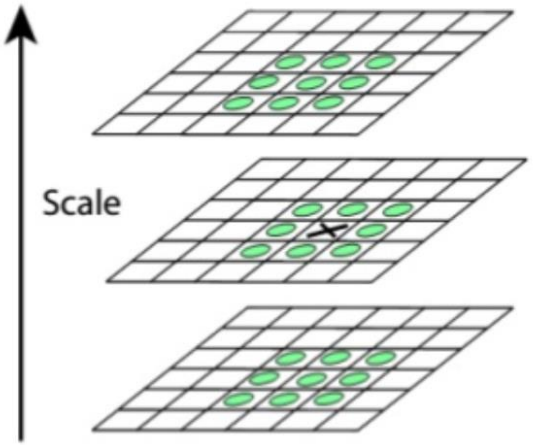
- Scale-space extrema detection
- Key point localization
- Orientation assignment
- Key point descriptor

Features Extraction – SIFT (principles)

Scale-space extrema detection

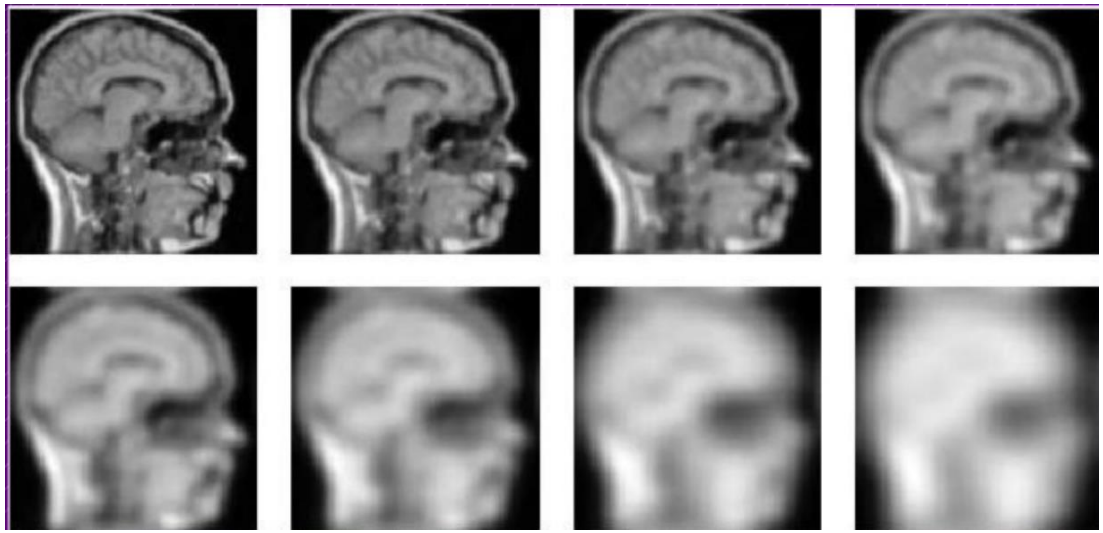
- Searching for stable features across all possible scales using a continuous function of scale known as scale space
- Convolve the input image with different Gaussian Kernel
- Subtract adjacent one from the other (DoG)





Features Extraction – SIFT (principles)

Scale-space extrema detection (Scale-space function)



$$L(x, y, \sigma) = G(x, y, \sigma) * I(x, y)$$

$$G(x, y, \sigma) = 1/(2\pi\sigma^2) \exp^{-(x^2+y^2)/\sigma^2}$$

Features Extraction – SIFT (principles)

Key point localization

- The next step is to perform a detailed fit to the nearby data for location, scale, and ratio of principal curvatures.

- This allows to reject points with low contrast (sensitive to noise) or poorly localized along an edge.

- Essentialy we remove some of the outliers.

Features Extraction – SIFT (principles)

Orientation assignment

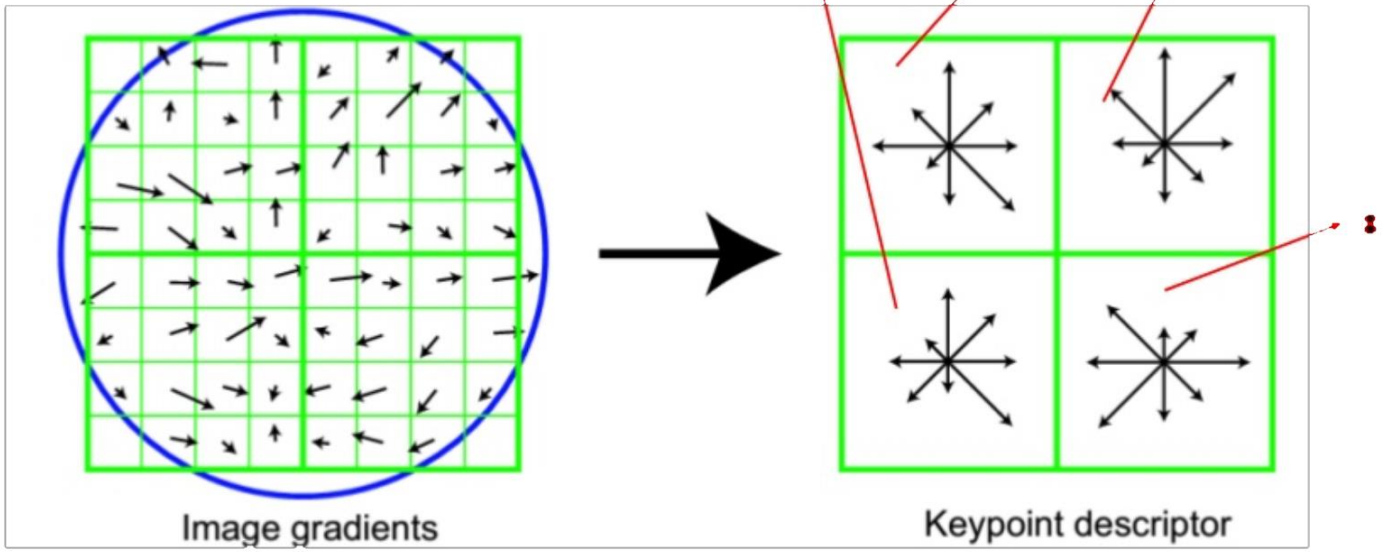
- Assign a consistent orientation to each key point based on local image properties.

- The key point descriptor can be represented relative to this orientation and therefore achieve invariance to image rotation.

Features Extraction – SIFT (principles)

Key point descriptor

- Created by the gradient magnitude and orientation at each image sample point in a region around the key point location
- Weighting by a Gaussian window follows
- 8+8+8+8 = 32 variable vectors



Features Extraction – SIFT (principles)

- As a result, each key point has three attributes:
  - Key point location (x,y)
  - Orientation direction
  - Associated 32 (or 128) variable vector

- These make each point distinct and identifiable from the others.

Other Detectors and Descriptors

Key point detector:

- Harris corner detector
- Harris-Laplace, scale and affine invariant version
- LoG
- FAST
- BRISK (descriptor included)
- ORB (descriptor included)

Key point descriptor:

- HOG
- GLOH
- LESH
- BRISK
- ORB
- FREAK

Feature (Key) Points Matching

- Brute Force
  - L1 (city block), L2, Hamming distance
- FLANN based matcher
- Ratio test
- KP_1.distance < KP_2. distance * 0.25 => KP_1 good feature

PnP – OpenCV Approach (2D ~ 3D)

- cv::Mat camera_matrix = cv::Mat::zeros(3, 3, CV_64FC1);
- double & fx = camera_matrix.ptr<double>(0)[0];
- double & fy = camera_matrix.ptr<double>(1)[1];
- double & cx = camera_matrix.ptr<double>(0)[2];
- double & cy = camera_matrix.ptr<double>(1)[2];
- camera_matrix.ptr<double>(2)[2] = 1;
- fx = camera->focal_length();
- fy = fx; // (1.0 / tan(DEG2RAD(42.185f) / 2.0)) * (camera->height() / 2.0);
- cx = camera->width() / 2.0;
- cy = camera->height() / 2.0;

$$s\ m' = A[R|t]M'$$

$$s \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}$$

PnP – OpenCV Approach (2D ~ 3D)

- cv::Mat distortion_coefficients = cv::Mat::zeros(4, 1, CV_64FC1);

- cv::Mat rvec = cv::Mat::zeros(3, 1, CV_64FC1);
- cv::Mat tvec = cv::Mat::zeros(3, 1, CV_64FC1);

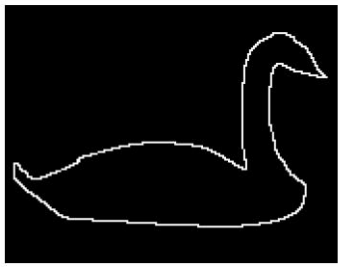
- cv::solvePnP**Ransac**(cv::Mat(good_matches_ms), cv::Mat(good_matches_is), camera_matrix, distortion_coefficients, rvec, tvec, false);

RANSAC – random sample consensus

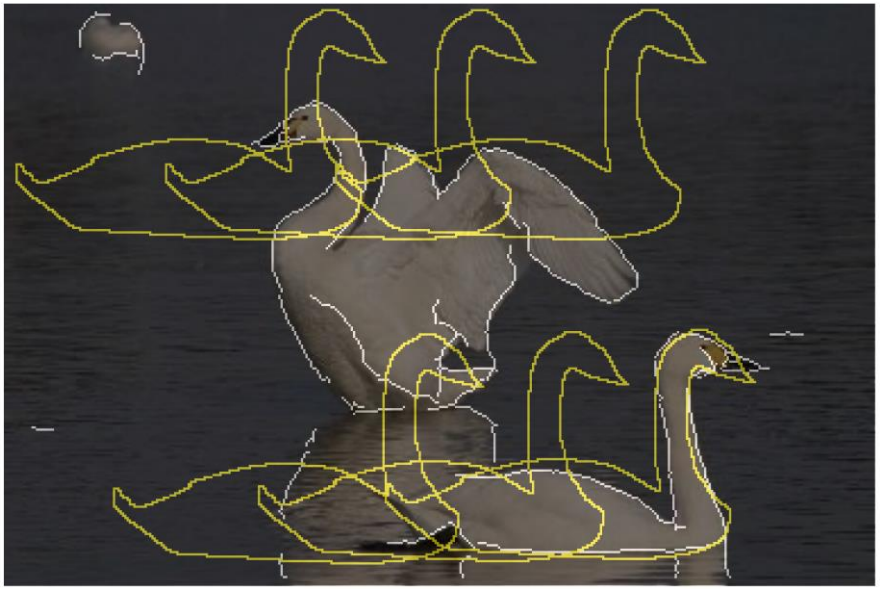
- cv::Mat R;
- cv::Rodrigues(rvec, R); // $\mathbf{v}_{\mathrm{rot}} = \mathbf{v}\cos\theta + (\mathbf{k}\times\mathbf{v})\sin\theta + \mathbf{k}(\mathbf{k}\cdot\mathbf{v})(1-\cos\theta)$.
- R = R.t();
- tvec = -R * tvec;

v is a vector in R3 and k is a unit vector describing an axis of rotation about which v rotates by an angle Θ according to the right hand rule.

Chamfer Matching



template shape



query image

$$d_{CM}(U,V) = \frac{1}{n} \sum_{\mathbf{u}_i \in U} \min_{\mathbf{v}_j \in V} |\mathbf{u}_i - \mathbf{v}_j|.$$

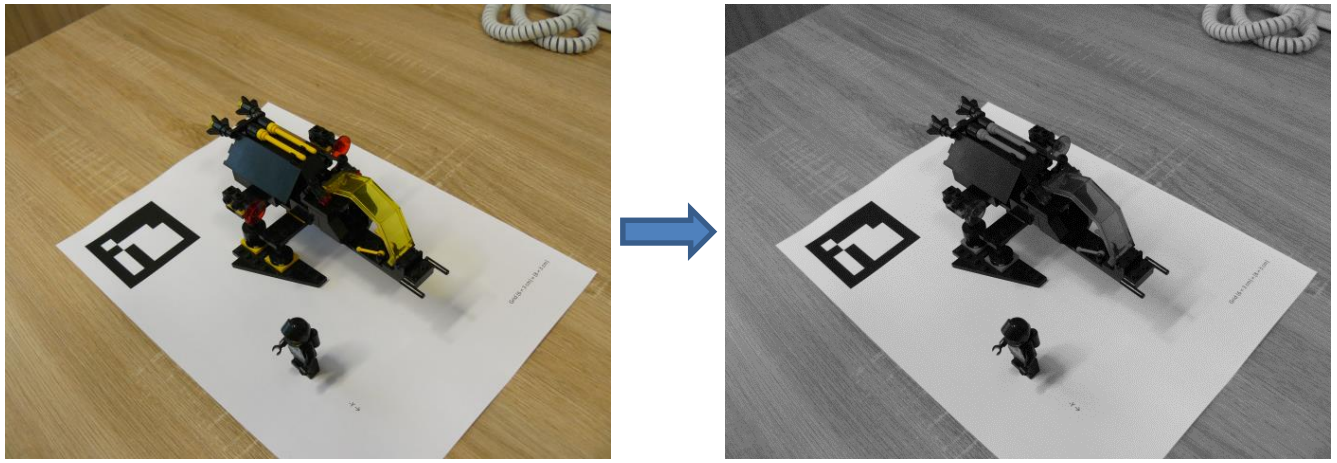
$$d_{DCM}(U,V) = \frac{1}{n} \sum_{\mathbf{u}_i \in U} \min_{\mathbf{v}_j \in V} |\mathbf{u}_i - \mathbf{v}_j| + \lambda |\phi(\mathbf{u}_i) - \phi(\mathbf{v}_j)|$$

Augmented Reality | Prerequisites

- IDE: **Visual Studio 2015** (vc 14)
- Library: **OpenCV 3.1** x64 dll
- **Image Watch** - watch window for viewing in-memory bitmaps when debugging

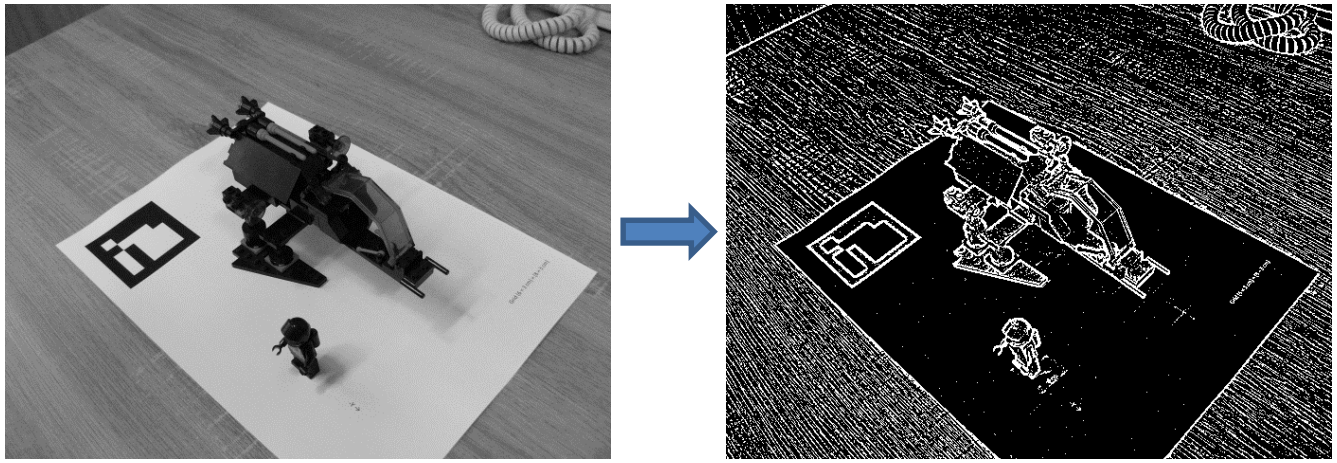
- Convert input image into grayscale

```
cvtColor( image_, image_gray_, CV_BGRA2GRAY );
```



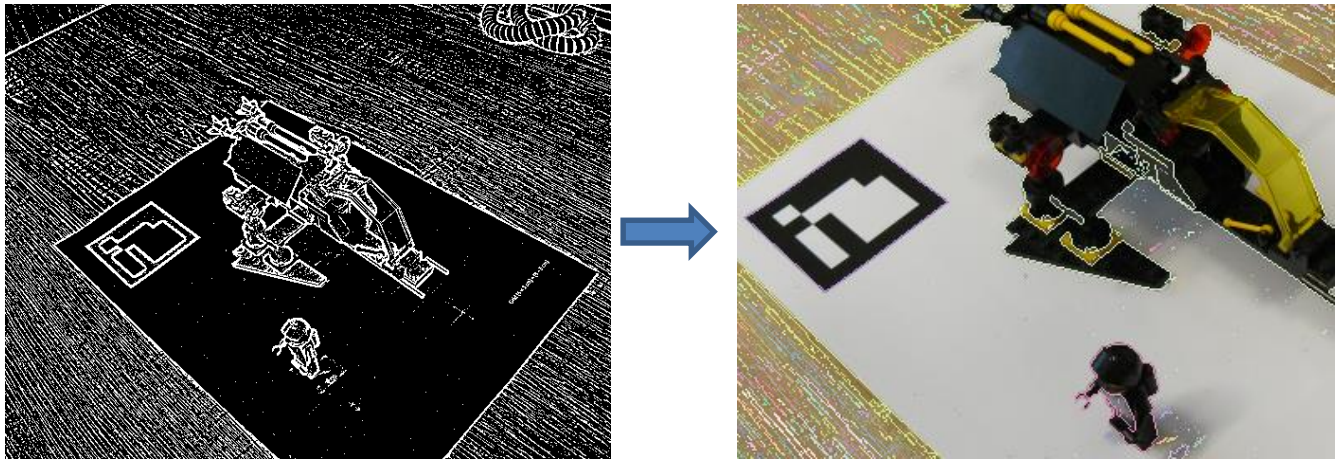
- Threshold grayscale image

```
adaptiveThreshold( image_gray_, image_thresholded, 255.0,
ADAPTIVE_THRESH_GAUSSIAN_C, THRESH_BINARY_INV, 7, 3.0 );
```



- Find contour

```
vector<vector<Point>> contours_candidates;
findContours( image_bw_.clone(), contours_candidates,
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE );
```



- Contours simplifying

For each contour with at least 4 vertices:

Approximate contour (a polygonal curve) with another polygonal curve with specified precision

```
vector<Point2f> approximated_contour;
approxPolyDP( contours_candidates[i], approximated_contour,
0.02 * arcLength( contours_candidates[i], true ), true );
```

Epsilon … the maximum distance between the original curve and its approximation.

- Contours pruning

Valid contour has:

1) at least 4 vertices (as we are looking for rectangles)

2) to have a reasonable area

```
contourArea( contour );
```

3) to be convex

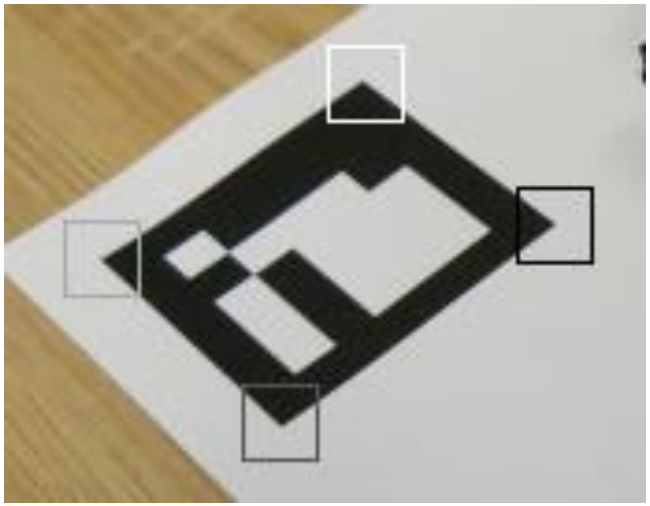
```
isContourConvex ( contour );
```

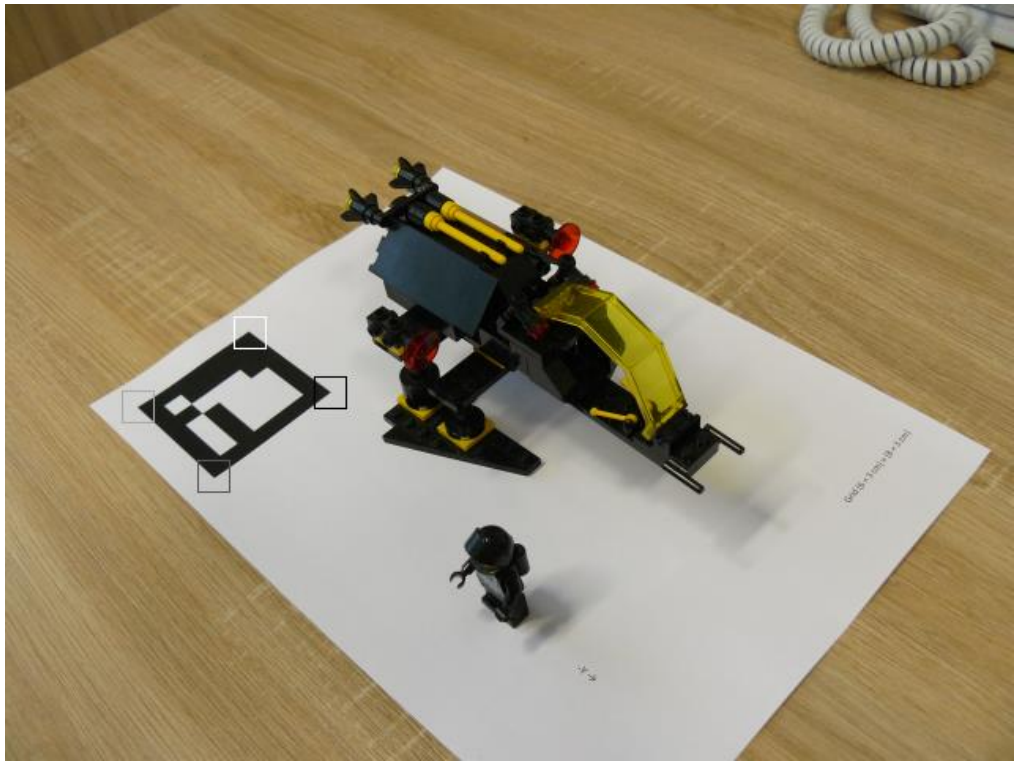
4) to be oriented in some prescribed way (cw or ccw)

```
convexHull( contour, contour_ccw, false );
```

- Contours pruning

This process should end up with a small number of contours





- Markers extraction

1) For each contour get perspective transformation

```
Mat is2ms = getPerspectiveTransform( contours_[i], corners_ );
```

```
vector<Point2f> corners_ = { Point( 0, 0 ), Point( MARKER_SIZE - 1, 0 ), Point( MARKER_SIZE - 1, MARKER_SIZE - 1 ), Point( 0, MARKER_SIZE - 1 ) };
```

2) Get unwarped image of potentional marker

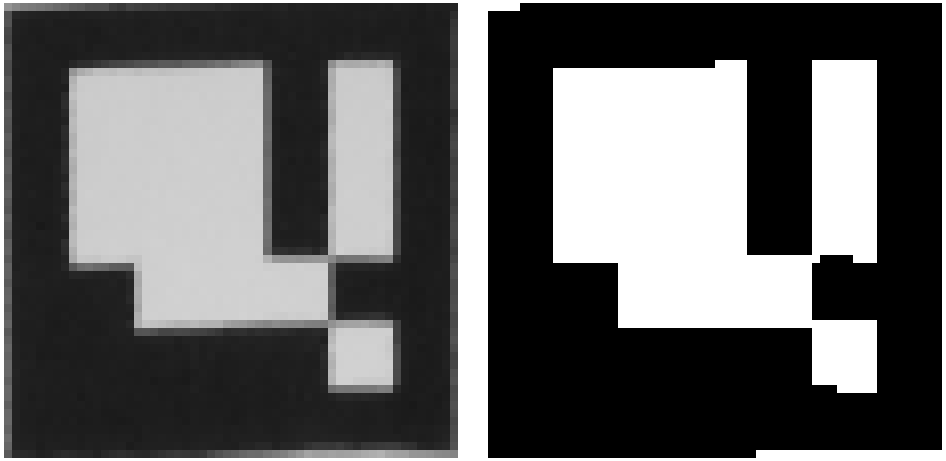
```
warpPerspective( image_gray_, marker, is2ms, Size( MARKER_SIZE, MARKER_SIZE ) );
```

3) Threshold (maximize extra-class variance and minimize intra-class variance)

```
threshold( marker, marker, 127.0, 255.0, THRESH_OTSU );
```

- Markers extraction

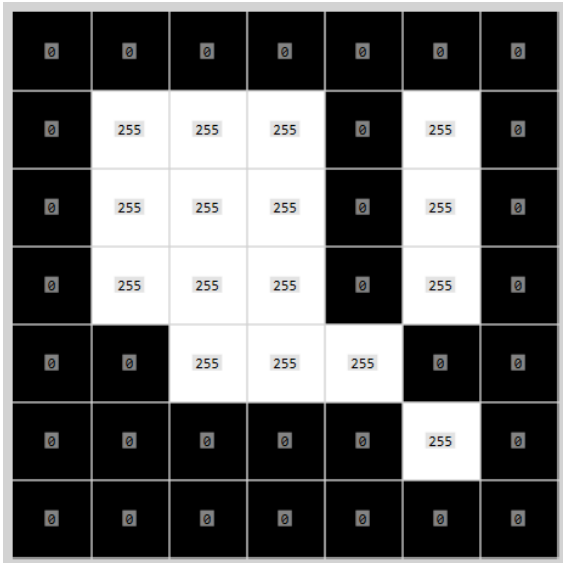
This process should end up with a list of images of individual markers



- Decode markers

1) For each potential marker get it‘s bit mask

```
resize( markers_[i].image, marker, Size( MARKER_BINS, MARKER_BINS ) );
threshold( marker, marker, 127.0, 255.0, THRESH_BINARY );
```



- Decode markers

2) Rotate bit mask until Hamming distance equals to zero

```
Mat m_rotation = getRotationMatrix2D( Point2f( (
MARKER_BINS - 1 )*0.5f, ( MARKER_BINS - 1 )*0.5f ), -
90.0, 1.0 ); // (+) ccw rotation, (-) cw rotation
warpAffine( marker, marker, m_rotation, marker.size() );
```

Note that we have to rotate (shift) contour vertices appropriately

```
rotate( contours_[i].begin(), contours_[i].end() - 1,
contours_[i].end() );
```

- Refine corners

We need to obtain the image coordinates of marker corners as precisely as possible

```
cornerSubPix( image_gray_, markers_[i].corners_is, Size( 5, 5 ),
cvSize( -1, -1 ), TermCriteria( CV_TERMCRIT_ITER, 30, 0.1 ) );
```

- Refine corners

We need to obtain the image coordinates of marker corners as precisely as possible

```
cornerSubPix( image_gray_, markers_[i].corners_is, Size( 5, 5 ),
cvSize( -1, -1 ), TermCriteria( CV_TERMCRIT_ITER, 30, 0.1 ) );
```

- Get the correspondences

We need to obtain the image coordinates of marker corners as precisely as possible

```
Mat rvec, tvec;
solvePnP( corners_ws_, marker.corners_is, m_camera,
distortion_coefficients, rvec, tvec );
Mat m_R( 3, 3, CV_64FC1 );
Rodrigues( rvec, m_R );

Mat m_T;
hconcat( m_R, tvec, m_T );
Mat m_M = m_camera * m_T;
```

- Get the correspondences

We need to obtain the image coordinates of marker corners as precisely as possible

```
Mat rvec, tvec;
solvePnP( corners_ws_, marker.corners_is, m_camera,
distortion_coefficients, rvec, tvec );
Mat m_R( 3, 3, CV_64FC1 );
Rodrigues( rvec, m_R );

Mat m_T;
hconcat( m_R, tvec, m_T );
Mat m_M = m_camera * m_T;
```

- Get the correspondences

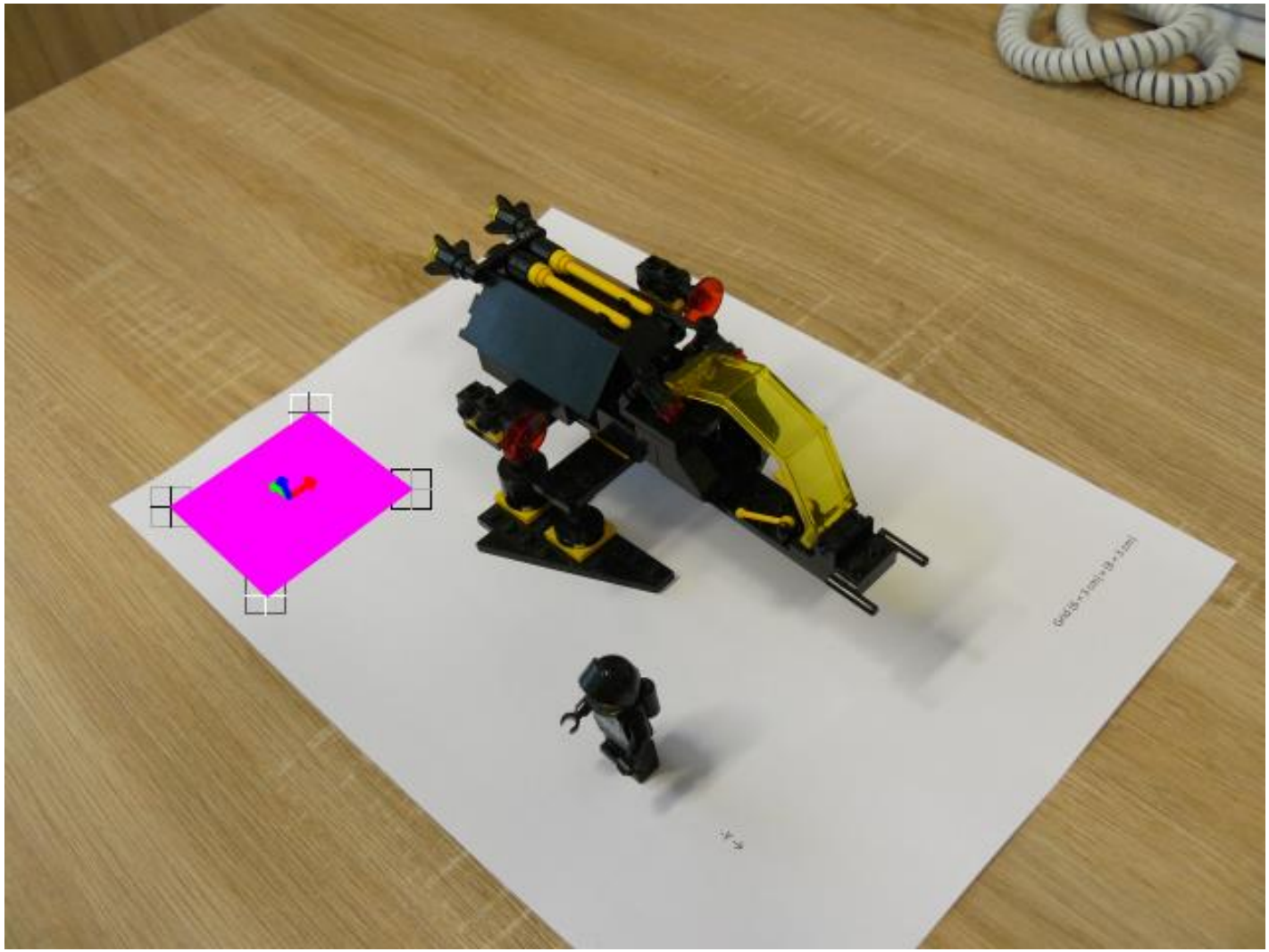
We need to obtain the image coordinates of marker corners as precisely as possible

```
Mat rvec, tvec;
solvePnP( corners_ws_, marker.corners_is, m_camera,
distortion_coefficients, rvec, tvec );
Mat m_R( 3, 3, CV_64FC1 );
Rodrigues( rvec, m_R );

Mat m_T;
hconcat( m_R, tvec, m_T );
Mat m_M = m_camera * m_T;
```

```
Mat point_is = m_M * point_ws;
point_is /= point_is.at<double>( 2, 0 );
```

Augmented Reality | Searching for Markers



- Camera matrix

```
Mat camera_matrix = Mat::zeros( 3, 3, CV_64FC1 );
double & fx = camera_matrix.ptr<double>( 0 )[0];
double & fy = camera_matrix.ptr<double>( 1 )[1];
double & cx = camera_matrix.ptr<double>( 0 )[2];
double & cy = camera_matrix.ptr<double>( 1 )[2];
camera_matrix.ptr<double>( 2 )[2] = 1.0;
const Size size = image_.size();
const double fov_y = DEG2RAD( 42.185 );
fx = ( size.height * 0.5 ) / tan( fov_y * 0.5 );
fy = fx;
cx = size.width * 0.5;
cy = size.height * 0.5;
```