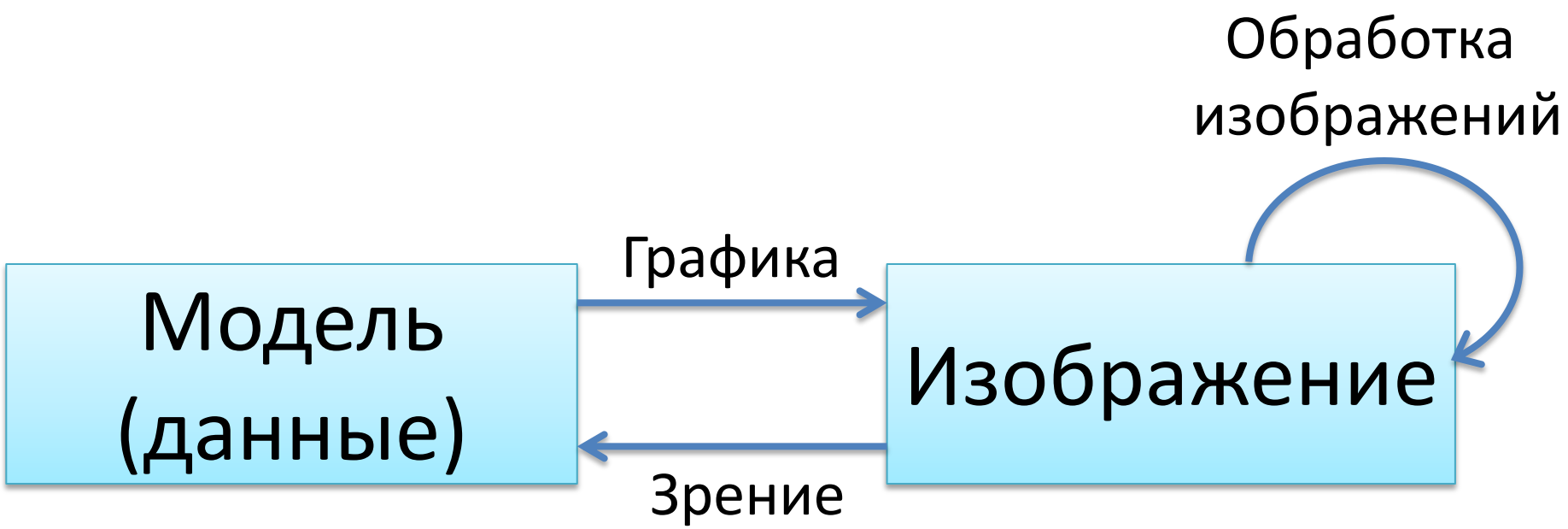
SINGULARIS LAB

2016 CEE-SEC(R)

Распознавание маркеров дополненной реальности в реальных условиях

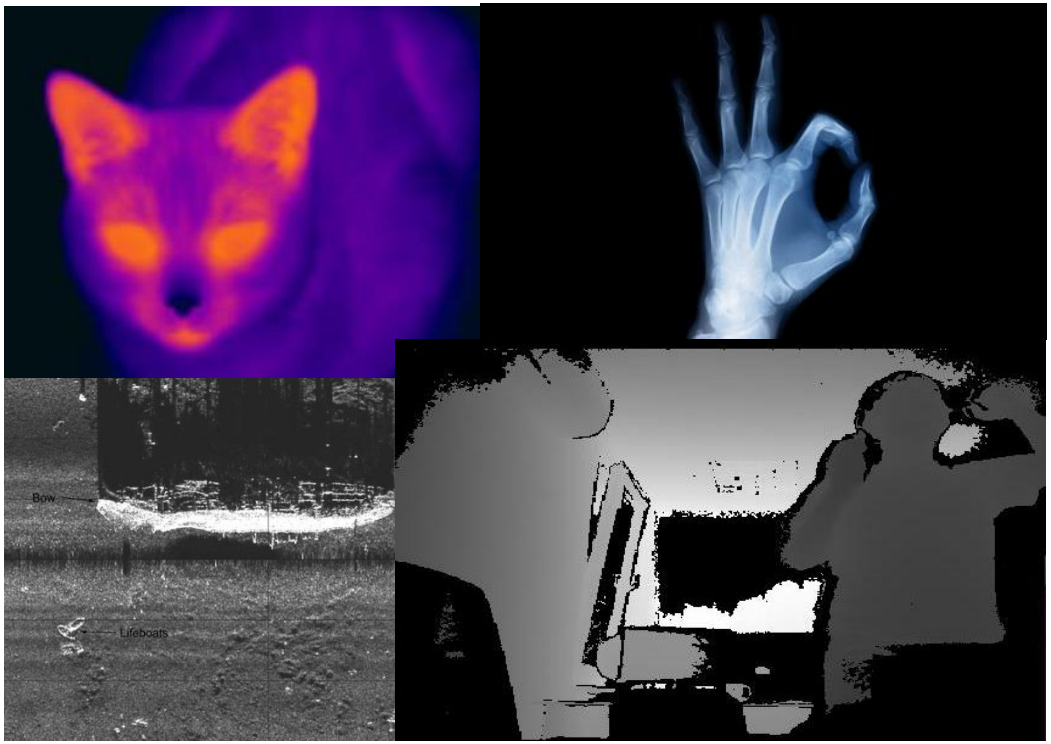
Катаев Александр
Ведущий инженер-программист, к.т.н.
Алексеев Алексей
Инженер-программист
Singularis Lab, Ltd.

Что такое компьютерное зрение



Изображения

- Видимые изображения (черно белые и цветные)
- Изображения температуры (IR)
- Изображения плотности (рентген)
- Изображения глубины (расстояния)
- …



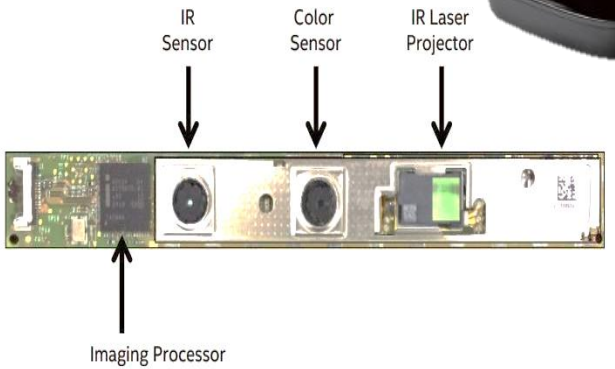
Источники изображений

Изображение глубины

- Лазерные сканеры
- Structured Light сенсоры

Изображение глубины

- Мощные SDK:
  - Отслеживание пользователя
  - Управление жестами
  - Идентификация лица
  - Распознавание голоса
  - Сканирование объектов
  - …
- Встраиваются в планшеты и ноутбук

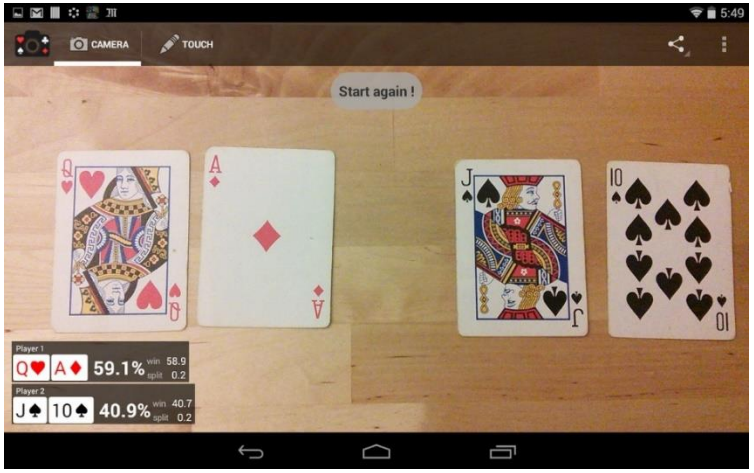


Дополненная реальность

http://www.youtube.com/watch?v=KGSa73fPCQA

http://www.youtube.com/watch?v=gH9CbrCw6vQ

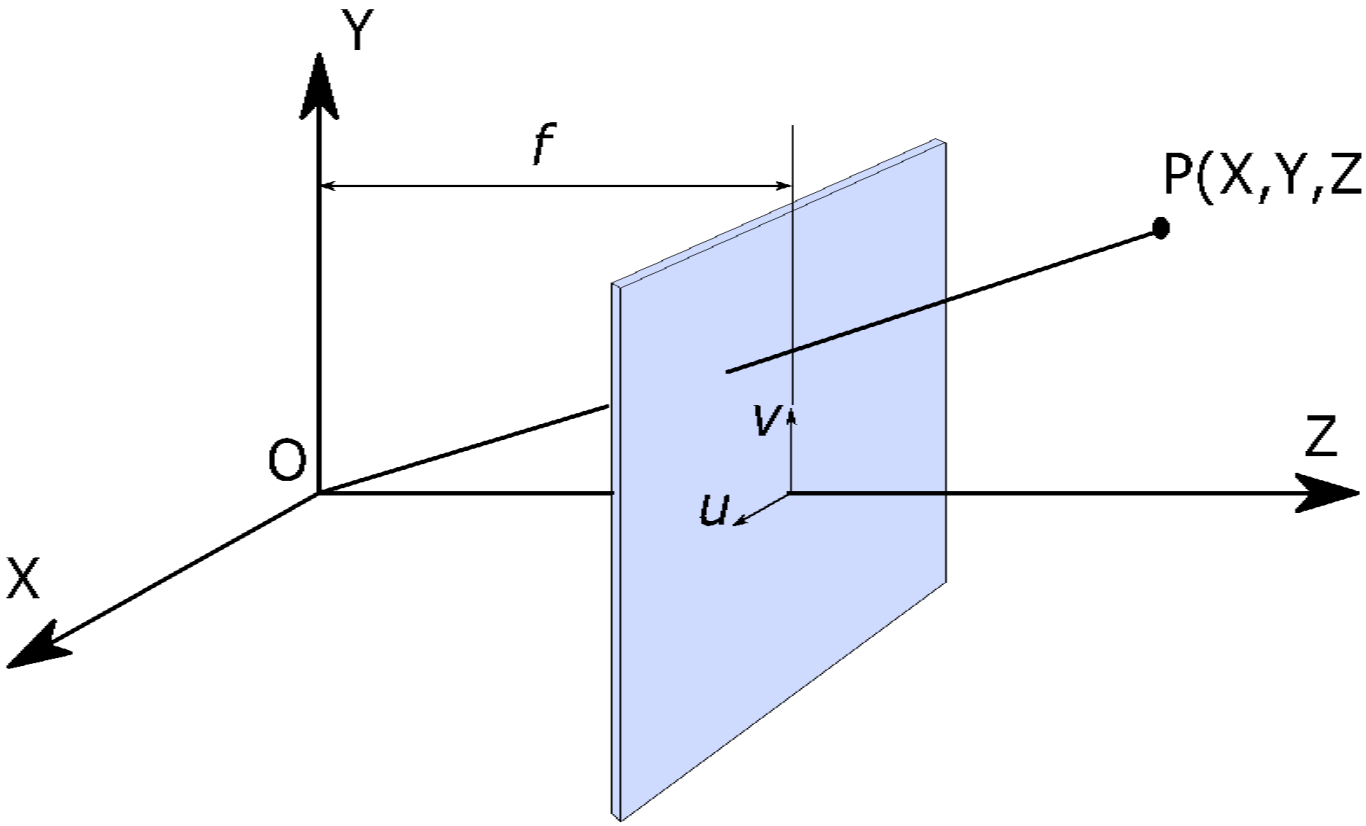
Дополненная реальность



http://www.youtube.com/watch?v=iEzIgc1GJjU

Дополненная реальность

Пинхол модель камеры

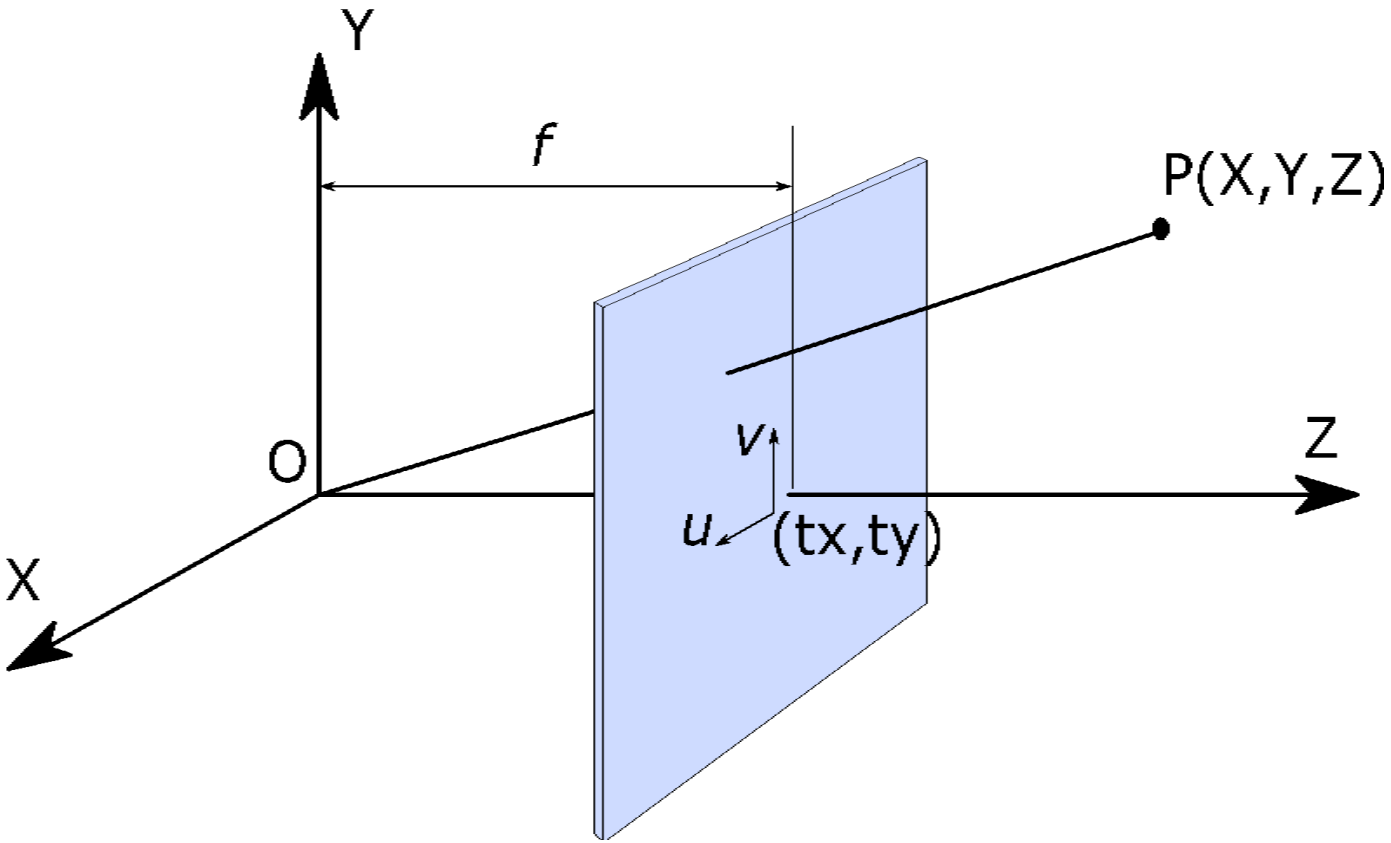


$$\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y}$$

$$u = \frac{fX}{Z}$$

$$v = \frac{fY}{Z}$$

Пинхол модель камеры



$$\frac{f}{Z} = \frac{u}{X} = \frac{v}{Y}$$

$$u = \frac{fX}{Z} + t_x$$

$$v = \frac{fY}{Z} + t_y$$

Пинхол модель камеры

$$u = m_u\left(\frac{fX}{Z} + t_x\right) = \frac{f_x X}{Z} + c_x$$

$$v = m_v\left(\frac{fY}{Z} + t_y\right) = \frac{f_y Y}{Z} + c_y$$

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = KP$$

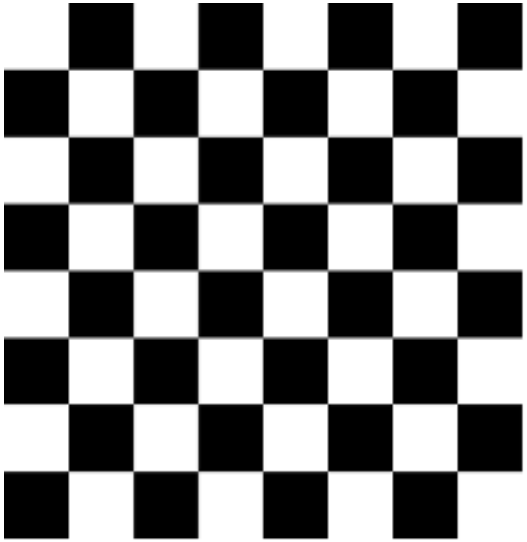
K=$\begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$ - матрица камеры

3d → 2d

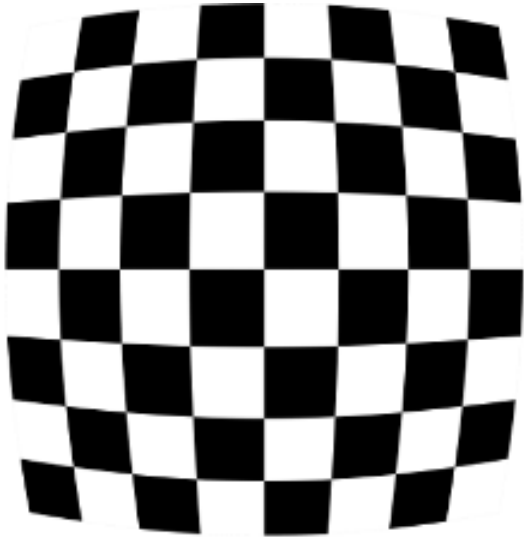
$$s\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

$\begin{pmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{pmatrix}$- матрица трансформации, включающая поворот R и перенос T

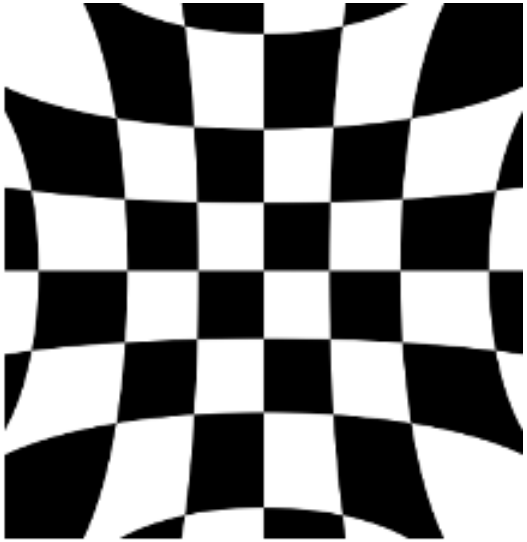
Дисторсия



No distortion



Positive radial distortion
(Barrel distortion)



Negative radial distortion
(Pincushion distortion)

Open Source Computer Vision Library

Модель Дисторсии OpenCV

$$x' = \frac{X}{Z}, y' = \frac{Y}{Z}$$

$$x'' = x' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + 2p_1 x' y' + p_2(r^2 + 2x'^2)$$

$$y'' = y' \frac{1 + k_1 r^2 + k_2 r^4 + k_3 r^6}{1 + k_4 r^2 + k_5 r^4 + k_6 r^6} + p_{1(r^2 + 2y'^2)} + 2p_2 x' y'$$

$$r^2 = x'^2 + y'^2$$

$$u = f_x * x'' + c_x$$

$$v = f_y * y'' + c_y$$

$$D = (k_1, k_2, p_1, p_2[, k_3[, k_4, k_5, k_6]])$$

Параметры камеры

- Внутренние (intrinsic)

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

$$D = \left(k_1, k_2, p_1, p_2\left[, k_3\left[, k_4, k_5, k_6\right]\right]\right)$$

- Внешние (extrinsic)

  Матрица поворота R

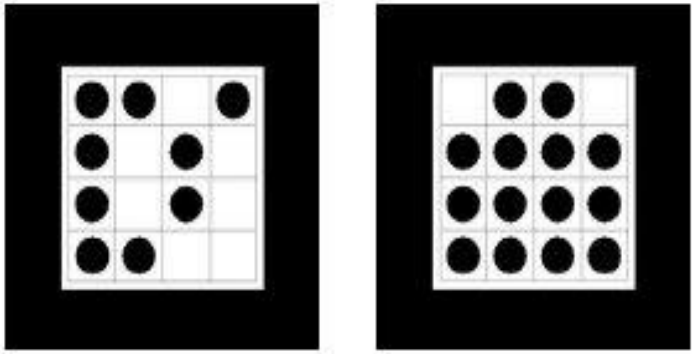
  Вектор переноса системы координат T

Специальные маркеры

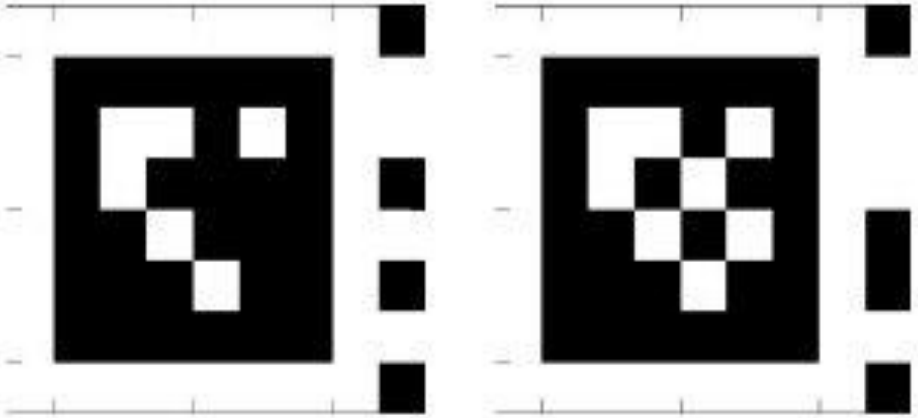
ArToolKit(ATK)

Institut Graphische Datenverarbeitung (IGD)

Siemens Corporate Research (SCR)

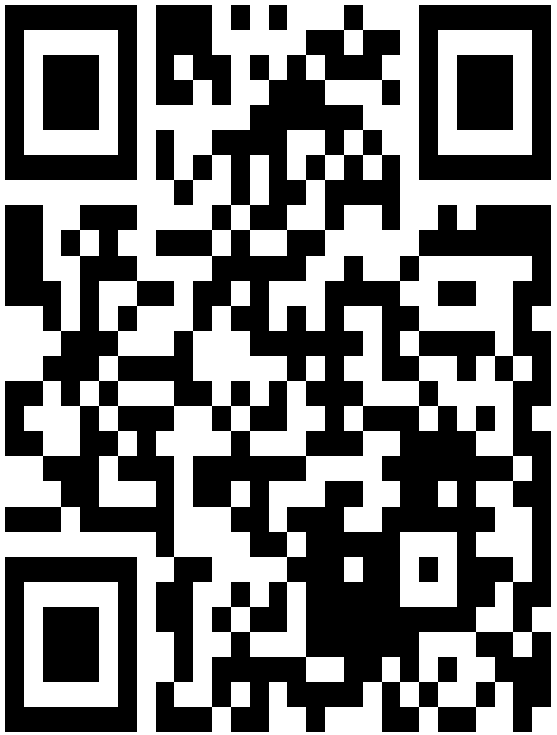


Hoffman marker system (HOM)

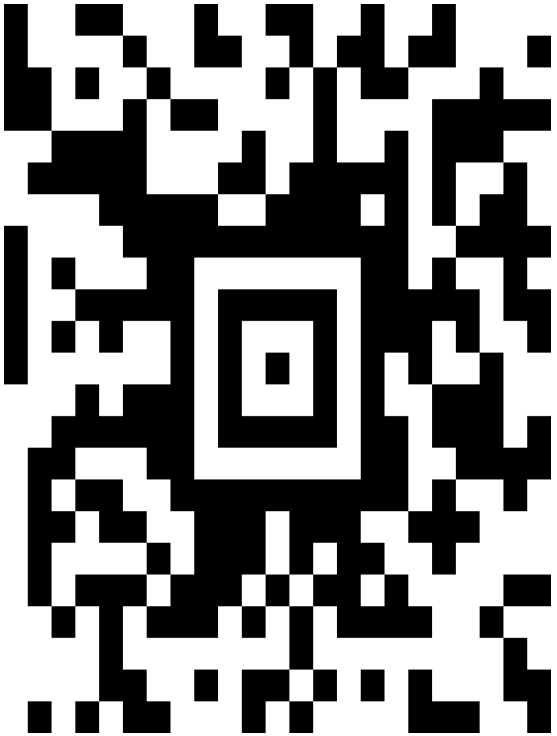


Специальные маркеры

QR code



Aztec Code



Дополненная реальность с OpenCV

Нам понадобятся:

- Python 2/3
- OpenCV, numpy, numpy-stl пакеты для python
- Веб-камера
- Заготовленные маркеры
- Монохромный медведь

Исходники

https://github.com/Kwentar/SECR

Первая программа

```
import cv2

img = cv2.imread('test.jpg')
cv2.imshow('Image', img)
cv2.waitKey()
```

План

1 Калибровка камеры

1.1 Создание выборки для калибровки

1.2 Калибровка

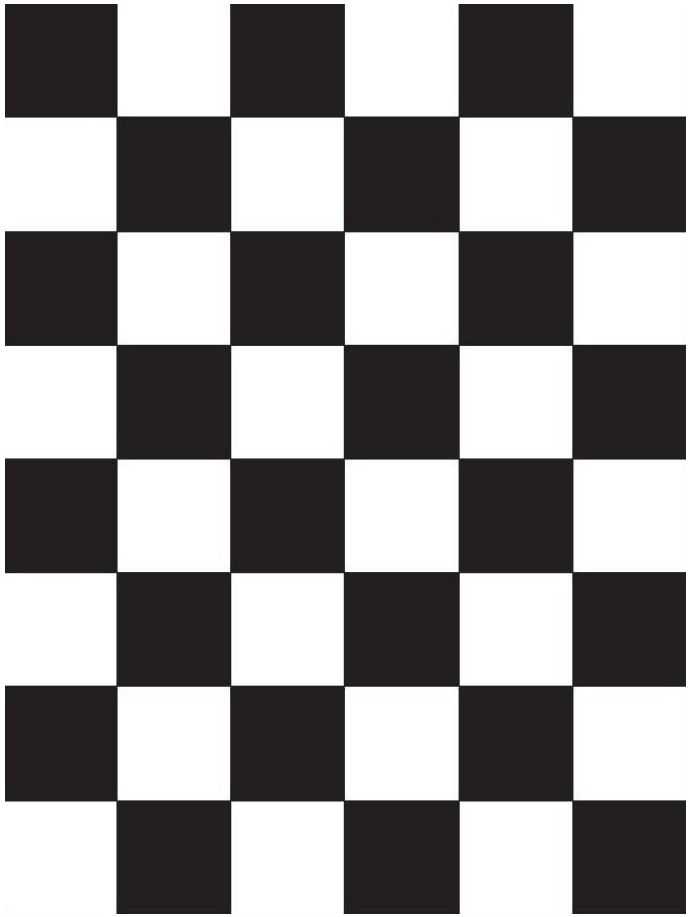
2 Детектирование модели и вывод 3D объектов

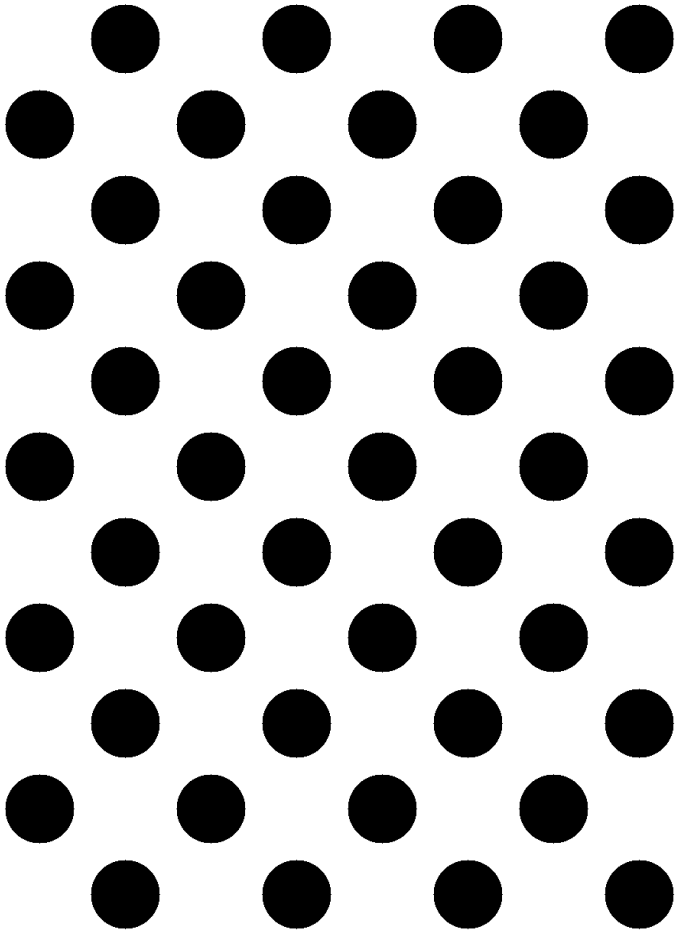
2.1 Вывод осей

2.2 Вывод куба

2.3 Вывод STL-модели

Калибровочные шаблоны





Качество калибровки

Требуется:

- Плотная бумага, твердое основание
- Качественный принтер (нелинейность протяжки!)
- Множество снимков (10-30) под разными углами, на разных расстояниях
- Фиксированная оптическая система*

Создание выборки для калибровки. Получение кадров с камеры

```
import cv2

cap = cv2.VideoCapture()  # объект камеры
cap.open(0)  # «открываем» камеру
while True:
    ret, frame = cap.read()  # считываем кадр
    if ret:
        cv2.imshow('frame', frame) # показываем
        c = cv2.waitKey(1)
        if c & 0xFF == ord('q'): # выход по ‘q’
            break
```

Создание выборки для калибровки. Поиск шахматки

```
cv2.imshow('frame', frame) # показываем
chessboard_size = (7,9)
criteria = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
ret, corners = cv2.findChessboardCorners(gray,
chessboard_size, None)
c = cv2.waitKey(1)
```

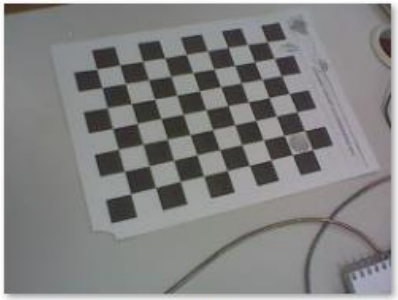
Создание выборки для калибровки. Вывод найденной шахматки

```
ret, corners = cv2.findChessboardCorners(gray,
chessboard_size, None)
if ret:
  cv2.cornerSubPix(gray, corners, (11, 11),
(3,3), criteria)
  cv2.drawChessboardCorners(gray,
chessboard_size, corners, ret)
  cv2.imshow('chess', gray)
  c = cv2.waitKey(1)
```

Создание выборки для калибровки. Сохранение снимков

```
import os
...
index = 0
while True:
    ret, frame = cap.read()
...
if c & 0xFF == ord('q'):
    break
elif c & 0xFF == ord(' '):
    photos_dir='images/'
    cv2.imwrite(os.path.join(photos_dir,
'{}.png'.format(index)), frame)
    index += 1
```

Создание выборки для калибровки. Пример выборки


0.png


1.png


2.png


5.png


6.png


7.png


10.png


11.png


12.png

Процесс калибровки

`obj_points` - координаты точек объекта в 3D (модельные координаты

`img_points` - координаты точек объекта на изображении (проекции)

`imageSize` - размер изображения в пикселях

cv2.calibrateCamera

`rms,` - ошибка репроецирования (хорошо, если < 1)

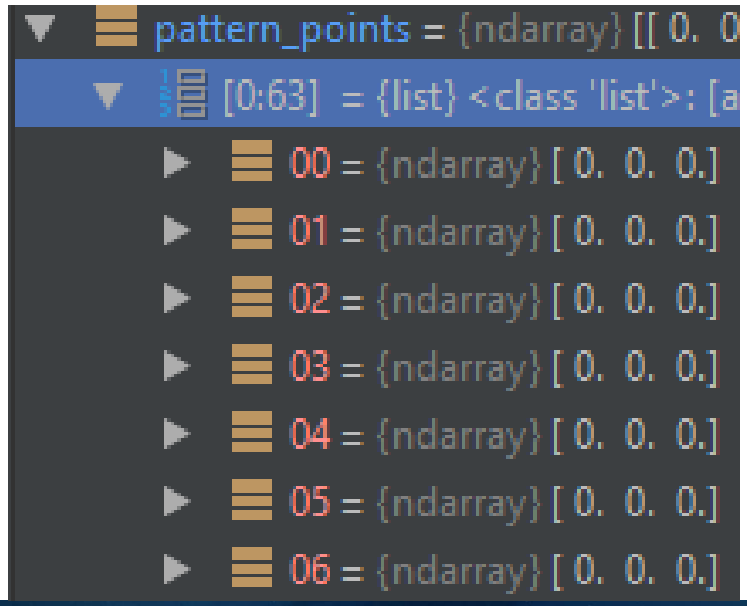
`camera_matrix,` - матрица камеры K

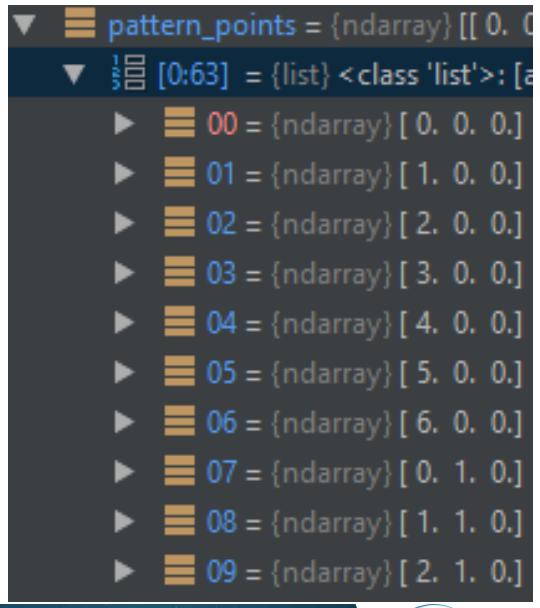
`dist_coefs,` - коэфициенты дисторсии D

`rvecs, tvecs` - внешние параметры

Калибровка камеры

```
chessboard_size = (7,9)
pattern_points =
np.zeros((np.prod(chessboard_size), 3), np.float32)
pattern_points[:, :2] =
np.indices(chessboard_size).T.reshape(-1, 2)
```





Калибровка камеры. Чтение снимков

```
obj_points = []  # 3d точки
img_points = []  # 2d точки
images = [os.path.join(photos_dir, file_) for
file_ in os.listdir(photos_dir)]
h, w = 0, 0
for file_name in images:
    img = cv2.imread(file_name, 0)
    if img is None:
        print("Failed to load", file_name)
        continue
```

Калибровка камеры. Чтение снимков

```
obj_points = []  # 3d точки
img_points = []  # 2d точки
images = [os.path.join(photos_dir, file_) for
file_ in os.listdir(photos_dir)]
h, w = 0, 0
for file_name in images:
    img = cv2.imread(file_name, 0)
    if img is None:
        print("Failed to load", file_name)
        continue
```

Калибровка камеры. Поиск шахматки

```
if img is None:
        print("Failed to load", file_name)
        continue
h, w = img.shape[:2]
found, corners = cv2.findChessboardCorners(img,
chessboard_size)
if found:
    term = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_COUNT, 30, 0.1)
    cv2.cornerSubPix(img, corners, (5, 5),(3,3),
term)
    img_points.append(corners.reshape(-1, 2))
    obj_points.append(pattern_points)
```

Калибровка камеры. Поиск шахматки

```
if img is None:
        print("Failed to load", file_name)
        continue
h, w = img.shape[:2]
found, corners = cv2.findChessboardCorners(img,
chessboard_size)
if found:
    term = (cv2.TERM_CRITERIA_EPS +
cv2.TERM_CRITERIA_COUNT, 30, 0.1)
    cv2.cornerSubPix(img, corners, (5, 5),
(3,3), term)
    img_points.append(corners.reshape(-1, 2))
    obj_points.append(pattern_points)
```

Калибровка камеры

```
for file_name in images:
    ...
rms, camera_matrix, dist_coefs, rvecs, tvecs =
cv2.calibrateCamera(obj_points, img_points, (w,
h), None, None)
```

	0	1	2
0	750.18744	0.00000	353.08649
1	0.00000	726.17982	214.22557
2	0.00000	0.00000	1.00000

camera_matrix

	0	1	2	3	4
0	0.38135	-1.57555	-0.00852	0.00686	2.18726

dist_coef

Сохранение параметров

```
result_filename = 'test.npz'
np.savez(result_filename,
camera_matrix=camera_matrix,
dist_coefs=dist_coefs)
```

Детектирование модели и вывод объектов

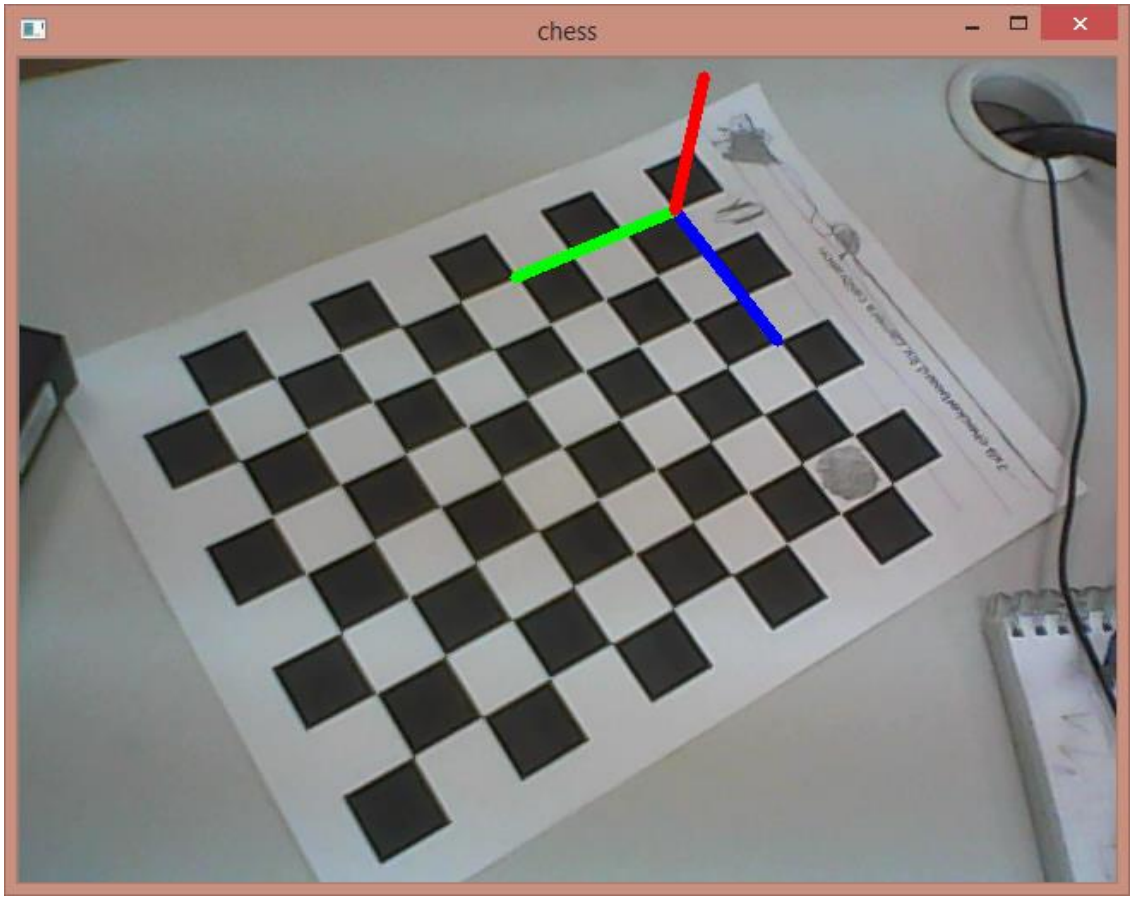
Проблема Perspective-n-Point:

По набору 3D координат точек (в мировой системе координат) и набору их 2D проекций найти параметры камеры R, T (и *f*)

Методы в OpenCV:

solvePnP, solvePnPRansac

Вывод в 3D осей координат



Загрузка внутренних параметров

```
with np.load('test.npz') as X:
    camera_matrix = X['camera_matrix']
    dist_coefs = X['dist_coefs']
    pattern_points = np.zeros((np.prod(chessboard_size),
3), np.float32)
    pattern_points[:, :2] =
np.indices(chessboard_size).T.reshape(-1, 2)
    axis = np.float32([[0, 0, 0], [3, 0, 0], [0, 3, 0],
[0, 0, -3]]).reshape(-1, 3)
    cap = cv2.VideoCapture()
    cap.open(0)
    while True:
        ret, frame = cap.read()
        ...
    cap.release()
```

Построение набора 3D точек (модели)

```
with np.load('test.npz') as X:
    camera_matrix = X['camera_matrix']
    dist_coefs = X['dist_coefs']
    pattern_points = np.zeros((np.prod(chessboard_size),
3), np.float32)
    pattern_points[:, :2] =
np.indices(chessboard_size).T.reshape(-1, 2)
    axis = np.float32([[0, 0, 0], [3, 0, 0], [0, 3, 0],
[0, 0, -3]]).reshape(-1, 3)
    cap = cv2.VideoCapture()
    cap.open(0)
    while True:
        ret, frame = cap.read()
        ...
    cap.release()
```

Вывод в 3D осей координат

```
with np.load('test.npz') as X:
    camera_matrix = X['camera_matrix']
    dist_coefs = X['dist_coefs']
    pattern_points = np.zeros((np.prod(chessboard_size),
3), np.float32)
    pattern_points[:, :2] =
np.indices(chessboard_size).T.reshape(-1, 2)
    axis = np.float32([[0, 0, 0], [3, 0, 0], [0, 3, 0],
[0, 0, -3]]).reshape(-1, 3)
    cap = cv2.VideoCapture()
    cap.open(0)
    while True:
        ret, frame = cap.read()
        ...
    cap.release()
```

Поиск модели

```
ret, frame = cap.read()
if ret:
    cv2.imshow('frame', frame)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    found, corners = cv2.findChessboardCorners(gray,
chessboard_size, None)
    if found:
        cv2.cornerSubPix(gray, corners, (11, 11), (3, 3),
criteria)
        ...
        cv2.imshow('chess', frame)
    c = cv2.waitKey(1)
    if c & 0xFF == ord('q'):
        break
```

PnP

```
cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
ret, rvecs, tvecs = cv2.solvePnP(pattern_points, corners, camera_matrix, dist_coefs)
img_pts, jac = cv2.projectPoints(axis, rvecs, tvecs, camera_matrix, dist_coefs)
draw_lines(frame, img_pts)

cv2.imshow('chess', frame)
```

Вывод в 3D осей координат

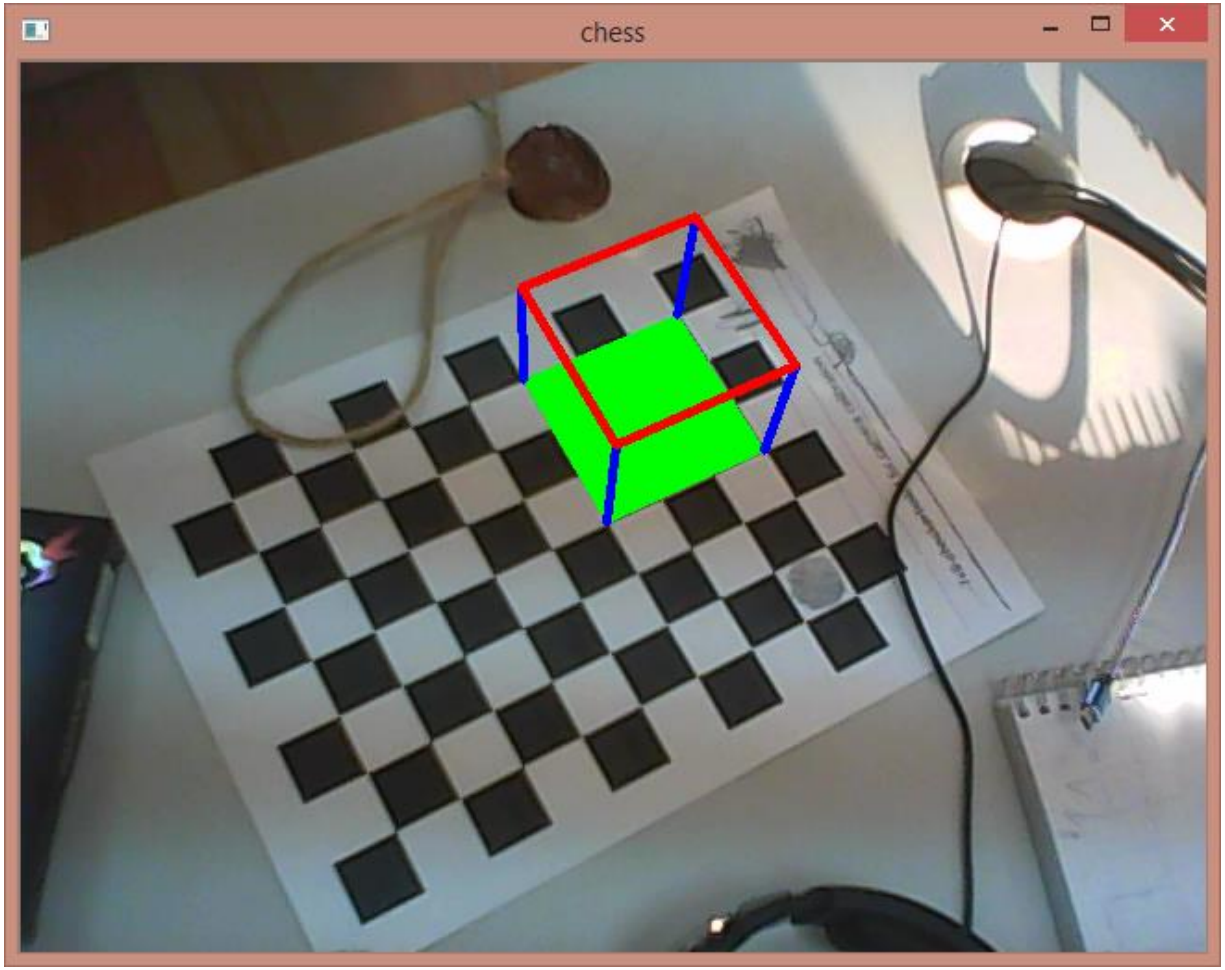
```
cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)
ret, rvecs, tvecs = cv2.solvePnP(pattern_points, corners, camera_matrix, dist_coefs)
img_pts, jac = cv2.projectPoints(axis, rvecs, tvecs, camera_matrix, dist_coefs)
draw_lines(frame, img_pts)

cv2.imshow('chess', frame)
```

Вывод в 3D осей координат

```
def draw_lines(img,  img_pts):
    for i, color in zip(range(1, 4), [(0, 0, 0), (0, 255, 0), (255, 0, 0)]):
        cv2.line(img, tuple(img_pts[0].ravel()), tuple(img_pts[i].ravel()), color, 5)
```

ravel: [[0,1,2]] -> [0,1,2]

Вывод в 3D Куба



Вывод в 3D Куба

```
axis = np.float32([[0, 0, 0], [3, 0, 0], [0, 3, 0], [0, 0, -3]]).reshape(-1, 3)
cube_points = np.float32([[0, 0, 0], [0, 3, 0], [3, 3, 0], [3, 0, 0], [0, 0, -3], [0, 3, -3], [3, 3, -3], [3, 0, -3]])

def draw_cube_model(img, img_pts):
    img_pts = np.int32(img_pts).reshape(-1, 2)
    cv2.drawContours(img, [img_pts[:4]],-1, (0,255,0),-3)
    for i, j in zip(range(4), range(4, 8)):
        cv2.line(img, tuple(img_pts[i]), tuple(img_pts[j]), 255, 3)
        cv2.drawContours(img, [img_pts[4:]], -1, (0, 0, 255), 3)
```

Вывод 3D STL модели

Открытие модели:

```
from stl import mesh
your_mesh = mesh.Mesh.from_file('Moon.stl')
```

Чтение треугольников:

```
for vector in your_mesh.vectors:
    img_pts, jac = cv2.projectPoints(vector, rvecs, tvecs,
camera_matrix, dist_coefs)
    draw_triangle(frame, imgpts)
```

Рисование:

```
def draw_triangle(img, points):
    points = np.int32(points).reshape(-1, 2)
    cv2.drawContours(img,[points],-1,(0,255,0), -3)
    cv2.line(img, tuple(points[0]),tuple(points[1]),255,1)
    cv2.line(img, tuple(points[1]),tuple(points[2]),255,1)
    cv2.line(img, tuple(points[2]),tuple(points[0]),255,1)
```

Вывод 3D STL модели

Открытие модели:

```
from stl import mesh
your_mesh = mesh.Mesh.from_file('Moon.stl')
```

Чтение треугольников:

```
for vector in your_mesh.vectors:
    img_pts, jac = cv2.projectPoints(vector, rvecs, tvecs,
camera_matrix, dist_coefs)
    draw_triangle(frame, img_pts)
```

Рисование:

```
def draw_triangle(img, points):
    points = np.int32(points).reshape(-1, 2)
    cv2.drawContours(img,[points],-1,(0,255,0), -3)
    cv2.line(img, tuple(points[0]),tuple(points[1]),255,1)
    cv2.line(img, tuple(points[1]),tuple(points[2]),255,1)
    cv2.line(img, tuple(points[2]),tuple(points[0]),255,1)
```

Вывод 3D STL модели

Открытие модели:

```
from stl import mesh
your_mesh = mesh.Mesh.from_file('Moon.stl')
```

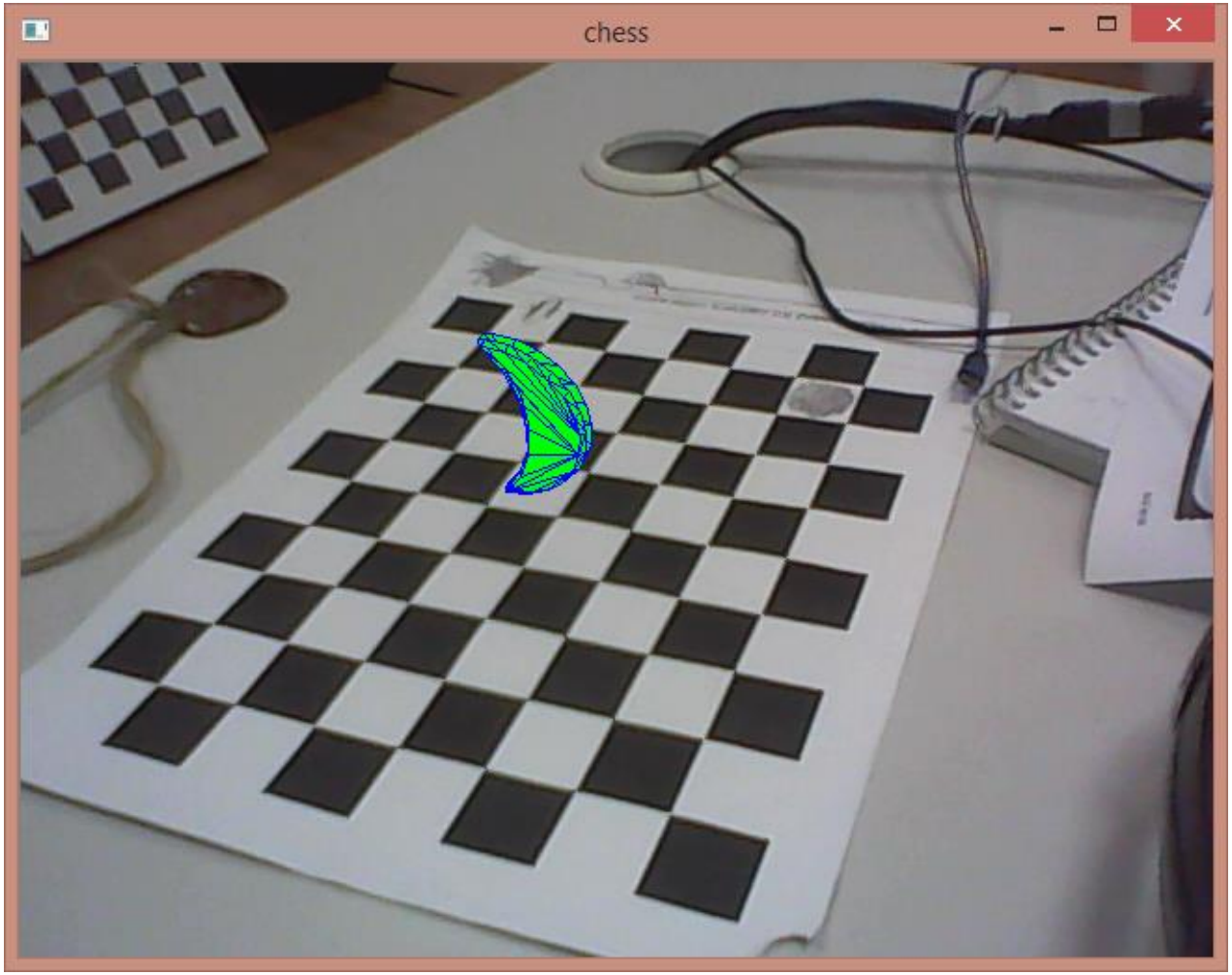
Чтение треугольников:

```
for vector in your_mesh.vectors:
    img_pts, jac = cv2.projectPoints(vector, rvecs, tvecs,
camera_matrix, dist_coefs)
    draw_triangle(frame, imgpts)
```

Рисование:

```
def draw_triangle(img, points):
    points = np.int32(points).reshape(-1, 2)
    cv2.drawContours(img,[points],-1,(0,255,0), -3)
    cv2.line(img, tuple(points[0]),tuple(points[1]),255,1)
    cv2.line(img, tuple(points[1]),tuple(points[2]),255,1)
    cv2.line(img, tuple(points[2]),tuple(points[0]),255,1)
```

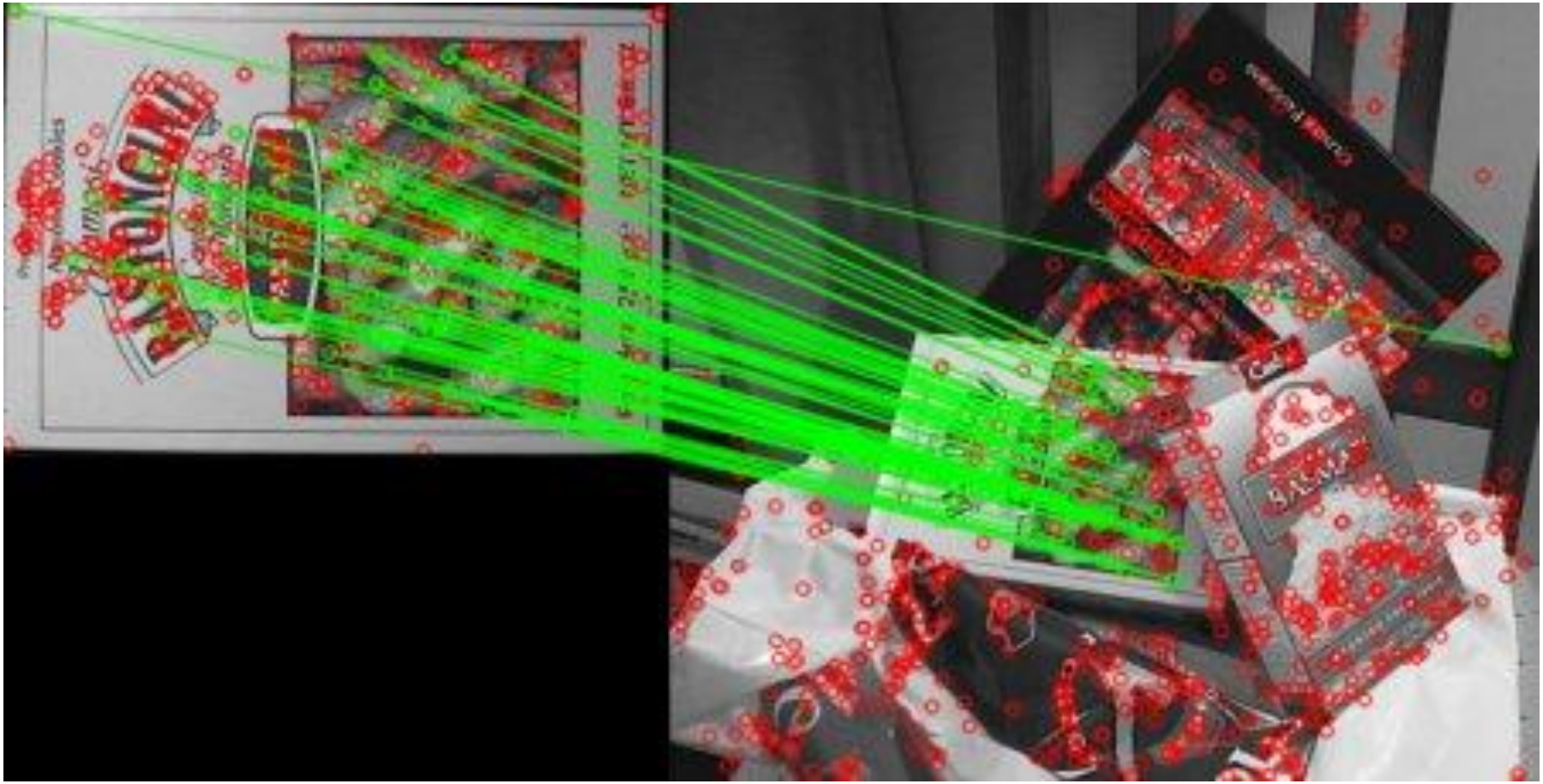
Вывод 3D STL модели



Детектирование более сложных моделей

- Детектирование маркеров
  - Специальные методы, эффективно работающие со специальными маркерами. Быстро, качественно, в прошлом году.
- Детектирование произвольного объекта
  - Универсальные методы поиска объектов, например, по ключевым точкам. Медленнее, интереснее.

Сопоставление ключевых точек



Методы сопоставления

- **BruteForce** – полный перебор ($O(N^2)$)
- Fast Approximate Nearest Neighbor Search Library (**FLANN**) – быстро, но приблизительно
- **FLANN + постобработка**

Поиск точек и вычисление дескрипторов

```
def getDes(image, nFeatures=1500):
    orb = cv2.ORB(nFeatures)
    kp = orb.detect(image,None)
    if kp is None:
        return [],[]
    return orb.compute(image, kp)
```

Сопоставление точек (BF)

```
def getMatches(des_marker, des_image):
    matcher = cv2.BruteForceMatcher()
    return matcher.match(des_image,des_marker)
```

FLANN + постобработка

```
def getMatches(des_marker, des_image):
    FLANN_INDEX_LSH = 6
    index_params = dict(algorithm = FLANN_INDEX_LSH, table_number = 6, key_size = 12,
multi_probe_level = 1)
matcher = cv2.FlannBasedMatcher(index_params,{})

    # находим по 2 ближайших дескриптора для каждой точки
    # два раза: маркер к картинке и обратно (они будут разными, т.к. FLANN)
    matches1to2 = matcher.knnMatch(des_image,des_marker,k=2)
    matches2to1 = matcher.knnMatch(des_marker,des_image,k=2)

    #ratio test
    …

    #symmetry test
    good = …
    return good
```

Постобработка (фильтрация соответствий)

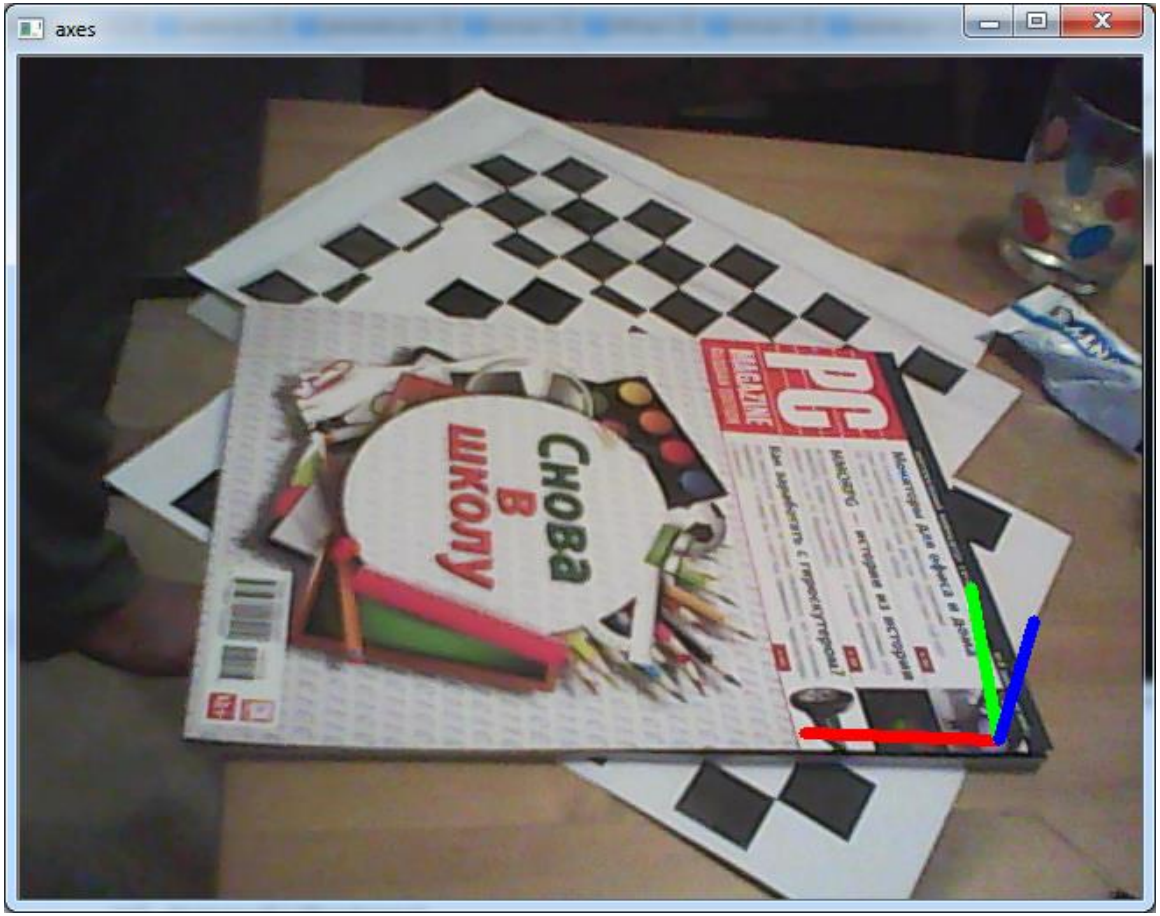
```
# выкидываем точки с менее чем 2 соответствиями
matches1to2 = [x for x in matches1to2 if len(x) == 2]
matches2to1 = [x for x in matches2to1 if len(x) == 2]

#ratio test – выкидываем такие, в которых не очень уверены
ratio = 0.8
good1to2 = [m for m,n in matches1to2 if m.distance < ratio *n.distance]
good2to1 = list([m for m,n in matches2to1 if m.distance < ratio *n.distance])

#symmetry test – выкидываем несимметричные соответствия
good = []
for m in good1to2:
   for n in good2to1:
      if m.queryIdx == n.trainIdx and n.queryIdx == m.trainIdx:
         good.append(m)

 return good
```

Сопоставление точек

PnP Ransac

```
kp_marker, des_marker = getDes(marker)
kp_image, des_image = getDes(frame)

matches = getMatches(des_marker, des_image)

# формирование массивов с координатами
pattern_points = [kp_marker[pt.trainIdx].pt for pt in matches]
pattern_points = np.array([(x/50.0,y/50.0,0) for x,y in pattern_points], dtype=np.float32)
image_points = np.array([kp_image[pt.queryIdx].pt for pt in matches], dtype=np.float32)

# PnP RANSAC
rvecs, tvecs, inliers = cv2.solvePnPRansac(pattern_points, image_points, camera_matrix,
dist_coefs)
```

Результат

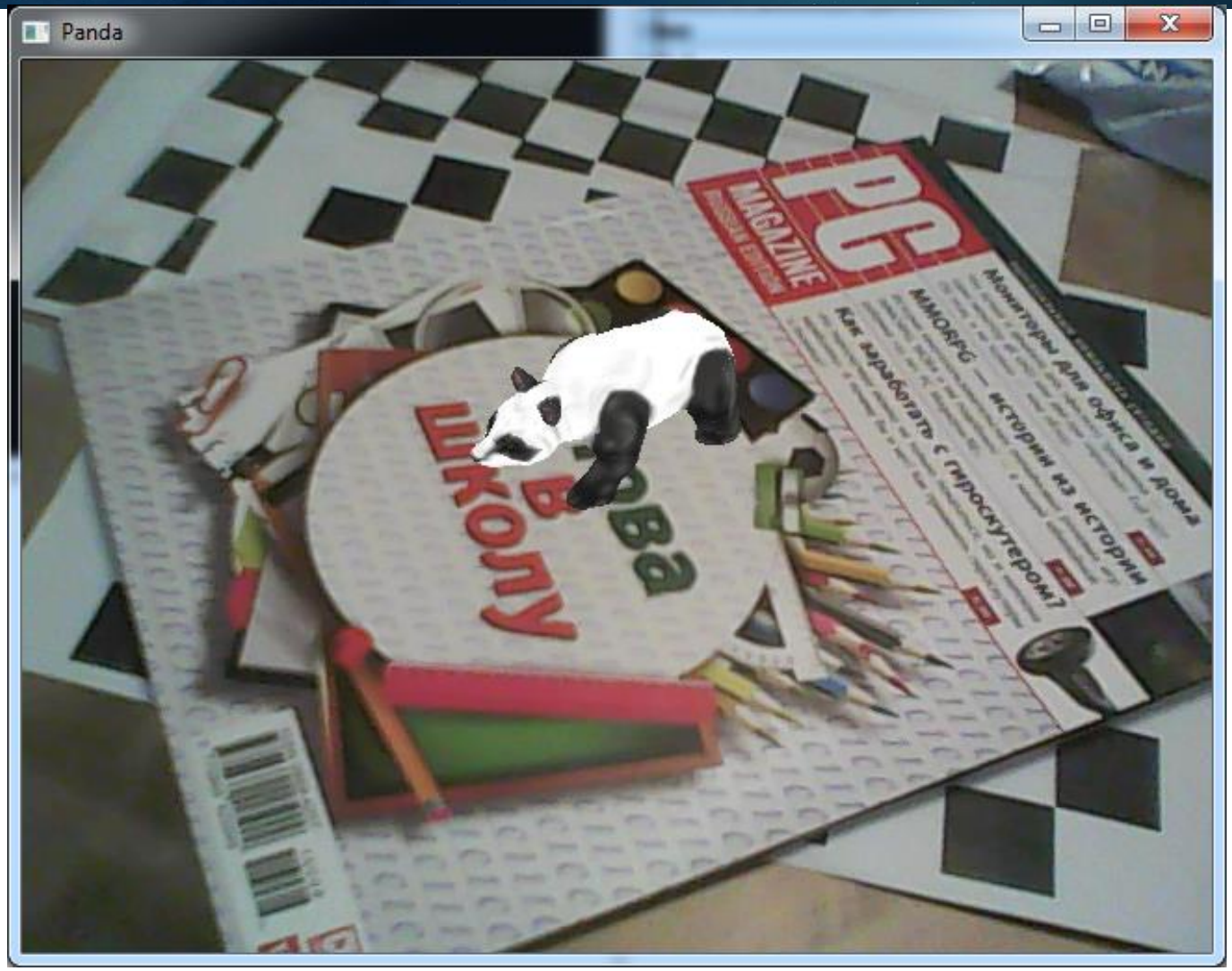


Вывод 3д модели

- Установить камеру
  - fovx, fovy, f, (cx, cy), a = cv2.calibrationMatrixValues(self.K, (w,h), w, h)
- Сформировать на основе rvec и tvec трансформацию

```
T = tvecs.ravel(), R = rvecs.ravel()
RotM,_ = cv2.Rodrigues(R)
RotM = RotM.T
RotM = Mat3(RotM[0,0],RotM[0,1],RotM[0,2],
            RotM[1,0],RotM[1,1],RotM[1,2],
           -RotM[2,0],-RotM[2,1],-RotM[2,2])
self.modelroot.setMat(Mat4(RotM, Vec3(T[0],T[1],T[2])))
```

Монохромный медведь



Спасибо за внимание

Александр Катаев

- alexander.kataev@singularis-lab.com

Алексей Алексеев

- aleksey.alekseev@singularis-lab.com

https://www.singularis-lab.com/

https://www.linkedin.com/company/singularis-lab-llc

http://habrahabr.ru/company/singularis

http://vk.com/singularis_lab