CTU
CZECH TECHNICAL UNIVERSITY IN PRAGUE

Robotics: Introduction to perception

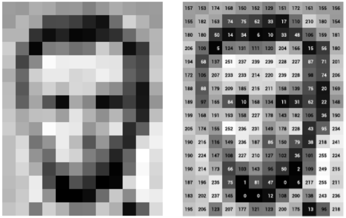
Vladimír Petrík

vladimir.petrik@cvut.cz

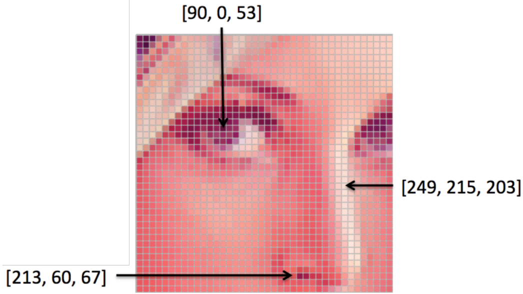
23.10.2023

What is image?

- Camera connected to computer produces images
- Image is array of numbers[1]



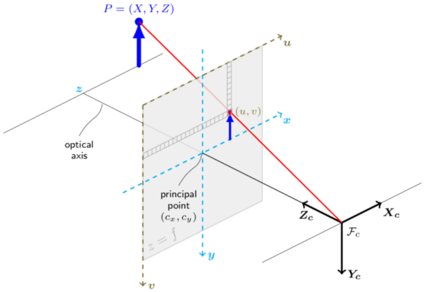
157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

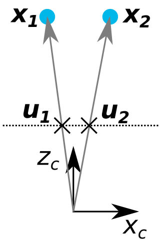


[1] Images are from: `https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html`

How is the image formed?

- Perspective camera
  - pinhole camera model[2]
  - projects spatial point $\boldsymbol{x}_c$ into image point $\boldsymbol{u} = \begin{pmatrix} u & v \end{pmatrix}^\top$ by intersecting
    - image plane and
    - the line connecting $\boldsymbol{x}_c$ with the projection center
  - all points on a ray project to the same pixel





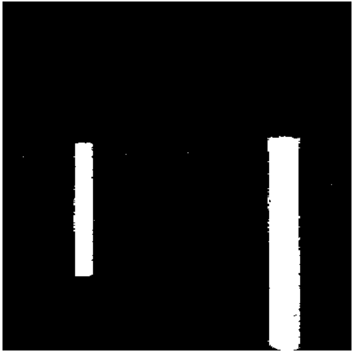
[2] docs.opencv.org

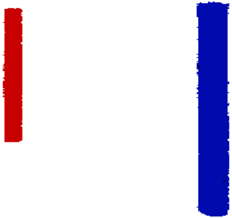
Projection of pinhole camera

- $\boldsymbol{u}_H = K\boldsymbol{x}_c$
  - $\boldsymbol{u}_H$ is pixel in homogeneous coordinates
  - if $\boldsymbol{u}_H = \begin{pmatrix} u_H & v_H & w_H \end{pmatrix}^\top$, then pixel coordinates are $\begin{pmatrix} u_H/w_H & v_H/w_H \end{pmatrix}^\top$
  - alternatively, we can represent it as: $\lambda \left(u, v, 1\right)^\top = K\boldsymbol{x}_c$
- K is camera matrix
  - $K = \begin{pmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{pmatrix}$
  - what does λ represent?
    - λ is non-zero real number
    - if you know λ value, you can compute Cartesian coordinate $\boldsymbol{x} = \lambda K^{-1}\boldsymbol{u}$
    - otherwise, only ray is computable
  - how to find K from points?

What we can study on images?

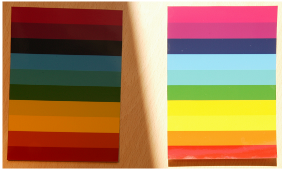
- Segmentation masks (where are the objects of interest)
- Objects classification (labeling)

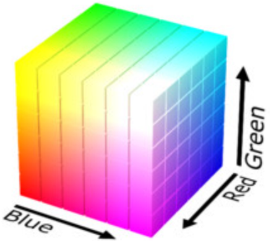


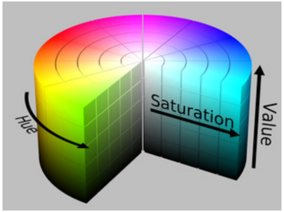


Segmentation masks - color thresholding

- Thresholding
  - RGB pixel values for coordinates $\boldsymbol{u}$: $I_{\text{RGB}}(\boldsymbol{u})$
  - $M(\boldsymbol{u}) = 1$, if $I_{\text{RGB}}(\boldsymbol{u}) = \begin{pmatrix} 0 & 255 & 0 \end{pmatrix}^\top$?
  - $M(\boldsymbol{u}) = 1$, if $\boldsymbol{\tau}_l < I_{\text{RGB}}(\boldsymbol{u}) < \boldsymbol{\tau}_u$, for all channels
  - $M(\boldsymbol{u}) = 1$, if $\boldsymbol{\varphi}_l < I_{\text{HSV}}(\boldsymbol{u}) < \boldsymbol{\varphi}_u$, for all channels
- Post-processing
  - compute connected components
  - remove small or deformed segments
  - assign label based on thresholds



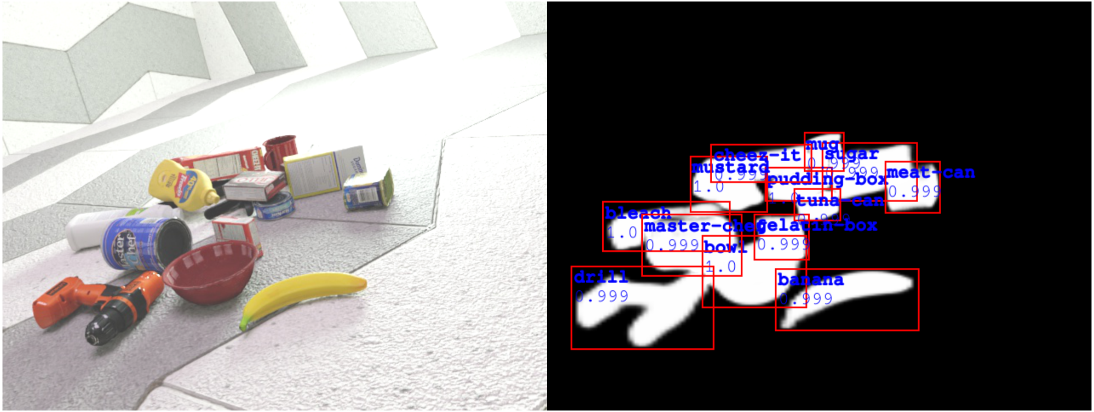




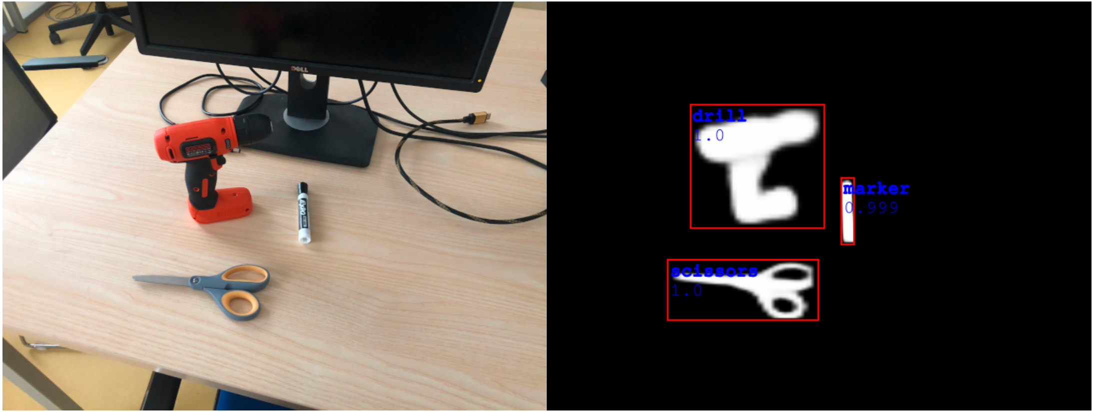
Segmentation masks for known 3D objects

- Neural Network (*e.g.* Mask R-CNN)
- Training inputs:
  - dataset of images, masks and labels, or
  - dataset of known 3D objects (meshes)
  - quality depends on the training data (augumentations)
- Inference:
  - Input: image
  - Output: segmentation mask, bounding box, label, and confidence

Mask R-CNN results

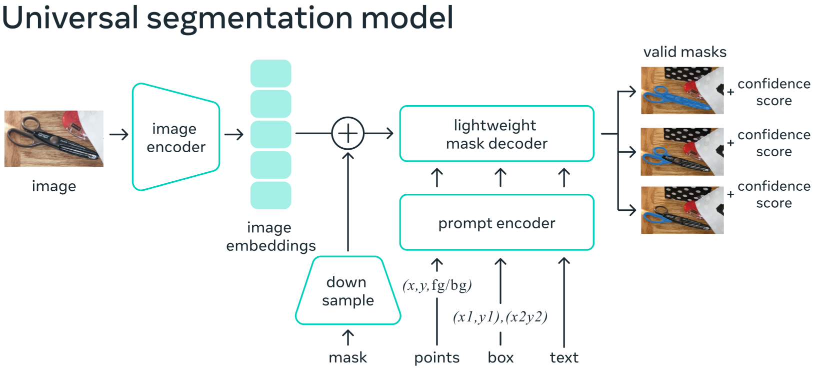


Mask R-CNN results

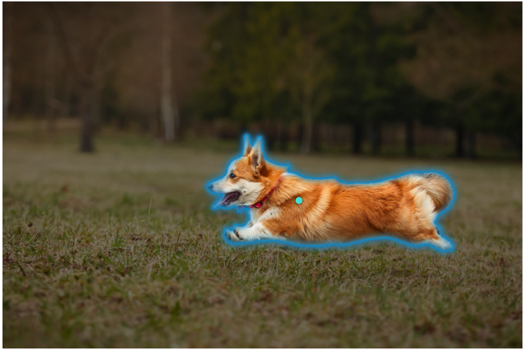


Segmentation masks without re-training

- Segment Anything Model (SAM)
  - segment any object, in any image, with a single click
  - dataset of 10M images, 1B masks



SAM results



SAM results

Segmentation

- Segmentation finds objects in image
  - segmentation mask
  - bounding box
  - label
  - confidence score
- Information only in image space
- How to use it in robot space?

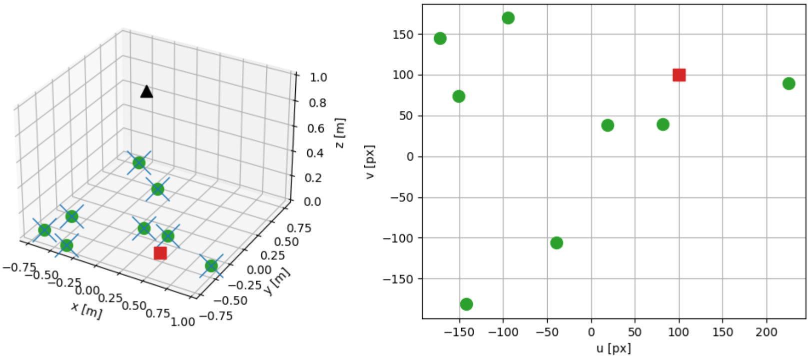
External camera

- Assume camera mounted rigidly to the reference frame
  - if we know K and T_{RC}, how to project points $\boldsymbol{x}_R$ to image?
- Unknown K and T_{RC} and planar problem
  - *e.g.* cubes with the same high on table desk
  - what is the position of cube on 2D table w.r.t. 2D image/pixels coordinates?
  - analyzed by **homography**

Homography

- Homography matrix H is 3×3 matrix that maps points from one plane to another
  - image plane to table desk
  - one image plane to another image plane (different view)
- $s \begin{pmatrix} x & y & 1 \end{pmatrix}^\top = H \begin{pmatrix} u & v & 1 \end{pmatrix}^\top$
  - x, y are coordinates in the first plane
  - u, v are coordinates in the second plane
- 9 elements but only 8 DoF, usually added constraint $h_{33} = 1$
- How to find H?
  - `H, _ = cv2.findHomography(U, X)`
  - U, X are $N \times 2$ correspondence points
  - *e.g.* measure manually
    - position of cube center w.r.t. table corner
    - position of cube center in image

Homography example

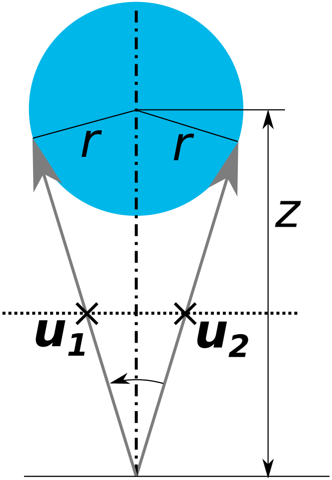


Non-planar pose estimation

- Homography maps only plane to plane
- More general object pose estimation in **camera** frame
  - get depth by mapping from area in pixels to depth for fixed size objects
  - get depth by additional scene information, *e.g.* known size/model of the objects
  - RGBD camera
  - additional markers

Using prior knowledge about size

- We know radius is fixed
- From detected pixels $\boldsymbol{u}_1, \boldsymbol{u}_2$, we can compute rays $\boldsymbol{x}_1, \boldsymbol{x}_2$: $\frac{1}{\lambda_i}\boldsymbol{x}_i = K^{-1}\boldsymbol{u}_i$
- Angle between vectors: $\cos\alpha = \frac{\frac{1}{\lambda_1\lambda_2}}{\frac{1}{\lambda_1\lambda_2}} \frac{\boldsymbol{x}_1 \cdot \boldsymbol{x}_2}{\|\boldsymbol{x}_1\|\|\boldsymbol{x}_2\|}$
- Depth: $z = \frac{r}{\sin(\alpha/2)}$

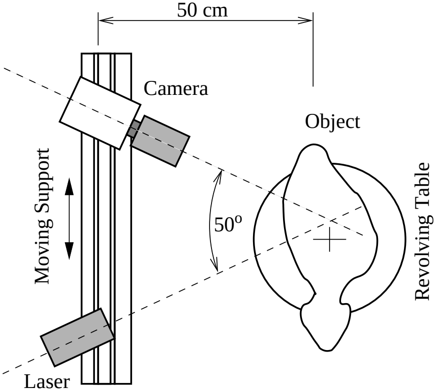


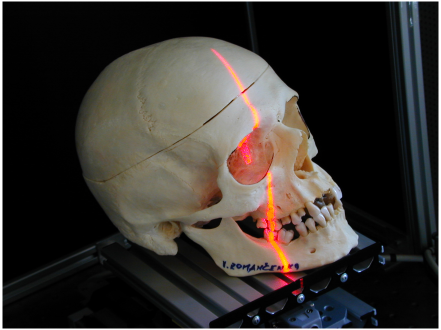
Using depth sensor

- RGB**D** sensors
  - RGB image $(H \times W \times 3)$
  - Depth map $(H \times W \times 1)$, distance in meters for each pixel
  - Structured point cloud $(H \times W \times 3)$, $\begin{pmatrix} x_c & y_c & z_c \end{pmatrix}$ for each pixel

How depth sensor works

- Laser projects pattern and camera recognizes it
- Depth information is computed using triangulation



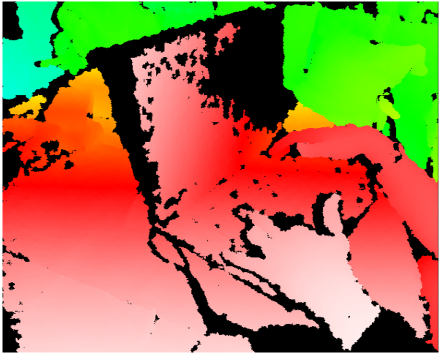


2D depth sensors

- Based on the structured light
- Projects 2D infra red patterns
- One projector and two cameras (RGB + IR)

Issues with depth sensors

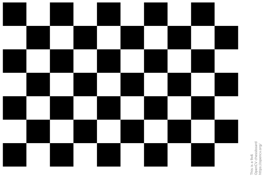
- Depth reconstruction is not perfect (black areas in the image[3])
- In python represented by NaN
- Not every pixel in RGB has reconstructed depth value
- RGB and Depth data are not aligned (you need to calibrate them)

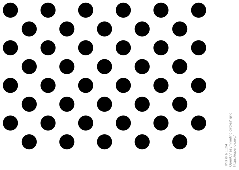



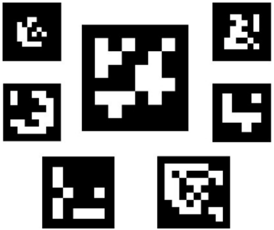
[3] https://commons.wikimedia.org, User:Kolossos

Additional markers

- Can we compute the pose of patterns[4]?
  - the size and structure needs to be known
  - subpixel accuracy
  - it has to be completely visible
- Can we compute the pose of ArUco markers?
  - less accurate than regular patterns
  - provides marker id and the pose
  - it has to be completely visible

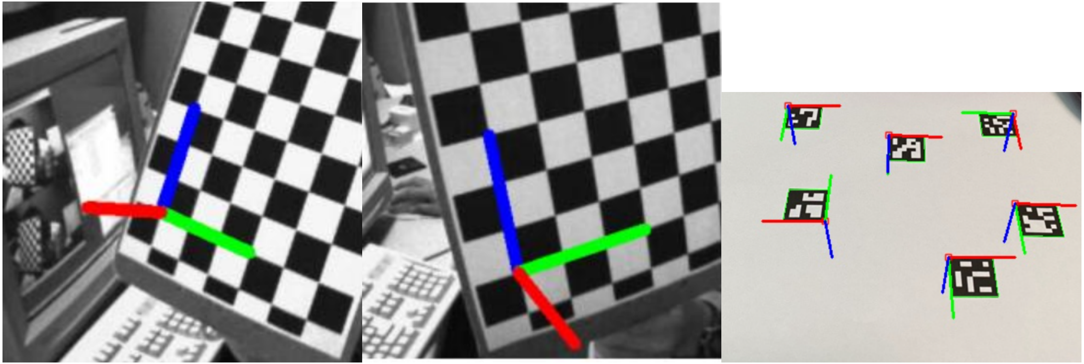






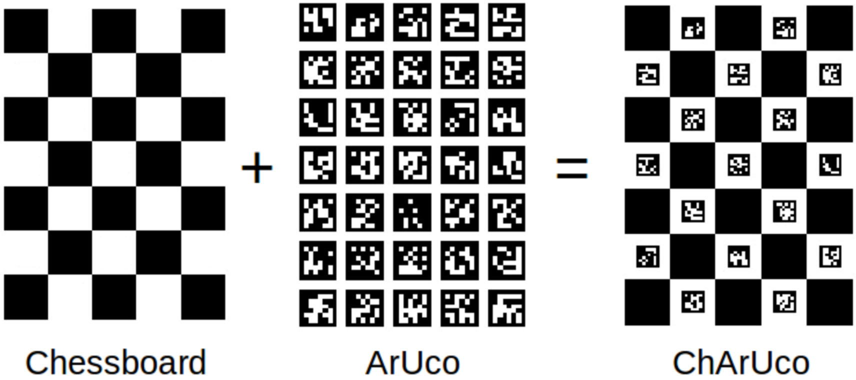
[4] `docs.opencv.org`

Markers pose example



ChArUco board for calibration

- Combines accuracy of pattern with detections of ArUco
- Partial visibility detections



Charuco definition

Camera matrix estimation with boards

- We can estimate camera matrix from correspondences in image space and spatial space
  - collect images of the board from different views
  - detect boards
  - compute correspondences between image points and board frame points
  - ```
    _, K, dist_coeffs, rvecs, tvecs = cv2.calibrateCamera(
        obj_points, img_points, img_shape)
    ```
- In addition we get
  - distortion coefficients that compensates defects of objective
    ```
    Knew, roi = cv.getOptimalNewCameraMatrix(K, dist_coeffs,
        img_shape, 1, img_shape)
    img_undistorted = cv.undistort(img, K, dist_coeffs, None, Knew)
    ```
  - $SE(3)$ poses of boards in camera frame
    ```

Pose estimation from RGB(D)

- Pose estimation methods
  - use prior knowledge about the task, *e.g.* fixed height objects on a plane
  - use prior knowledge about the objects (size)
  - use depth sensor
  - use ArUco markers
- Where is robot?
  - homography estimates poses of objects w.r.t. plane frame
  - other methods estimate poses in camera frame
  - we need to estimate/calibrate T_{RC}

HandEye calibration

- Camera can be mounted w.r.t.
  - robot base frame (eye-to-hand calibration)
  - gripper frame (eye-in-hand calibration)
- Solve $A^i X = Y B^i$
  - measurements: $A^i, B^i \in SE(3)$
  - estimated parameters: $X, Y \in SE(3)$
- `X, Y = calibrateRobotWorldHandEye(A, B)`
- Eye-to-hand calibration
  - $A^i = T^i_{\mathsf{RG}}$
  - $B^i = T^i_{\mathsf{CT}}$
  - $X = T_{\mathsf{GT}}$
  - $Y = T_{\mathsf{RC}}$
- Eye-in-hand calibration
  - $A^i = T^i_{\mathsf{CT}}$
  - $B^i = T^i_{\mathsf{GR}}$
  - $X = T_{\mathsf{TR}}$
  - $Y = T_{\mathsf{CG}}$

Summary

- Image representation
- Projection to/from image
- Segmentation in image space
- Homography
- Pose estimation from image
- Camera calibration

Laboratory

- No new homework this week
- Homography estimation on toy example in Python/OpenCV
- HandEye calibration on toy example in Python/OpenCV