POLITECNICO DI MILANO

Real-time Pose Estimation

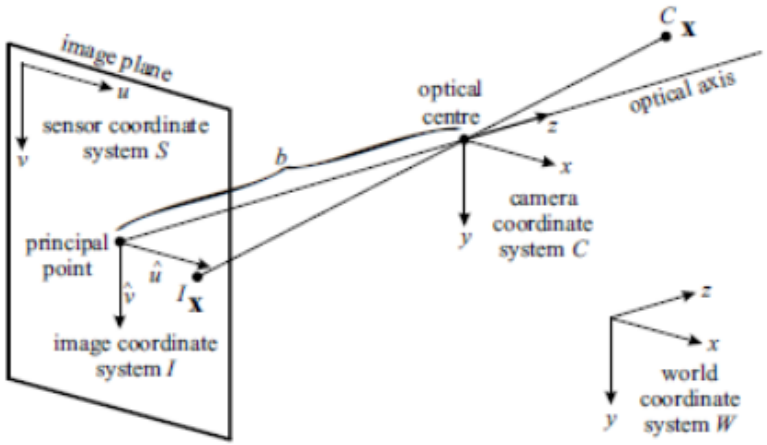
Vision Based 3D Measurements

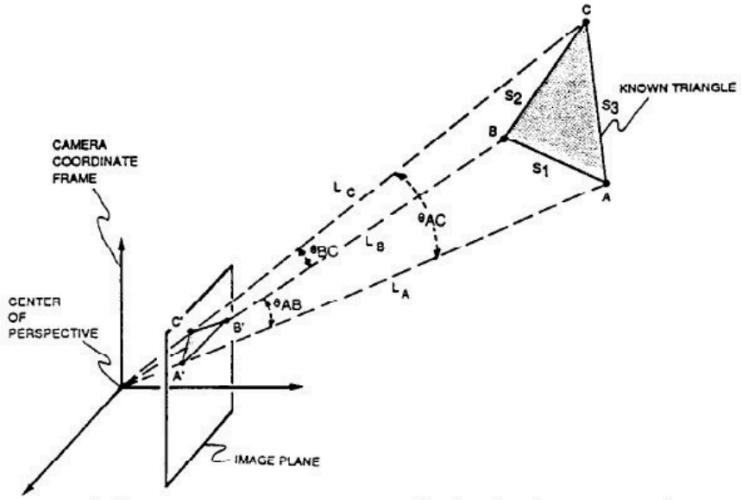
10616961 Fan Zhang
10569695 Chenzhuo Li

3D Pose Estimation Problem

3D pose estimation is the problem of determining the transformation(6 DOF: position and orientation) of an object in a 2D image which gives the 3D object.

The goal is to estimate the **position** and **orientation** of an object of **known geometry** in motion with respect to a fixed reference system.





3D Pose Estimation Problem

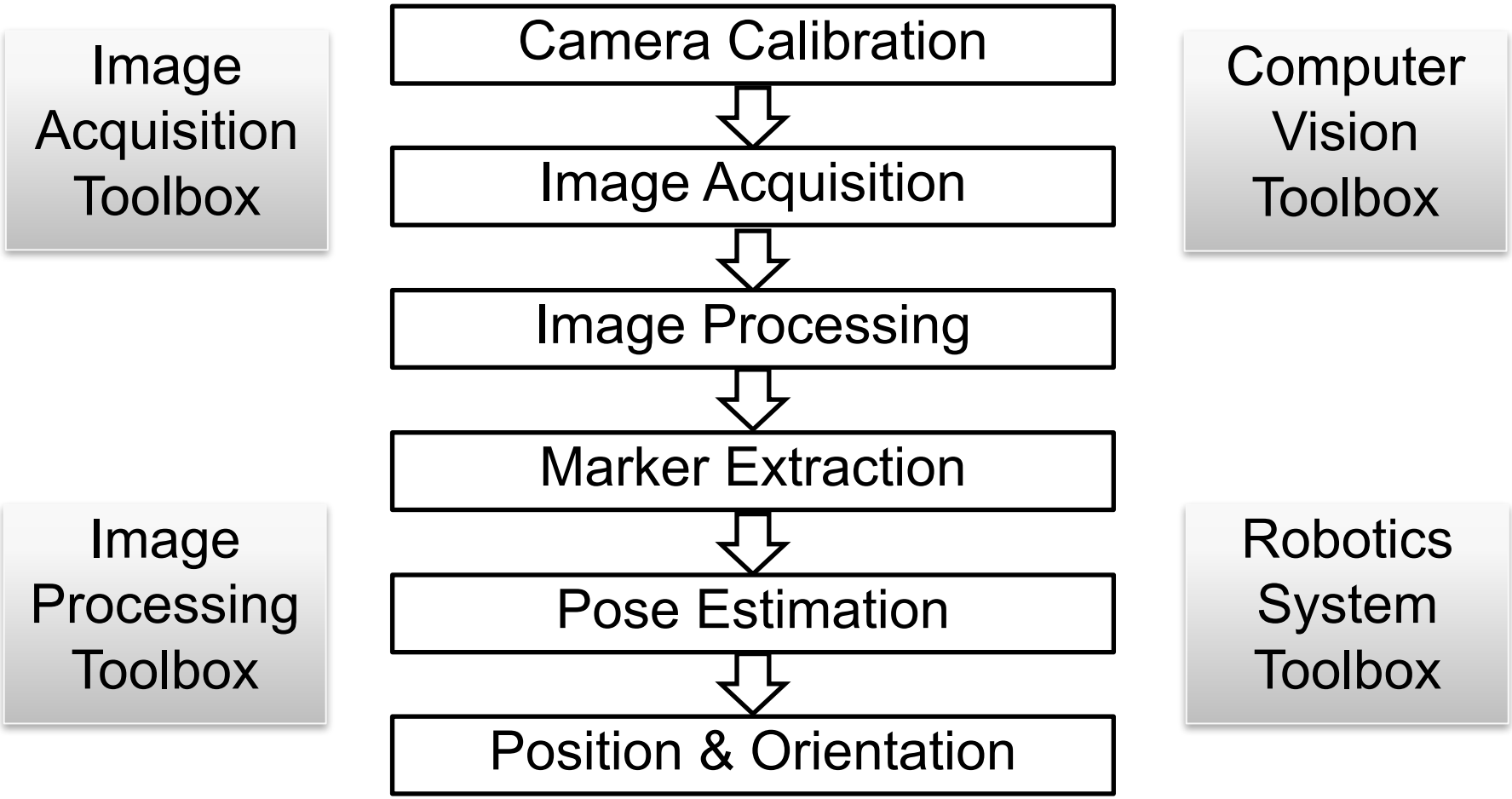
- Camera calibration parameters
- A rigid body of known geometry
- A digital image of the rigid body

Pose Estimation

- The camera body relative position (**translation vector**)
- The camera body relative orientation (**rotation matrix**)

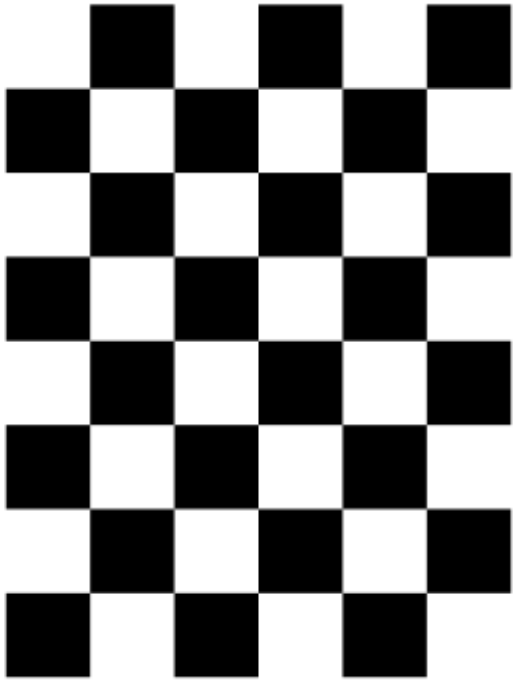
Pose Estimation Solution

Overview



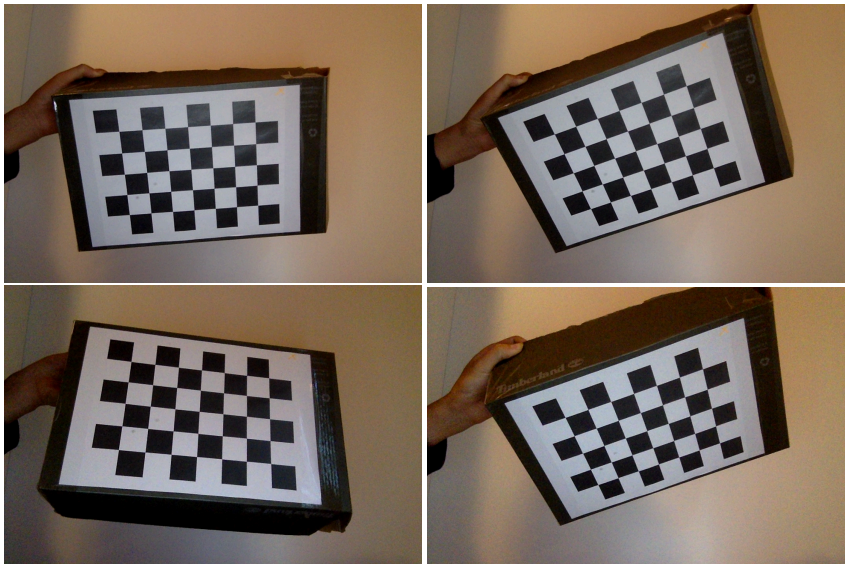
Pose Estimation Solution

Camera Calibration



Chessboard

30mm × 30mm square



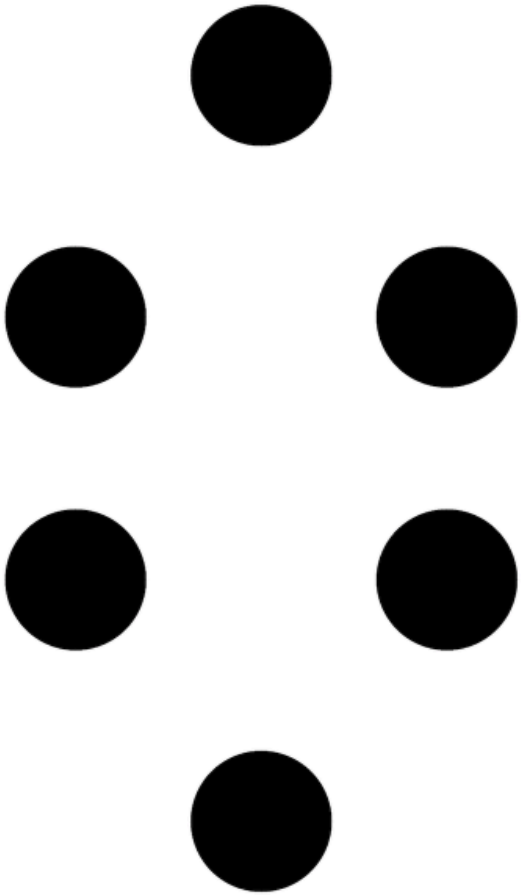
Using MATLAB camera calibrator application to get the intrinsic matrix of web camera and save it.

What we need is:

cameraParams.IntrinsicMatrix

Pose Estimation Solution

Object with known geometry



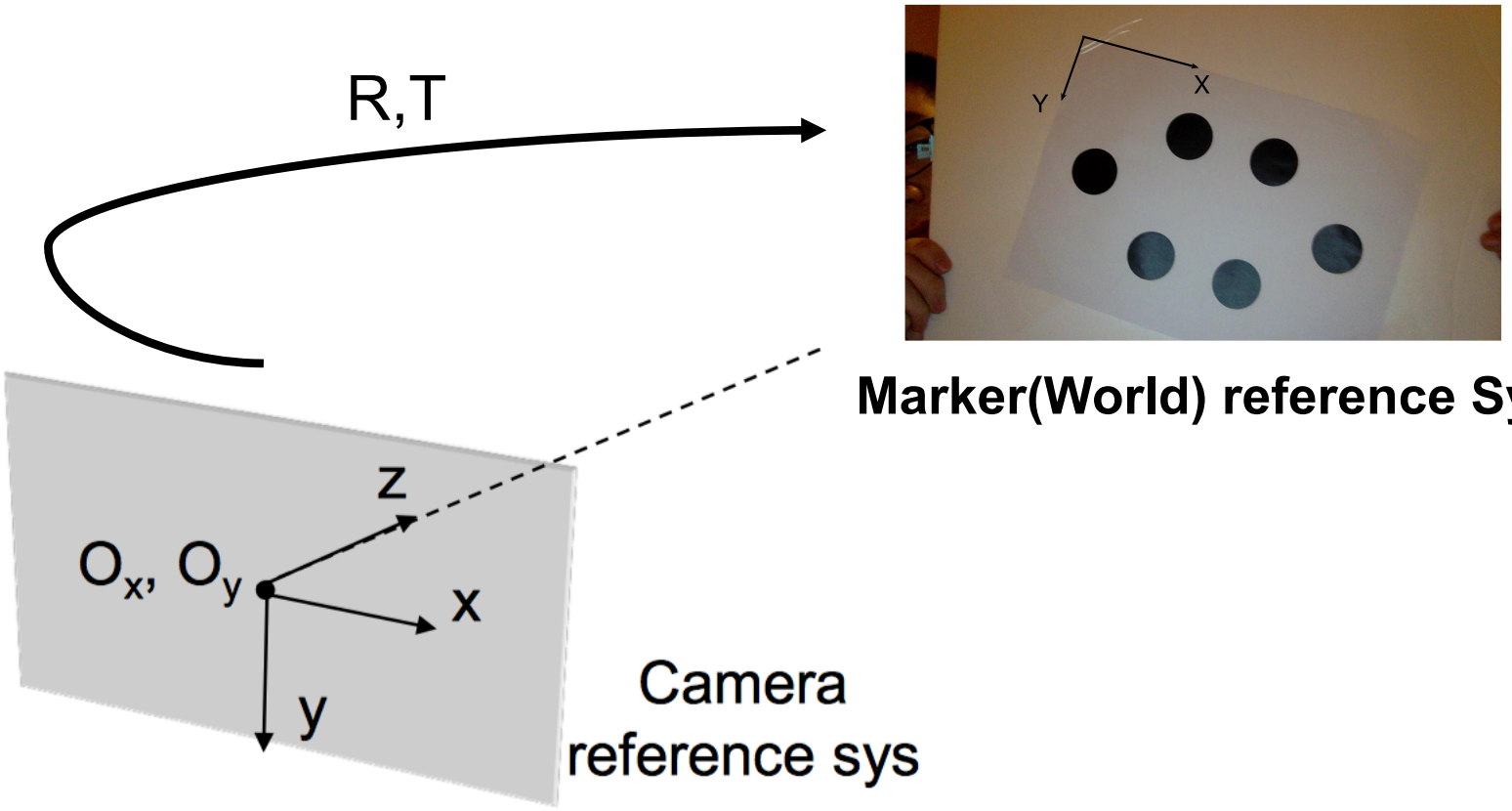
Generate marker with OpenCV

Size: 297mm × 210mm (A4)

Circle radius: 20mm

Pose Estimation Solution

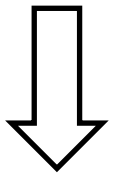
Object with known geometry



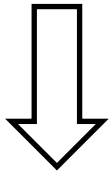
Pose Estimation Solution

Image Acquisition

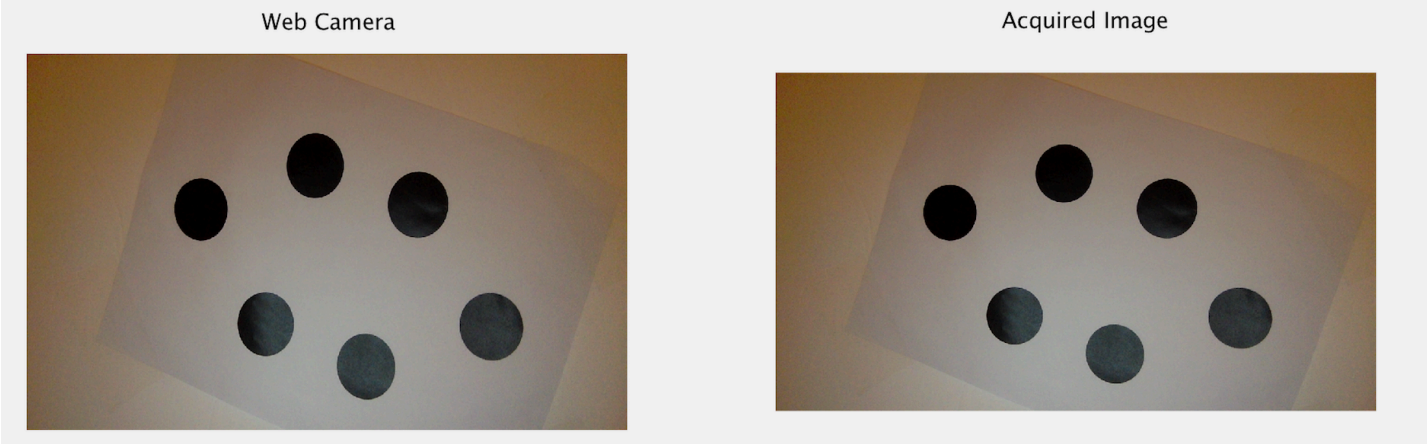
Using web camera to acquire images



```
vid = videoinput('macvideo', 1, YCbCr422_1280x720')
```



```
imgOriginal = getsnapshot(vid)
```



N.B. The acquisition procedure may change due to different platform.

Pose Estimation Solution

Image pre-processing --- background remove

Functions

- **I = rgb2gray(RGB)**

  converts the truecolor image RGB to the grayscale intensity image I.

- **IM2 = imerode(IM,SE)**

  erodes the grayscale image IM, returning the eroded image, IM2. SE is a structuring element object returned by the STREL function.

- **IM2 = imdilate(IM,SE)**

  dilates the grayscale image IM, returning the dilated image, IM2. SE is a structuring element object returned by the STREL function.

- **Z = imsubtract(X,Y)**

  subtracts each element in array Y from the corresponding element in array X and returns the difference in the corresponding element of the output array Z.

- **J = imadjust(I)**

  increases the contrast of the output image J.

Pose Estimation Solution

Image pre-processing --- binary image creation

Functions

- **LEVEL = graythresh(I)**

  computes a global threshold (LEVEL) that can be used to convert an intensity image to a binary image.

- **BW = imbinarize(I)**

  binarizes image I with a global threshold.

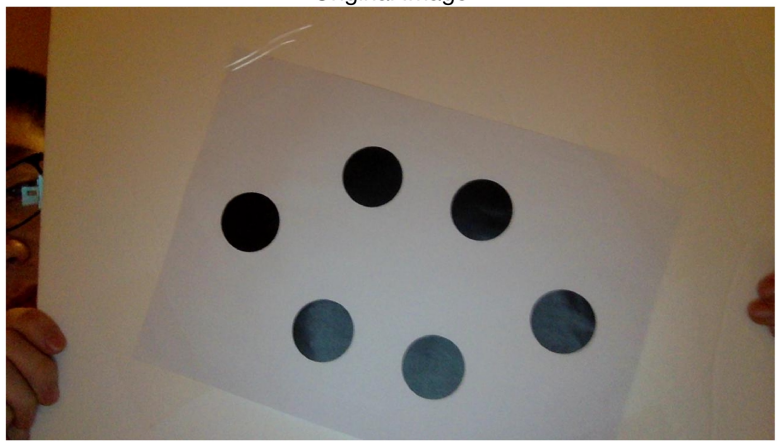
- **IM2 = imclose(IM,SE)**

  performs morphological closing on the grayscale or binary image IM with the structuring element SE.

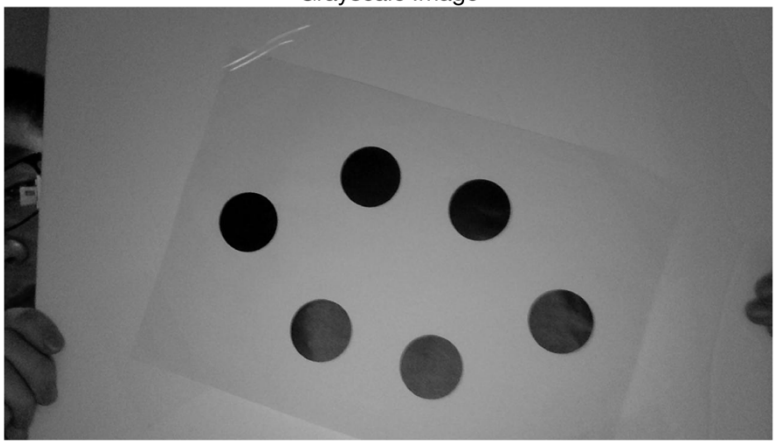
Pose Estimation Solution

Image pre-processing

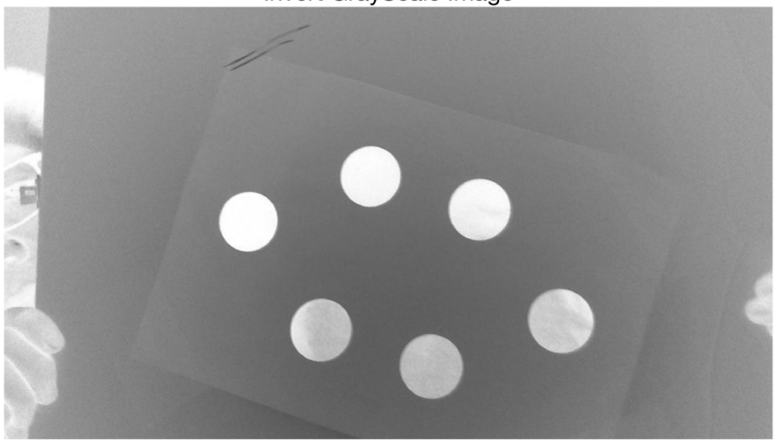
Original Image



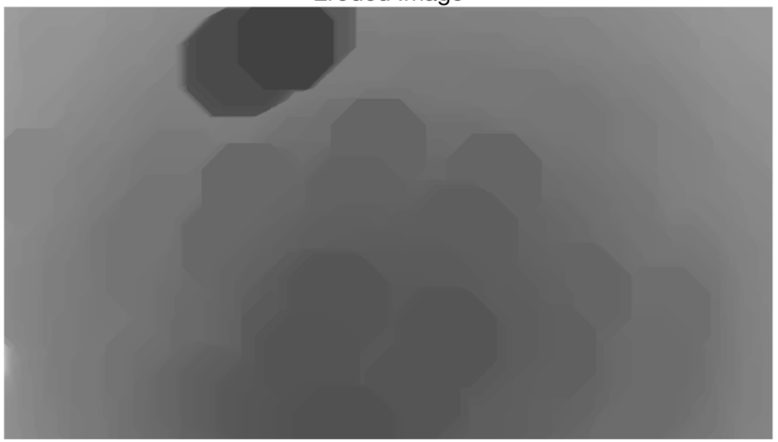
Grayscale Image



Invert GrayScale Image



Eroded Image



→ Image Background …

Pose Estimation Solution

Image pre-processing

Dilated Image

Front Image

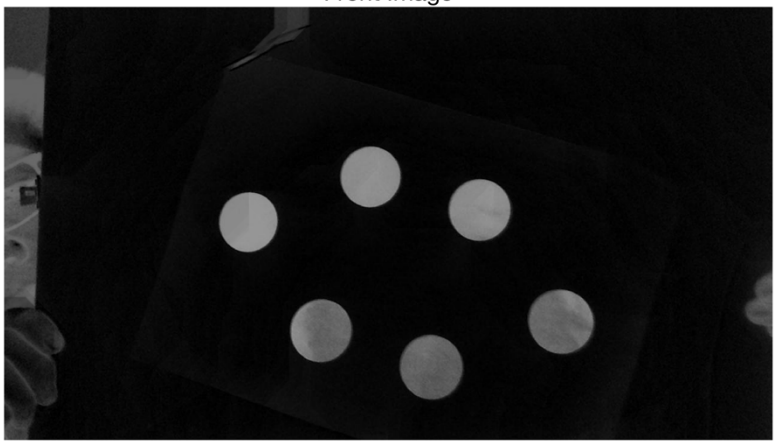
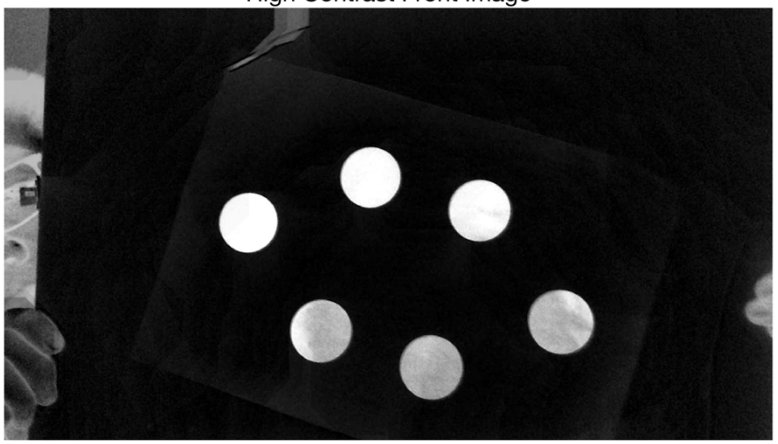


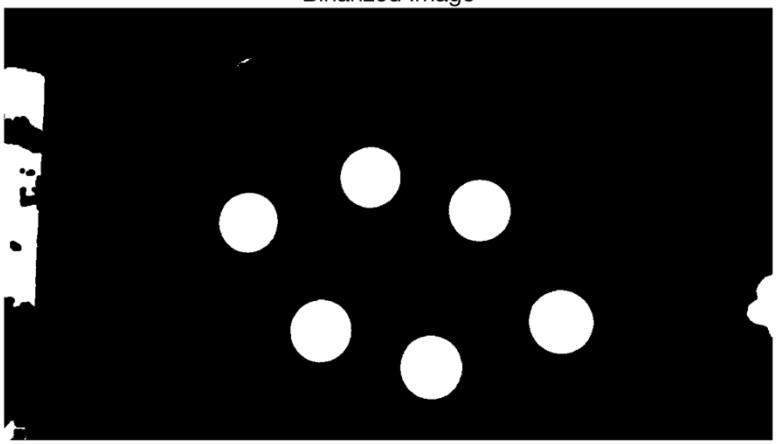
Image Background

Subtract

High Contrast Front Image



Binarized Image



Pose Estimation Solution

Image pre-processing --- extracting markers

Functions

- **L = bwlabel(BW,N)**

  returns a matrix L containing labels for the connected components in BW. N = 4 or 8, where 4 specifies 4-connected objects and 8 specifies 8-connected objects.

- **STATS = regionprops(BW,PROPERTIES)**

  measures a set of properties for each connected component (object) in the binary image BW. PROPERTIES can be a comma-separated list.

Since the eccentricity, area, and centroid of each component are known, we use **eccentricity** and **area** as thresholds to distinguish the markers from noise (in practice the values are set to 0.8 and 3000 to achieve high accuracy), and set the **centroids** as marker points.

Pose Estimation Solution

Pose estimation

Functions

- **[worldOrientation, worldLocation] = estimateWorldCameraPose(imagePoints, worldPoints, cameraParams)**

  returns the orientation and location of a calibrated camera in the world coordinate system in which worldPoints are defined.

  N.B. We change the value of **MaxReprojectionError** here to balance the accuracy and the fault-tolerance.

Mathematical calculation

- **Object orientation matrix** is the transpose of the camera orientation matrix. $\mathbf{R_{Obj} = (R_{Cam})^T}$
- **Object translation matrix** is the minus product of camera location and object orientation matrix. $\mathbf{T_{Obj} = - T_{Cam} \times (R_{Cam})^T}$

Pose Estimation Solution

Perspective-n-Point

Perspective-n-Point is the problem of estimating the pose of a calibrated camera given a set of n 3D points in the world and their corresponding 2D projections in the image. The camera pose consists of 6 degrees-of-freedom (DOF) which are made up of the rotation (roll, pitch, and yaw) and 3D translation of the camera with respect to the world.

When ***n* = 3**, the P*n*P problem is in its minimal form of P3P and can be solved with three point correspondences. However, with just three point correspondences, P3P yields many solutions, so a fourth correspondence is used in practice to remove ambiguity.

Gao, Xiao Shan, et al. "Complete solution classification for the perspective-three-point problem." IEEE Transactions on Pattern Analysis & Machine Intelligence 25.8(2003):930-943.

Pose Estimation Solution

Important parameters

- in marker extraction
  - **Eccentricity**

    circle: $e = 0$

    ellipse: $0 < e < 1$

    In practice, this value is set to 0.8
  - **Area**

    In practice, the areas of marker regions are larger than 3000. The value of this parameter may change with different camera in order to achieve the high accuracy.
- in pose estimation
  - **MaxReprojectionError**

    This parameter needs to be appropriate. If it is too small, not enough inliers in imagePoints and worldPoints can be found. If the value is too large, the result may not reliable.

Pose Estimation Solution

Pose post-processing

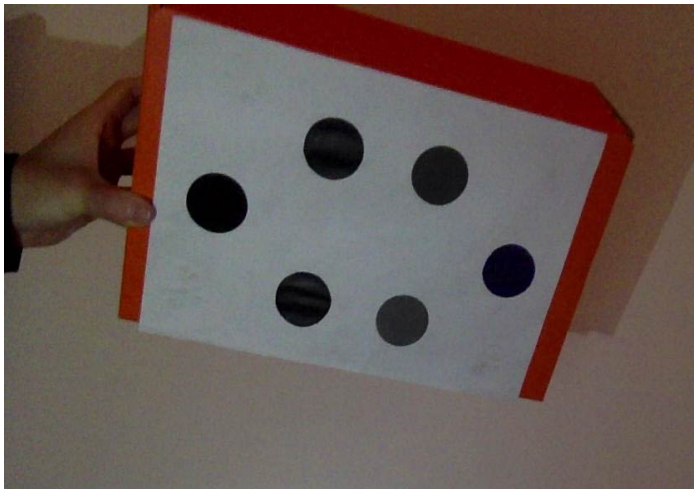
Function

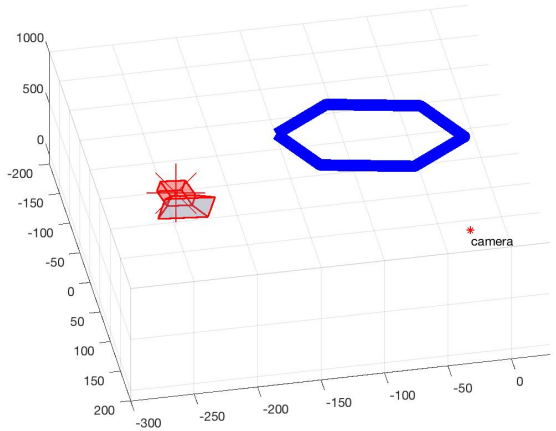
- **EUL = rotm2eul(R)**

  converts a 3D rotation matrix, R, into the corresponding Euler angles, EUL.

- **TreDimReconstruct(objectMarkerCoordinatesinCRS, originofWRSinCRS, savePath, objectOrientationinCRS)**

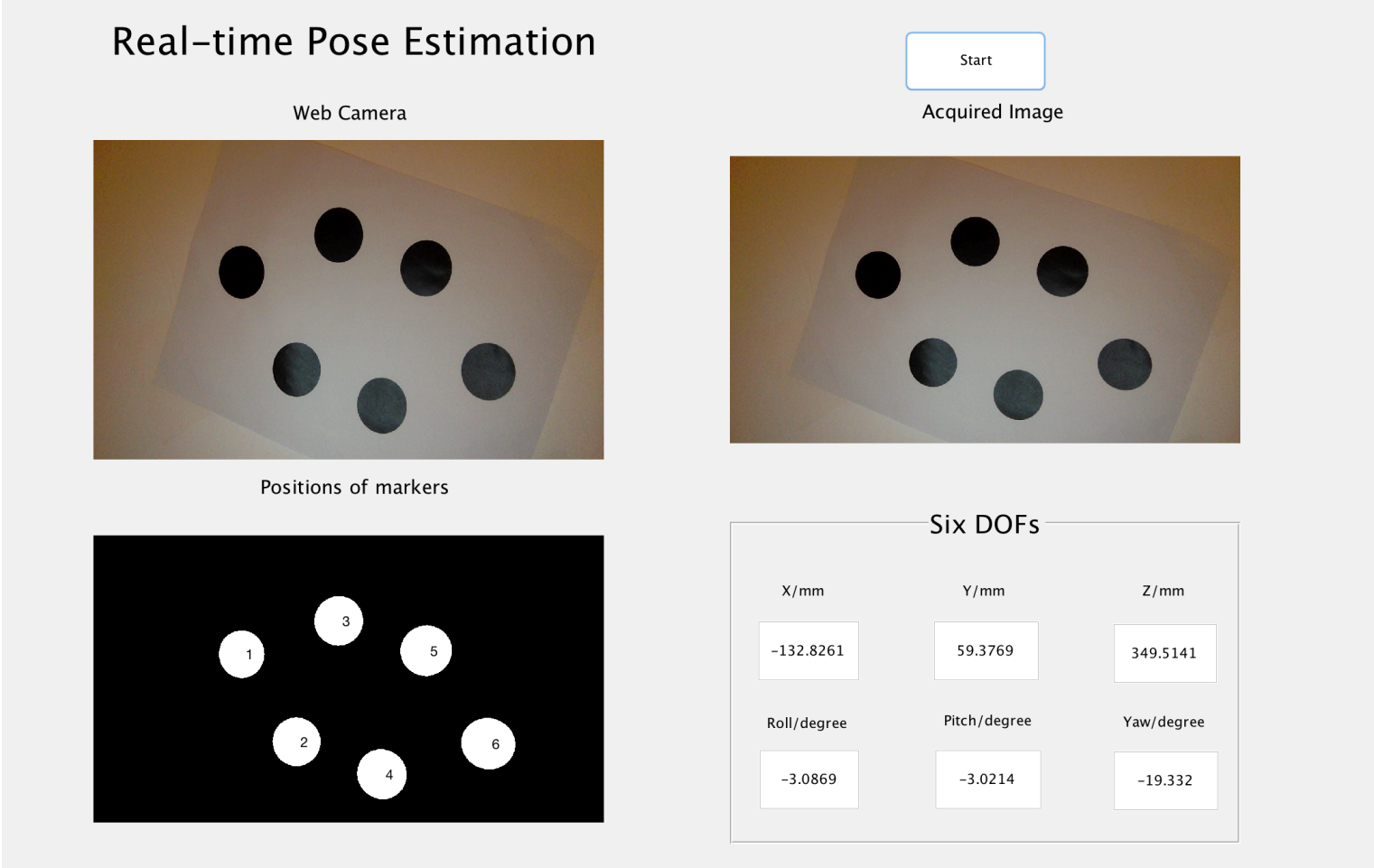
  reconstruct object, camera in 3D, demonstrating the position and orientation.





GUI: Visualize the result

In order to make the result more clear and specific, we create a GUI in MATLAB



Grazie !