Detail of Tracker.sln

Yuji Oyamada

[1]Graduate school of science and technology
Keio University

[2]Computer Aided Medical Procedure & Augmented Reality (CAMP)
Technische Universität München

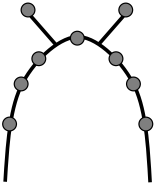
April 20, 2012

What's Tracker.sln?

Solves tracking problem as extrinsic parameters estimation problem.

Given

- Pre-computed camera parameters.
- Prior knowledge on target object, markers on arc:
  - 3D position of each marker.
  - Size of the marker.



The arc with 9 markers for Tracker.sln

The idea of Tracker.sln?

- 3D position estimation for detected blobs:
  - Compute 3D position of detected blobs.
  - Given known marker size and intrinsic parameters.
  - Detected blob may contain non-marker object, say noise.
- Extrinsic parameter estimation from detected blobs:
  - Find best match between known 3D position and estimated one.
  - Refinement removes the noise from detected blobs.

Very rough flow

1. track→Initialise() initializes the running system.
2. track→Track() computes extrinsic parameters given input image.
    2.1 ReadImage() reads image from camera, image file, or video file.
    2.2 UndistortImage() removes lens distortion.
    2.3 ExtractMarkers() detects markers on infra-red image.
    2.4 GetDepth() computes 3D position of each detected marker.
    2.5 MatchMarkers() computes extrinsic parameters.

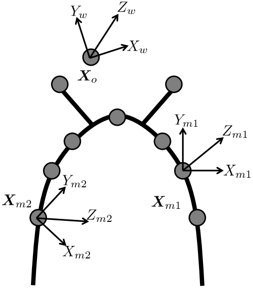
Function track→Initialize()

System initialization sets:

- Known intrinsic parameters (=calibration matrix) as m_K.
- Known distortion parameters as m_kc.
- Known 3D position of the arc as m_master3dArcPoints.

Marker-oriented coordinate system

- Define the marker-oriented coordinate for each marker.
- Important key to remove the noise from detected markers.



World coordinate

$$\boldsymbol{X}_{o,w} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{X}_{m1,w} = \begin{bmatrix} X_{m1} \\ Y_{m1} \\ Z_{m1} \end{bmatrix} \qquad \boldsymbol{X}_{m2,w} = \begin{bmatrix} X_{m2} \\ Y_{m2} \\ Z_{m2} \end{bmatrix}$$

Marker1-oriented coordinate

$$\boldsymbol{X}_{o,m1} = \boldsymbol{P}_{m1}\boldsymbol{X}_{o,w} \qquad \boldsymbol{X}_{m1,m1} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} \qquad \boldsymbol{X}_{m2,m1} = \boldsymbol{P}_{m1}\boldsymbol{X}_{m2,w}$$

Marker2-oriented coordinate

$$\boldsymbol{X}_{o,m2} = \boldsymbol{P}_{m2}\boldsymbol{X}_{o,w} \qquad \boldsymbol{X}_{m1,m2} = \boldsymbol{P}_{m2}\boldsymbol{X}_{m1,w} \qquad \boldsymbol{X}_{m2,m2} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}$$

World and Marker-oriented coordinate sytesm

Marker-oriented coordinate system: Algorithm

Compute markers position in each marker-oriented coordinate:

1. Set markers in world coordinate m_master3dArcPoints.
2. Compute each marker-oriented coordinate m_modelsOfMaster3dArcPoints:
   2.1 GetNeighbours() finds 2 neighbors for each marker.
   2.2 MakeModel() computes marker-oriented coordinate:
      2.2.1 first neighbor for X axis, second one for Y axis, dot product of them for Z axis.
      2.2.2 converts world coordinate to the marker-oriented coordinate.

Function track→Track()

Computes extrinsic parameters given input infra-red image:

1. Detects markers by blob detection ▸ ExtractMarkers().
2. Computes 3D position of the detected blobs ▸ GetDepth().
3. Computes extrinsic parameters from the computed 3D position ▸ MatchMarkers().

Marker Extraction

A function ExtractMarkers() does marker extraction.
Detect markers on the arc using the following conditions:

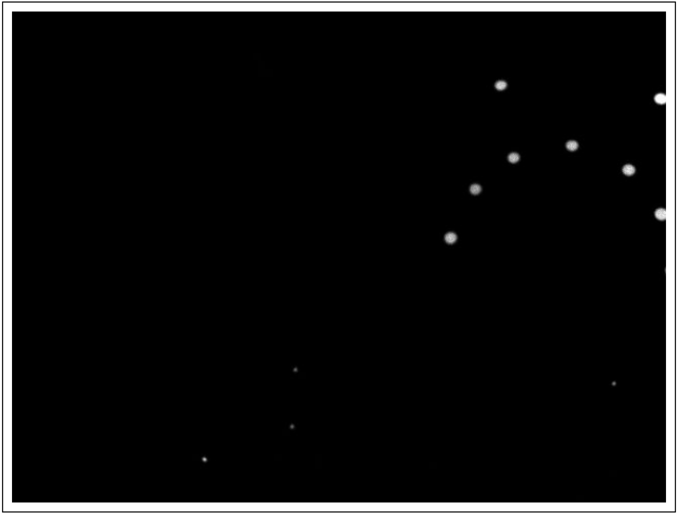
- Ideally, only markers are visible in infra-red image.
- Circles can be robustly detectable under perspective transform.

Marker Extraction: Algorithm

Detects markers and computes their position and size.

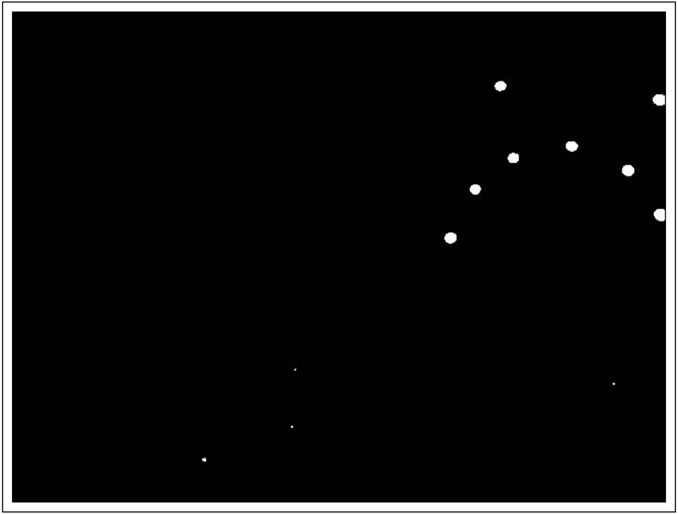
1. Extracts markers on the arc based on
   - Simple thresholding (cvThreshold())
   - Blob detection (cvBlobResult())
2. Compute statistical infomation of each detected blob:
   - (m_XMean[*i*], m_YMean[*i*]): Mass center of *i*-th blob.
   - area[*i*]: Area of *i*-th blob.
   - m_eigenVectors[*i*]: Eigenvectors of *i*-th blob.
   - m_eigenValues[*i*]: Eigenvalues of *i*-th blob.
3. Fit ellipse to the corresponding blob using eigenvectors:
   - m_majorSemiAxis[*i*]: Major axis of ellipse fitted to *i*-th blob.
   - m_minorSemiAxis[*i*]: Minor axis of ellipse fitted to *i*-th blob.

Marker Extraction: Result



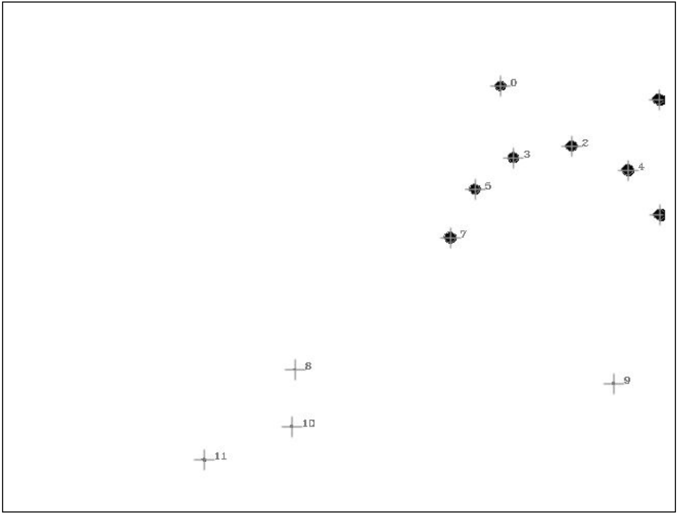
Input image

Marker Extraction: Result



Thresholded image

Marker Extraction: Result



Detected blobs and their position

3D position estimation

A function $\mathrm{GetDepth}()$ computes 3D position m_distanceVector even with wrong blobs exist.

Using

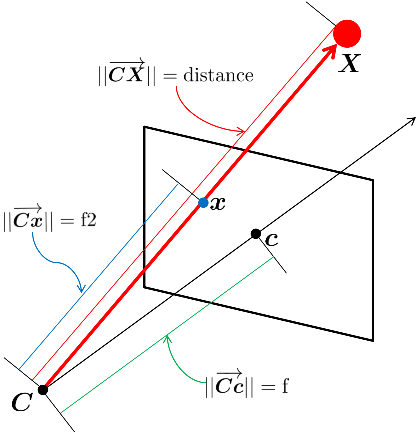
- Known
  - m_CameraMatrix ($=\mathbf{K}$): Intrinsic parameter including f.
  - m_markerRadius ($=\mathrm{radius}(\mathbf{X})$): Marker size in 3D coordinate.
- Estimated
  - (m_XMean,m_YMean): Mass center of extracted blobs.
  - m_majorSemiAxis ($=\mathrm{radius}(\mathbf{x})$): Blob size in 2D image coordinate.

Depth computation is based on

$$\frac{\text{Marker size in real world}}{\text{Marker size in image}} = \frac{\text{Marker depth in real world}}{\text{Focal length}} \tag{1}$$

3D position estimation: Camera geometry

Simple camera geometry to understand the function GetDepth().



$$\boldsymbol{x} = \begin{bmatrix} \text{middlePoint}[0] \\ \text{middlePoint}[1] \\ 1] \end{bmatrix} = \boldsymbol{K}[\boldsymbol{R}|\boldsymbol{t}]\boldsymbol{X}$$

$$\begin{aligned} \overrightarrow{\boldsymbol{Cx}} &= \boldsymbol{K}^{-1}\boldsymbol{x} \\ &= \text{invK} * \text{middlePoint} \\ &= \text{x2} \end{aligned}$$

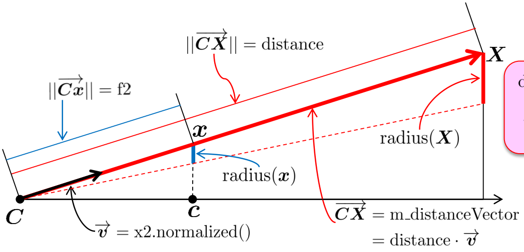
$$||\overrightarrow{\boldsymbol{CX}}|| = \text{distance}$$
$$||\overrightarrow{\boldsymbol{Cx}}|| = \text{f2}$$
$$||\overrightarrow{\boldsymbol{Cc}}|| = \text{f}$$

$$\text{radius}(\boldsymbol{X}) = \text{m\_markerRadius}$$
$$\text{radius}(\boldsymbol{x}) = \text{m\_majorSemiAxis[i]}$$

Definition in Tracker.sln

3D position estimation: Algorithm

1. Compute invK $= \mathbf{K}^{-1}$.
2. Compute vector x2 $= s\overrightarrow{\mathbf{Cx}} = \mathbf{K}^{-1}\mathbf{X}$ up to scale ambiguity.
3. Compute distance $= \left\|\overrightarrow{\mathbf{CX}}\right\|$.
4. Compute unit vector $\vec{v} = \frac{\overrightarrow{\mathbf{Cx}}}{\left\|\overrightarrow{\mathbf{Cx}}\right\|}$.
5. Compute distanceVector $=$ distance $\cdot\, \vec{v} = \overrightarrow{\mathbf{CX}}$.



distance : f2 = radius(X) : radius(x)

$\rightarrow$ distance $= \frac{\text{radius}(X) * \text{f2}}{\text{radius}(x)}$

Extrinsic parameter estimation

Even with noise in detected blobs, a function MatchMarkers() computes extrinsic parameters:

- m_rotationMatrix: Rotation matrix.
- m_translationVector: Translation vector.

Using

- Known
  - m_modelsOfMaster3dArcPoints: Marker-oriented coordinate.
  - m_K: Intrinsic parameters.
  - m_kc: Lens distortion parameters.
- Estimated
  - m_distanceVector: 3D position of each blob.

Extrinsic parameter estimation: Algorithm

Compute extrinsic parameters based on the coordinate system.

1. Check whether the detected blobs is close to its previous position. If yes, go to final step.
2. Convert computed blobs position to blob-oriented coordinates.
3. Match detected blobs to known markers.
4. Compute extrinsic parameters using best matching result.

Extrinsic parameter estimation: Step 1

Uses sequential information if available.

- A function getMarkerMatch() checks whether detected blobs are near to their previous position.
- If enough matching result is obtained, go to final step.

Extrinsic parameter estimation: Step 2

Compute blobs position in each blob-oriented coordinate:

1. Given detected blobs in world coordinate m_distanceVector.
2. Compute each blob-oriented coordinate modelsOfCalculatedArc3dPoints[0]:
   2.1 GetNeighbours() finds 2 neighbors for each blob.
   2.2 MakeModel() computes blob-oriented coordinate:
      2.2.1 first neighbor for X axis, second one for Y axis, dot product of them for Z axis.
      2.2.2 converts world coordinate to the blob-oriented coordinate.

Extrinsic parameter estimation: Step 3

A function GetMatch() computes matching scores of all pair of markers and blobs:

- m_modelsOfMaster3dArcPoints[k]: Markers position in k-th marker-oriented coordinate.
- modelsOfCalculatedArc3dPoints[0]: Blobs position in i-th blob-oriented coordinate.

Here, mis-detected noise in blobs are removed:

1. Assumption that noise appears smaller than markers.
2. Smaller size noise corresponds to a marker positioning very far.
3. Matching metric is based on distance between marker and blob.

Extrinsic parameter estimation: Step 4

Compute extrinsic parameters using best matching result between known markers and detected blobs.

Camera parameters

Tracker.sln considers the following camera model.

- Projection matrix is consists of intrinsic parameters and extrinsic one.
- Consider lens distortion.

Projection matrix

$$[m\_P] = [m\_K][m\_R|m\_t], \tag{2}$$

where

- m_P: 3×4 Projection matrix
- m_K: 3×3 Calibration matrix (intrinsic parameters)
- m_R: 3×3 Rotation matrix (extrinsic parameters)
- m_t: 3×1 Translation vector (extrinsic parameters)

Calibration matrix (intrinsic parameters)

Calibration matrix m_K can be decomposed as

$$\left[m\_K\right] = \begin{bmatrix} alphaX & 0 & pX \\ 0 & alphaY & pY \\ 0 & 0 & 1 \end{bmatrix} \tag{3}$$

where

- alphaX: Focal length in x-direction
- alphaY: Focal length in y-direction
- pX: X-coordinate of principal point [pixel]
- pY: Y-coordinate of principal point [pixel]

Lens distortion parameters

See OpenCV function explanation.

m_kc

- Lens distortion parameters.
- 1×4 vector, double as $m\_kc = \begin{bmatrix} k\_1 & k\_2 & p\_1 & p\_2 \end{bmatrix}$,

where

- k_1 and k_2 are radial distortion parameters and
- p_1 and p_2 are tangential distortion parameters.

m_dstMapx and m_dstMapy

- Undistortion map for all sequences.
- height$\times$width matrix, double.