5
Computer vision for hand recognition

5.1
Cameras

A *camera*, in its everyday meaning, is a device for taking *photographs*, which in their turn are 2D representations of a 3D scene in the form of either 1) a raster image file, 2) a printout or 3) a transparent slide. In this work, we're mainly interested in digital cameras. They are capable to produce raster image files (see Section 5.2 for more on digital images), suitable for computer processing.

Real cameras can be modeled using various mathematical models. In mathematical terms, a *camera model* (frequently called *camera* as well) is defined as a mapping between the 3D (Euclidean) world being observed and the resulting 2D image:

$$\text{camera: } \text{3D scene} \longrightarrow \text{2D image}$$

For the purposes of this exposition, one mathematical model in particular satisfies our needs: the **pinhole camera** model.

5.1.1
Pinhole camera model

The pinhole camera (Figure 5.1) consists of a hollow box, whose side has been perforated by a small hole, called *pinhole*. (Alternative name for pinhole is *optical center*, designated by $\vec{C}$ in Figure 5.1). Light emanating from the scene enters the pinhole and gets projected onto the inner surface (*screen*) opposite to the hole. The screen has a light-sensitive surface (either an CCD or CMOS chip in the case of digital cameras, and photographic film in the case of analog cameras) which enables the camera to record the 3D scene.

Due to the geometry of the image-creating process (which can be easily understood looking at Figure 5.1), the projected image is reversed (i.e. upside-down). Also, the smaller the hole is, the projected image is sharper due to the

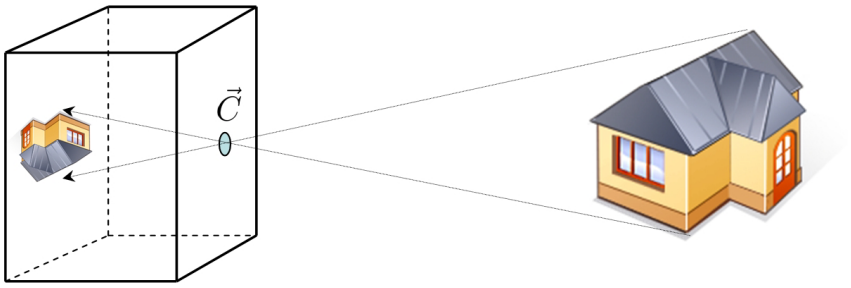

Figure 5.1: Pinhole camera, with pinhole (i.e. optical center) at $\vec{C}$

smaller number of light rays falling onto one specific spot in the image plane; and vice versa, light rays emanating from one particular position in 3D scene fall on just one spot on the screen. On the other hand, very small pinholes lead to the aberration of the image because light entering the box starts to suffer the phenomenon of *wave difraction*. Also, too small a hole permits just a very low amount of light energy to enter the box, which leads to exposure times which are simply too long for many purposes.

Since the projected image is upside down, we sometimes replace the screen with another (hypothetical) screen between the optical centre $\vec{C}$ and the 3D scene, thus creating a *virtual image* which has the same orientation as the 3D scene:

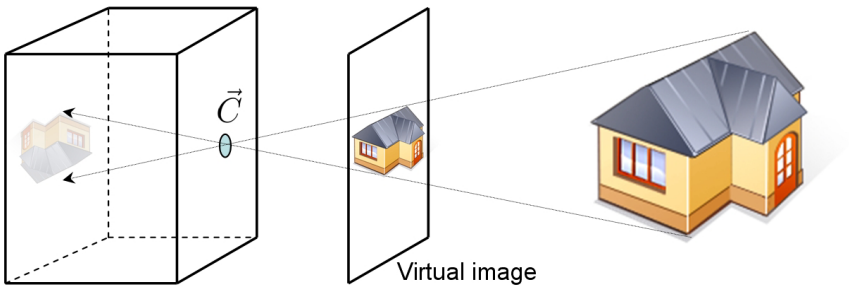

Figure 5.2: Pinhole camera with screen in the front of $\vec{C}$

5.2 Digital images

Digital images we refer to in this work are the so-called *intensity images*, two-dimensional discrete arrays of picture elements (also called pixels) with M rows and N columns, where each of the $M \times N$ pixels measure the amount of visible electromagnetic energy (i.e. light) that fell onto that position at the moment the image was taken.

An image can also be considered a planar (2D) coordinate system, where the origin is fixed at the upper-left corner, and where each 2D point (u, v) is

Figure 5.3: A digital image consisting of 48×43 pixels

defined by its horizontal distance u from the origin and its vertical distance v from the origin.

5.3 Mono vision

This section deals with mono-vision, which is a vision obtained using just one camera. We'll determine mathematically how a specific 3D point P with the associated position vector $\vec{X} = (X, Y, Z)$ gets projected into a specific 2D pixel point $\vec{u} = (u, v)$ in the raster image.

5.3.1 Relevant coordinate systems

There are four coordinate systems (Figure 5.4) involved in the computation of the 2D raster image of a 3D scene:

1. **World coordinate system (WCS)** — this is our global, absolute 3D system. In this work, WCS is fixed on the table, so that $+x$ points to the right, $+y$ away from the user and $+z$ up. We designate the origin of WCS by O.

2. **Camera coordinate system (CCS)** — this 3D coordinate system is fixed so that its origin O_c is at the camera's optical centre. No imagine that you're peeping through the camera's optical finder. In this case, $+x$ is to the right, $+y$ is up, and $+z$ points to the user.

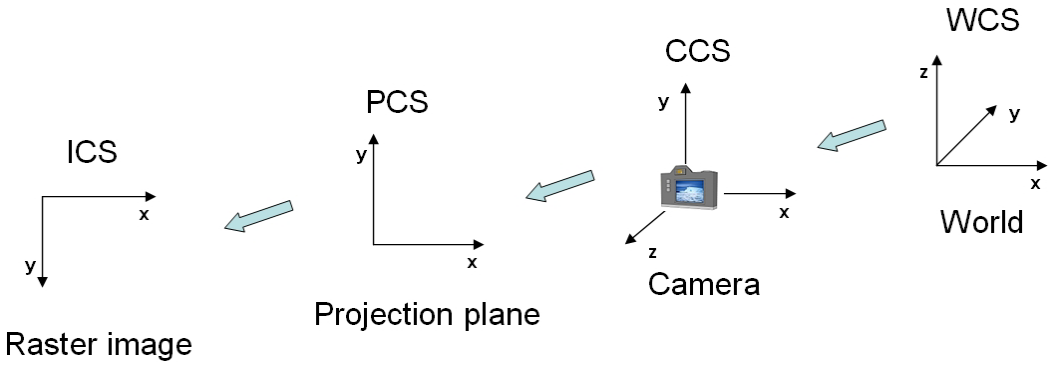

Figure 5.4: Coordinate systems in the world–camera–projection–raster image chain.

3. **Projection plane coordinate system (PCS)** — this is a 2D coordinate system embedded into the projection plane. The origin O_p is fixed at the orthogonal projection of the camera's optical centre onto the projection plane, $+x$ axis points right, and $+y$ points up.

4. **Raster image coordinate system (ICS)** — this is a final 2D coordinate system, which expresses the position of a point in PCS relative to the grid defined by the rectangular array of pixels. In this work, the origin $o = (o_x, o_y)$ of ICS is fixed at the upper left corner of the image produced in PCS. The $+x$ points therefore to the right, and $+y$ points down.

5.3.2 Transformations between coordinate systems

As we have seen, we need to deal with four coordinate systems: WCS, CCS, PCS and ICS. Therefore, we have to consider three coordinate transformations between (three of) them, in order to understand mathematically how the 2D raster image forms from the 3D scene:

1. CCS $\longleftarrow\!\shortmid$ WCS
2. PCS $\longleftarrow\!\shortmid$ CCS
3. ICS $\longleftarrow\!\shortmid$ PCS

Schematically:

$$\boxed{\text{Raster image (2D)}} \longleftarrow\!\shortmid \boxed{\text{Projection plane (2D)}} \longleftarrow\!\shortmid \boxed{\text{Camera (3D)}} \longleftarrow\!\shortmid \boxed{\text{World (3D)}}$$

or equivalently:

$$\text{ICS} \longleftarrow\!\shortmid \text{PCS} \longleftarrow\!\shortmid \text{CCS} \longleftarrow\!\shortmid \text{WCS}$$

or, using coordinates:

$$(u, v) \longmapsto (u_p, u_p) \longmapsto (X_c, Y_c, Z_c) \longmapsto (X, Y, Z)$$

CCS $\longmapsto$ WCS

Here, 3D world (i.e. expressed in WCS) coordinates (X, Y, Z) are being re-computed as 3D camera coordinates (X_c, Y_c, Z_c) (i.e. CCS coordinates). Let P be a 3D point, $\vec{X} = (X, Y, Z)$ its representation in WCS, and $\vec{X}_c = (X_c, Y_c, Z_c)$ be the presentation (i.e. coordinates) of P in CCS. It holds:

$$\vec{X}_c \longmapsto \vec{X}$$

$$\vec{X}_c = R\vec{X} + \vec{t}$$

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} X \\ Y \\ Z \end{bmatrix} + \begin{bmatrix} t_1 \\ t_2 \\ t_3 \end{bmatrix}$$

where R is an 3×3 rotation matrix that rotates WCS relative to CCS (i.e. its columns are coordinates of the unitary vectors that make up a base in WCS, relative to the vector base of CCS), and $\vec{t} = (t_1, t_2, t_3)^\tau$ is the vector $\overrightarrow{O_c O}$ (the vector from the camera's origin to the world's origin). See Figure 5.5 that depicts the transition from WCS to CCS.

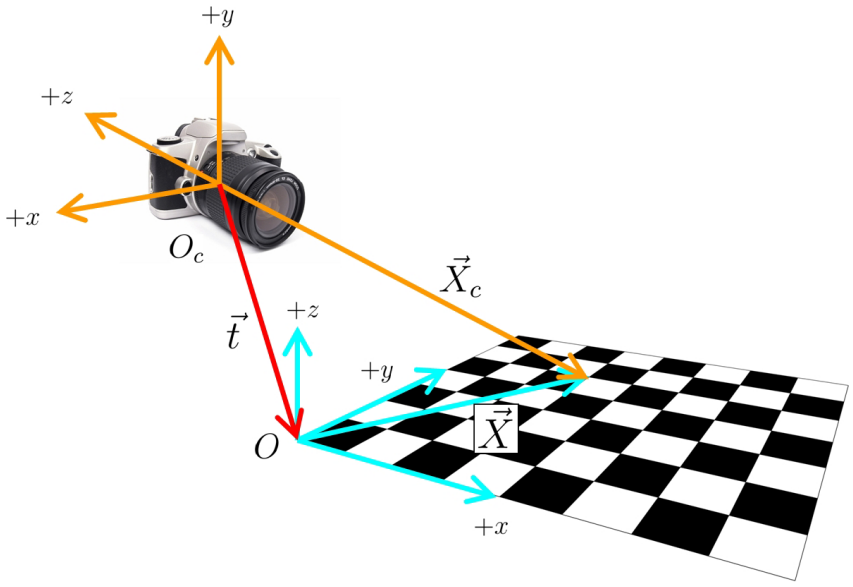

Figure 5.5: Going from 3D world to 3D camera coordinates (CCS $\longmapsto$ WCS)

The rotation matrix R and translation vector $\vec{t}$ are also called *extrinsic parameters* of the camera. Extrinsic parameters determine the camera's

location and orientation relative to WCS.

PCS $\longmapsto$ CCS

Having now calculated CCS coordinates $\vec{X}_c = (X_c, Y_c, Z_c)$, we can compute 2D coordinates $\vec{u}_p = (u_p, v_p)$ of the projection of $\vec{X}_c$ onto the camera's projection plane. We'll use the pinhole camera model.

$$\vec{u}_p \longmapsto \vec{X}_c$$

$$(u_p, v_p) = \left(f\frac{X_c}{Z_c},\ f\frac{Y_c}{Z_c} \right) \tag{5-1}$$

For the computation in Eq. 5-1 to take place, we of course must know the value of f (focal length of the camera, expressed in meters). Parameter f is one of the so-called *intrinsic parameters* of the camera.

ICS $\longmapsto$ PCS

Finally, the projected 2D point $\vec{u}_p = (u_p, v_p)$ can now be expressed relative to the pixel array grid as a 2D point $\vec{u} = (u, v)$. Note that $\vec{u}_p = (u_p, v_p)$ is expressed in meters, while $\vec{u} = (u, v)$ is expressed in pixels.

$$\vec{u} \longmapsto \vec{u}_p$$

$$(u, v) = \left(-\frac{u_p}{s_x} + o_x, -\frac{v_p}{s_y} + o_y \right)$$

$$\begin{bmatrix} u \\ v \end{bmatrix} = \begin{bmatrix} -\frac{u_p}{s_x} + o_x \\ -\frac{v_p}{s_y} + o_y \end{bmatrix}$$

For this computation, we must know the values of s_x and s_y (dimensions of one sensor element in the CCD/CMOS chip, expressed in meters), and o_x and o_y (translation values for the raster image origin). Parameters s_x, s_y, o_x, o_y also belong to the set of intrinsic parameters of the camera.

Composing transformations PCS $\longmapsto$ CCS and ICS $\longmapsto$ PCS

Composing transformations PCS $\longmapsto$ CCS and ICS $\longmapsto$ PCS we obtain the transformation ICS $\longmapsto$ CCS:

$$\text{ICS} \longmapsto \text{PCS} \longmapsto \text{CCS}$$

that is, we go from the 3D camera system CCS to the 2D raster image system directly:

$$\text{ICS} \longleftarrow\!\shortmid \text{CCS}$$

$$(u, v) = \left(-\frac{f}{s_x} \cdot \frac{X_c}{Z_c} + o_x, -\frac{f}{s_y} \cdot \frac{Y_c}{Z_c} + o_y \right)$$

If we now define $f_x = \frac{f}{s_x}$ and $f_y = \frac{f}{s_y}$ we obtain

$$(u, v) = \left(-f_x \cdot \frac{X_c}{Z_c} + o_x, -f_y \cdot \frac{Y_c}{Z_c} + o_y \right)$$

When working with vision setups, we rarely get to know focal length f of the camera exactly (only when we have the camera manufacturer data). Instead, using calibration techniques (see Section 5.3.3) we usually obtain just f_x and f_y (focal lengths expressed in pixels). Using the same calibration techniques we also obtain o_x and o_y (expressed in pixels as well), which are coordinates of the image center (principal point), which is the intersection between the image plane and the optical axis (line through O_c, perpendicular to the image plane).

Considering this, in this work we frequently ignore PCS and go straight from CCS (3D camera system) to ICS (2D raster image system). Thus the pipeline looks like this:

$$\boxed{\text{Raster image (2D)}} \longleftarrow\!\shortmid \boxed{\text{Camera (3D)}} \longleftarrow\!\shortmid \boxed{\text{World (3D)}}$$

or equivalently:

$$\text{ICS} \longleftarrow\!\shortmid \text{CCS} \longleftarrow\!\shortmid \text{WCS}$$

or, using coordinates:

$$(u, v) \longleftarrow\!\shortmid (X_c, X_c, Z_c) \longleftarrow\!\shortmid (X, Y, Z)$$

5.3.3 Mono-camera calibration

The (mono) camera calibration is a process where all the parameters of a camera are being determined. The parameters include *intrinsic* camera parameters and *extrinsic* camera parameters.

- **Intrinsic camera parameters**:
  - $f_x \in \mathbb{R}, f_y \in \mathbb{R}$: two focal lengths in the x- and y-direction (in pixels)
  - $o_x \in \mathbb{N}, o_y \in \mathbb{N}$: two integer x- and y-coordinates of the image center (in pixels). Note that when we are working with subpixel precision, we have $o_x \in \mathbb{R}, o_y \in \mathbb{R}$ (in pixels, but we use real numbers here)

  - $\alpha = \frac{s_y}{s_x}$: pixel aspect ratio (pixel deformation)(dimensionless)
  - k_1, k_2: two radial distortion coefficients (dimensionless)

- **Extrinsic camera parameters**:

  - rotation matrix R (dimensionless) and
  - translation vector $\vec{t}$ (in meters).

While there exist many calibration techniques, i.e. methods to extract both the extrinsic and intrinsic camera parameters listed above, we focus on the method by Zhang [19].

5.3.4 Zhang's camera calibration method

Zhang's camera calibration method [19] enables the user to obtain intrinsic and extrinsic camera parameters taking several (at least two) snapshots of a planar pattern (for example, a checkerboard consisting of a grid of alternating black and white squares, or any other planar pattern with easily distinguishable features). For example, Figure 5.5 shows a camera observing an 8×7 checkerboard pattern.

The method first finds an initial solution using a closed-form expression. This solution is then refined using the Levenberg-Marquardt algorithm, which is a nonlinear minimization method.

5.4 Stereo vision

Stereo vision, or *stereopsis*, is the main mechanism through which humans perceive spatial depth. In stereopsis, a 3D point P gets projected into two different locations on both eyes' retinas. The difference between these two locations, called *stereo disparity*, is then processed by the brain which, in conjuction with other factors (like for example eye-to-eye distance), estimates the depth coordinate of the point P.

In the computational context, stereo vision is obtained using two cameras. We say that two cameras, when employed to compute stereo, make up a *stereo rig*. Just like a in the case of mono vision, stereo vision has its own geometry. Figure 5.6 shows the geometry of a stereo rig.

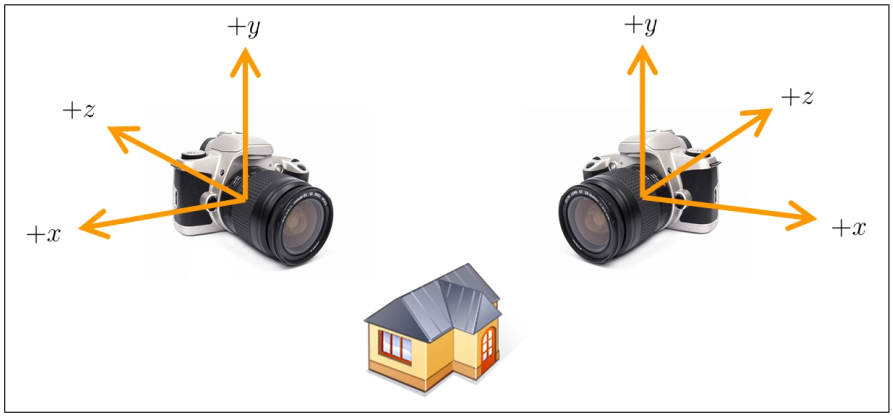

Figure 5.6: Stereo rig. If the two cameras take a snapshot at the same instant, the two photos make a stereo pair of photos.

5.4.1 Stereo 3D reconstruction

The expression "stereo 3D reconstruction" refers to the process of determining the 3D structure and 3D position of an observed object, given a number N of correspondences $\{(\vec{u}_1, \vec{u}'_1), (\vec{u}_2, \vec{u}'_2), \ldots, (\vec{u}_N, \vec{u}'_N)\}$ in the left and right image of the stereo input stream.

Reconstruction based on triangulation

Triangulation is a method of determining the position of a fixed point using another two fixed points a known distance apart. Therefore, triangulation is a stereo 3D reconstruction of just one corresponding pair. Having one correspondence $(\vec{u}_1, \vec{u}'_1)$ of one observed (photographed) point feature P, we can use triangulation to determine the 3D location (X, Y, Z) of this feature.

Supposing that a point P is visible in both images, and that we know the pixel coordinates $\vec{u}$ and $\vec{u}'$ of both projections of the point P in both images, we can easily compute two rays in space — one ray passing through the left camera's optical centre and $\vec{u}$, and the second ray through the right camera's optical centre and $\vec{u}'$. The triangulation problem is then equivalent to finding the intersection of these two rays in space.

However, there exist several problems with this approach. If we knew $\vec{u}, \vec{u}'$ and camera's parameters exactly, of course we would be able to recover P easily, using formulas of elementary vector algebra. The problems are the following:

– **Floating-point arithmetics errors** — since we use floating-point arithmetics (because all the underlying parameters are actually continuous physical values), we induce numerical errors to the triangulation

process. As a consequence of this, the two rays can never intersect exactly.

– **Measurement errors** — we can never measure $\vec{u}, \vec{u}'$ and determine camera's parameters exactly. As a consequence, we can never compute the intersection of two rays exactly.

Due to the aforementioned problems, which leads to the fact that the two rays *do not cross* in space, we must find other solutions to determining 3D point $\vec{X}$ using triangulation. An overview of available triangulation methods can be found in [20]. In this work, we will mention (and later implement) two triangulation methods: a simple one, called **Mid-point triangulation method**, and the optimal one (under the assumption of Gaussian noise), called **Polynomial triangulation method**.

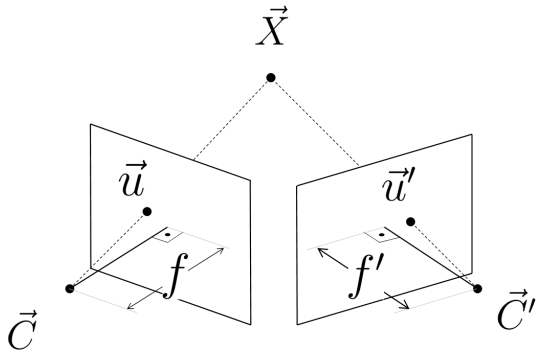

Figure 5.7: Triangulation. Knowing 3D positions of optical centers $\vec{C}, \vec{C}'$, focal lengths f, f' and 2D positions $\vec{u}, \vec{u}'$, we can determine 3D position $\vec{X}$ using various triangulation methods (for example, *mid-point* and *polynomial* triangulation methods).

Mid-point triangulation method This is probably the simplest and fastest possible triangulation method, however with serious shortcomings (see [20]).

Suppose that the corresponding (matched) 2D points are $\vec{u} = (u, v)$ and $\vec{u}' = (u', v')$, i.e. $\vec{u}, \vec{u}'$ are images of an 3D point $\vec{X} = (X, Y, Z)$ in the left and right image. The point $\vec{X}$ lies at the interesection of two rays: first ray from C through $\vec{u}$ and the second ray from C' through $\vec{u}'$ (see Figure 5.8). However, since the two rays do not actually meet in space due to the aforementioned issues, we can only *approximate* the intersection of two rays. In this method, mid-point method, we approximate this intersection with the point that lies at the *minimum distance to both rays.*

To extract $\vec{X}$, suppose first that we work in the left camera's coordinate system. Let r, r' be two rays:

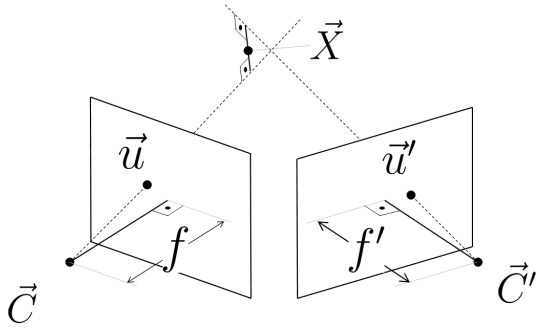

Figure 5.8: Mid-point triangulation method, which finds the point $\vec{X}$ as the point that lies at the minimum distance to both rays: first ray from C through $\vec{u}$, and the second ray from C' through $\vec{u}'$.

- $r = \{\alpha \cdot \vec{u} \mid \alpha \in \mathbb{R}\}$ the ray through C and $\vec{u}$, and
- $r' = \{\hat{\vec{t}} + \beta \cdot \hat{R}^\tau \cdot \vec{u}' \mid \beta \in \mathbb{R}\}$ the ray through C' and $\vec{u}'$.

... where $\hat{R} = R'R^\tau$, and $\hat{\vec{t}} = \vec{t} - \hat{R}^\tau \vec{t}'$. Let $\vec{a}$ be a vector orthogonal to both r and r'. Point $\vec{X}$ is now in the middle of the segment parallel to $\vec{a}$ that joins both rays r and r'.

Now let $\vec{Y} = \alpha_0 \cdot \vec{u}$ one endpoint of the segment, and $\vec{Z} = \hat{\vec{t}} + \beta_0 \cdot \hat{R}^\tau \cdot \vec{u}'$ the other endpoint of the segment. Parameters α_0 and β_0 are now computed solving the following system of linear equations:

$$\alpha_0 \vec{u} - \beta_0 R^\tau \vec{u}' + \gamma(\vec{u} \times (R^\tau \vec{u}')) = \vec{t}$$

The desired 3D point $\vec{X}$ is now simply

$$\vec{X} = \frac{\vec{Y} + \vec{Z}}{2}$$

Polynomial triangulation method This method [20] gives an optimal global solution to the triangulation problem. Further, the algorithm employed is non-iterative and simple in concept, has low computation requirements and has superior performance compared with other methods.

Formulated as a least-squares minimization problem, the method computes image points $\hat{u}, \hat{u}'$ such that

$$d(u, \hat{u})^2 + d(u', \hat{u}')^2 \to \min, \qquad \hat{u}'^\tau F \hat{u} = 0$$

where $d(*, *)$ is the Euclidean distance function and F the fundamental matrix of the stereo rig. Assuming Gaussian error distribution (tracking of u, u' is noisy because of digitization errors), the points $\hat{u}, \hat{u}'$ are the most likely values for

true image correspondences. Since the corresponding rays through $\hat{u}, \hat{u}'$ meet *exactly* in 3D space, we can now find easily x (the global position of the hand in the workspace) using other triangulation methods, for example the mid-point triangulation described above in Section 5.4.1.

5.5 Color spaces

Since we are interested in visual hand recognition, and human hand is of course covered with skin, we also have to consider the color of human skin when trying to detect a hand in an image. According to [21], human skin color can be conveniently represented and processed in the following color spaces:

- **Basic color spaces** (RGB, normalized RGB, CIE-XYZ) — these are the so-called "default" color spaces, because they are ubiquitous and their properties are well know and defined.
- **Perceptual color spaces** (HSI/HSV/HSL, TSL) — the HSI/HSV/HSL model models "perceptual" qualities like *hue*, *saturation* and *intensity* (also called *brightness*, *lightness* or *value*). The TSL model quantifies *tint* (hue with white added), *saturation* and *lightness*.
- **Orthogonal color spaces** (YCbCr, YIQ, YUV, YES) — these reduce the redundancy present in the RGB model, and model the color with as statistically independent components as possible. Luminance and chrominance components are separated, therefore these spaces are the favorable choice for skin detection.
- **Perceptually uniform color spaces** (CIE-Lab, CIE-Luv) — in these spaces, the luminance L and the chroma ab or uv are obtained through a non-linear mapping of XYZ coordinates. The advantage of these spaces is that they can represent color in a perceptually uniform way. The downside is that computing CIE-Lab, CIE-Luv colors is computationally expensive.
- **Other color spaces** — color ratios like R/G and $R/G + R/B + G/B$.

5.6 Human skin modeling

Human skin *detection* can be viewed as a two-class classification problem — given an image I in color space C, and given a pixel $I(x, y)$ in image I at position (x, y) with color $c \in C$, a detector f_C outputs 1 if the pixel $I(x, y)$ is a skin pixel, and 0 otherwise (a non-skin pixel). The domain $D(f)$ of classifier

f is therefore the color space C, and its range $R(f)$ is the set $\{0, 1\}$:

$$f_C : C \rightarrow \{0, 1\}$$

$$f_C(c) = \begin{cases} 1 & \text{if } c \text{ deemed a skin color} \\ 0 & \text{otherwise} \end{cases}$$

Again according to [21], human skin detection methods can be classified as:

- **Explicit skin-color space thresholding** — skin colors of different individuals cluster in a small region in color space. This method thus simply marks this region; if a pixel falls within this region, it is deemed a skin pixel. Has good skin detection rates, at the expense of high false positives. Simple and fast, but with many limitations (e.g. illumination must be controlled, threshold values differ for color spaces and different illumination levels, is less accurate in case of shadows, and in case the background contains objects with colors similar to skin color). Because of the limitations, this approach is usually complemented with a dynamic adaptation approach.
- **Histogram model with naive Bayes classifiers** — here a 2D or 3D color histogram is used to represent skin tones. This method is stable, unaffected by occlusions and changes in view, and can be used to differentiate a large number of objects. Slightly better than GMM or MLP (see below). However, needs a very large training set, and has high storage requirements.
- **Gaussian classifiers (SGM, GMM)** — Again, since skin colors of different individuals cluster in a small region in color space, we can model this skin distribution by a multivariate normal Gaussian distribution (this is the so-called Single Gaussian Model — SGM) or by a sum of individual Gaussians (the so-called Gaussian Mixture Models — GMM). This approach generalizes well, with less training data, and has a small storage requirements. Slightly inferior to histograms with naive Bayes classifiers, however GMMs are popular because they can generalize very well with less training data.
- **Elliptical boundary model** — Has performance slightly better than GMM, and the computational complexity is as simple as training a SGM. The downside is that it performs binary classifications only.
- **Multi-layer perceptron (MLP)** — a type of feed-forward neural network. Outperforms (together with Bayesian classifiers) Gaussian models

and Explicit skin-color space thresholding. Has very low storage requirements.

- **Self-organizing map (SOM)** — a type of neural network. Consistently better than GMMs.
- **Maximum entropy (MaxEnt)** — a statistical method for estimating probability distributions from data.
- **Bayesian network (BN)** — directed acyclic graphs that allow efficient and effective representation of the joint probability density functions.

5.7 Image features

Detecting image features in an image is a low-level but fundamental task, prerequisite for almost any higher-level computer-vision algorithm, like for example camera calibration, line detection or tracking. The term *local feature* designates a *local* property of the image, for example an edge, cross, closed curve (ellipse, or circle), KLT feature or SIFT feature (more will be said about KLT and SIFT in the text that follows).

If a feature is located in a region of an image, we call this region a "local interest region". And taking this local interest region into consideration, we are then able to compute a "local descriptor". Many types of descriptors have been proposed so far in the literature. As a rule, local descriptors must be invariant to image scale and rotation. In further text, terms "local descriptor" and "local feature" are considered synonymous.

From the viewpoint of vision-based tracking (see also Section 5.11 on page 56), features represent "hooks" onto which we can latch and observe their displacements from frame to frame. By tracking these features, therefore, we can track the objects to whom these features belong to. See Figure 5.9.

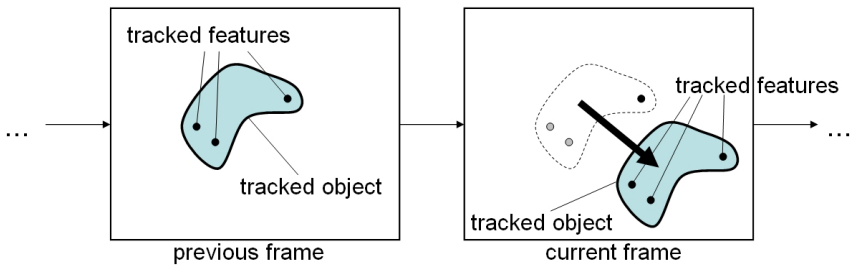

Figure 5.9: Tracking an object by tracking its features

The positive aspects of local features are:

1. **Abundance** — even tiny objects can give rise to a large number of features.

2. **Computationally cheap** — local features are generally cheap to compute, which leads, in general, to real-time tracking performance.

3. **Robustness** — features are robust to: occlusion, noise, small changes in viewpoint, and changes in illumination.

Some representative classes of features, listed from the oldest to the newest work, include:

- **Corners (Harris detector)** — the basic idea is that shifting a small window (for example, of 9×9 pixels) around a pixel should result in large change in intensity, in any direction [22].
- **KLT features** — Kanada-Lucas-Tomasi (KLT) features, also called "Good features to track", are "good" in a well-defined formal, mathematical sense. In this approach, an image feature is deemed a KLT feature only if it can be tracked in a simple, fast and accurate fashion. See Appendix C on page 116 and reference papers [23] [24] [25] .
- **SIFT features** — SIFT stands for "Scale Invariant Features Transform". This method maximizes difference of Gaussians over space and scale [26]. Note that this method has been patented.
- **SURF features** — SURF stands for "Speed Up Robust Features", and is a scale- and rotation-invariant interest point detector and descriptor. It approximates or outperforms earlier methods with respect to repeatability, distinctiveness and robustness, but can be computed and compared much faster [27].

Note that there exist many more types of local features. For the performance evaluation of various types of features, please refer to [28], which shows that SIFT outperforms all methods. However note that the publication date of this performance evaluation [28] is in 2005, while the SURF paper [27] was published one year later (in 2006) and claims performance superior to SIFT.

5.8 Hand detection

This is the process whereby a hand is detected (localized) in an image, using computer vision techniques. Basically, this process tries to answer the question "Is there a hand in this image?". (Is the answer is affirmative, then this process also returns the location of the hand in the image.) There exist various approaches to detecting and localizing a hand in an image:

– **Human skin detection** — human skin has a characteristic color signature which can be used to detect it (the skin) in an image.

Pros: skin color is surprisingly uniform (race - white/yellow/red/black, or being suntanned doesn't matter, since the hue does not change), so color-based detection is achievable.

Cons: other skin-colored object (like for example user's face, or other users' hands and/or faces, and so on) can enter into the image, and thus confuse the detection process. Workarounds would include 1) restricting the work area so that it contains hands only, or 2) to use hand's salient and discriminating features to dinstinguish it from other skin-colored objects.

Other potential problem is insufficient illumination, whereby too dark a workspace prevents efficient detection of human skin's characteristic hue. Potential workaround would include using supplementary infrared cameras.

– **Motion detection** — human hands usually move at greater speeds compared with other object in the scene.

Pros: when the background is static, motion-based detection is a practical and fast method to detect hands.

Cons: does not work when either the camera or parts of the background move, which is frequently the case.

– **Classifier-based detection** — this is a machine-learning approach. A classifier is a mapping $f : X \rightarrow Y$ from a feature space X to a discrete set of labels Y. Classifiers can be seen as decision systems which accept values of some features or characteristics of a situation as input and produce as an output a discrete label related to the input values.

One representative of this approach is the Viola-Jones method [29], which also uses AdaBoost [30] in order to construct so-called "strong classifiers" from a combination of "weak classifiers". Viola-Jones method can be

applied to any type of object; effectiveness at detecting specifically hands has also been investigated [31].

Pros: fast, high detection accuracy, very large and very complex set of features possible (although in this work we are interested in hands only), robust (works under wide range of conditions: variations of illumination, scale, pose and camera).

Cons: a learning process (sometimes a very prolonged one) is needed. However for hands this learning process can be significantly reduced [31].

– **Hybrid detection methods** — here two or more approaches are being combined, in order to increase the robustness and reliability of hand detection. For example, skin detection can be combined with motion detection. Or, Viola-Jones method can be combined with skin detection [31], and so on.

Pros: increased detection rate.

Cons: increased processing load.

Since Viola-Jones detection method has been used in the prototype application (see Chapter 6, page 58), technicalities are given in detail in Appendix B on page 109.

5.9 Hand segmentation

After the hand has been detected (localized) in the image, the hand must be segmented i.e. the background must be extracted from the region containing hand's image. In other words, pixels belonging to the hand must be separated from all the remaining pixels. Approaches and techniques:

– **Thresholding** — here a special auxiliary black & white image is created for each frame, whereby pixels belonging to the hand are white, and all the rest are black. Now combining this auxiliary image with the original image (utilizing the binary operation AND), the pixels belonging to the hand are being filtered out as they are, while all the remaining pixels are filtered out/set to 0 (black).

– **Morphological functions** — functions like "dilate" and "erode" that work on black & white images (like the auxiliary one obtained through thresholding). Dilate operation causes objects to grow in size (get "thicker"), and erosion causes elements to shrink (get "thinner"). These functions serve to "improve" the hand region ultimately segmented from the image.

– **Blobs** — here contiguous regions of the image are being identified and labelled. The region ("blob") which contains the location of detected hand is now the region depicting the hand.

5.10 Hand pose estimation

Now, given the extracted (segmented) hand region (blob), we can proceed with estimating the parameters determining the hand's posture in 3D space. See Figure 5.10 for a classification of pose estimation approaches.

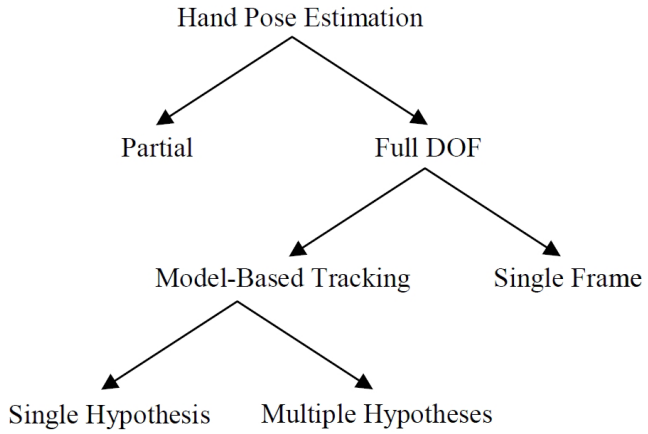

Figure 5.10: Taxonomy of hand pose estimation approaches

Partial hand pose estimation

Partial hand pose estimation relies on hand models with reduced number of d.o.f., therefore it does not try to recover the full set of 26 d.o.f. As such it relies on extracting *appearance-based* features like fingertips, orientation of fingers, global position of the hand, contours (sillhouettes), contour centroids, and curvature in order to reconstruct *directly* (i.e. without the help of a virtual 3D hand model) the pose of the partial model.

Full d.o.f. hand pose estimation

Differently from the partial hand pose estimation, the full d.o.f. hand pose estimation does not try to compute pose directly from extracted features, but instead *uses the extracted features in order to execute a search in the parameter space, so that a certain type of error can be minimized.*

- **Model-based pose estimation** — During the search, the 3D hand model (collocated into a certain position and certain orientation in virtual 3D space) is being projected onto a 2D image plane, and features extracted from this image plane are then compared with features extracted from the source video image. Can be further subdivided into:
  - **Single-hypothesis model-based pose estimation** — this one is based on restricted (local) search and retains only one, the best, estimate at each input image. The search is being done either a) optimization methods, or b) applying physical forces on the 3D hand model.
  - **Multiple-hypotheses model-based pose estimation** — retains several hypotheses about the hand pose. If one estimate gives a too big an error, the next best one is used. A big majority of this type of pose estimation utilizes Bayesian filtering or derivation thereof, specifically a) particle filters, b) tree-based filters, c) Bayesian networks, d) template database search, e) other approaches.
- **Single-frame pose estimation** — in this approach just one image (or, in the case of multiple-camera setups, the set of images taken at the same instant) is being used. Therefore, past parameter states are not being used in order to restrict the scope of the search — the whole (global) parameter space is being searched. An obvious advantage is that no analysis of past states is necessary, and disadvantage that the complete (and therefore computationally more expensive) search in parameter space must be made, although there may be no real need for that. Approaches:
  - **Classifiers** — please refer to Section 5.8. Classifiers can also work for pose estimation, not just detection, because classes of training data can be tagged by a full, predetermined set of pose data. So when a classifier detects a hand pattern in the image, we will automatically know the pose too. For this to work, a virtual 3D hand model must be used in order to produce training data (because it's practically impossible to determine pose data from pre-existing photos of real hands).
  - **Database indexing** — a large number of training samples (i.e. images of the hand model projected in every possible way) can be saved into a database, and then indexed in a special way. The database is then searched for in order to retrieve the nearest neighbour of the input image.

- **2D-3D mapping** — this is another learning approach. In a nutshell, here certain features are being computed from the (2D) input image (like moments of the hand contour, invariant to the rotation and scaling), and the mapping from this set of 2D features to the set of 3D poses is then being taught to a special machine-learning architecture named "Special Mapping Architecture".
- **Inverse kinematics** — here the position of fingertips is being used in order to compute joint angles. This reduces the problem to a typical inverse kinematics problem, known for example in the field of robotics. The difficulties in this approach are a) to detect fingertips reliably, and 2) to solve the inverse kinematics problem (i.e. computing the joint angles) correctly.

5.11 Hand tracking

Tracking is the process of estimating the position of a tracked object, taking its previous position into consideration.

Since the hand is an articulate 3D structure, we can choose to track the hand either in its original 3D space (in this case we say that we employ *model-based* hand tracking), or in the 2D projection image plane (in this case we employ the so-called *appearance-based* tracking). We can also talk about *hybrid tracking*, a recent mode of tracking which combines elements of model- and appearance-based tracking.

5.11.1 Appearance-based hand tracking

Various tracking methods in this class differ in what *cues* they use for tracking — some, for example, use just one cue like skin color or hand motion, and another use a combination of cues (for example, skin color *and* motion). For example, in Camshift [32] just the hand's color is being used as a cue; in CONDENSATION [33], hand contours + hand motion, and in ICONDENSATION [34] hand contours + hand motion + skin color; in "Flock of Features" [35] a combination skin color + KLT features [25], [24] (see Section 5.11 for more on KLT features).

For example, KLT tracking takes advantage of the fact that images in a video sequence are usually similar to each other. Due to the small time interval between the frames, objects being tracked haven't had the time to travel large distances, or the shift (translation vector) between an object's images in the previous and current image is small. This fact leads to an algorithm which

extracts this translation vector thus tracking the object. For technical details on tracking based on KLT features, please refer to Appendix C on page 116.

5.11.2 Model-based hand tracking

Here the back-projection of a predefined 3D parametric hand model is being matched against the input video frame. At each frame, extracted features are being compared with the current 3D model, and the matching error computed; if the error is too large, the 3D model is adjusted in the attempt to decrease the error — if the error is still too big, we repeat the model adjustment, otherwise we found a good matching and the tracking was successful. Examples include the classic DigitEyes system [5], where a 27-d.o.f. hand model is being tracked.

5.11.3 Hybrid hand tracking

Here elements of both the model-based and appearance-based tracking are combined in an effort to get the best of two worlds. Shimada *et al* in [36] and Athitsos and Sclaroff in [37] synthesize a large number of 2D views of a software 3D hand model, and tag each of these views by the corresponding, exact hand pose vector. After this preprocessing step, appearance-based matching methods are used to process real images.