Low-level feature extraction

Corner extraction

Image Feature Extraction Techniques

Piotr Fulmański

FACULTY OF MATHEMATICS AND COMPUTER SCIENCE
University of Lodz

Curvature

Curvature

Definition

Intuitively, you can consider ***curvature*** **as the rate of change in edge direction**.

This rate of change characterises the points in a curve -- **points where the edge direction changes rapidly are corners**, whereas points where there is little change in edge direction correspond to "straight" lines.

Such extreme points are very useful for shape description and matching, since they represent significant information with reduced data.

Curvature for planar curves

Definition

Curvature is normally defined by considering a parametric form of a planar curve.

The parametric contour $v(t) = x(t)U_x + y(t)U_y$ describes the **points in a continuous curve as the end points of the position vector**.

Here, the values of t define an arbitrary parameterisation, $U_x = [1,0]$ and $U_y = [0,1]$ are the standard unit vectors in Cartesian 2D coordinate system.

Changes in the position vector are given by the tangent vector function of the curve $v(t)$:

$$v'(t) = x'(t)U_x + y'(t)U_y.$$

Curvature for planar curves

Definition

This vectorial expression has a simple intuitive meaning.

If you think of the trace of the curve as the motion of a point and t is related to time, the tangent vector defines the instantaneous motion. At any moment, the point moves with a speed given by:

$$|v'(t)| = \sqrt{x'(t)^2 + y'(t)^2}$$

in the direction:

$$\phi(t) = \tan^{-1}\left(\frac{y'(t)}{x'(t)}\right).$$

Curvature for planar curves

Definition

The *curvature* at a point $v(t)$ describes the changes in the direction $\phi(t)$ with respect to changes in arc length. That is:

$$\kappa(t) = \frac{d\phi(t)}{ds},$$

where s is arc length along the edge itself.

Curvature is given with respect to arc length because a curve parameterised by arc length maintains a constant speed of motion.

Here ϕ is the angle of the tangent to the curve.

That is, $\phi = \theta \pm 90°$, where θ is the gradient direction defined by well known formula introduced in previous part:

$$\theta(x, y) = e_{dir}(x, y) = \tan^{-1}\left(\frac{M_y(x, y)}{M_x(x, y)}\right)$$

That is, **if we apply an edge detector operator to an image, then we can compute θ to obtain a normal direction for each point in a curve. The tangent to a curve is given by an orthogonal vector.**

Curvature for planar curves

Computing

By considering that:

$$\phi(t) = \tan^{-1}\left(\frac{y'(t)}{x'(t)}\right),$$

then the curvature at a point $v(t)$ is given by:

$$\kappa(t) = \frac{x'(t)y''(t) - y'(t)x''(t)}{\left(x'(t)^2 + y'(t)^2\right)^{\frac{3}{2}}}.$$

This relationship is called the *curvature function*, and it is the standard measure of curvature for *planar* curves.

Curvature for planar curves

Computing

For curves in digital images, the derivatives must be computed from discrete data. This can be done in three main ways.

1. The most obvious approach is to calculate curvature by directly computing the difference between angular direction of successive edge pixels in a curve.

2. A second approach is to derive a measure of curvature from changes in image intensity.

3. Finally, a measure of curvature can be obtained by correlation.

Computing curvature by differences in edge direction

Computing curvature

Computing differences in edge direction

To compute curvature in digital images you can **measure the angular change along the curve's path**. This approach merely computes the difference in edge direction between connected pixels forming a discrete curve.

That is, it approximates the derivative:

$$\kappa(t) = \frac{d\phi(t)}{ds}$$

as the direction difference between neighbouring pixels. As such, curvature is simply given by:

$$k(t) = \phi_{t+1} - \phi_{t-1}$$

where the sequence $\ldots, \phi_{t-2}, \phi_{t-1}, \phi_t, \phi_{t+1}, \phi_{t+2}, \ldots$. represents the gradient direction of a sequence of pixels defining a curve segment. Gradient direction can be obtained as the angle given by an edge detector operator.

Computing curvature

Computing differences in edge direction

Alternatively, gradient direction can be computed by considering the positions of pixels in the sequence. That is, by defining:

$$\phi_t = \frac{y_{t-1} - y_{t+1}}{x_{t-1} - x_{t+1}}$$

where (x_t, y_t) denotes pixel t in the sequence. **Since edge points are only defined at discrete points, this angle can only take eight values, so the computed curvature is very ragged.** This can be smoothed by considering the difference in mean angular direction of n pixels on the leading and trailing curve segment:

$$k_n(t) = \frac{1}{n}\sum_{i=1}^{n}\phi_{t+i} - \frac{1}{n}\sum_{i=1}^{n}\phi_{t-i} = \frac{1}{n}\sum_{i=1}^{n}(\phi_{t+i} - \phi_{t-i}).$$

The average gives some immunity to noise. The value of n, defines a compromise between accuracy and noise sensitivity.

Computing curvature

Computing differences in edge direction

From the previous slide it seems to be simple process: it should be enough to use *magnitude* and *angle* computed for example with the Canny edge operator.

The problem is, how to determine pixels belonging to one curve?

The simplest option is:

1. For each pixel with coordinates $p = (x_p, y_p)$ belonging to some edge (i.e. a pixel with magnitude greater than zero) define its close neighborhood (small region surrounding it) $R(p)$, as we did in case of *nonlocal means* filter described in *Advanced smoothing filters* part.

2. Then consider all pixels from $R(p)$ as pixels from the same curve as pixel p.

Of course, this is a great oversimplification but it may works in your case.

Computing curvature

Computing differences in edge direction

The following code implements detecting curvature by angle differences. Notice that to calculate angle difference a dot product is used. See an explanation of this on next slide.

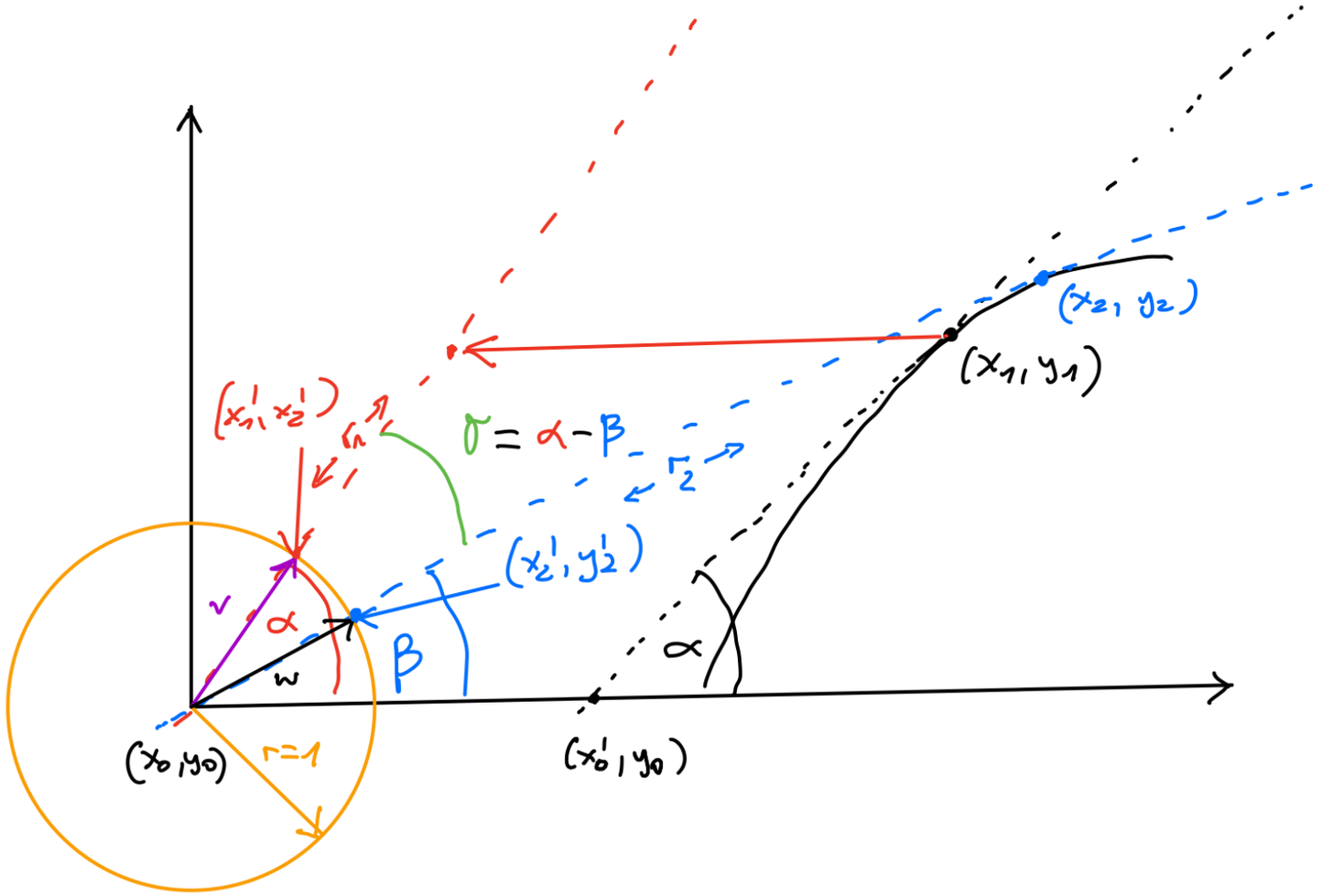
```
for x,y in itertools.product(range(0, width), range(0, height)):
  # Edge
  if magnitude[y,x] > 0:

  # Consider neighbor edges
  edgesNeigbor = []

  for wx,wy in itertools.product(range(-windowDelta, windowDelta+1), \
range(-windowDelta, windowDelta+1)):
    if magnitude[y+wy, x+wx] > 0 :
      edgesNeigbor.append((y+wy,x+wx))

# Use dot product to measure angle difference
np = len(edgesNeigbor)
for p in range(0, np):
  y1 = (edgesNeigbor[p])[0]
  x1 = (edgesNeigbor[p])[1]

  curvature[y,x] += 1.0-(cos(angle[y1,x1]) * cos(angle[y,x]) \
  + sin(angle[y1,x1]) * sin(angle[y,x]))

if np > 0:
  curvature[y,x] /= np
```

Computing curvature

Computing differences in edge direction

How to use dot product to calculate angle difference?

If we have two tangent lines, one at point (x_1, y_1) and angle α and second at point (x_2, y_2) and angle β, calculating their angle difference $\alpha - \beta$ is equivalent to calculating angle γ between them:



Computing curvature

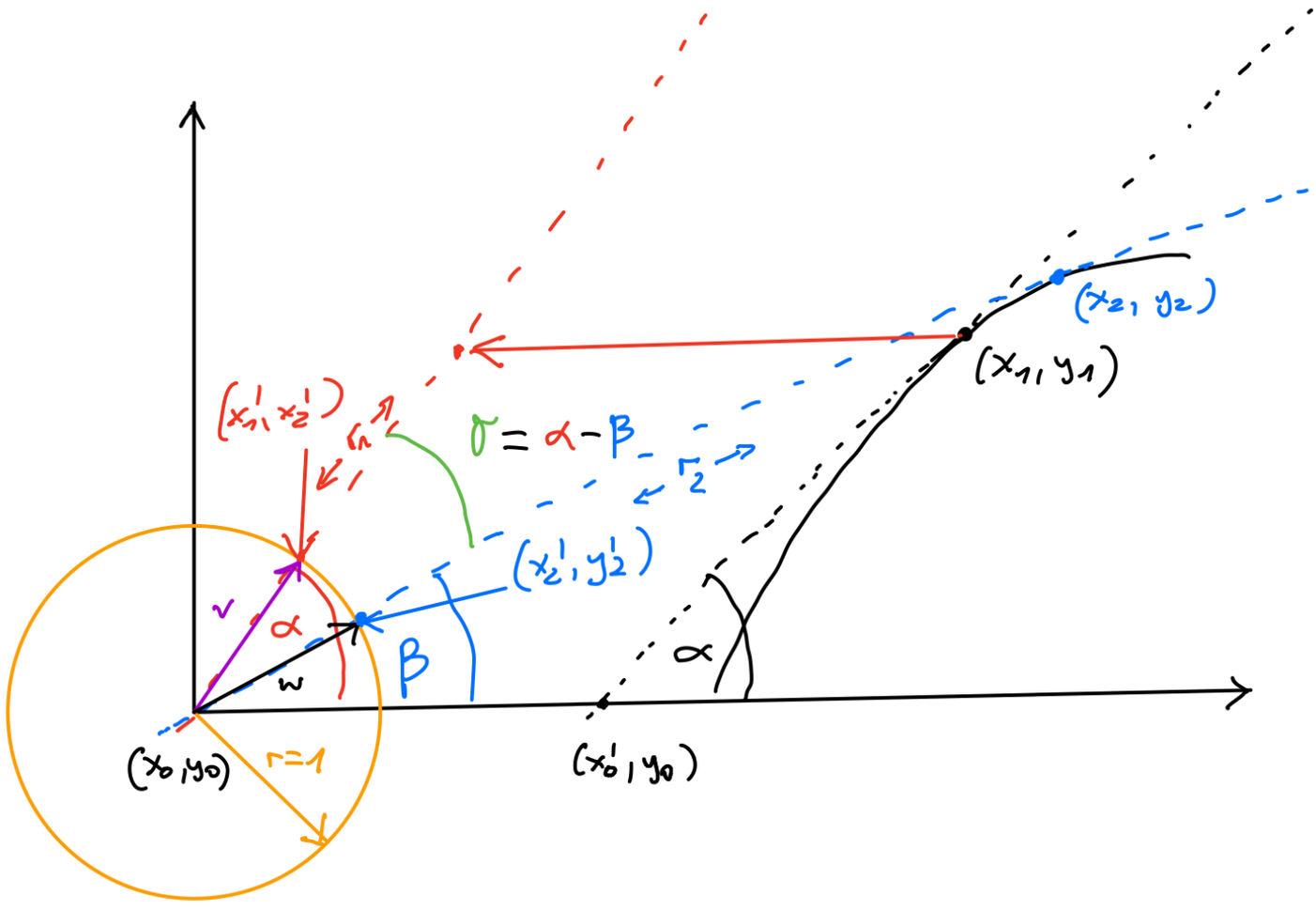
Computing differences in edge direction

Both points can be represented by the circle equation in parametric form:

$$(x_1, y_1) = (x_0' + r_1 \cos\alpha, y_0 + r_1 \sin\alpha),$$

$$(x_2, y_2) = (x_0 + r_2 \cos\beta, y_0 + r_2 \sin\beta).$$

Because we care only about directions, we can consider points (x_1', y_1') and (x_2', y_2') such that both preserves their angle and both are located at the circle centered at the point $(0,0)$, with radius equal to 1.



With this conditions:

$$(x_1', y_1') = (\cos\alpha, \sin\alpha),$$

$$(x_2', y_2') = (\cos\beta, \sin\beta)$$

and one tangent line has direction given by vector v of length 1:

$$v = [\cos\alpha, \sin\alpha]$$

while second tangent line has direction given by vector w also of length 1:

$$w = [\cos\beta, \sin\beta].$$

Computing curvature

Computing differences in edge direction

How to use dot product to calculate angle difference?

As we know, the dot product of two Euclidean vectors v and w is defined by:

$$v \cdot w = \|v\| \ \|w\| \cos\theta$$

where θ is the angle between v and w. If vectors are normalized to a length equal to 1, this formula simplifies to:

$$v \cdot w = \cos\theta.$$

Thus θ can be computed as:

$$\theta = \arccos(v \cdot w).$$

Computing curvature

Computing differences in edge direction

From previous slide you know that angle θ between two vectors v and w is given by formula:

$$\theta = \arccos(v \cdot w)$$

In our case:

$$v \cdot w = [\cos\alpha, \sin\alpha] \cdot [\cos\beta, \sin\beta] = \cos\alpha\cos\beta + \sin\alpha\sin\beta$$

so

$$\gamma = \arccos(v \cdot w) = \arccos(\cos\alpha\cos\beta + \sin\alpha\sin\beta)$$

The domain and the range of $\arccos$ are given below:

$$\arccos : [-1,1] \longrightarrow [0,\pi]$$

As you can see, $\arccos$ transforms monotonically interval $[-1,1]$ into $[0,\pi] \approx [0,3.14]$. Computing trigonometric function and their inverses is time consuming task. Saying the truth, in this case we don't need exact value -- all the transformations which are monotonic and spans in positive range of values are acceptable, as capable of expressing presence of the edge.

In our case the following transformation is used:

$$k(\alpha,\beta) = 1 - (\cos\alpha\cos\beta + \sin\alpha\sin\beta)$$

so:

$$k : [-1,1] \longrightarrow [0,2]$$

and it preserves descending nature of $\arccos$ function: if product part $\cos\alpha\cos\beta + \sin\alpha\sin\beta$ changes from -1 to 1, values of $\arccos$ changes from π to 0; the same is for k as it changes from 2 to 0.

Computing curvature by changes in intensity (differentiation)

Computing curvature

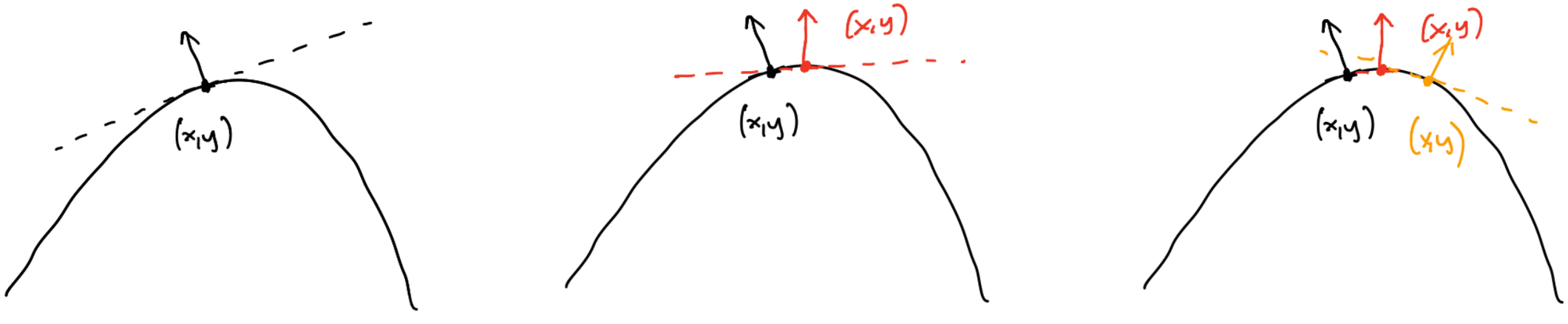
Measuring curvature by changes in intensity (differentiation)

We can derive the **curvature as a function of angular changes** in the discrete image. This derivation can be based on the measure of changes in image intensity.

We can represent the curve direction at each image point as the function $\phi(x, y)$. According to the definition of curvature:

[...] ***if we apply an edge detector operator to an image, then we can compute θ to obtain a normal direction for each point in a curve. The tangent to a curve is given by an orthogonal vector [...]***

we should compute the **change in these direction values normal to the image edge**



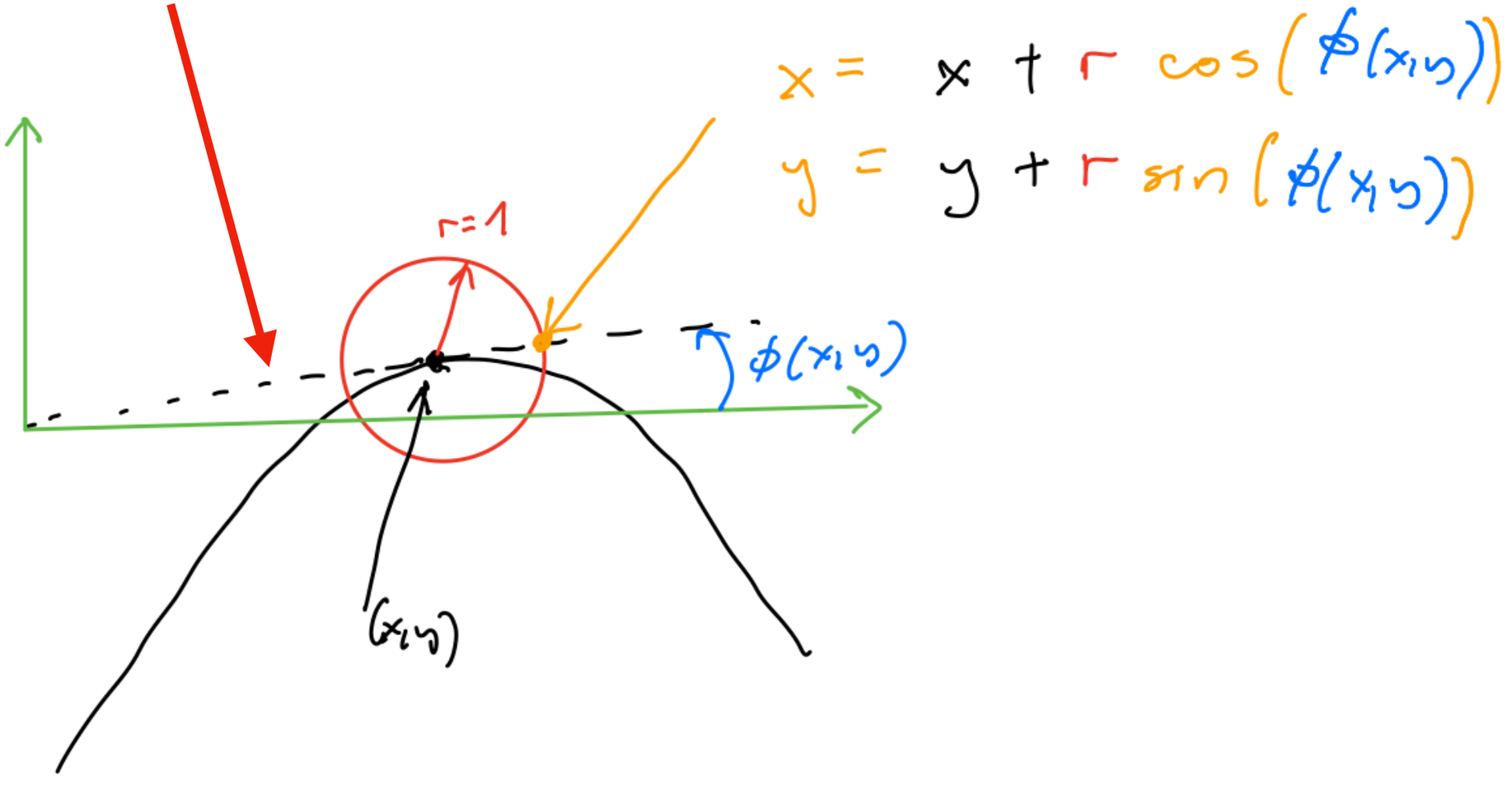
Computing curvature

Measuring curvature by changes in intensity (differentiation)

The **curve** at an edge **can be locally approximated by** the points given by the parametric **line** defined by:

$$x(t) = x + t\cos(\phi(x, y))$$

$$y(t) = y + t\sin(\phi(x, y))$$



Computing curvature

Measuring curvature by changes in intensity (differentiation)

Thus, the curvature is given by the change in the function $\phi(x, y)$ with respect to t:

$$\kappa_\phi(x, y) = \frac{\partial \phi(x, y)}{\partial t} = \frac{\partial \phi(x, y)}{\partial x}\frac{x(t)}{\partial t} + \frac{\partial \phi(x, y)}{\partial y}\frac{y(t)}{\partial t}$$

From previous material (see *Prewitt edge detection operator* in *Low-level feature extraction. Edge detection*) you know that edge magnitude and edge direction (gradient direction, so it is perpendicular to an edge) can be calculated as:

$$e_m(x, y) = \sqrt{M_x(x, y)^2 + M_y(x, y)^2},$$

$$\theta(x, y) = e_{dir}(x, y) = \tan^{-1}\left(\frac{M_y(x, y)}{M_x(x, y)}\right),$$

where M_x is the vertical template and M_y is the horizontal template defined by operator.

Computing curvature

Measuring curvature by changes in intensity (differentiation)

Very basic property of vectors in 2D space is that given a non zero vector:

$$u = [a, b]$$

the vector:

$$v = [b, -a]$$

is perpendicular to u.

The unit vector is then obtained by dividing by its norm:

$$v = (\frac{b}{\|v\|}, -\frac{a}{\|v\|}),$$

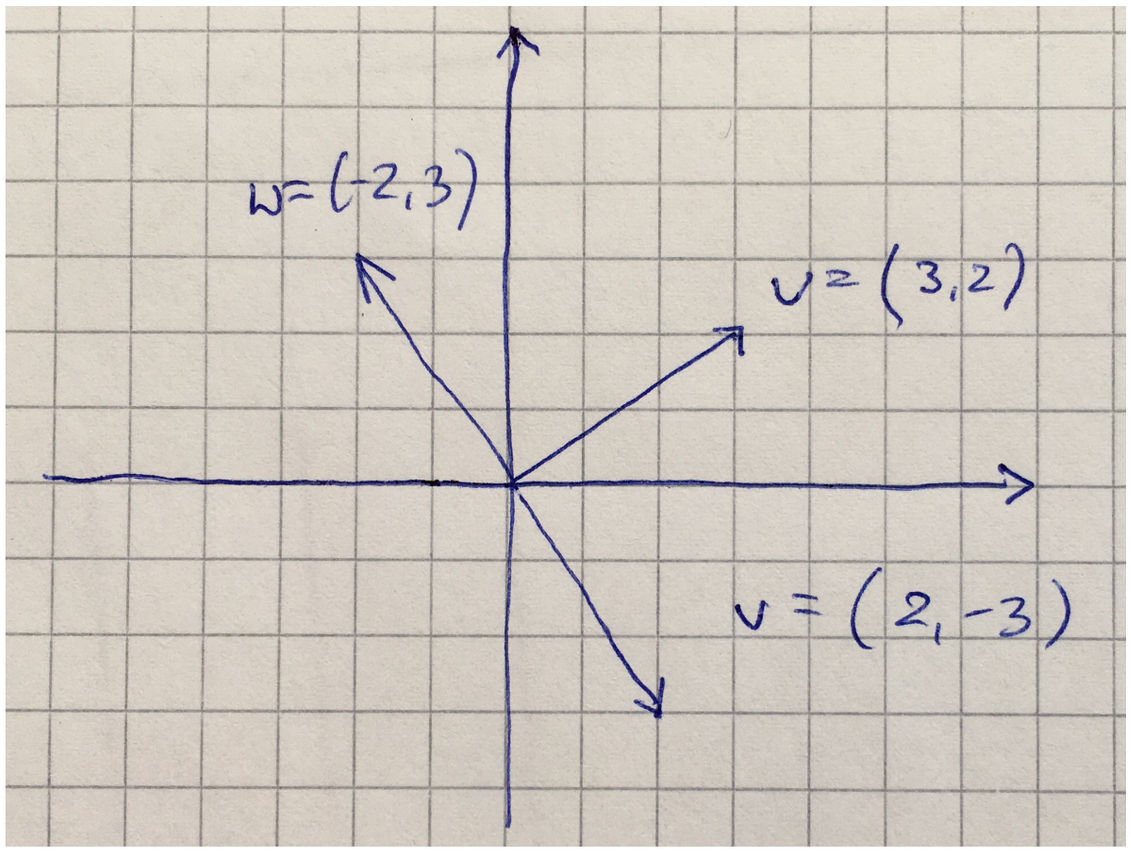
where:

$$\|v\| = \sqrt{a^2 + b^2}$$

It is worth to note, that vector:

$$w = [-b, a]$$

is the other perpendicular to u.



Computing curvature

Measuring curvature by changes in intensity (differentiation)

In consequence, if direction:

$$e_{dir}(x,y) = \tan^{-1}\left(\frac{M_y(x,y)}{M_x(x,y)}\right)$$

is defined by vector $[M_x, M_y]$ perpendicular to a curve at a point (x,y), then vector $[-M_y, M_x]$ defines direction tangent at the same point:

$$\phi(x,y) = \tan^{-1}\left(\frac{M_x(x,y)}{-M_y(x,y)}\right).$$

Because:

$$\sin(\alpha) = \frac{y}{r}, \cos(\alpha) = \frac{x}{r}, \tan(\alpha) = \frac{y}{x}, \tan^{-1}\left(\frac{y}{x}\right) = \alpha, \text{ where } r = \sqrt{x^2+y^2},$$

then:

$$\sin(\phi(x,y)) = \frac{M_x}{\sqrt{M_x^2+M_y^2}}, \cos(\phi(x,y)) = \frac{-M_y}{\sqrt{M_x^2+M_y^2}}$$

Computing curvature

Measuring curvature by changes in intensity (differentiation)

As you remember, derivative of $\tan^{-1}$ function (arcus tangens) is given by:

$$\tan^{-1}(a) = \frac{1}{1-a^2}.$$

Because in our case:

$$\phi(x,y) = \tan^{-1}\left(a(x,y)\right),$$

where:

$$a(x,y) = \frac{M_x(x,y)}{-M_y(x,y)},$$

so:

$$\frac{\partial \phi}{\partial x} = \frac{\partial \phi}{\partial a}\frac{\partial a}{\partial x} = \frac{1}{1+a(x,y)^2}\frac{\partial}{\partial x}a(x,y) = \frac{1}{1+\frac{M_x^2}{\left(-M_y\right)^2}}\frac{-\frac{\partial M_x}{\partial x}M_y + M_x\frac{\partial M_y}{\partial x}}{M_y^2} = \frac{-\left(M_y\frac{\partial M_x}{\partial x} - M_x\frac{\partial M_y}{\partial x}\right)}{M_y^2+M_x^2}$$

$$\frac{\partial \phi}{\partial y} = \frac{\partial \phi}{\partial a}\frac{\partial a}{\partial y} = \frac{1}{1+a(x,y)^2}\frac{\partial}{\partial y}a(x,y) = \frac{1}{1+\frac{M_x^2}{\left(-M_y\right)^2}}\frac{-\frac{\partial M_x}{\partial y}M_y + M_x\frac{\partial M_y}{\partial y}}{M_y^2} = \frac{M_x\frac{\partial M_y}{\partial y} - M_y\frac{\partial M_x}{\partial y}}{M_y^2+M_x^2}$$

Computing curvature

Measuring curvature by changes in intensity (differentiation)

Both derivatives calculated in previous slide:

$$\frac{\partial \phi}{\partial x} = \frac{-1}{M_y^2 + M_x^2}\left(M_y \frac{\partial M_x}{\partial x} - M_x \frac{\partial M_y}{\partial x}\right)$$

$$\frac{\partial \phi}{\partial y} = \frac{1}{M_y^2 + M_x^2}\left(M_x \frac{\partial M_y}{\partial y} - M_y \frac{\partial M_x}{\partial y}\right)$$

are needed to calculate curvature:

$$\kappa_\phi(x,y) = \frac{\partial \phi(x,y)}{\partial t} = \frac{\partial \phi(x,y)}{\partial x}\frac{\partial x(t)}{\partial t} + \frac{\partial \phi(x,y)}{\partial y}\frac{\partial y(t)}{\partial t}$$

Two other missing component can be easily calculated from previously given definitions:

$$x(t) = x + t\cos(\phi(x,y))$$

$$y(t) = y + t\sin(\phi(x,y))$$

In consequence respective derivatives are given by:

$$\frac{\partial x(t)}{\partial t} = \cos(\phi(x,y)),$$

$$\frac{\partial y(t)}{\partial t} = \sin(\phi(x,y))$$

and finally:

$$\frac{\partial x(t)}{\partial t} = \frac{-M_y}{\sqrt{M_x^2 + M_y^2}},$$

$$\frac{\partial y(t)}{\partial t} = \frac{M_x}{\sqrt{M_x^2 + M_y^2}}.$$

Computing curvature

Measuring curvature by changes in intensity (differentiation)

Last step is to combine all the components:

$$\frac{\partial \phi}{\partial x} = \frac{-1}{M_y^2 + M_x^2}\left(M_y \frac{\partial M_x}{\partial x} - M_x \frac{\partial M_y}{\partial x}\right), \frac{\partial \phi}{\partial y} = \frac{1}{M_y^2 + M_x^2}\left(M_x \frac{\partial M_y}{\partial y} - M_y \frac{\partial M_x}{\partial y}\right)$$

$$\frac{\partial x(t)}{\partial t} = \frac{-M_y}{\sqrt{M_x^2 + M_y^2}}, \frac{\partial y(t)}{\partial t} = \frac{M_x}{\sqrt{M_x^2 + M_y^2}}.$$

in curvature formula:

$$\kappa_\phi(x,y) = \frac{\partial \phi(x,y)}{\partial t} = \frac{\partial \phi(x,y)}{\partial x}\frac{\partial x(t)}{\partial t} + \frac{\partial \phi(x,y)}{\partial y}\frac{\partial y(t)}{\partial t}$$

$$\kappa_\phi(x,y) = \frac{-1}{M_y^2 + M_x^2}\left(M_y \frac{\partial M_x}{\partial x} - M_x \frac{\partial M_y}{\partial x}\right)\frac{-M_y}{\sqrt{M_x^2 + M_y^2}} + \frac{1}{M_y^2 + M_x^2}\left(M_x \frac{\partial M_y}{\partial y} - M_y \frac{\partial M_x}{\partial y}\right)\frac{M_x}{\sqrt{M_x^2 + M_y^2}}$$

$$= \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(M_y^2 \frac{\partial M_x}{\partial x} - M_x M_y \frac{\partial M_y}{\partial x}\right) + \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(M_x^2 \frac{\partial M_y}{\partial y} - M_x M_y \frac{\partial M_x}{\partial y}\right)$$

$$= \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(M_y^2 \frac{\partial M_x}{\partial x} - M_x M_y \frac{\partial M_y}{\partial x} + M_x^2 \frac{\partial M_y}{\partial y} - M_x M_y \frac{\partial M_x}{\partial y}\right)$$

+

Computing curvature

Measuring curvature by changes in intensity (differentiation)

The formula:

$$\kappa_{\phi}(x,y) = \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(M_y^2\frac{\partial M_x}{\partial x} - M_x M_y\frac{\partial M_y}{\partial x} + M_x^2\frac{\partial M_y}{\partial y} - M_x M_y\frac{\partial M_x}{\partial y}\right)$$

defines a *forward* measure of curvature along the edge direction. We can actually use an alternative direction to the measure of curvature. We can differentiate *backwards* (in the direction of $-\phi(x,y)$ which gives $\kappa_{-\phi}(x,y)$. In this case, we consider that the curve is given by:

$$x(t) = x + t\cos(-\phi(x,y))$$

$$y(t) = y + t\sin(-\phi(x,y))$$

Thus:

$$\kappa_{-\phi}(x,y) = \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(M_y^2\frac{\partial M_x}{\partial x} - M_x M_y\frac{\partial M_y}{\partial x} - M_x^2\frac{\partial M_y}{\partial y} + M_x M_y\frac{\partial M_x}{\partial y}\right)$$

Computing curvature

Measuring curvature by changes in intensity (differentiation)

Two further measures can be obtained by considering the forward and a backward differential along the normal. These differentials cannot be related to the actual definition of curvature, but can be explained intuitively. If we consider that curves are more than one pixel wide, differentiation along the edge will measure the difference between the gradient angle between interior and exterior borders of a wide curve. In theory, the tangent angle should be the same. However, in discrete images there is a change due to the measures in a window. If the curve is a straight line, then the interior and exterior borders are the same. Thus, gradient direction normal to the edge does not change locally. As we bend a straight line, we increase the difference between the curves defining the interior and exterior borders. Thus, we expect the measure of gradient direction to change. That is, if we differentiate along the normal direction, we maximise detection of gross curvature.

Computing curvature

Measuring curvature by changes in intensity (differentiation)

The value $\kappa_{\perp\phi}(x, y)$ is obtained when:

$$x(t) = x + t\sin(\phi(x, y))$$

$$y(t) = y + t\cos(\phi(x, y))$$

Thus:

$$\kappa_{\perp\phi}(x, y) = \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(M_x^2\frac{\partial M_y}{\partial x} - M_x M_y\frac{\partial M_y}{\partial x} - M_x M_y\frac{\partial M_y}{\partial y} + M_y^2\frac{\partial M_x}{\partial y}\right)$$

In a backward formulation along a normal direction to the edge, we obtain:

$$\kappa_{-\perp\phi}(x, y) = \frac{1}{\left(M_x^2 + M_y^2\right)^{\frac{3}{2}}}\left(-M_x^2\frac{\partial M_y}{\partial x} + M_x M_y\frac{\partial M_x}{\partial x} - M_x M_y\frac{\partial M_y}{\partial y} + M_y^2\frac{\partial M_x}{\partial y}\right)$$

.

Computing curvature

Measuring curvature by changes in intensity (differentiation)

Code below shows an implementation of the four measures of curvature. The arrays `Mx` and `My` store the gradient obtained by the convolutions of the original image with Sobel kernels in horizontal and vertical directions. The arrays `Mxx`, `Mxy`, `Myx` and `Myy` contain the convolutions of `Mx` and `My` with Sobel kernels.

```
for x,y in itertools.product(range(0, width), range(0, height)):
  # If it is an edge
  if magnitude[y,x] > 0:
    Mx2,My2,MxMy = mX[y,x]*mX[y,x], mY[y,x]*mY[y,x], mX[y,x]*mY[y,x]

    if Mx2 + My2 !=0.0:
      p = 1.0/ pow((Mx2 + My2), 1.5)

    elif op == "T":
      curvature[y,x] = p * (My2 * mXx[y,x] - MxMy * mYx[y,x] + \
      Mx2 * mYy[y,x] - MxMy * mXy[y,x])

    elif op == "TI":
      curvature[y,x] = p * (-My2 * mXx[y,x] + MxMy * mYx[y,x] - \
      Mx2 * mYy[y,x] + MxMy * mXy[y,x])

    elif op == "N":
      curvature[y,x] = p * (Mx2 * mYx[y,x] - MxMy * mYx[y,x] - \
      MxMy * mYy[y,x] + My2 * mXy[y,x])

    else: #if op == "NI":
      curvature[y,x] = p * (-Mx2 * mYx[y,x] + MxMy * mXx[y,x] + \
      MxMy * mYy[y,x] - My2 * mXy[y,x])

    curvature[y,x] = fabs(curvature[y,x])
```

Computing curvature by computing correlation

Computing curvature

Measuring curvature by computing correlation

Corners are regions in the image with large variation in intensity in all the directions.

Harris corner detector -- see [2, 3, 4, 5]

Bibliography

Bibliography

1. How to Find Perpendicular Vectors in 2 Dimensions, https://www.wikihow.com/Find-Perpendicular-Vectors-in-2-Dimensions retrieved, 2021-11-21

2. Harris Corner Detection, https://theailearner.com/2021/09/25/harris-corner-detection/ retrieved, 2021-11-21

3. Show that the determinant of A is equal to the product of its eigenvalues, https://math.stackexchange.com/q/507660 retrieved, 2021-11-21

4. Trace equals sum of eigenvalues in Trace (linear algebra), https://en.wikipedia.org/wiki/Trace_(linear_algebra)#Trace_equals_sum_of_eigenvalues retrieved, 2021-11-21

5. Mark S. Nixon, Alberto S. Aguado, *Feature Extraction and Image Processing for Computer Vision*, Academic Press (Elsevier), 2020 (4th edition), Section: 4.4.1.4 Moravec and Harris detectors