```
begin
    using PlutoUI
    using Plots
    using Images
    using LinearAlgebra
end
```

step (generic function with 1 method)

```
function step(x)
    if x>0
        return 1.0
    else
        return 0.0
    end
end
```

momentum_of_inertia (generic function with 1 method)

```
function momentum_of_inertia(img::Matrix{RGB{N0f8}})
    # Preprocess
    A = step.(1 .- Gray.(img))
    m = size(A)

    # Evaluate Centroid
    sum_x = 0
    sum_y = 0
    for ix in 1:m[2]
        for iy in 1:m[1]
            x = ix
            y = m[1] - iy + 1
            sum_x += x*A[iy,ix]
            sum_y += y*A[iy,ix]
        end
    end
    x0 = sum_x / sum(A)
    y0 = sum_y / sum(A)

    # Evaluate Moment of Inertia
    Ix = 0
    Iy = 0
    Ixy = 0
    for ix in 1:m[2]
        for iy in 1:m[1]
            x = ix
            y = m[1] - iy + 1
            Ix += (y - y0)^2 * A[iy,ix]
            Iy += (x - x0)^2 * A[iy,ix]
            Ixy += (x - x0) * (y - y0) * A[iy,ix]
        end
    end

    # Normalize
    Ix = Ix / sum(A)
    Iy = Iy / sum(A)
    Ixy = Ixy / sum(A)

    # Search of Symmetry (Gradient Descent)
    sita = 0
    for n in 1:1000
        Ix_new = [Ix Iy Ixy] * [cos(sita)^2 sin(sita)^2 -2 * sin(sita) * cos(sita)]'
        tangent = - sin(2 * sita) * Ix + sin(2 * sita) * Iy -2 * cos(2 * sita) * Ixy
        sita -= 0.02 * tangent
    end

    # Evaluate Components
    I_p = 0
    I_n = 0
    for ix in 1:m[2]
        for iy in 1:m[1]
            x = ix
            y = m[1] - iy + 1
```

```
            x_new = (cos(sita) * (x-x0) + sin(sita) * (y-y0)) * A[iy,ix] / sum(A)
            if x_new > 0
                I_p += x_new^2
            end
            if x_new < 0
                I_n += x_new^2
            end
        end
    end

    O_p = 0
    O_n = 0
    for ix in 1:m[2]
        for iy in 1:m[1]
            x = ix
            y = m[1] - iy + 1
            x_new = (cos(sita + pi/2) * (x-x0) + sin(sita + pi/2) * (y-y0)) *
            A[iy,ix] / sum(A)
            if x_new > 0
                O_p += x_new^2
            end
            if x_new < 0
                O_n += x_new^2
            end
        end
    end

    # Make Decision
    if abs(I_p - I_n) > abs(O_p - O_n)
        if I_p < I_n
            angle = sita
        else
            angle = sita + pi
        end
    else
        if O_p < O_n
            angle = sita + pi/2
        else
            angle = sita + pi*3/2
        end
    end

    return angle

end
```