Hough Transform:

تبدیل هاف (Hough Transform) یک روش ریاضی و الگوریتمی در پردازش تصویر است که برای تشخیص اشکال هندسی ساده مانند خطوط، دایره‌ها و بیضی‌ها استفاده می‌شود. ایده‌ی اصلی این روش بر پایه‌ی نگاشت نقاط تصویر به یک فضای پارامتری است. به‌طور مثال، یک خط در تصویر را می‌توان در مختصات کارتزین با معادله $y=mx+b$ توصیف کرد. تبدیل هاف هر نقطه از لبه‌ی تصویر که معمولاً توسط الگوریتم‌هایی مثل Canny شناسایی شده است را به یک منحنی در فضای پارامترها منتقل می‌کند. محل‌هایی که این منحنی‌ها همدیگر را قطع می‌کنند، نشان‌دهنده‌ی وجود یک شکل هندسی واقعی در تصویر است.

مزیت بزرگ تبدیل هاف این است که حتی در شرایطی که یک شکل در تصویر ناقص، نویزی یا دارای اشکال باشد، همچنان می‌تواند آن را شناسایی کند. برای مثال، در یک تصویر جاده‌ای که خطوط وسط جاده به‌صورت مقطع و نیمه‌پاک شده دیده می‌شوند، الگوریتم هاف همچنان قادر است امتداد این خطوط را تشخیص دهد. دلیل این توانایی، رویکرد تجمعی یا رأی‌گیری (voting) در فضای پارامترهاست، یعنی هر نقطه‌ی لبه‌ای به تمام شکل‌های ممکن رأی می‌دهد و در نهایت شکلی انتخاب می‌شود که بیشترین رأی را کسب کرده است.

در پیاده‌سازی عملی در OpenCV ، تبدیل هاف به دو شکل اصلی ارائه می‌شود: تشخیص خطوط و تشخیص دایره‌ها. در حالت خطوط، توابعی مانند `cv2.HoughLines` و `cv2.HoughLinesP` استفاده می‌شوند. نسخه‌ی احتمالی یا Probabilistic Hough یعنی `HoughLinesP` سریع‌تر است و انتهای دقیق خطوط را مشخص می‌کند، در حالی که نسخه‌ی استاندارد فقط زاویه و فاصله‌ی خطوط از مرکز مختصات را بازمی‌گرداند. در حالت دایره، از `cv2.HoughCircles` استفاده می‌شود که علاوه بر تشخیص مکان مرکز، شعاع دایره را نیز محاسبه می‌کند.

به طور کلی، تبدیل هاف یک ابزار قدرتمند در بینایی ماشین و پردازش تصویر است، به‌ویژه در کاربردهایی مثل بینایی رباتیک، سیستم‌های کمک‌راننده خودرو، پردازش نقشه‌ها و تشخیص اشیاء. این الگوریتم اگرچه محاسباتی نسبتاً سنگین دارد، اما دقت و توانایی‌اش در شناسایی اشکال حتی در تصاویر با کیفیت پایین باعث شده همچنان پرکاربرد باشد. ترکیب این روش با تکنیک‌های دیگر مثل فیلترگذاری یا یادگیری ماشین، امکان ایجاد سیستم‌های دقیق‌تر و سریع‌تر را فراهم می‌کند.

تابع `cv2.HoughLines` نسخه‌ی استاندارد تبدیل هاف است که خطوط را در فضای پارامترها به‌صورت (ρ,θ) برمی‌گرداند. در این روش هر خط به شکل بی‌نهایت امتداد داده شده در تصویر توصیف می‌شود و خود کاربر باید با استفاده از این پارامترها آن را رسم کند. دقت بالایی دارد اما محاسبات سنگین‌تری نسبت به نسخه‌ی احتمالی انجام می‌دهد و برای مواقعی مناسب است که صرفاً نیاز به تشخیص حضور خطوط داریم، نه طول و موقعیت دقیق آن‌ها.

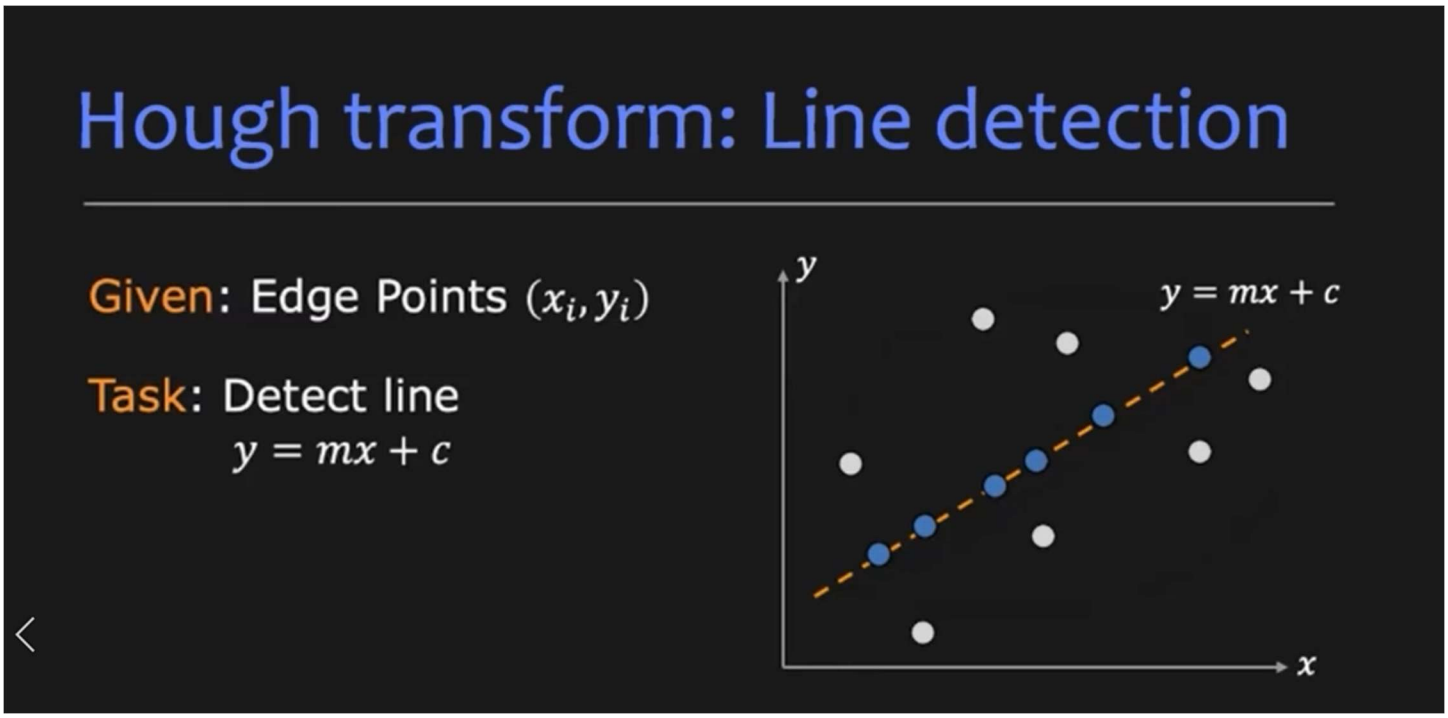
در مقابل، `cv2.HoughLinesP` نسخه‌ی احتمالی (Probabilistic) است که مستقیماً مختصات ابتدا و انتهای خطوط $(x1, y1, x2, y2)$ را برمی‌گرداند. این روش سریع‌تر است چون همه‌ی نقاط لبه را بررسی نمی‌کند و بیشتر

برای کاربردهای عملی مثل تشخیص خطوط جاده یا اشیای خطی مناسب است. نتیجه‌ی این روش خط‌های واقعی و محدود در تصویر است، نه خطوط بی‌نهایت، بنابراین استفاده‌ی آن برای پردازش و تحلیل در مراحل بعدی ساده‌تر خواهد بود.

y=mx+b :

معادله‌ی **y=mx+b** یک **فرم خطی** در هندسه و ریاضیات است که برای توصیف خط مستقیم در صفحه‌ی دوبعدی به کار می‌رود. در این معادله:

- xو yمتغیرهای مستقل و وابسته هستند (مختصات یک نقطه روی خط).
- mضریب شیب (slope) است که میزان **تندی یا زاویه‌ی خط** نسبت به محور x را مشخص می‌کند.
- bعرض از مبدأ (intercept) است و نشان می‌دهد که خط در چه نقطه‌ای محور y را قطع می‌کند.



با استفاده از این متد می توانیم تبدیل هاف خطی را انجام دهیم:

```
lines = cv2.HoughLines(edges,1,np.pi/180,125)
```

- روی خروجی لبه‌ها تبدیل هاف اجرا می‌شه.
- پارامترها:
  - 1 دقت فاصله‌ی ρ\rhoρ پیکسل.
  - np.pi/180 دقت زاویه‌ی thetaθ یک درجه.
  - 125 آستانه‌ی حداقل رأی لازم برای اینکه یک خط معتبر محسوب بشه.

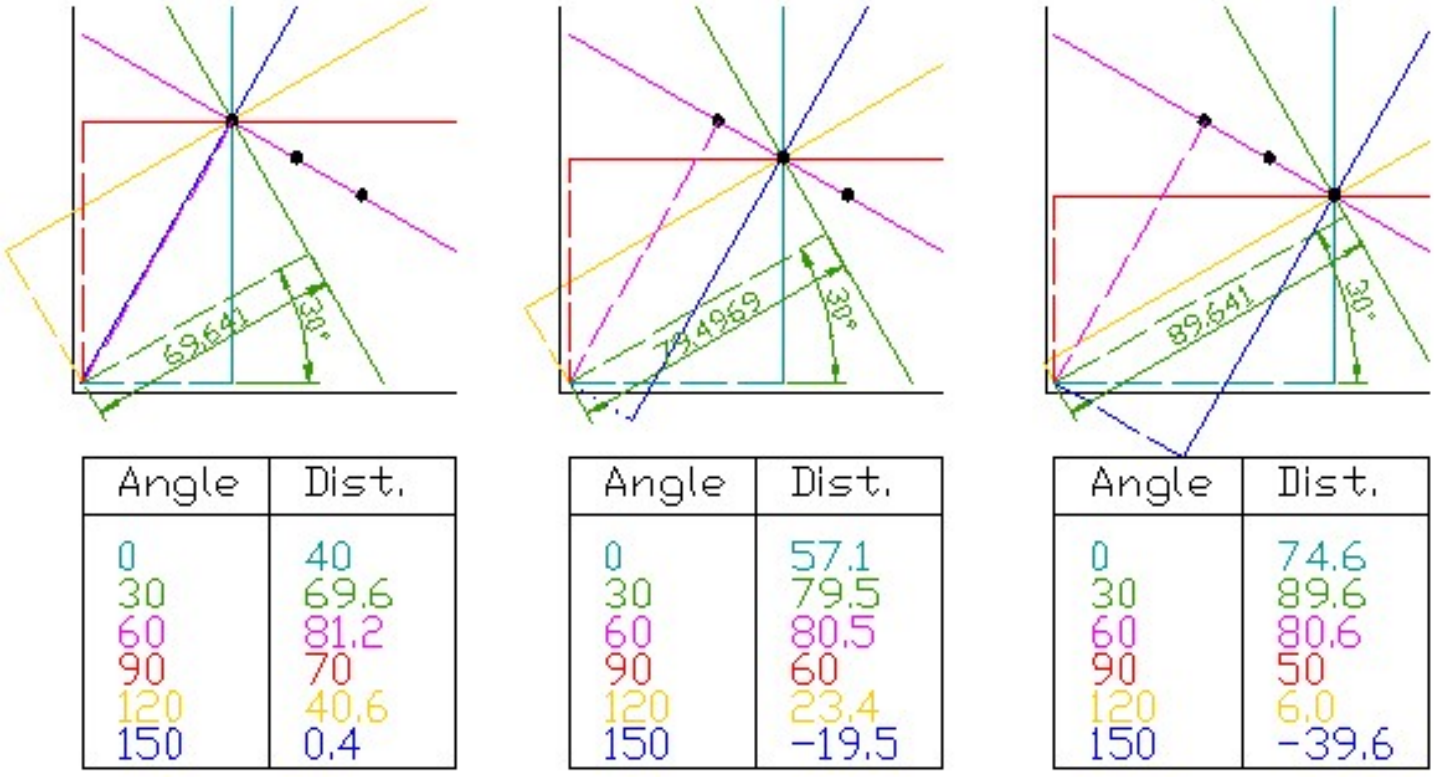
```
img = cv2.imread('images/sudoku.jpg')
gray = cv2.cvtColor(img,cv2.COLOR_BGR2GRAY)
edges = cv2.Canny(gray,50,150)
lines = cv2.HoughLines(edges,1,np.pi/180,125)
for line in lines:
    rho,theta = line[0]
    a = np.cos(theta)
    b = np.sin(theta)
    x0 = a*rho
    y0 = b*rho
    x1 = int(x0 + 1000*(-b))
    y1 = int(y0 + 1000*(a))
    x2 = int(x0 - 1000*(-b))
    y2 = int(y0 - 1000*(a))
    cv2.line(img,(x1,y1),(x2,y2),(0,0,255),2)
plt.imshow(img[...,::-1])
```

این کد ابتدا تصویر را از مسیر داده‌شده می‌خوانَد و برای ساده‌تر شدن پردازش، آن را از فضای رنگی BGR پیش‌فرض OpenCV به تصویر تک‌کاناله خاکستری تبدیل می‌کند. تبدیل به خاکستری باعث می‌شود اطلاعات شدت روشنایی حفظ شود و محاسبات بعدی (که بر پایه تغییرات شدت است) سریع‌تر و پایدارتر انجام شوند. این گام مقدمه تمام روش‌های کلاسیک تشخیص لبه/شکل است و معمولاً قبل از آن می‌توان یک فیلتر ملایم‌سازی مثل (GaussianBlur هم اعمال کرد تا نویزهای ریز کمتر در نتایج اثر بگذارند.

در گام بعد، آشکارساز لبه Canny با آستانه‌های 50 و 150 اجرا می‌شود .Canny عملاً گرادیان شدت را محاسبه می‌کند، نقاطی را که تغییر شدت بزرگی دارند به‌عنوان «لبه» نگه می‌دارد و بقیه را حذف می‌کند. دو آستانه پایین/بالا کنترل می‌کنند که کدام گرادیان‌ها به‌صورت قطعی پذیرفته، کدام رد و کدام هم با «هیسترزیس» بررسی شوند. خروجی این مرحله یک تصویر دودویی از لبه‌هاست که ورودی ایدئال برای تبدیل هاف محسوب می‌شود، چون هاف فقط به مختصات نقاط لبه نیاز دارد.

سپس `cv2.HoughLines`روی نقشه لبه‌ها اعمال می‌شود. این تابع به‌صورت رأی‌گیری در فضای پارامتری (ρ,θ)(\rho,\theta)کار می‌کند: هر نقطه لبه به تمام خطوط ممکنی که می‌تواند از آن عبور کند رأی می‌دهد. دقت فاصله (پارامتر اول) روی 1 پیکسل و دقت زاویه (پارامتر دوم) روی یک درجه $(\pi/180)$(\pi/180)تنظیم شده است. آستانه 125 حداقل تعداد رأی لازم برای معتبر دانستن یک خط است. خروجی، آرایه‌ای از زوج‌های (ρ,θ)(\rho,\theta)است که هرکدام نماینده یک خطِ کشف‌شده در تصویرند؛ با برداشتن `line[0]` به این دو پارامتر دسترسی پیدا می‌کنیم.

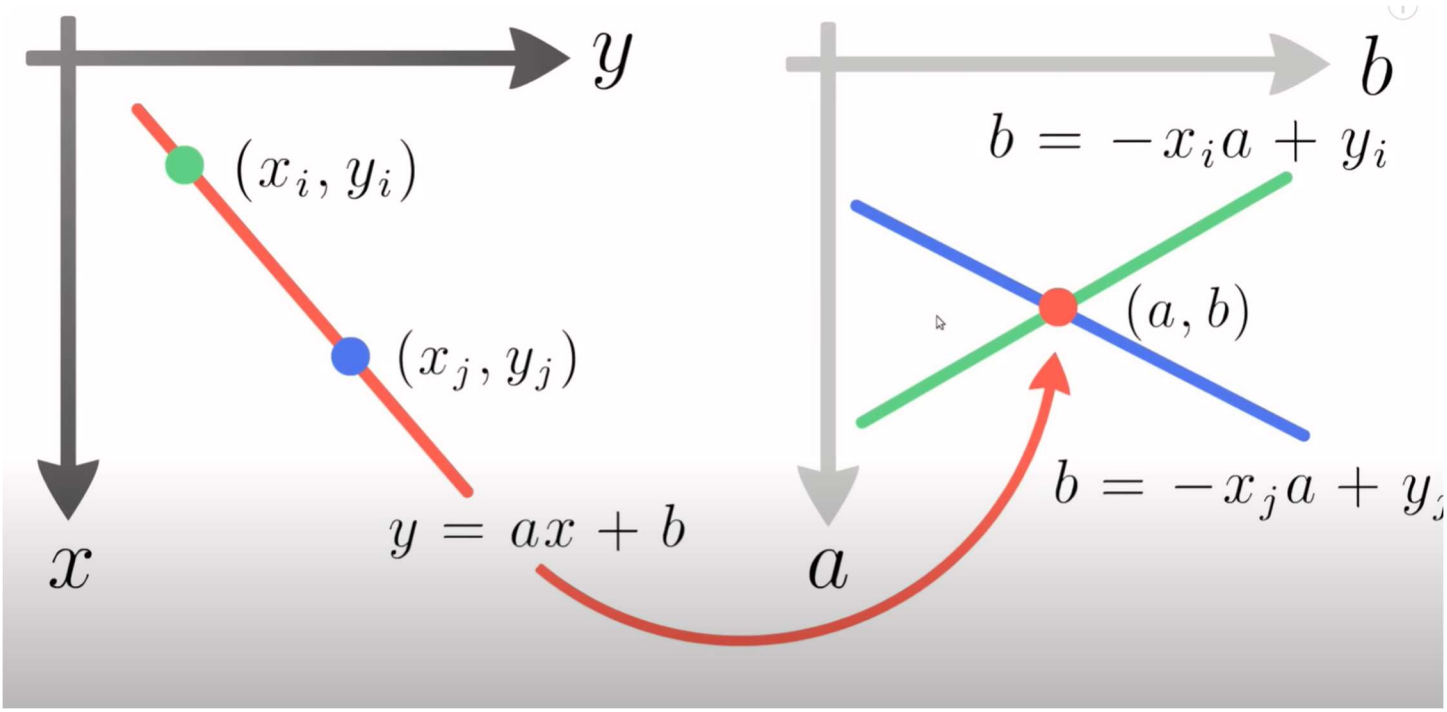
این حلقه یکی یکی روی خطوطی که تابع هاف پیدا کرده حرکت می‌کند. در هر تکرار، ابتدا مقادیر ρ\rhoو θ\thetaمربوط به همان خط گرفته می‌شود؛ یعنی فاصله خط از مبدأ و زاویه آن. بعد با استفاده از کسینوس و سینوس زاویه، بردار نرمال خط ساخته می‌شود. سپس نقطه‌ای روی خط به نام $(x0,y0)$(x_0,y_0)محاسبه می‌شود که دقیقاً روی همان خط قرار دارد. برای اینکه بتوانیم خط را روی تصویر بکشیم، باید دو نقطه دورتر روی همان خط داشته باشیم. بنابراین با اضافه و کم کردن یک مقدار بزرگ (اینجا 1000 پیکسل) در راستای عمود بر خط، دو نقطه $(x1,y1)$(x_1,y_1)و $(x2,y2)$(x_2,y_2)ساخته می‌شوند. در نهایت، تابع `cv2.line`این دو نقطه را به‌هم وصل می‌کند و یک خط قرمز ضخامت‌دار روی تصویر رسم می‌شود.



Angle	Dist.
0	40
30	69.6
60	81.2
90	70
120	40.6
150	0.4

Angle	Dist.
0	57.1
30	79.5
60	80.5
90	60
120	23.4
150	-19.5

Angle	Dist.
0	74.6
30	89.6
60	80.6
90	50
120	6.0
150	-39.6



هاف احتمالی:

```
lines = cv2.HoughLinesP(opening, 1, np.pi/180 , lineThresh, None, minLineLength, maxLineGap)
```

تابع `cv2.HoughLinesP` نسخه‌ی احتمالی تبدیل هاف است و ورودی‌هایش هرکدام نقشی مهم در نتیجه‌ی نهایی دارند.
توضیح پارامترها:

1. **edges**
   این همان تصویر دودویی لبه‌هاست خروجی الگوریتمی مثل Canny. فقط نقاط لبه (سفید) بررسی می‌شوند تا مشخص شود کدامشان روی یک خط قرار می‌گیرند.
2. **1**
   این پارامتر مربوط به دقت فاصله‌ی ρ است. یعنی هر بار که در فضای هاف رأی‌گیری می‌شود، فاصله‌ها با چه دقتی محاسبه شوند. مقدار 1 یعنی دقت یک پیکسل.
3. **np.pi/180**
   این دقت زاویه θ است. مقدار `np.pi/180` یعنی زاویه‌ها با گام 1 درجه بررسی شوند. هرچه این مقدار کوچک‌تر باشد (مثلاً نیم‌درجه)، دقت بالاتر ولی سرعت کمتر خواهد شد.

4. **50**

این آستانه‌ی حداقل رأی است. خطی فقط وقتی معتبر شناخته می‌شود که حداقل 50 نقطه از لبه‌ها روی آن بیفتند. اگر این مقدار خیلی کم باشد، خطوط زیادی (حتی ناخواسته) پیدا می‌شوند؛ اگر خیلی زیاد باشد، ممکن است بعضی خطوط واقعی تشخیص داده نشوند.

5. **minLineLength=20**

حداقل طول یک خط معتبر است. یعنی اگر خط کشف‌شده کوتاه‌تر از 20 پیکسل باشد، نادیده گرفته می‌شود. این کمک می‌کند خط‌های خیلی کوچک و نویزی حذف شوند.

6. **maxLineGap=50**

بیشترین فاصله‌ی مجاز بین دو بخش از یک خط است تا همچنان یک خط واحد محسوب شوند. مثلاً اگر خطی وسطش شکسته یا ناقص باشد، اگر فاصله‌ی شکاف کمتر از 50 پیکسل باشد، این دو قسمت به‌عنوان یک خط در نظر گرفته می‌شوند.

```
7. for line in lines:
8.     x1, y1, x2, y2 = line[0]
9.     cv2.line(img, (x1, y1), (x2, y2), (255, 0, 0), 3)
```

تابع `cv2.HoughLinesP` یک آرایه برمی‌گرداند که در آن هر عنصر مربوط به یک خط کشف‌شده است. این حلقه روی همه این خطوط حرکت می‌کند و هر بار یک خط را از مجموعه می‌گیرد.

هر خطی که برگردانده می‌شود به‌صورت یک آرایه چهارتایی است که مختصات **ابتدای خط (x1, y1)** و **انتهای خط (x2, y2)** را نگه می‌دارد. با این دستور، این چهار مقدار از داخل `line[0]` استخراج و در متغیرهای جداگانه ذخیره می‌شوند.

تابع `cv2.line` یک خط روی تصویر رسم می‌کند.

- `Img` تصویر مقصد که همان کپی از تصویر اصلی است.
- `(x1, y1)` و `(x2, y2)`: مختصات نقطه شروع و پایان خط.
- `(255, 0, 0)` رنگ خط در فضای BGR است ← یعنی آبی خالص.
- `3` ضخامت خط به واحد پیکسل.