





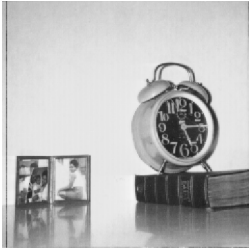

Kittipong Tapyou 65070501003

Homework 06 : Hough Transform

Load Image

```
In[1]:= imgs = { , , ,  };
```

```
grayImgs = ColorConvert[#, "Grayscale"] & /@ imgs
```

Out[2]= { , , ,  }

Preprocessing : Noise Reduction





```
In[3]:= gaussianFilter = 1/16 * {{1, 2, 1}, {2, 4, 2}, {1, 2, 1}};
```

```
gaussianFilter // MatrixForm
```

Out[4]//MatrixForm=

$$\begin{pmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{pmatrix}$$

```
In[5]:= denoiseImgs = ImageConvolve[#, gaussianFilter] & /@ grayImgs
```

Out[5]= { , , ,  }

ทำการ denoise image โดยการทำให้ gaussian blur โดยใช้ filter ขนาด 3*3 เพื่อไม่ให้กรณีของการทำ

highpass นั้นมีจุดหรือ noise ปรากฏ

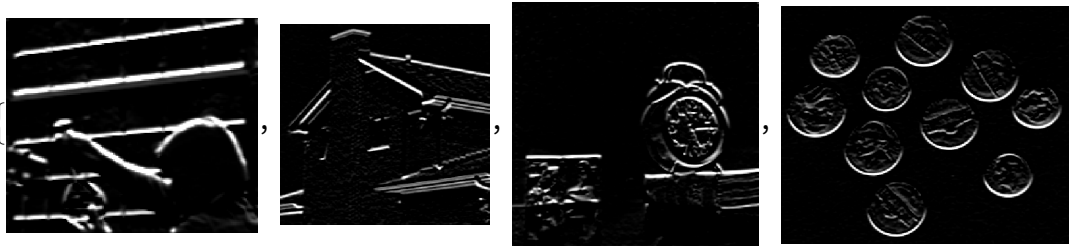
Gradient Computation

1st derivative : Sobel

```
In[6]:= SobelX = {{-1, -2, -1}, {0, 0, 0}, {1, 2, 1}};
SobelY = {{-1, 0, 1}, {-2, 0, 2}, {-1, 0, 1}};
```

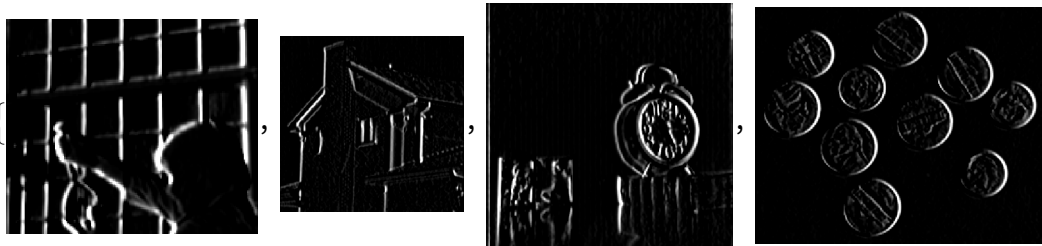
```
In[8]:= Gx = ImageConvolve[#, SobelX] & /@ denoiseImgs
```

Out[8]=



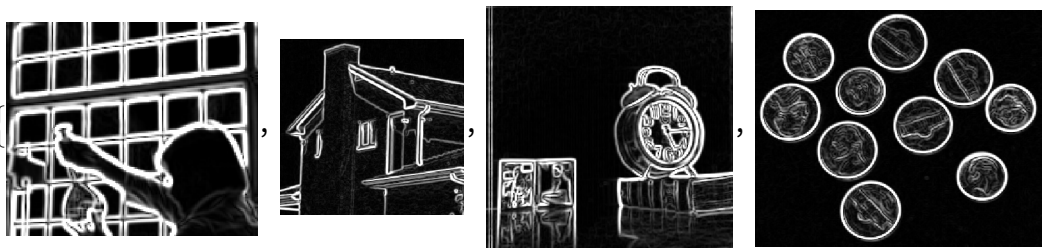
```
In[9]:= Gy = ImageConvolve[#, SobelY] & /@ denoiseImgs
```

Out[9]=



```
In[10]:= G = Sqrt[Gx^2 + Gy^2]
```

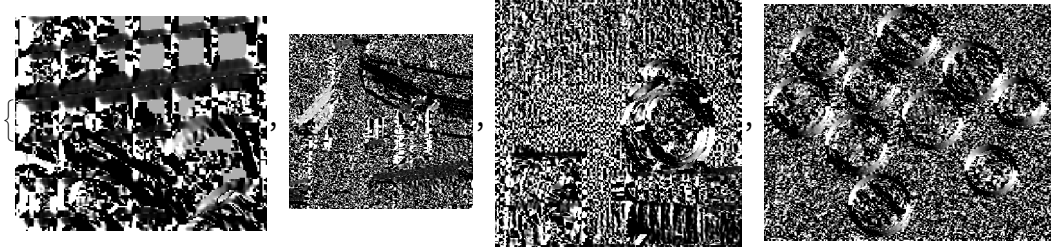
Out[10]=



ใช้ 1st derivative filter : Sobel ในการทำหา high pass ในแกน X และ Y ของ image
จากนั้นทำการรวมกันของทั้งสองแกน โดยการหา Norm ของภาพทั้งสองภาพ

```
In[11]:=  $\theta = \text{ArcTan}[Gy/Gx]$ 
```

```
Out[11]=
```



หาทิศทางของการเปลี่ยนแปลง โดยการหาจาก $\arctan Gy/Gx$

Edge Localization

```
In[49]:= edgeImgs = Binarize[#, 0.9] & /@ G
```

```
Out[49]=
```



ทำการ binarize รูปภาพ highpass โดยที่ให้ค่า threshold คือ 0.9 ทำให้ได้ภาพที่เป็น edge อย่างเดียว

Line Detection

In[13]:=

```

Clear[GetHoughLine]
GetHoughLine[img_] := Module[
  {dims, temp, sample, ρmax, res, thetas, cosT, sinT, nDeg, nR, x, y, j, ρi, idx, po
  temp = Reverse /@Position[ImageData[img], 1];
  dims = ImageDimensions[img];
  sample = {#[[1]], dims[[2]] - #[[2]]} & /@ temp;

  ρmax = Norm[ImageDimensions[img]];
  nDeg = 180;
  nR = Round[2 * ρmax];
  res = ConstantArray[0, {nR, nDeg}];
  pos = ConstantArray[{0,0,0,0,0}, {nR, nDeg}];

  thetas = Subdivide[-90, 89, nDeg - 1];
  cosT = Cos[thetas Degree];
  sinT = Sin[thetas Degree];

  Do[
    {x, y} = sample[[i]];
    Do[
      ρi = x cosT[[j]] + y sinT[[j]];
      idx = Round[ρi + ρmax + 1];
      If[1 ≤ idx ≤ nR,
        res[[idx, j]]++;
        pos[[idx, j]] = {res[[idx, j]], thetas[[j]], ρi, j, idx}
      ],
      {j, nDeg}
    ],
    {i, Length[sample]}
  ];
  Return[{res, pos}];
]

```

ทำการแปลง hough transform โดยการ โดยใช้สมการเส้นตรงที่เป็น polar approach เพื่อจำกัด search space ของข้อมูล โดยที่ theta มีค่า -90 ถึง 89 และ rho มีค่าตั้งแต่ - Norm(W,H) ถึง Norm(W, H) โดยการสร้างเส้นใน hough transform นั้น ใช้ pixel representation โดยการสร้างภาพเปล่า จากนั้นเติมด้วยค่าของสมการเส้นตรงในแต่ละ theta, rho แต่เนื่องจากภาพที่เป็น edge image นั้นอยู่ใน image space ทำให้ต้องแปลงข้อมูลจาก image space เป็น cartesian space ก่อน จาก (x, y) -> (y, x-h)

In[26]:=

```

nLines = {30, 50, 20};
minDist = {8, 10, 10};

```

เมื่อทำ hough transform แล้วนั้น จะทำการหาจุดที่เราต้องการเลือก โดยการนำ pixel based hough transform ในขั้นตอนก่อนหน้ามาทำการหาว่า pixel ใหนมีค่า intensity สูงสุด ซึ่งแสดงถึงส่วนที่มีจุดตัดกันมากที่สุด โดยหากหาค่า max มากตินั้น จะทำให้ได้เพียงแค่สุดเดียว ทำให้ต้องหา max หลายๆค่า(จำนวนจุดอยู่ใส่ตัวแปร nLines)

```

In[17]:= DropClosePoints[data_List,  $\delta$  : 1.] := Module[{res = {}, p},
  Do[
    p = pt;
    If[AllTrue[res, Norm[p - #] >  $\delta$  &],
      AppendTo[res, p]
    ],
    {pt, data}
  ];
  Return[res];
];

```

เมื่อได้จุดมาแล้วนั้น มีโอกาสสูงมาก ๆ ที่จะมีจุดใกล้กัน หรือมีเส้นซ้ำกันหลายเส้น เนื่องจาก edge อาจจะหนา ทำให้ต้อง drop จุดใน hough space ลง โดยการหา distance ของแต่ละจุดใน hough space ตัวอย่างเช่น ภาพแรก เลือกมาจำนวน Top 30 จาก hough space จากนั้นนำ 30 จุดนั้นมาทำการหา pair-wise distance โดยที่แต่ละจุดนั้นต้องห่างกัน 8 เท่านั้น เพื่อไม่ให้เกิดจุดที่ซ้ำกัน

```

In[50]:= houghData = GetHoughLine[#] & /@ edgeImgs[;;-2];

```

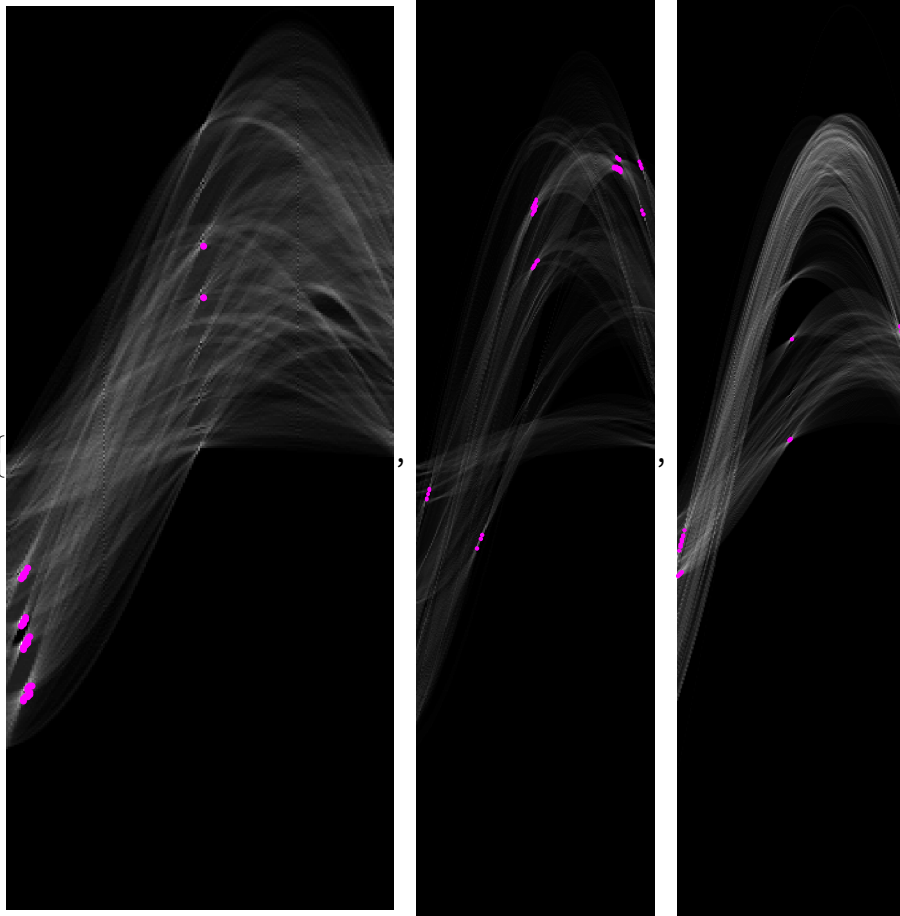
```

In[51]:= peakVal = SortBy[Flatten[#, 2], 1, First] & /@ houghData;
allPeakVal = peakVal[#[#][[-nLines[#[#];]] & /@ Range[Length[edgeImgs]-1];
selectedVal = DropClosePoints[allPeakVal[#[#][All, {2, 3}], minDist[#[#]] & /@ Range[Length

```

```
In[54]:= houghSpace = HighlightImage[ImageAdjust[Image[Reverse[houghData[#, 1]]]], Point[allPea
```

```
Out[54]=
```



จากตัวอย่างด้านบนจะเป็นภาพของ hough space ของภาพที่ 1 2 และ 3 โดยที่จุดสีชมพูแสดงถึงจำนวนจุดที่มีค่ามากที่สุด(จุดที่เป็นคำตอบ)

```
In[55]:= rTheta = # + ConstantArray[{0.01, 0}, Length[#]] & /@ selectedVal;
mAndC = Table[
    {-Cos[#[[1]] Degree]/Sin[#[[1]] Degree], #[[2]] /Sin[#[[1]] Degree]} & /@ rTheta[[i]]
    ,{i, Length[edgeImgs]-1}
];
```

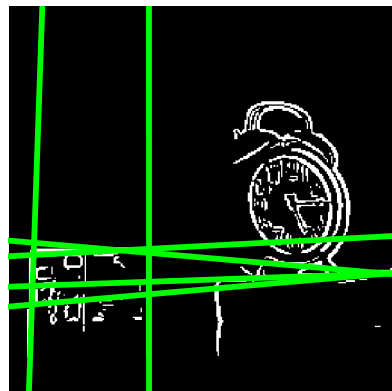
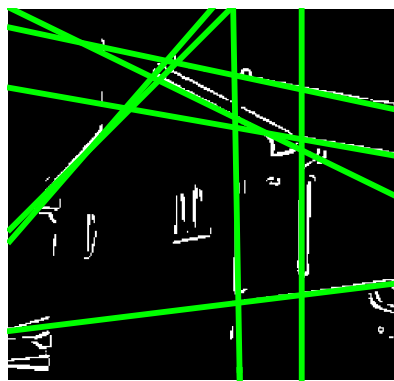
เมื่อเราได้จุดที่ไม่เกิดการซ้ำกันแล้วนั้น ก็ทำการแปลง rho และ theta ให้อยู่ในรูปของ m และ c โดยที่ theta อาจจะมีค่า เป็น 0 ได้ ทำให้ต้องบวก theta ด้วย ϵ เพื่อให้สามารถหาค่าของ cot ได้ โดยในตัวอย่าง ϵ มีค่าเท่ากับ 0.01

```

In[57]:= resultLine = Show[
  edgeImgs[[#]],
  Table[Plot[mAndC[[#, i, 1]] x + mAndC[[#, i, 2]], {x, 0, ImageDimensions[edgeImgs[[#]]][[1]] & /@ Range[Length[edgeImgs]-1]}
]

```

Out[57]=



จากรูปตัวอย่าง เกิดจากการนำสมการเส้นตรงที่ได้จากการทำ hough transform มา plot รวมกับรูปภาพ(เส้นสีเขียว)

In[220]:=


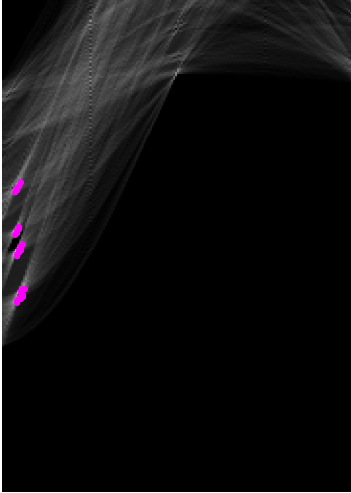

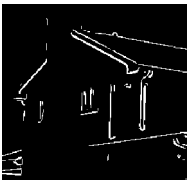
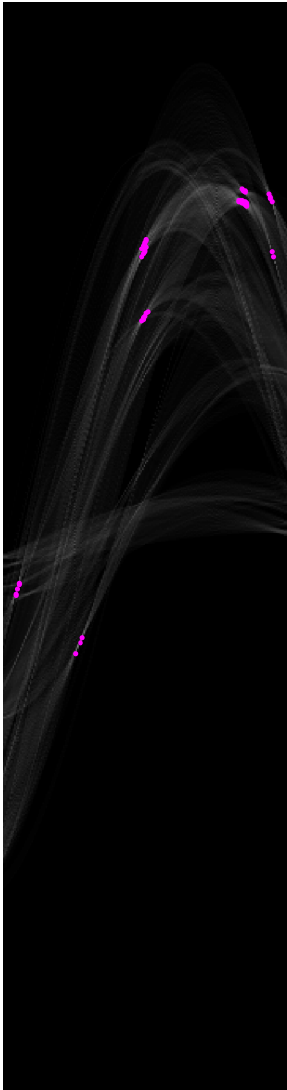
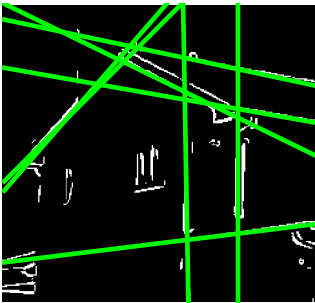
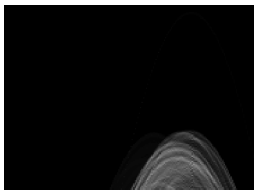
```

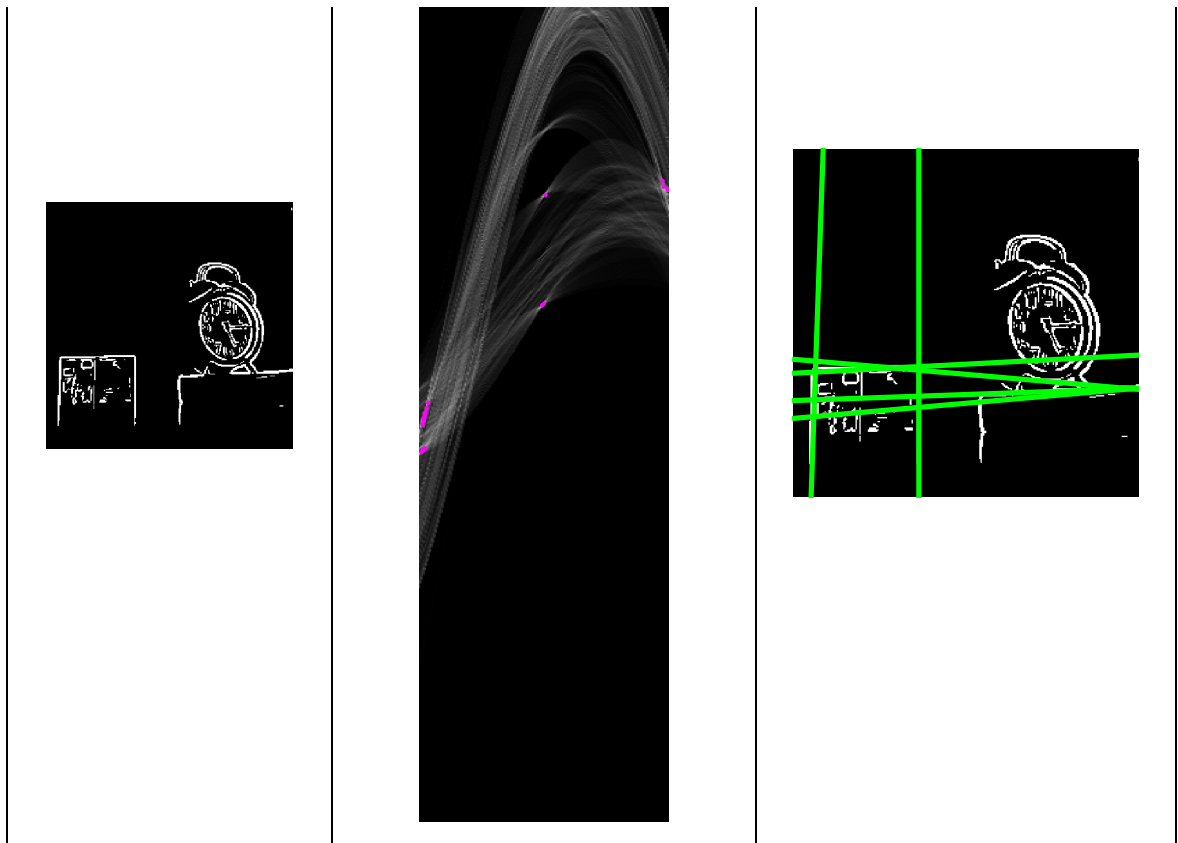
Grid [
  Transpose[{Prepend[edgeImgs[;;-2], "Edge Img"], Prepend[houghSpace, "Hough Space"], P
  Spacings-> {3,2},
  Dividers->All
]

```

Out[220]=

Edge Img	Hough Space	Line



จากตารางด้านบนเป็นตารางเปรียบเทียบการทำ line detection โดยใช้ hough transform โดยที่เห็นได้ว่าภาพแรกนั้น มีบางเส้นที่ไม่ถูก detect หากต้องการให้ทำการ detect ได้ดีนั้น อาจจะต้องทำการจัดการ parameter คือ จำนวนเส้นที่เลือก และ distance ของ rho และ theta เพื่อให้สามารถตรวจจับเส้นได้มากกว่าเดิม หรืออาจจะจัดการกับภาพ โดยการ ปรับ parameter ของการทำ edge detection เช่นการเปลี่ยนขนาด ของ filter เพื่อให้สามารถ detect edge ได้เพิ่มขึ้น

Circle Detection

In[194]:=

```

Clear[GetHoughCircle]
GetHoughCircle[img_,  $\theta$ _] := Module[
  {dims, tan, imgDat, imgCoor, temp, sample, w, h, res, thetas, tani, a, b, bi, x, y}
  temp = Reverse /@ Position[ImageData[img], 1];
  dims = ImageDimensions[img];
  sample = {#[[1]], dims[[2]] - #[[2]]} & /@ temp;

  imgDat = ImageData[ $\theta$ ];
  tan = Tan[imgDat];

  {w, h} = dims;
  a = Range[1, w, 1];
  b = Range[1, h, 1];
  res = ConstantArray[0, {h, w}];
  pos = ConstantArray[{0, 0, 0, 0}, {h, w}];

  Do[
    {x, y} = sample[[i]];
    tani = -1 / tan[[h - y, x]];
    Do[
      bi = Round[(j tani) + y - (x tani)];
      If[1 ≤ bi ≤ h,
        res[[bi, j]]++;
        pos[[bi, j]] = {res[[bi, j]], tani, j, bi}
      ],
      {j, a}
    ],
    {i, Length[sample]}
  ];
  Return[{res, pos, sample}];
]

```

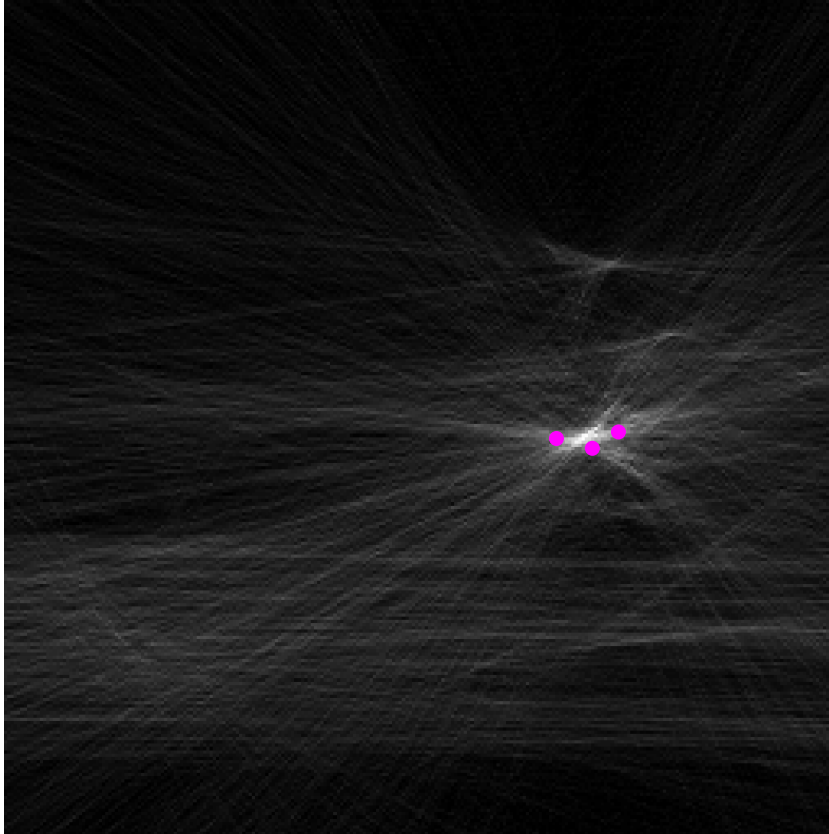
ทำ circle detection โดยใช้สมการ $b = a \tan(\theta) + y - x \tan(\theta)$ โดย bound ของ search space คือ a in $[0, W]$ และ b in $[0, H]$ โดย $\tan(\theta)$ ได้จากการทำ 1st derivative จาก sobel filter แต่ θ ที่ได้นั้น เป็นมุมของ tangent ไม่ใช่ normal ดังนั้น \tan ที่ใช้คำนวณต้องเป็น $-1/\tan$ ที่เกิดจากการทำ 1st derivative

In[178]:=

```
clock = GetHoughCircle[edgeImgs[[3]],  $\theta$ [[3]]];
centroid = SortBy[Flatten[clock[[2]], 1], First][[-80;;, {3, 4}]];
centroid = DropClosePoints[centroid, 9];
HighlightImage[ImageAdjust[Image[Reverse[clock[[1]]]], Point[centroid]]
```

{256, 256}

Out[181]=



จากผลลัพธ์ที่ได้นั้น เกิดจากการทำ hough transform โดยเลือก pixel value ที่มีค่า 80 อันดับแรก จากนั้นเลือกเฉพาะจุดที่มีระยะห่างมากกว่า 9 เท่านั้น เพื่อไม่ให้เกิดจุดที่ซ้ำกัน จากนั้นรูปจะให้ได้ a, b 3 จุด

In[182]:=

```
rList = Sqrt[(clock[[3]][All, 1] - #[[1]])^2 + (clock[[3]][All, 2] - #[[2]])^2] & /@ centroid;
rHist = BinCounts[#, {0, Max[#, 1]}] & /@ rList;
rHat = First /@ (Ordering[#, -1] & /@ rHist)
```

Out[184]=

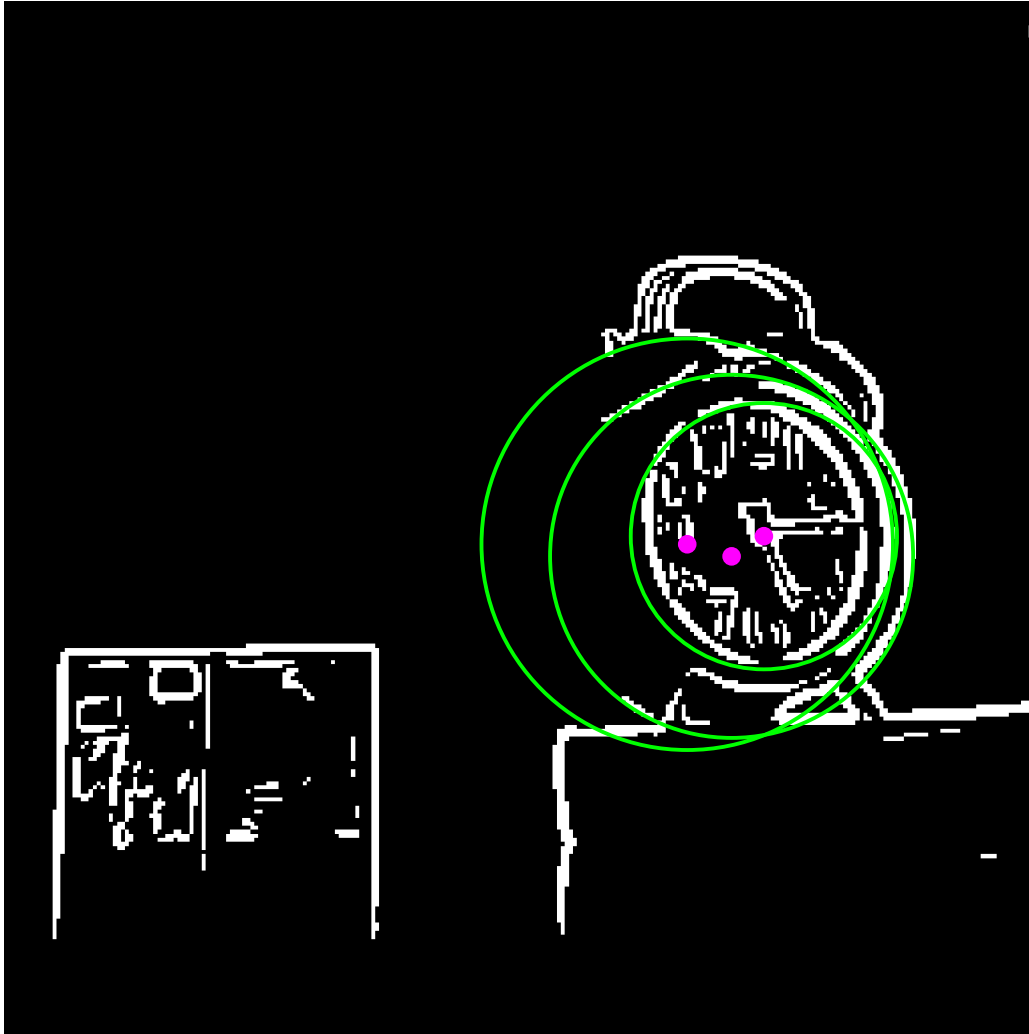
{45, 33, 51}

เมื่อได้ a, b แล้ว ก็ทำการหา r จาก $r = \sqrt{(x-a)^2 + (y-b)^2}$ โดยทำการหารัศมีจากทุกจุด กับ a, b จากนั้นทำการ สร้าง histogram เพื่อดูค่าเฉลี่ยของ bin ที่มีโหวตเยอะสุด หาก r ใดมี bin count มากจุด ใน a, b ใดๆ แสดงว่า r นั้นคือคำตอบ

In[185]:=

```
Show[
  HighlightImage[edgeImgs[[3]], Point[centroid]],
  Table[
    Graphics[{Thick, Green, Circle[{centroid[[i, 1]], centroid[[i, 2]], rHat[[i]]}]}],
    {i, Length[rList]}]
]
```

Out[185]=



จากรูปนาฬิกา เมื่อทำ hough transform แล้วพบว่ามียวงกลม 3 วง โดยจุดศูนย์กลางคือจุดสีชมพู และวงกลมเป็นเส้นสีเขียว

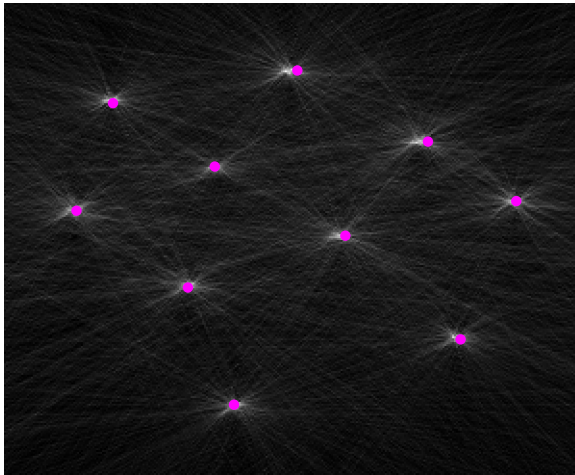
In[216]:=

```

coin = GetHoughCircle[edgeImgs[[4]],  $\theta$ [[4]]];
centroid = SortBy[Flatten[coin[[2]], 1], First][[-80;;;, {3, 4}]];
centroid = DropClosePoints[centroid, 9];
HighlightImage[ImageAdjust[Image[Reverse[coin[[1]]]], Point[centroid]]

```

Out[219]=



นำ hough transform มาลองทำกับรูปเหรียญโดยเลือกจุดมากทั้งหมด 80 จุดและ ระยะห่างจุดคือ 9 โดยตรวจจับวงกลมได้ทั้งหมด 10 ตามจุดสีชมพูใน hough space

In[190]:=

```

rList = Sqrt[(coin[[3]][All, 1] - #[[1]])^2 + (coin[[3]][All, 2] - #[[2]])^2] & /@ centroid;
rHist = BinCounts[#, {0, Max[#, 1]}] & /@ rList;
rHat = First /@ (Ordering[#, -1] & /@ rHist)

```

Out[192]=

```
{25, 30, 27, 30, 27, 29, 25, 25, 28, 25}
```

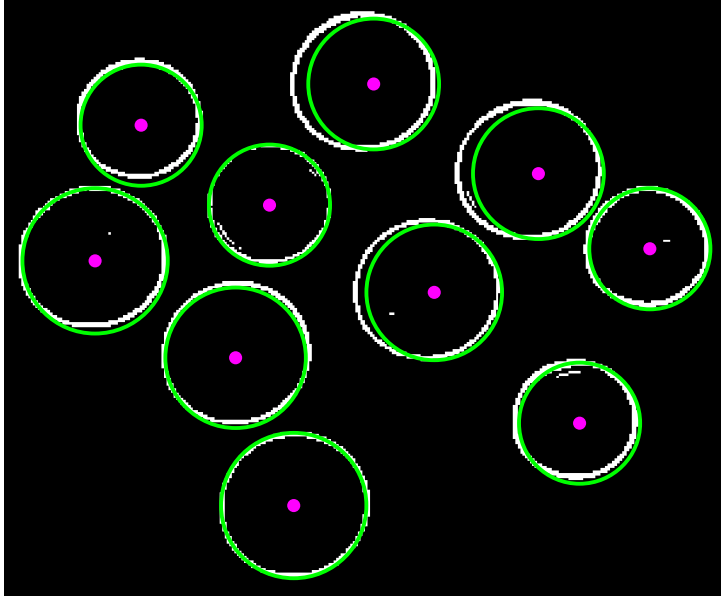
โดยรัศมีของวงกลมแต่ละวงนั้น คือ {25,30,27,30,27,29,25,25,28,25}

ซึ่งแสดงว่าวงกลมแต่ละวงมีรัศมีพอกัน

In[193]:=

```
Show[
  HighlightImage[edgeImgs[[4]], Point[centroid]],
  Table[
    Graphics[{Thick, Green, Circle[{centroid[[i, 1]], centroid[[i, 2]], rHat[[i]]}],
    , {i, Length[rList]}]
]
```

Out[193]=



จากรูปเมื่อลอง plot วงกลม บนรูปเหรียญแล้วจะเห็นว่าวงกลมที่ได้นั้น ค่อนข้าง fit กับ edge ของเหรียญ แต่จะมีบางวงที่ overlap กับขอบบ้าง เกิดจากการเลือก a, b จากการเลือก 80 จุด และหาระยะห่าง แต่ที่ implement ไว้นั้น ตรวจจับระยะห่างแล้วเลือก จุดแรกที่มีระยะห่างพอๆกัน ดังนั้นจุดแรกที่ได้ อาจจะไม่ใช่จุดศูนย์กลางที่เหมาะสมจริงๆ