Cubieboard + Kinect

Então, eu comprei recentemente uma Cubieboard para usá-la como um servidor de arquivos. Se você olhar atentamente para a imagem do Post em Destaque (ou se você já clicou no link), você pode perceber por que eu escolhi esta placa em particular. Precisamente, porque tem uma porta SATA para ligar um disco rígido padrão ou SSD a ele. Isso deve oferecer uma melhoria de desempenho enorme sobre USB conectado HDD de (embora a Cubieboard ainda só tenha uma interface Ethernet 100MBit / s).

Sobre a Cubieboard

Esta placa é apenas um pouco mais caro que um Raspberry Pi , mas vem com muito mais poder de processamento (A10 @ 1GHz) com o dobro da RAM (1 GB) e ROM on-board do Flash (4 GB - pré-carregado com Android 4).

Ele também tem Audio In / Out analógico, saída HDMI, 2x USB host, slot micro SD (útil para inicializar outro sistema operacional fora de um cartão SD), um receptor de IR, a porta SATA acima mencionado e 96 IO pinos que incluem pinos para I2C, SPI, RGB / LVDS, CSI / TS, FM-IN, ADC, CVBS, VGA, SPDIF-out, R-TP, etc ...

Portanto, é um pacote muito arrumado e é quase uma vergonha desperdiçá-la apenas como um servidor de arquivos. Então, eu queria dar tudo o que o processamento não utilizada poder algo útil para fazer e decidiu ligar um Microsoft Kinect

Sobre o Kinect

Até agora o Kinect deve ser um dispositivo bem conhecido de todos vocês. É uma nova onda na comunidade hacker / maker, porque pela primeira vez houve um sensor de visão 3D de baixo custo que poderia ser utilizado para outras tantas coisas além de sua finalidade (jogos no Xbox 360). Todo mundo que está interessado em ver todos os projetos interessantes que têm sido feitos usando um Kinect , é só acessar este site do Kinecthacks.

Assim, as principais características do Kinect são:

1) Visão Laser para mapeamento de campo em profundidade baseado em uma resolução de 640 × 480 pixels;

2) RGB Color Video Stream com uma resolução de 640 × 480 pixels;

3) Servo para inclinar a cabeça do sensor Kinect;

4) Acelerômetro de 3 eixos

Alguém poderia perguntar por que o Kinect inclui um servo de inclinação e o acelerômetro, o meu palpite é que os desenvolvedores Microsoft Kinect nunca pretendeream que este dispositivo seja limitada a jogos XBox, porque para mim esses recursos realmente só fazem sentido quando você está construindo um robô.

O que eu fiz até agora...

Felizmente os desenvolvedores Cubieboard fornecem uma imagem do Ubuntu / Linaro que você pode apenas flash para um cartão micro SD.

A Cubieboard vai reinicializar o Android sem um cartão micro SD inserido e quando há um cartão SD inicializável presente, ele será inicializado. Graças à distribuição moderna Linux a instalação dos pacotes necessários é tão fácil quanto parece ser. No console como root basta fazer o seguinte:

```
sudo apt-get install freenect python-freenect python-numpy python-opencv
```

Após isso, você terá que desativar o driver do kernel para Kinect para que freenect possa reivindicar as interfaces.

```
sudo modprobe -r gspca_kinect
sudo modprobe -r gspca_main
echo "blacklist gspca_kinect" >> /etc/modprobe.d/blacklist.conf
```

E isso é tudo, agora o Kinect pode ser conectado a uma das portas USB da Cubieboard e então você pode começar a acessá-lo via Python.

Hoje eu consegui ler a informação de profundidade a partir do sensor 3D, leia imagens do RGB Cam, incline a cabeça do sensor e faça algumas manipulações sobre os dados recuperados com OpenCV. Para começar a usar o Kinect com OpenCV em Python tem que começar importando todos os módulos necessários.

```
import freenect, cv, numpy
```

Leitura de **Depth Data**

Ler o Depth Data do Sensor Kinect é bastante simples:

```
data = freenect.sync_get_depth()
data = data[0].astype(numpy.uint8)
```

Agora, se você (ao contrário de mim) tem um monitor conectado ao seu Cubieboard (ou talvez você está fazendo isso em algum outro tipo de computador), você deve ser capaz de exibir diretamente os dados de profundidade que você acabou capturando com OpenCV assim:

```
cvMat = cv.fromstring(data)
cv.ShowImage("depth", cvMat)
```

Por outro lado, você pode querer salvar a imagem de profundidade no disco, de modo que você possa vê-la em outro computador:

```
cvMat = cv.fromstring(data)
cv.SaveImage("depth.png", cvMat)
```

O resultado pode ser o seguinte:

Você pode facilmente manipular dados de imagem com OpenCV, por exemplo, este código desenha um círculo no centro da imagem (320, 240) com a cor preta (0,0,0) e um raio de 50 Pixels:

```
# first convert cvMat to iplImage
img = cv.CreateImage(cv.GetSize(cvMat), cv.IPL_DEPTH_8U, 1)  # cv.GetSize() returns a
tuple, in this case (640,480) ; For a depth image we only have one color channel, hen
the ,1 at the end
cv.SetData(img, cvMat.tostring())
# now draw a Circle on that image
cv.Circle(img, (320,240), 50, (0,0,0), 0, 8, 0)
# Save to file
cv.SaveImage("depth_manipulated.png", img)
```

E o resultado fica assim:

Agora você pode facilmente fazer a mesma coisa com os dados RGB (embora temos 3 canais de cor para RGB, ao contrário de apenas um para a profundidade). Agora uma coisa interessante seria para sobrepor a imagem de profundidade à imagem RGB e ver como isso parece. Por conveniência, nós primeiro definimos um par de funções para pegar os dados que precisamos.

```
def getrgb():
    data = freenect.sync_get_video()
    data = data[0].astype(numpy.uint8)
    cvMat = cv.fromarray(data)
    img = cv.CreateImage(cv.GetSize(cvMat), cv.IPL_DEPTH_8U, 3) #3 channels for RGB
color!
    cv.SetData(img, cvMat.tostring())
    return img

def getdepth():
    data = freenect.sync_get_depth()
    data = data[0].astype(numpy.uint8)
    cvMat = cv.fromarray(data)
    img = cv.CreateImage(cv.GetSize(cvMat), cv.IPL_DEPTH_8U, 1) #1 channel for depth!
    cv.SetData(img, cvMat.tostring())
    return img

def newimg(size):
  return cv.CreateImage(size, cv.IPL_DEPTH_8U, 3) #create an empty image with 3 color
channels
```

Como você pode ver, pegando um quadro de vídeo a partir do Kinect é tão simples como pegar os dados de profundidade. A principal diferença (dados wise) é que o vídeo RGB vem com 3 canais de cor, enquanto os dados de profundidade vem com apenas um canal de cor. Eu enfatizo sobre essa diferença, porque você pode ter problemas quando quiser combinar as duas imagens.

Antes que você possa fazer isso você vai precisar converter os dados de profundidade em uma imagem que também tem 3 canais de cor. Então, aqui está o código para combinar as duas imagens.

```
depth = getdepth()
rgb = getrgb()
dstDepthImg = newimage(cv.GetSize(depth))
finalImage = newimage(cv.GetSize(depth))
cv.Merge(depth, depth, depth, None,dstDepthImg) #this is basically assigning the 1
channel of the depth image to all 3 channels of the empty dstDepthImg
# and finally adding the rgb image to the reformatted depth image
cv.Add(dstDepthImg, rgb, finalImage)
cv.SaveImage("combined_data.png", finalImage)
```

O resultado vai se parecer com isso:

E agora para a parte final do código eu vou mostrar-lhe como escrever alguns quadros de dados em um arquivo de vídeo.

```
format = cvFOURCC('M', 'J', 'P', 'G') # MJPEG - Motion JPEG - encoding
fps = 24 #frames per second
resolution = (640,480)  # this is the resolution that cv.GetSize() will always return
when working with data from the Kinect
# first create a video writer instance that'll write the video to output.avi
writer = cv.CreateVideoWriter("output.avi", format, fps, resolution)
# now write 100 frames in a loop and draw an ever growing circle on each frame
for frame in xrange(0,100):
    img = getrgb()
    cv.Circle(img, (320,240), frame, (255,255,0), 0, 8, 0)
    cv.WriteFrame(writer, img)
```

Fonte: http://neon-society-electronics.com/?p=62

Compartilhe e Divirta-se