به نام خدا

گزارش پروژه ی درس سیستم های نهفته و بیدرنگ

عنوان پروژه : Simple Motion Detection

اعضای گروه :

علی دینداامال - 9773118

علیرضا پشم فروش - 9773106

استاد : دکتر محمد جواد رشتی

بهار ۱۴۰۱

فهرست مطالب

شرح پروژه :

هدف این است که میخواهیم حرکات را شناسایی بکنیم، به این صورت که با استفاده از دوربین تعبیه شده در برد Raspberry Pi ، حرکات را شناسایی کرده و حرکات شی یا شخص را ضبط و ذخیره می کنیم.

سخت افزار مورد نیاز پروژه :

- Raspberry Pi 4 model B – 4G RAM
- ماژول دوربین برای Raspberry Pi
- SD card - 32G
- آداپتور مربوطه
- Buzzer

Raspberry Pi 4 model B - 4G RAM

انتخاب سیستم عامل :

با توجه قابلیت پشتیبانی کردن سیستم عامل های 64 بیتی توسط بورد، از سایت Raspberry Pi آخرین ورژن 64 بیتی *Raspberry Pi OS image* را دانلود کرده و سپس آن را بر روی SD card میسوزانیم (burn).

کامپایل کردن سیستم عامل :

به طور کلی هدف از کامپایل کردن یک سیستم عامل این است که شخصی سازی صورت گیرد و تغییرات مورد نیاز در سیستم عامل را با توجه به نیازهایی که پروژه تعیین میکند، اعمال کنیم.

کامپایل کردن در Raspberry Pi به دو صورت انجام میگیرد :

1. به صورت *cross-compiling*
2. به صورت مستقیم بر روی Raspberry Pi

که برای این پروژه از روش دوم استفاده شده است. در ادامه نحوه کامپایل کردن و مراحل آن ذکر شده است.

مراحل کامپایل :

در ابتدا git و دیگر نیازمندی ها را در سیستم عامل خود نصب میکنیم. کد مربوطه به صورت زیر است:

```
sudo apt install git bc bison flex libssl-dev make libncurses-dev
```

سپس باید سورس کد را از گیت هاب مربوط به Raspberry pi، clone بکنیم:

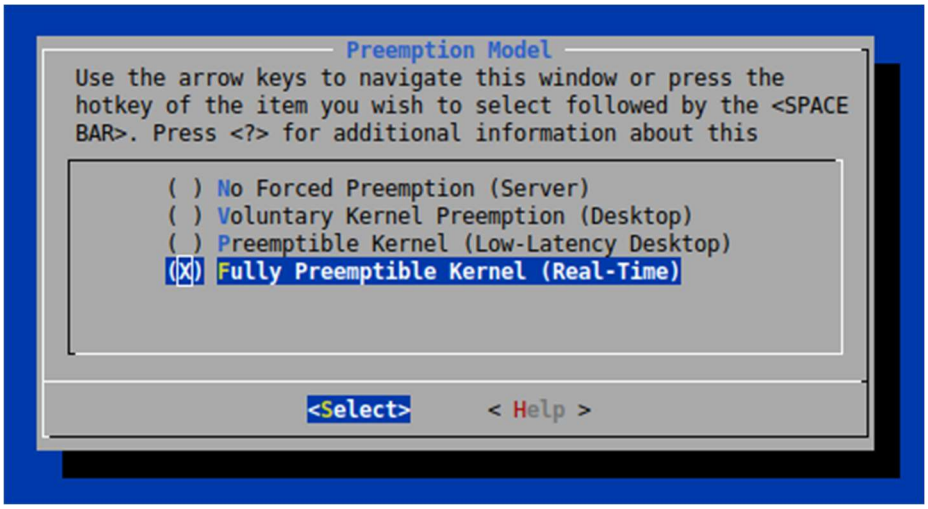
git clone --depth=1 https://github.com/raspberrypi/linux

با وارد کردن دستور بالا آخرین branch فعال را میتوانیم دانلود بکنیم، فقط باید دقت کرد که –depth=1 را اضافه کنیم زیرا با حذف کردن این بخش تمام *repository* را دانلود خواهیم کرد!

قبل از کانفیگ کردن، نیاز هست که عمل *patch* کردن را بر روی سیستم عامل انجام داد. پس آخرین ورژن *PREEMPT_RT*[1] را از سایت *wikilinux* دانلود میکنیم و باید دقت شود که ورژن *patch* باید با ورژن کرنل مچ شوند. سپس با استفاده از دستور زیر، patch دانلود شده را بر روی سورس کد میگذاریم :

zcat patch-5.15.40-rt43.patch.gz | patch -p1

بعد از انجام دستور ، *make manuconfig* را اجرا میکنیم . بعد از باز شدن منو، به قسمت General setup و بعد Preemption Model میرویم و گزینه ی **Fully Preemptible Kernel** را انتخاب میکنیم. سپس تغییرات را ذخیره میکنیم و خارج میشویم.



[1] . Basically, PREEMPT_RT allows user-space code to preempt operating system tasks.

حال باید عمل کانفیگ کردن را انجام دهیم. با توجه به 64 بیتی بودن سیستم عامل و 4B بودن نوع بورد دستور زیر را وارد میکنیم :

```
KERNEL=kernel8
make bcm2711_defconfig
```

همانطور که گفته شد باید توجه کرد که مدل بورد برای قسمت **KERNEL** رعایت شود.

و در آخر برای نصب کردن ماژول های کرنل دستورات زیر را وارد میکنیم :

```
make -j4 Image.gz modules dtbs
sudo make modules_install
sudo cp arch/arm64/boot/dts/broadcom/*.dtb /boot/
sudo cp arch/arm64/boot/dts/overlays/*.dtb* /boot/overlays/
sudo cp arch/arm64/boot/dts/overlays/README /boot/overlays/
sudo cp arch/arm64/boot/Image.gz /boot/$KERNEL.img
```

سپس Raspberry Pi را ریبوت میکنیم تا با کرنل کامپایل شده اجرا شود.

توجه : منظور از پرچم -j4 این است که عمل کامپایل را بین 4 هسته تقسیم میکند تا سریعتر انجام گیرد.

توصیف نرم افزار پروژه :

ایده کلی پروژه :

هر ویدیو شامل مجموعه ای از فریم ها است و یک دوربین بطور مثال میتواند در هر ثانیه 20 فریم را ضبط کند. از آنجایی که ما قصد داریم که حرکات را شناسایی و ضبط بکنیم، پس ایده باید این باشد که اگر حرکتی رخ داد، بین دو فریم متوالی باید تفاوت هایی ظاهر شود. پس میتوانیم مقادیر پیکسل های دو فریم متوالی را مدام چک کنیم. اگر اختلاف بین این دو صفر شود، میفهمیم که همه چیز در حالت ایستا یا استاتیک قرار دارند و حرکتی رخ نداده است، با این حال در دنیای

واقعی این اختلاف صفر مطلق نیست و عددی نزدیک به صفر است چون همیشه مقداری نویز وجود دارد!

در حالت دوم اگر حرکتی رخ دهد، اختلاف مقادیر دو فریم میتواند عددی منفی و یا مثبت شود. و در اخر این مقدار به دست آمده را باید با یک *threshold* مقایسه کنیم و باید دقت کرد که قدر مطلق این اختلاف لحاظ شود. اگر مقدار به دست آمده از *threshold* بیشتر باشد حرکت شناسایی میشود.

شرح کد :

```python
import cv2
import datetime
import numpy as np
from gpiozero import Buzzer
from time import sleep

out = None
def capture():
    global out
    time = datetime.datetime.now()
    out = cv2.VideoWriter(f'output_{time.strftime("%Y-%b-%d_%H:%M:%S")}.avi',fourcc, 20.0, (640, 480))

buzzer = Buzzer(17)
cap = cv2.VideoCapture(0)
last_mean = 0
frame_rec_count = 0
fourcc = cv2.VideoWriter_fourcc(*'XVID')
capturing = False
while(True):
    ret, frame = cap.read()
    #cv2.imshow('frame',frame)
    gray = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)
    result = np.abs(np.mean(gray) - last_mean)
    print(result)
    last_mean= np.mean(gray)
    #sensitivity
    if result > 2 :
        print("Motion detected!")
        #do not capture another footage while capturing one
        if not capturing:
            print("Started recording.")
            capture()
            capturing = True
    #appending frames to make an avi file
    if capturing:
        out.write(frame)
        #buzzer works while capturing
        if frame_rec_count in [0,20,40,60,80,99]:
            buzzer.on()
        elif frame_rec_count in [10,30,50,70,90]:
            buzzer.off()
        frame_rec_count = frame_rec_count + 1
    #checks if 5 seconds has elapsed
    if frame_rec_count == 100:
        frame_rec_count = 0
        capturing = False
```

در این پروژه از چند کتابخانه در زبان پایتون استفاده شده است :

open cv2: از این کتابخانه برای پردازش تصاویر و ویدیو ها شناسایی پترن های خاصی استفاده میکنیم و مناسب اعمال بی درنگ است.

NumPy: کتابخانه ای است برای استفاده کردن از اعمال ریاضی.

Datetime: کتابخانه ای است برای اعمال مربوط به زمان.

gpiozero: کتابخانه ای برای کار کردن با *buzzer*

همانطور که مشاهده میشود یک حلقه while در پروژه داریم ولی قبل از اینکه به سراغ آن برویم، یک متغیر بنام *last_mean* تعریف میکنیم و مقدار آن را برابر صفر میگذاریم. به این علت که در شروع چیزی به نام فریم قبلی برای مقایسه وجود ندارد، پس یک مقدار اولیه برابر صفر داریم.

در ادامه مقدار *absolute* اختلاف دو فریم را در متغیر *result* قرار میدهیم و مقدار آن را چاپ میکنیم.

اکنون میتوانیم متوجه شویم که اگر حرکتی رخ ندهد وارد شرط ذکر شده در بالا نمیشویم، اما اگر حرکتی شناسایی شود با مقدار threshold که برابر با 2 است مقایسه میشود و پس از درست بودن شرط وارد شده و ادامه دستورات اجرا میشوند.

حال که حرکت را شناسایی کردیم، میخواهیم ویدیوی مربوط به زمانی که حرکت انجام شده است را ضبط بکنیم. برای مشخص کردن زمان ضبط کردن لازم است که تعداد فریم هایی را که میخواهیم ضبط کنیم مشخص شوند. به طور مثال اگر بخواهیم 5 ثانیه بعد از شناسایی حرکت را ضبط کنیم، لازم است که تعداد فریم های مورد نظر را برابر 100 قرار دهیم (با توجه به اینکه دوربین ما در هر ثانیه 20 فریم ضبط میکند). پس متغیری به نام *frame_rec_count* تعریف میکنیم.

برای ضبط کردن میتوانیم از کتابخانه ی *opencv* نیز استفاده بکنیم. یک متغیر از نوع بولی به نام *capturing* تعریف میکنیم و مقدار اولیه آن را برابر False میگذاریم. اگر شرط مربوط به مقایسه بین اختلاف دو فریم و threshold درست باشد، مقدار متغیر *capturing* به True تغییر میکند

و در این if یک شرط دیگر وجود دارد که مانع از این میشود که در هنگام ضبط کردن، اگر حرکتی شناسایی شود ضبط را قطع کرده و شروع به ضبط حرکت جدید بکند. و با استفاده از capture() شروع به ضبط کردن همراه با تاریخ و دیگر مشخصات میکنیم.

Buzzer تعریف شده برای پروژه از نوع *active* است. ابتدا در خطوط مربوط به تعریف متغیر ها، آن را تعریف کرده ایم . نحوه کارکرد این ماژول به این صورت است که در فریم های ۲۰،۴۰،۶۰ و ... فعال میشود و صدا تولید میکند و در فریم های۱۰،۳۰و... غیر فعال است.

بررسی پارامتر بی درنگ :

برای انجام تست از ***RT-Tests*** استفاده شده است. *RT-Tests* یک بسته ابزاری است که دارای برنامه هایی زیادی جهت تست کردن ویژگی های بی درنگ یک سیستم است.

برنامه های آن شامل موارد زیر هستند:

- cyclicteset: latency detection
- cyclicdeadline
- deadline_test
- hackbench
- pip_stress
- pi_stress
- pmqtest
- ptsematest
- queuelat
- rt-migrate-test
- signaltest
- sigwaittest
- ssdd
- svsematest

که برای این پروژه از *cyclictest* استفاده شده است .

cyclictest به طور مکرر و دقیق اختلاف (interval) بین زمان بیدار شدن در نظر گرفته شده (ست شده) برای یک *thread*، و زمانی که عملا آن *thread* بیدار میشود را اندازه میگیرد.

برای انجام تست، دستور زیر را وارد میکنیم :

```
cyclictest -l5000000 -m -S -p90 -i200 -h400 -q > output
```

که توضیحات آن ها به شرح زیر است :

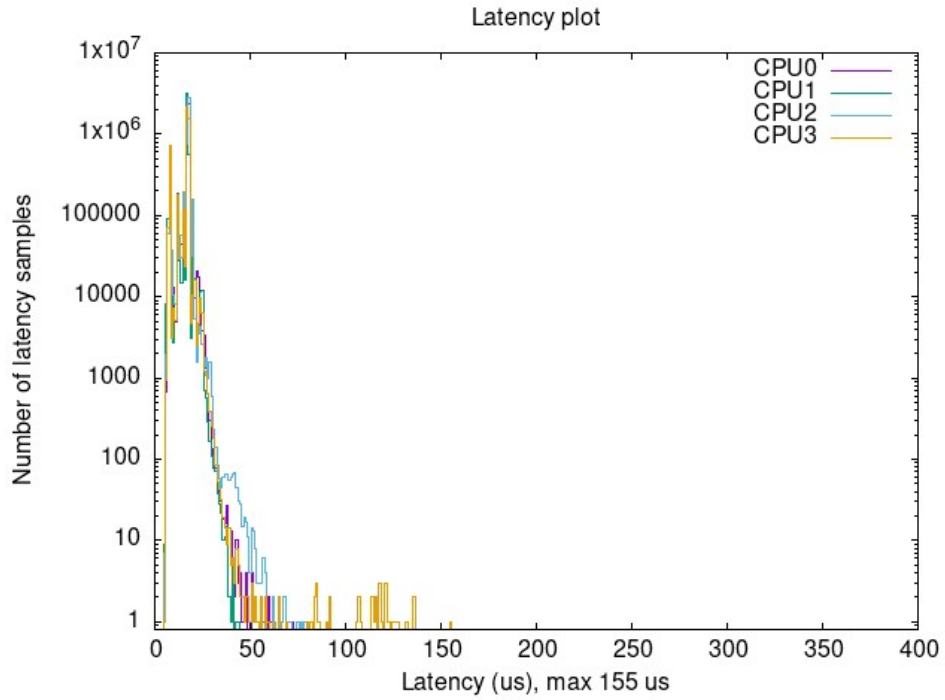
- **-l5000000: 5M iterations (about 15 min!);**
- **-m: lock current and future memory allocations (prevent being paged out?)**
- **-S: Standard SMP testing: options -a -t -n and same priority of all threads (Raspberry Pi has 4 Cores then 4 Threads)**
- **-p90: priority of highest priority thread set to 90 (for the 4 threads, then: 90 89 88 87)**
- **-i200: interval for the first thread (in us).**
- **-h1000: dump histogram for max latency (up to 1000us).**
- **-q: print a summary only on exit.**

مراحل این تست طبق دستور العمل های سایت زیر انجام شده است که روشی استاندارد برای تست latency را بیان کرده است:

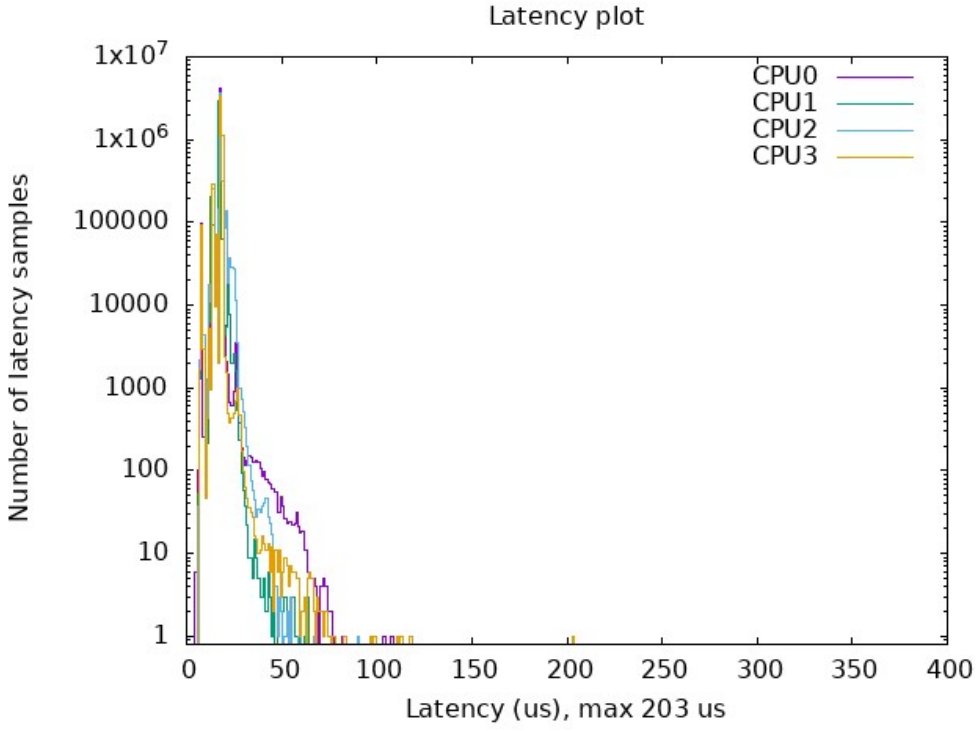
https://www.osadl.org/Create-a-latency-plot-from-cyclictest-hi.bash-script-for-latency-plot.0.html

توجه: در تست استاندارد تعداد iterationها ۱۰۰ میلیون در نظر گرفته شده است که این تست به مدت ۵ ساعت و ۳۳ دقیقه زمان میبرد! پس در این پروژه جهت کوتاهتر شدن این فرایند، تعداد iteration ها را ۵ میلیون در نظر گرفته ایم و باید متذکر شویم که اگر به تعداد ۱۰۰ میلیون در نظر میگرفتیم ، میتوانستیم به جواب درست تر و مطلوب تری برسیم!

این تست به دو صورت انجام گرفته است؛ یک بار بر روی *PREEMPT-RT kernel* و بار دیگر بر روی *Standard kernel* که نتایج را در زیر مشاهده میکنید :



PREEMPT-RT kernel



Standard kernel

منابع :

https://wiki.linuxfoundation.org/realtime/documentation/technical_basics/start

https://sokacoding.medium.com/simple-motion-detection-with-python-and-opencv-for-beginners-cdd4579b2319

https://wiki.linuxfoundation.org/realtime/start

https://www.raspberrypi.com/documentation/computers/linux_kernel.html

http://robskelly.com/2020/10/14/raspberry-pi-4-with-64-bit-os-and-preempt_rt

https://www.osadl.org/Create-a-latency-plot-from-cyclictest-hi.bash-script-for-latency-plot.0.html?&no_cache=1&sword_list[0]=script

https://lemariva.com/blog/2019/09/raspberry-pi-4b-preempt-rt-kernel-419y-performance-test

https://projects.raspberrypi.org/en/projects/physical-computing/8

https://www.osadl.org/Create-a-latency-plot-from-cyclictest-hi.bash-script-for-latency-plot.0.html