Проект по Вградени автономни системи

Android приложение –

Судоку разпознаване и решаване

Борис Минев

Фн: 24050

Android

Android е операционна система, разработена от Google за мобилни устройства. Основата ѝ се гради върху модифициран Линукс кернел. Първоначалната му разработка е направена от Android Inc, фирма която по-късно бе закупена от Google, а още по-късно от Open Handset Alliance.

Операционната система се състой основно от Java приложения, работещи върху Java Framework. Основния Java код (libraries) е подкаран под т.нар. Dalvik Virtual Machine, който във версия 2.2 (Froyo) поддържа JIT компилиране. Общия код на Андроид се състой от 12 милиона реда, от които 3 милиона са XML, 2.8 милиона са C, и 2.1 милиона са на Java.

Версии на Android:

1. Android 1.5 Cupcake
2. Android 1.6 Donut
3. Android 2.0 Eclair
4. Android 2.01 Eclair
5. Android 2.1 Eclair
6. Android 2.2.x Froyo
7. Android 2.3 Gingerbread
8. Android 3.0,1,2 Honeycomb
9. Android 4.0.3,4 Ice Cream Sandwich
10. Android 4.1,2 Jelly Bean

Приложения (Applications)

Android OS идва с комплект от основни програми като Email клиент, SMS програма, календар, интерент браузър и други. Всички програми са написани на програмният език Java.

Application Framework

Чрез предоставянето на отворена платформа за разработване на приложения, Android предоставя възможност на разработчиците да изграждат изключително богати и иновативни приложения. Програмистите са свободни да се възползват от хардуера на устройството, достъп до местоположението на устройството, да стартират background services, да добавят известия в status bar-а и много, много други неща!

Програмистите имат пълен достъп до същите фреймуърк API-та използвани от вградените програми. Архитектурата на приложенията е проектирана, така че да улесни многократната употреба на компоненти; всяко приложение може да публикува своите възможности и всяко друго може да ги използва. Същият този механизъм позволява приложенията да бъдат разменяни от потребителя. Например, ако не ви харесва вградената Email програма може да си свалите друга и да я направите по подразбиране.

В основата на всяко приложение са набор от сервизи и системи включващи:

Богат и разширяващ се набор от Views, които могат да бъдат използвани за изграждането на приложения, включващи: Lists (списъци), Grids (мрежи), Text Boxes (текстови кутии), Buttons (бутони) и дори възможност за вграждане на уеб браузъра в приложението.

Content Providers, които дават възможност на приложенията да получават достъп до данни от други приложения (като например програмата Contacts) или да споделят тяхните общи данни.

Resource Manager - предоставя достъп до ресурси различни от код, като например стрингове, графики или layout файлове.

Notification Manager, който дава възможност на приложенията да показват различни известявания в status bar-а.

Activity Manager - служи за управление на lifecycle-а (цикълът на живот) на приложението и предоставя обща навигация в backstack-а.

Библиотеки (Libraries)

Android включва набор от C/C++ библиотеки използващи се от различни компоненти на системата. Тези възможности са на разположение на програмистите през Android фреймуърк-а. Някои от библитеките:

System C library - BSD-произовдна имплементация на системната C библиотека (libc) пригодена за вградените Linux-базирани устройства.

Media Libraries - базирана на OpenCORE на PacketVideo; библиотеки поддържащи възпроизвеждане и запис на много от популярни видео и аудио формати, както също така и на статични снимкови файлове, включващи: MPEG4, H.264, MP3, AAC, AMR, JPG, и PNG.

Surface Manager - управлява достъпът на показване на подсистемите и безпроблемно копозира 2D и 3D графични слоеве от различни приложения.

LibWebCore - модерен уеб браузър engine, който се грижи за браузърът на Android както и за вграждащото се web view.

SGL - основният 2D графичен engine.

3D libraries - имплементация базирана на OpenGL ES 1.0 API-та; библиотеките използват както хардуерното 3D ускорение (ако е на разположение) или включеният, високо оптимизиран 3D растерайзър.

FreeType - bitmap и векторно рендиране на шрифтове.

SQLite - мощен и лек engine на релационни бази от данни, които са на разположение на всички приложения.

Android Runtime

Android включва набор от основни библиотеки, които предоставят голяма част от функционалността, която е на разположение в основните библиотеки на езикът за програмиране Java.

Всяко Android приложение се стартира в свой процес със своя инстанция на Dalvik виртуалната машина. Dalvik е написана, така че устройството да може да стартира множество от виртуални машини ефективно. Dalvic VM стартира Dalvik Executable (.dex) формат, който е оптимизиран за минимален отпечатък върху паметта. Виртуалната машина е базирана на регистри и стартира класове компилирани чрез компилатора на програмният език Java и трансформирани в .dex формат чрез включеният "dx" инструмент.

Dalvik виртуалната машина разчита на Linux ядрото за основна функционалност като threading и low-level управление на паметта.

Linux Kernel

Android разчита на Linux версия 2.6 за основни системни сервизи като сигурност, управление на паметта, управление на процесите, мрежовият стак и драйвър модела. Ядрото също така играе ролята на абстрактен слой между хардуера и останалата част от софтуерният стак.

Емулатори

Емулатор е хардуерно или софтуерно устройство (или и двете), което имитира (още емулира) функциите една компютърна система(гост) в друга компютърна система(домакин) по такъв начин, че емулираното поведение да наподобява максимално поведението на реалната(гост) системата.

Емулация се свързва със способността на компютърна програма в електронно устройство да емулира (имитира) друга програма или устройство. Много принтери например са проектирани да емулират Hewlett-PackardLaserJet принтер, тъй като по-голямата част от софтуерът е разработен за HP принтери. Ако принтер различен от HP емулира HP принтер, то той ще може да използва всеки софтуер писан за HP принтер и по този начин ще достигне до същите резултати при принтиране.

Хардуерен емулатор е емулатор, който имитира формата на дадено хардуерно устройство.

В теоретичен смисъл Church-Turing тезисът предполага, че всяка оперативна среда може бъде емулирана във всяка друга. На практика това се оказва доста трудно, в частност когато точното поведение на системата, която трябва да бъде емулирана не е документирано и трябва да бъде възстановено(налучкано) по обратен път (reverse engineering). Също така тезисът не казва нищо за времевите ограничения. Ако емулаторът не работи бързо колкото оригиналният хардуер – емулираният софтуер е възможно да работи много по-бавно отколкото би работи на оригиналният хардуер.

Структура на емулатор

В общия случай емулаторът се разделя на модули, които които съответстват на емулираната компютърна подсистема. Най-често емулаторът се състои от следните модули:

- Процесорен емулатор / симулатор
- Модул на подсистемата с паметта
- Различни емулатори на входно-изходни (I/O) устройства

Шините често не се емулират поради проблеми със производителността или сложността на системата и затова периферните устройства комуникират директно със процесорът или с подсистемата с паметта.

Възможно е подсистемата с паметта да бъде редуцирана до масив от елементи при емулация, но това бързо се превръща в проблем в момента в който някое място от логическата памет на компютъра не отговаря на физическата памет.

Като резултат повечето емулатори имплементират поне две процедури за четене и писане в логическата памет, които имат задължението да мапват всеки достър до правилното място на правилният обект.

Процесорен симулатор

Процесорният симулатор е често най-сложната част от емулаторът. Много емулатори са написани с използването на предварително пакетирани процесорни симулатори с цел да се постигне добра и ефективна емулация на специфична машина.

Най-простата форма на процесорен симулатор са интерпретаторите. Интерпретаторът представлява компютърна програма, която следва изпълнението на емулираният програмен код и за всяка машинна инструкция изпълнява операции на процесорът домакин (host), които са саментично еквивалентни на оригиналните инструкции.

Това е възможно чрез назначаване на променливи за всеки регистър и флаг от симулираният процесор. По този начин логиката на симулираният процесор може да бъде директно преведена в софтуерно алгоритми, създавайки

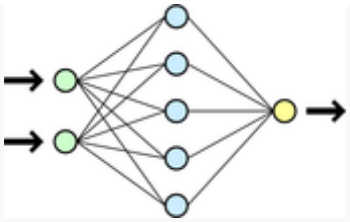
софтуерна ре-имплементация, която много наподобява оригиналната хардуерна имплементация.

Емулатори на входно-изходни (I/O) устройства.

Както споменахме по-рано – повечето емулатори не емулират шината. Затова всяко входно-изходно устройство се реазглежда като специален случай и не се предоставя постоянен интерфейс за виртуални периферни устройства. Това понякога се оказва полезно за производителността, тъй като всеки входно-изходен модул може да бъде приспособен към характеристиките на емулираното устройство.

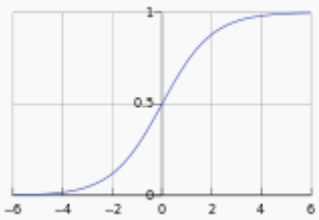
Невронни мрежи

Невронната мрежа е модел за обработка на информация, вдъхновен от изучаването на биоелектричните мрежи в мозъка на човека и животните, образувани от неврони и техните синапси. В наши дни учените често наричат*изкуствените невронни мрежи* просто *невронни мрежи*.



Опростен изглед на изкуствена невронна мрежа

Математическия аналог на биологичната невронна мрежа представлява множество от взаимносвързани прости изчислителни елементи (неврони). Всеки неврон приема сигнали от другите (под формата на числа), сумира ги, като сумата минава през активационна функция (най-често използваната е сигмоидалната функция $y=f(x)=1/(1+e^{-x})$), и така определя своята активация (степен на възбуда), която се предава по изходящите връзки към другите неврони. Всяка връзка има тегло, което умножавайки се със сигнала, определя неговата значимост (сила). Теглата на връзките са аналогични на силата на синаптичните импулси, предавани между биологичните неврони. Отрицателна стойност на теглото съответства на подтискащ импулс, а положителна - на възбуждащ.



сигмоидалната функция $y=f(x)=1/(1+e^{-x})$

В невронната мрежа обикновено винаги съществуват входен и изходен слой от неврони, във входния се въвежда информацията към мрежата, след това сигналите от входните неврони преминават през един или няколко слоя от междинни (скрити) неврони, според топологията на невронната мрежа, като сигналите накрая стигат до изходния слой, откъдето се чете получената информация.

Математически е доказано, че всяка невронна мрежа с поне един скрит слой от достатъчно на брой неврони между входния и изходния слой, може да моделира поведението на всяка съществуваща функция.

Теглата на връзките между невроните определят функционалността и поведението на невронната мрежа. За да бъде една невронна мрежа използваема и приложима към даден проблем, тя трябва да бъде предварително изучена.

Обучаването на една невронна мрежа се постигне чрез промяна на теглата на връзките между невроните и се осъществява чрез правила, които определят как да се променят тези тегла. Най-разпространеното сред тях е метода на обратното разпространение на сигнал за грешка (back-propagation), където за всеки изходен неврон се изчислява разликата от желаното му поведение, като се формира сигнал за грешка, който се движи назад към входния слой и по пътя си променя теглата на връзките така, че при следващата активация на мрежата грешката да бъде по-малка от сегашната. Този начин на "обучение" на мрежата обаче води до "забравяне"- мрежата бъде обучена да разпознава един елемент и впоследствие той не се повтори във входните данни, мрежата "забравя" този елемент.

Невронната мрежа в проекта е обучена да разпознава само числа. Лесно може да се обучи да разпознава букви и други символи както и различни шрифтове. Коефициентите на невронната мрежа се пазят и четат от файл, тъй като Java не позволява дефинирането на конструкции по-големи от 65KB в кода.

Laplacian of Gaussian Filter

Лапласовата мярка е двуизмерна изотропична мярка на 2-та частна производна на изображение. Лапласовата мярка приложена върху изображение подчертава регионите, в които има значителни различия в цветовете и затова се използва често за откриване на ръбове в изображение. Често Лапласовият филтър се прилага върху изображения, върху които преди това е бил приложен Гаусов филтър, който се изпозва за изглаждане на изображенията и намаляване на шума. Преди прилагането на тези филтри е необходимо върху изображението да бъде приложена трансформация, която го обръща в сива цветова схема.
Филтърът връща сиво изображение като резултат. Матрицата на филтърът, използван в проекта е дадена по-долу:

```
private final double[][] filter = {
      {0.00356759,0.00473254,0.00578289,0.00638064,0.00655581,0.00638064,0.00578
289,0.00473254,0.00356759},
```

```
	{0.00473254,0.00579836,0.00557669,0.0040679,0.00313232,0.0040679,0.0055766
9,0.00579836,0.00473254},
	{0.00578289,0.00557669,0.00152521,-0.00529969,-0.00891253,-
0.00529969,0.00152521,0.00557669,0.00578289},
	{0.00638064,0.0040679,-0.00529969,-0.0191446,-0.0262402,-0.0191446,-
0.00529969,0.0040679,0.00638064},
	{0.00655581,0.00313232,-0.00891253,-0.0262402,-0.0350557,-0.0262402,-
0.00891253,0.00313232,0.00655581},
	{0.00638064,0.0040679,-0.00529969,-0.0191446,-0.0262402,-0.0191446,-
0.00529969,0.0040679,0.00638064},
	{0.00578289,0.00557669,0.00152521,-0.00529969,-0.00891253,-
0.00529969,0.00152521,0.00557669,0.00578289},
	{0.00473254,0.00579836,0.00557669,0.0040679,0.00313232,0.0040679,0.0055766
9,0.00579836,0.00473254},
	 {0.00356759,0.00473254,0.00578289,0.00638064,0.00655581,0.00638064,
0.00578289,0.00473254,0.00356759}
};
```

Перспективна трансформация

Перспективната трансформация превръща четириъгълник във квадрат. Това се прави след като квадратите съдържащи числа са разпознати и преди да се подадат на невронната мрежа. След прилагането на перспективната трансформация върху намерените изображения, те са готови за разпонаване от невронната мрежа.

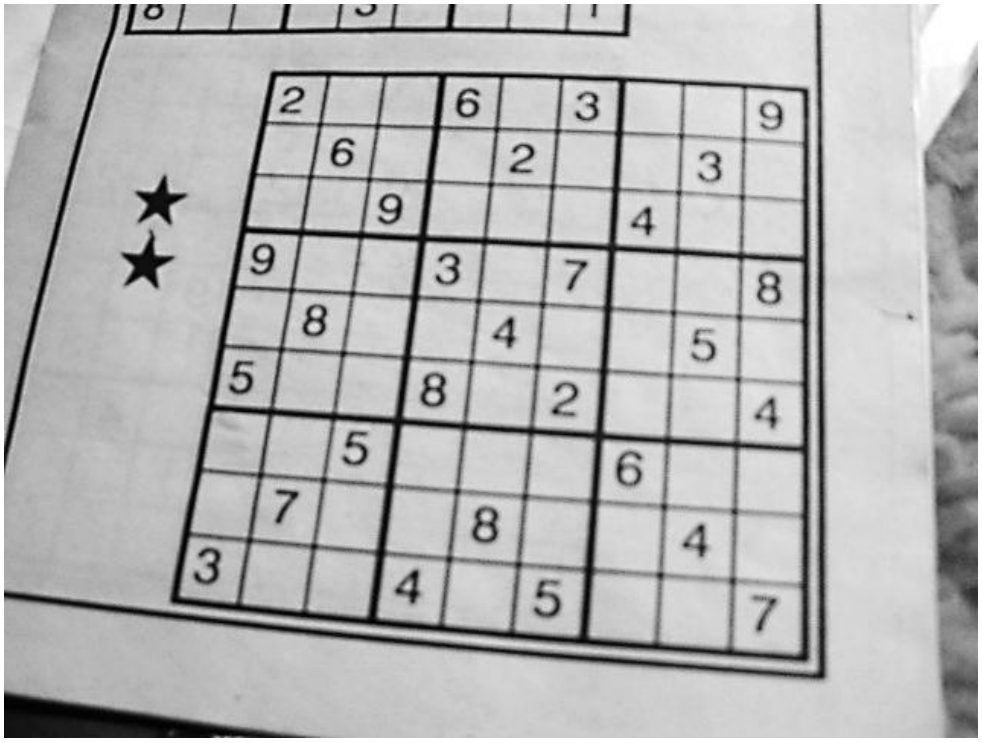
Стъпки на алгоритъмът

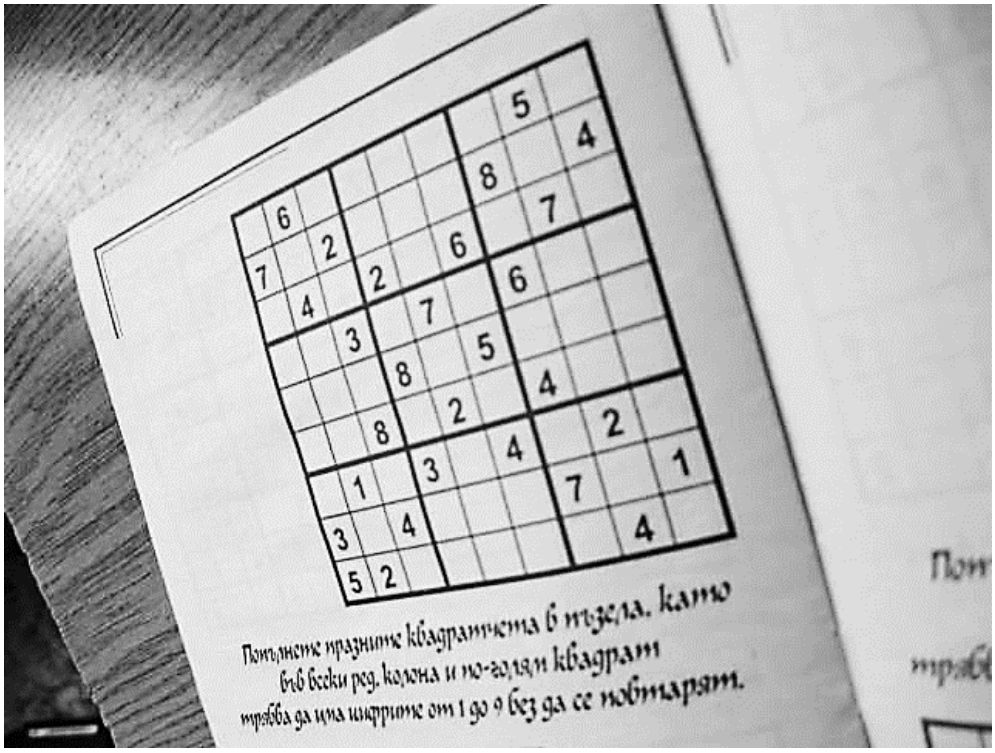
1. Смаляване на изображението до резолюция 640x480 (ако е по-голямо)
2. Прилагане на сив филтър върху смаленото приложение
3. Прилагане на Лапласов от Гаусов филтър върху новополученото сиво изображение
4. Конвертиране на обработеното изображение в бинарен вид
5. Намиране на рамката судокуто
6. Намиране на „квадратите“ (81) на судокуто (чрез свързани компоненти)
7. Прилагане на перспективна трансформация върху всеки един от намерените квадрати
8. Подаване на резултата (81 изображения) на невронната мрежа

Тестове

	5	6				2		
			3	8				
	4			7		9		1
		4	5					
	7	5	8		1	3	4	
					7	8		
6		2		4			5	
				9	6			
		1				7	8	

8								
		3	6					
	7			9		2		
	5				7			
				4	5	7		
			1				3	
		1					6	8
		8	5				1	
	9					4		




Попълнете празните квадратчета в тъмно, като
във всеки ред, колона и по-голям квадрат
трябва да има цифрите от 1 до 9 без да се повтарят.