Abstract

隨著科技的進步，越來越多的電子產品都有了語音辨識的力；舉例來說：手機上的語音撥號、PDA 上的語音命令以及銀行電話總機的身分證語音辨識…等。但是由於低階機器上的運算能力有限以及缺少了浮點數的運算，因此發展與應用都不夠普及。

本研究中針對這些限制提出了一連串的解決方式，並實作出 32 位元嵌入式系統上的語音辨識核心。語音辨識的過程可以大致上分成三個部分：擷取語音特徵、建立聲學模型和尋找最佳路徑；這三部分都有著大量的浮點數運算，因此我們主要使用的方式是將系統中的大部分運算改爲定點數，並利用查表法來加速部分步驟。本系統在辨識率降低大概 3%的情況下，分別使特徵擷取的過程加速了約 6.7 倍，尋找最佳路徑的過程加速了約 4.1 倍。

此外，還提出調整語音辨識過程中所用到的參數時所須注意的事項；例如：調整 Pruning bean-width 必須兼顧辨識率及辨識速度，並從中取得平衡；調整 Scaling factor 必須注意運算溢位。爲了方便偵測溢位，我們特別設計出一個新的 C++ class 以達到目的。希望未來設計完整的調整參數方式時可以將此觀點考量進去。

Due to the advancement of modern technologies, more and more digital devices are capable of recognizing speech for various applications, such as voice dialing on cell phones, voice commands on PDAs and voice-based telephone operators. However, the lack of floating-point arithmetic and the limited computing power of these mobile devices constrain the domain of speech-based applications.

This study proposes some methods to overcome these constraints. We have also implemented a recognition system on a 32-bit processor to show the feasibility of the proposed approach. In general, the process of speech recognition could be divided

into three steps, including feature extraction, acoustic model construction (training), and Viterbi search for the most-likely path (recognition). Since all of these time-consuming steps are floating-point operations, one straightforward way to reduce computation time is to use fixed-point operations instead. Moreover, we also built look-up tables to speed up the evaluation of some mathematical functions. The feature extraction is about 6.7 times faster and Viterbi decoding is about 4.1 times faster than their floating-point counterparts, while the recognition rate only drops about 3%.

We have also discussed the effects of several recognition parameters on the recognition results. For example, we have tried several values of the pruning beam-width in order to achieve a balance between the recognition rate and the computation time. We have also explored the scaling factor at various stages, which affects the occurrence of overflow. For better debugging, we have designed a new C++ class that can be used to detect overflows and handle the situation correctly. We sincerely hope that these proposed methods can pave a road to a better and more convenient world of speech-based applications.