Journal of Korea Game Society 2016 Apr; 16(2): 119-128
http://dx.doi.org/10.7583/JKGS.2016.16.2.119

게임 프로그래밍

컴퓨터 게임에서 HMM 기반의 명령어 신호 처리 시간 단축을 위한 방법

박도생, 김상철
한국외국어대학교 컴퓨터 및 전자시스템 공학부
phg1405@gmail.com, kimsa@hufs.ac.kr

A HMM-based Method of Reducing the Time for Processing Sound Commands in Computer Games

Dosaeng Park, Sangchul Kim
Computer Science and Engineering Major, Graduate School,
Hankuk University of Foreign Studies

요 약

컴퓨터 게임에서 대부분의 사용자 인터페이스 방법은 키보드, 마우스, 터치스크린이다. 사운드 형태 명령어의 전체 처리 시간은 크게 명령어 입력 시간과 인식 시간으로 구성된다. 본 논문은 명령어 신호 전체를 입력받지 않고 일부 앞부분 신호만을 받음으로써, 입력 시간을 줄여 전체 처리 시간을 단축하는 방법을 제안한다. 우리의 방법에서는 HMM(Hidden Markov Process)를 이용해 명령어 신호를 인식하는데, 전체 신호 및 부분 신호들에 대해 별도의 HMM을 구성한다. 플랫홈 게임의 대표 명령어들을 음성과 손바닥 소리로 표현해, 본 논문의 방법을 실험했다. 실험 결과, 인식률의 큰 저하 없이 명령어 처리 시간을 줄임을 알 수 있었다. 본 연구는 게임의 사용자 인터페이스 방법을 다양화하는데 기여할 것이다.

ABSTRACT

In computer games, most of GUI methods are keyboards, mouses and touch screens. The total time of processing the sound commands for games is the sum of input time and recognition time. In this paper, we propose a method for taking only the prefixes of the input signals for sound commands, resulting in the reduced the total processing time, instead of taking the whole input signals. In our method, command sounds are recognized using HMM(Hidden Markov Model), where separate HMM's are built for the whole input signals and their prefix signals. We experiment our proposed method with representative commands of platform games. The experiment shows that the total processing time of input command signals reduces without decreasing recognition rate significantly. The study will contribute to enhance the versatility of GUI for computer games.

Keyword : Command Sound(명령어 신호), HMM, User Response Time(사용자 반응시간), Recognition Rate(인식률)

Received: Mar, 10, 2016 Revised: Apr, 8, 2016
Accepted: Apr, 14, 2016
Corresponding Author: Sangchul Kim(Hankuk University of foreign Studies)
E-mail: kimsa@hufs.ac.kr
ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

게임 플레이에서 사용자 인터페이스는 몰입감과 흥미를 좌우하는 큰 요인들 중 하나다. 기존 대부분의 컴퓨터 게임은 키 스트로크나 마우스 클릭을 통해 명령어를 입력한다. 최근 체감형 스포츠 및 레저 게임의 경우, 사람의 몸동작을 카메라 또는 모션 센서를 이용해 인식하고 있지만, 아직 사운드를 입력 방안으로 채택한 게임은 많지 않다. 우리는 컴퓨터 게임의 입력 방안으로 사운드의 중요성이 점차 증대할 것으로 전망한다. 그 이유는 사용자의 다양한 입력 양식에 대한 욕구, 음악 게임이나 교육용 게임에서의 활용 가능성이 높기 때문이다.

사운드 형태의 명령어 (줄여서, 명령어 신호)를 이용한 게임 진행은 여러 가지 장점을 갖는다. 여러 키 스트로크(key stroke) 조합을 사용해야 하거나 이를 암기해야하는 불편함을 덜어준다. 외치는 고함 같은 소리는 키 스트로크나 마우스 클릭으로 표현할 수 없고, 게임에 몰입한 사용자의 감정을 잘 표현할 수 있다. 또한 명령어 신호는 키보드 명령어에 비해 입력 속도가 10%이상 빠르고 입력 에라가 현저히 낮다고 보고된다[1].

반면 명령어 신호 처리를 위해서는 해당 명령어 신호가 입력된 후 인식 단계를 거쳐 어떤 명령어인지를 파악하는 과정을 거친다. 따라서 이 과정에 소요되는 시간만큼의 지연을 피할 수 없다. 따라서 전략 게임이나 느린 템포의 플랫홈 게임과 같이 명령어 입력에 일정 시간 지연을 허용하는 게임 장르에 적합할 것이다.

명령어 신호의 처리 시간은 크게 신호 입력 시간과 인식 시간의 합으로 볼 수 있다. 일반적으로 입력 시간은 인식 시간보다는 훨씬 크다. 따라서 명령어 신호의 전체 처리 시간은 명령어의 입력 시간에 따라 결정되고, 입력 시간을 줄이는 것이 중요하게 된다.

본 논문에서는 HMM(Hidden Markov Model) [2] 기반의 사운드 인식 엔진을 이용해. 명령어 신호의 처리 시간을 감소시키는 방법을 제안한다.

본 논문에서 제안하는 방법의 핵심은 명령어 신호의 전체가 아닌 전반부 일부만을 입력받아 인식 직업을 수행하는 것이다. 따라서 명령어 신호 신호의 일부를 인식하기 위해 별도의 HMM들을 구축한다. 또한, 명령어의 부분 신호를 이용할 때, 부분 신호의 길이에 따라 인식률을 가장 높일 수 있는 HMM들을 파악하는 방법을 제안한다. 우리는 11개의 명령어 신호를 대상으로 제안된 우리의 방법의 특성을 분석하였다. 실험 결과, 전체 신호를 사용 시의 인식률과 비교해 거의 차이가 없이, 명령어 처리 시간을 크게 줄임을 알 수 있었다.

지금까지 음성 인식에 대한 연구는 상당히 많이 진행되어 각종 분야에서 실제 적용이 되고 있다. 최근 웹 검색을 위한 사용자 인터페이스에서 음성 인식은 높은 인식률을 보이고 있다[3]. 또한 음성 인식을 위한 컴퓨터 GUI에 대한 연구는 이제까지 많이 발표되었다[4]. 이들 연구는 파일처리나 프로그램 실행과 같은 운영체제 명령어를 마우스 클릭이나 키 스트로크 대신 음성으로 표현하는 것을 지원한다.

컴퓨터 게임의 조작을 위해 마우스나 키보드를 대신하는 방법에 대한 연구도 다소 발표되었다. 이들 연구에서는 대부분 음성이나 [1,6,7,8,9,10,12], 신체 동작으로[13,14]으로 게임 명령어를 표현하고 있다. [1]에서는 마작 보드 게임을 음성으로 진행하여, 컴퓨터와 사람간의 상호 작용에 대한 만족도가 크게 높아짐을 확인했다. [6, 7]에서는 음성 명령어는 게임 공간 대한 몰임감을 높이고 게임 플레어들이 게임 객체들을 조작하는 스타일에도 변화를 일으킴을 확인했다. [8]에서는 음성은 게임 플레이들간의 의사소통 방법을 변화시켜 게임 공간에서의 경험을 보다 사실적으로 느끼게 한다고 주장한다. [9]에서 저자들은 게임 플레이어들의 음성을 통해 그들의 감정을 판단하여 게임 진행에 활용하는 방법을 제안했다. 대부분의 앞선 연구들이 음성 사용의 장점을 주장하고 있는 반면에, [10]에서는 한계점을 지적하기도 했다. [10]에 따르면, 테트리스 게임에서 게임 진행 템포가 빨라지면 키보드

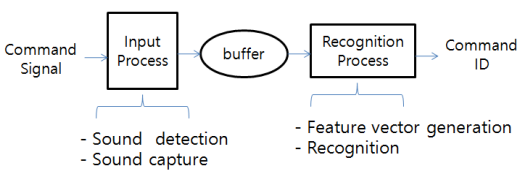
입력보다 게임 조작을 하는데 어려움이 있다고 보고했다. [11]에서는 퀴즈 로봇이 음성으로 퀴즈 문제를 내고 답을 이해하는데 어려움이 없음을 보여준다. 게임 진행에서는 일반 대화에서 보다 다소 빠른 템포의 음성을 사용한다고 할 수 있다. [12]에서는 다소 빠른 템포의 음성도 기존 음성 인식 기술로도 일반 대화 음성에서와 비슷한 인식률이 나옴을 보여준다.

대부분의 앞선 연구들은 음성 인식 분야에서 연구된 기술을 컴퓨터 게임의 사용자 인터페이스에 적용시킨 것으로 볼 수 있다. 음성 명령어를 인식하는 데 소요되는 시간 지연을 단축하는데 대한 연구는 거의 발표되지 않았다.

음성 및 각종 사운드 인식의 대표적인 기술로는 HMM과 신경망[5]을 들 수 있다. 신경망 기술은 HMM과 결합해서 많이 사용한다. HMM의 핵심은 스토캐스틱 프로세스를 이용해 주어진 입력에 대한 발생 확률을 구하는 것이다. HMM은 개별 단어뿐만이 아니라 문장인식에 적용되고 있다.

2. HMM 기반의 일반 명령어 신호 인식

사운드 인식을 이용하는 많은 응용 프로그램에서는 사운드 시작 버튼을 누르는 등과 같이 사운드 입력을 하는 시점을 응용 프로그램 입장에서는 알 수 있다. 하지만, 컴퓨터 게임의 경우, 진행을 위한 명령어 신호 신호는 언제라도 들어 올 수 있다. 일반적인 명령어 신호의 인식은 [Fig. 1]의 과정을 거쳐 진행되게 된다.

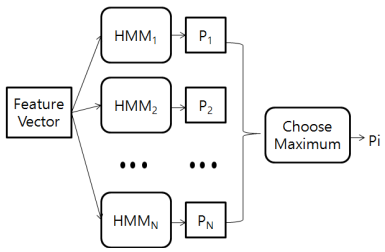


[Fig. 1] The procedure of processing command sound signals

입력 프로세스는 명령어 신호의 유무를 계속적으로 감시하고 있다. 신호를 감지하면 명령어 신호를 캡처해 버퍼에 넣는다. 인식 프로세스는 버퍼로부터 사운드 신호를 페치(fetch)한 후, 전처리, 특징 벡터 생성 및 인식 단계를 차례로 수행한다 [12]. 전처리 단계에서는 사운드 신호의 스펙트라 필트 (spectral tilt) 현상을 보완하기 위해 하이패스 필터링를 수행 후, 전체 신호를 여러 프레임들로 나눈다.

HMM을 이용한 사운드 인식에서는 특징 벡터 생성은 윈도우윙(windowing) 및 스펙트럼 계수 추출을 진행된다. 본 논문에서는 윈도우잉은 해밍 윈도우(hamming window)를, 스펙트럼 계수는 MFCC (Mel-frequency cepstral coefficients)를 사용한다.

HMM에서 인식과 관련한 주요 파라미터들로서 상태(state) 수, 상태들 간의 변이 확률(transition probability), 심볼(observation symbol)의 표현 방법 등이 있다. 우리의 경우에는 프레임 신호의 특징 벡터가 심볼이 된다. 명령어 신호를 HMM을 이용해서 인식하는 과정은 [Fig. 2]에 나타난다.



[Fig. 2] HMM-based recognition of voice commands

3. 명령어 신호의 일부를 이용한 명령어 인식 방법

명령어 신호의 부분 신호를 이용해 명령어를 인식하는 전반적인 과정은 [Fig. 1]과 비슷하지만, 입력 프로세스는 [Fig. 3]과 같이 진행된다. [Fig. 3]

의 Step 2에서 보다시피, 명령어 신호를 캡처할 때에 최대 시간을 설정해 놓는다. 우리는 명령어 신호를 세그먼트 단위로 나누어 부분 신호를 구성하는데, T_s는 세그먼트의 길이 (시간 단위)이다. J는 사용자가 지정하는 파라미터이다

```
do {
  Step 1: wait on an occurrence of
          a command signal;
  Step 2: capture the incoming signal
          either for Ts * J seconds or
          until the end of the signal;
  Step 3: store the signal into the buffer
} while (true);
```

[Fig. 3] The operation of input process

부분 명령어 신호를 사용하기 위해서는 부분 신호들로 훈련된 HMM들이 필요하다. 그런 훈련 작업은 [Fig. 4]의 ConstructHmms 프로시저에 기술되어 있다. 주어진 J에 대해, 그림 5의 BestHmms는 해당 길이의 부분 명령어 신호들의 인식하는데 적합한 HMM들을 선택하는 프로시저이다. 그림 4과 5에서 명령어들은 N개 타입으로 구성되고, 실험용 명령어 신호의 집합은 M개라고 가정한다.

Input: c_i^m, $1 \le i \le N$, $1 \le m \le M$, T_s
Output: *HMM*'s

Procedure *ConstructHmms:*

Step 1)
for i = 1 **to** N
  len_i = $AVERAGE_{1 \le m \le M}$ $duration(c_i^m)$;
  L_i = $ceiling(\frac{len_i}{T_s})$;
}

Step 2)
for i = 1 **to** N
  for j = 1 **to** L_i
    train $HMM_{i[j]}$ by using $c_{i[j]}^1$, ..., $c_{i[j]}^M$;

[Fig. 4] A Procedure of training HMMs

[Fig. 4]의 ConstructHmms 프로시저에서는 훈련용 명령어 신호들을 여러 길이(duration)의 부분 신호들로 나누고, 각 길이에 대해 별도의 HMM을 훈련시킨다. c_i^m는 m번째 명령어 집합에 있는 i 타입의 명령어 신호를 나타낸다. L_i는 i 타입 명령어 신호가 몇 개의 세그먼트로 구성된 지를 나타낸다. ceiling(·) 함수는 소수점이하를 정수 1로 올리는 올림 함수이다.

[Fig. 4]의 Step 2에서는 $c_{i[j]}^m$ 신호들을 이용해서, $HMM_{i[j]}$의 모델 파라미터를 훈련시킨다. $c_{i[j]}^m$는 c_i^m 신호의 앞부분으로서, 세그먼트 j개만큼의 길이를 갖는다. 즉 $HMM_{i[j]}$는 i 타입 명령어의 부분 신호들로 훈련된 HMM이다. 예를 들어, T_s가 0.05sec이고 L_i가 4이면 $c_{i[2]}^m$는 c_i^m 신호의 0.1sec 만큼의 앞부분을 나타낸다. 만약 어떤 m, i 및 j에 대해 j가 L_i보다 작으면, $c_{i[j]}^m$는 전체 신호인 c_i^m로 정의된다.

Input: c_i^m, $1 \le i \le N$, $1 \le m \le M$,
  T_s
  $HMM_{i[j]}$, $1 \le i \le N$, $1 \le j \le L_i$,
  J
Output: *HS*, *RR*

Procedure *BestHmms*

Step 1)
HS = ∅;

Step 2)
for each $c_{i[J]}^m$, $1 \le i \le N$, $1 \le m \le M$
  for v = 1 **to** N
    for w = 1 **to** L_v

```
        apply c^m_{i[J]} to HMM_{i[w]} and
            obtain P[m,i,v,w];
Step 3)
for m = 1 to M
  for i = 1 to N {
    A[m,i] =
        arg MAX_{1 ≤ v ≤ N, 1 ≤ w ≤ L_i}   P[m,i,v,w];
  }

Step 4)
for i = 1 to N  {
  for m = 1 to M {
    let A[m,i] be <a,b>;
    if(a == i)
        add b to S_i;
  }
  h_i = mode(S_i);

  HS = HS ∪ HMM_{i[h_i]} ;

}

Step 5)
Step 5-1)
  apply all c^m_{i[J]}'s (1 ≤ i ≤ N, 1 ≤ m ≤ M)
    to HMM's in HS;

Step 5-2)
  compute the recognition rate and store
 it into RR;

Step 6) output HS and RR;
```

[Fig. 5] A Procedure of finding the HMM's for partial input signals

[Fig. 5]의 BestHmms 프로시저에서는 세그먼트 수인 J에 대해, T_s * J 만큼 길이의 부분 명령어 신호들을 가장 잘 인식할 수 있는 HMM들을 찾고, 예상 인식률을 구한다.

입력 신호인 c_i^m들은 그림 4의 입력 신호들과는 다르다고 가정한다. 또한 명령어 신호 집합의 개수인 M도 그림 4의 경우와 다를 수 있다. 입력으로 주어진 *HMM*들은 ConstructHmms 프로시저에서 구한 것들이다.

[Fig. 5]의 Step 2에서는 각 $c_{i[J]}^m$들 모든 HMM들에 적용시켜, P[m,i,v,w]를 구한다. P[m,i,v,w]는 $c_{i[J]}^m$를 $HMM_{i[w]}$에 적용시켜 구한 확률이다.

[Fig. 5]의 Step 3에서는 각 명령어 신호 $c_{i[J]}^m$에 대해, 가장 높은 확률 값을 가지는 HMM을 찾아 그것의 인덱스 쌍을 벡터 형태로 A[m,i]에 저장한다. 예를 들어 만약 $HMM_{a[b]}$ 가장 높은 확률값을 출력한다면, A[m,i] = <a, b>가 된다.

Step 4에서는 각 명령어 타입 i에 대해서, 첫 번째 요소로 i를 갖는 A[m,i]들을 찾아 그들의 두번째 인자를 리스트 S_i에 저장한다. 만약 어떤 A[m,i] = <i, b>라고 하면, 이것은 $c_{i[J]}^m$에 대해 $HMM_{i[b]}$가 다른 HHM들보다 가장 큰 확률을 가지므로, 명령어 타입 i의 신호를 인식에 하는데 적합하다고 볼 수 있다. mode(S_i)는 S_i내에서 가장 빈도가 높은 값을 구하는 함수이다. 만약 h_i=3이면, 명령어 타입 i의 인식을 위한 HMM으로 $HMM_{i[3]}$이 선택되었음을 나타낸다.

Step 5에서는 HS에 저장된 HMM들을 이용해서 평균 인식률을 구하고, 이를 RR에 저장한다.

실제 응용에서는, 여러 J에 대해 BestHmms 프로시저를 수행하여, 명령어 타입별 가장 적합한 HMM과 인식률을 구한다. 그리고 인식률에 따라서 응용에 맞는 수준의 J를 선택해 사용하면 될 것이다.

4. 실 험

우리는 제안된 명령어 신호의 입력 시간 단축 방법을 실험을 통해 그 유용성을 분석했다. 플랫홈 게임에서 자주 사용될 수 있는 명령어 11개를 선정했다: 왼쪽, 오른쪽, 위로, 아래로, 앞으로, 뒤로, 점프, 시작, 그만, 짝, 짝짝.

이들 명령어는 박수 소리인 '짝'과 '짝짝'을 제외하고는 모두 음성으로 표현된다. '짝'과 '짝짝'을 포함한 이유는 게임 플레이에서 사람의 감정을 잘 표현할 수 있는 수단일 수 있다고 생각해서 이다. 그림 3과 4의 프로시저는 Java로 구현했고, 3.5GHz I5 CPU, 8G 메모리를 가진 PC에서 동작

시켰다.

입력 신호를 프레임으로 나눌 때, 프레임의 크기는 일반적인 값인 20ms로 했다. 프레임별 특징 벡터는 MFCC F 개, 1차 미분 계수 F개, 2차 미분 계수 F개, 스펙트럼 에너지 3개로 구성된다. 따라서 입력 벡터의 길이는 3*F + 3이 된다. 특징 벡터에 대해 벡터 양자화(Vector Quantization)를 이용했고, 벡터 테이블의 인덱스를 HMM에서의 심볼로 표현한다 심볼의 개수를 512로 정했다. HMM의 훈련과 인식에는 Baum-Welch 방식과 Viterbi 방식을 채택했다[2].

특징 벡터의 길이를 결정하는 F와 상태의 수인 S는 HMM 기반 인식의 주요 파라미터이다. 우리는 이들에 대해 다양한 값을 설정한 후 제안된 방법에 미치는 영향을 분석했다.

먼저 실험자 5명을 대상으로 명령어별 수십회의 음성 신호와 박수 신호를 확보했다. 신호의 각 샘플은 16비트로 표현했다. 이들을 적절히 나누어, 그림 4와 5의 프로시저를 실행하고 유용성을 검증하는데 사용했다. 우리의 실험에서는 화자 의존적인 인식을 가정하였기에, 각 실험은 실험자별로 진행되었고 아래 결과에 나오는 수치들은 평균값이다.

먼저 우리는 그림 4와 5에서 기술한 방법을 적용해, 여러 신호 길이에 대해, 인식에 사용할 HMM들을 구했다. T_s=0.04sec, S=6, F=9로 정했고, 사용한 명령어 신호의 최대 길이에 따라 최대 세그먼트 수는 12로 정했다. 그리고 각 J에 대해, T_s * J 만큼 길이의 명령어 부분 신호들을 대상으로 성능을 분석했다. 우리는 본 장에 기술된 모든 실험에서 T_s=0.04sec로 설정했다. 성능 평가 지표로 인식률인 RR, 명령어 신호의 입력 시간 감소률인 ITR(Input Time Reduction) 및 명령어 신호의 처리 시간 감소율인 TTR(Total Time Reduction)을 사용했다. ITR과 TTR은 다음과 같이 정의했다.

$$ITR = 1 - \frac{\sum_{i=1}^{N} duration\ of\ c_{i[J]}}{\sum_{i=1}^{N} duration\ of\ c_i}$$

$$TTR = 1 - \frac{\sum_{i=1}^{N} duration\ of\ c_{i[J]} + recognition\ time\ of\ c_{i[J]}}{\sum_{i=1}^{N} duration\ of\ c_i + recognition\ time\ of\ c_i}$$

[Fig. 6] (a)와 (b)는 첫 번째 실험 결과를 보여준다. 각 명령어 타입 i에 대해, RR(Our)는 [Fig. 5]의 방법을 적용했을 때, RR(ES)는 전체 신호로 훈련된 HMM을 적용했을 때, RR(random)은 무작위로 선택한 길이로 훈련된 HMM을 사용했을 때의 인식률이다.

그림에서, J=12는 명령어 신호 전체를 입력으로 사용한 경우를 나타내는데, RR(ES)의 경우에 인식률은 약 91%이다. 앞으로 이것을 RR_{normal} 로 부른다. RR_{normal}이 100%에 미치지 못하는 이유는 명령어 신호의 다양성으로 인한 것이다. 훈련 단계에 사용한 사운드 신호와 인식 단계에서 사용하는 신호가 음색과 템포 면에서 얼마나 유사한 지에 때라 인식률과 인식시간은 차이가 날 것이다. 참고로 실험에서는 이들을 일치하고자 했다.

[Fig. 6] (a)의 결과에서 보다시피, 우리의 방법이 전체 신호로 훈련된 HMM을 사용한 경우와 무작위 선택에 비해 인식률 면에서 나은 것을 알 수 있다. 우리 방법의 경우, J가 5 ~ 11 사이에서 인식률이 RR_{normal} 보다 3~4% 높다는 것을 알 수 있다. 이것은 부분 명령어 신호는 부분 신호들로 훈련된 HMM들을 사용하는 것이 인식률 면에서 유리함을 알 수 있다.

ITR은 정의에 따라 입력 신호의 길이에 크게 좌우되므로, 실험 결과에서 입력 신호의 길이가 짧아지면 그 만큼 ITR도 증가함을 확인할 수 있다. TTR은 ITR보다 조금 큰 값을 가진다. 실험을 통해 관찰한 인식 시간이 입력 시간에 비해 현저히 짧기 때문에, TTR의 정의에 따라 예측 가능한 결

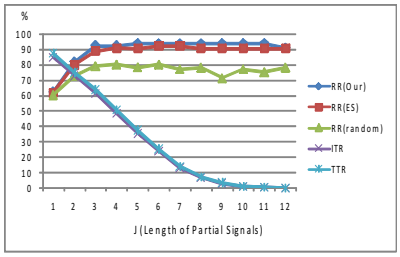
과이다. [Fig. 6] (b)는 입력 신호의 길이별 평균 인식 시간을 보여준다. J가 증가하면 인식 시간은 증가하지만, 수 ms 내외의 작은 값이다.

R(Our) 그래프를 보면, J가 3이하의 경우에는 입력 신호의 길이가 짧아지게 되면 인식률이 현저히 낮아지고, 3보다 큰 경우에는 인식률은 RR_{normal}과 큰 차이를 보이지 않지만 ITR과 TTR은 크게 증가한다. RR(ES) 그래프 역시 비슷한 경향을 보인다. TTR의 큰 증가는 게임 플레이에서 사용자 반응 시간의 단축을 나타내므로, 사용자들의 게임 만족도를 높일 것이다.

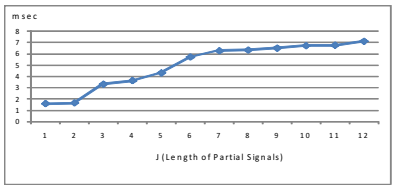
J=3의 경우 인식률은 91%, ITR은 약 65%에 이른다. 이것은 명령어 신호 전체의 35% 정도의 길이를 갖는 부분 신호를 사용한다는 것이다. 이것은 명령어 신호의 전체가 아닌 적당한 길이를 갖는 전반부만을 사용해도 전체 신호를 사용한 경우만큼의 인식률을 달성할 수 있음을 나타낸다. 그 이유 두 가지로 볼 수 있다. 하나는 많은 명령어 신호들은 앞부분이 서로 다르다는 것이다. 예를 들면 '앞으로'나 '뒤로'는 벌써 첫 번째 음절이 다르다. 또 다른 이유는 명령어 신호의 부분 신호들에 대한 HMM들을 별도로 훈련시켜 이들을 인식 작업에서 사용하기 때문으로 판단된다.

J=1에서는 인식률은 RR_{normal}에 비해 현저히 낮지만, ITR은 약 85%로 크게 높아진다. 이것은 인식률의 저하를 감수하면, 명령어 입력 시간은 상당히 줄일 있다는 것을 알 수 있다. 사용자는 게임 플레이에서 정확한 사운드로 명령어를 발음해 인식률을 높이고자 노력하는 것도 재미 요소 중 하나일 것이다.

[Fig. 6] (b)에서 보다시피, J가 증가하면 인식 시간이 늘어난다. J의 증가는 HMM 관점에서는 입력 심볼 열이 커지는 것이기 때문에, 더 긴 상태 변환을 해야 하기 때문이다.



(a) *RR, ITR, TTR*



(b) Recognition time

[Fig. 6] Results for partial signals of sound commands (T_s=0.04sec, S=6 F=9)

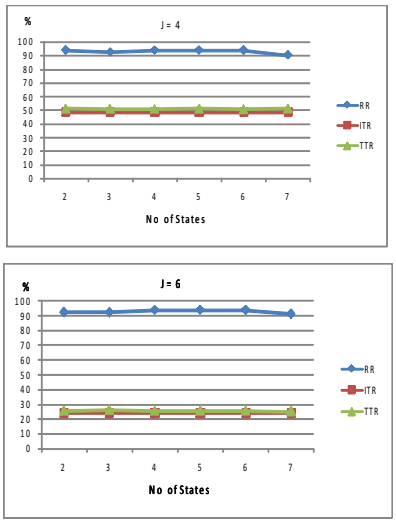
앞에서 언급한 대로 HMM의 상태 수 S는 HMM의 특징을 정하는 주요 파라미터들 중 하나이다. HMM의 적절한 상태 수는 응용 마다 다를 수 있고, 이것은 실험적으로 구해진다[15]. [Fig. 7]은 F=9로 설정한 후, 다양한 J에 대해 S 값이 우리의 방법에 미치는 영향을 정리한 것이다.

실험 결과, S가 변화해도 인식률, ITR 및 TTR은 큰 변화를 보이지 않는다. ITR과 TTR은 J에 영향을 주로 받기에 당연한 결과로 보인다. 그림 7에서, S=4 ~ 6인 경우의 인식률이 S=7보다 약 2~3%높아지는 경향을 보였다. 또한 S=2인 경우도 다른 경우와 비슷한 인식률을 보인다. 이것은 실험에 사용된 11개 정도의 명령어 신호들은 데이터 측면에서 유사성이 적기 때문에 작은 크기의 HMM로도 충분히 구분할 수 있음을 알 수 있다.

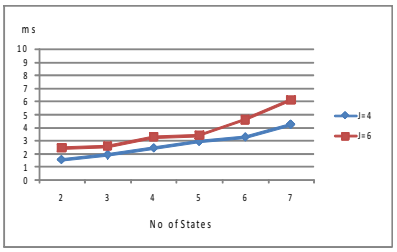
한글 단어의 음식 인식에서는 HMM의 상태 수로 6이 자주 사용된다. 게임 명령어 신호의 길이를 알아내기 위한 HMM에서는 6 정도를 사용하면 될 것이다.

[Fig. 7]의 (b)는 S이 인식 시간에 미치는 영향을 보여준다. S가 증가하면 인식 시간이 증가하는데, S = 7까지는 인식 시간이 10ms 미만이다. 우

리가 사용하는 Viterbi 방식에서 디코딩 시간은 S에 비선형적으로 비례하기에, S가 커지만 인식 시간은 크게 늘게 된다.



(a) *RR*, *ITR*, *TTR*



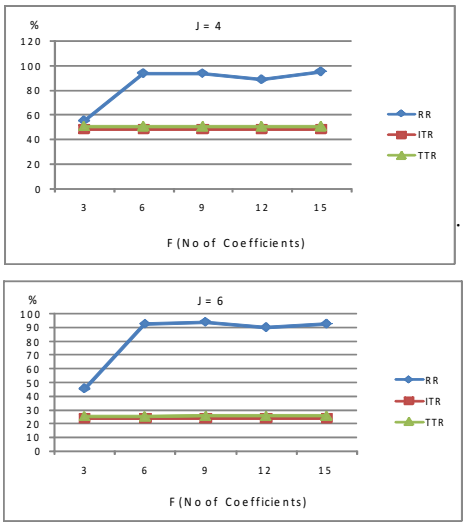
(b) Recognition time

[Fig. 7] Performance comparison w. r. t various *S*'s (*F*=9)

[Fig. 8]은 F가 제안된 방법에 미치는 영향을 정리한 것이다. S=6으로 설정하여 실험했다. F가 증가하면 특성 벡터는 길어지고, 그 결과 작은 값에 비해 입력 신호의 속성을 세밀하게 표현할 수 있다. 하지만 MFCC 개수의 증가가 인식률의 향상을 반드시 가져오지는 않기에[16], 적절한 F값은 실험적으로 구해야 한다.

실험 결과에 따르면, 각 J에 대해, F가 일정한 값 이상이 되면 F가 변화해도 인식률, ITR 및 TTR은 크게 변화하지 않았다. 하지만 F가 3이하가 되면 인식률은 크게 낮아진다. 그 이유는 특징 벡터의 길이가 짧아, 명령어 신호의 특징을 다른 신호들과 구분되도록 충분히 표현할 수 없기 때문일 것이다.

F가 증가해도 인식 시간에는 큰 변화가 없었다. 이것은 F가 커지면 따라서 특징 벡터의 경우의 수가 크게 증가하지만, 벡터 양자화를 사용해 때문에 심볼 값은 일정한 범주 내에서 고정되어 있기 때문이다.

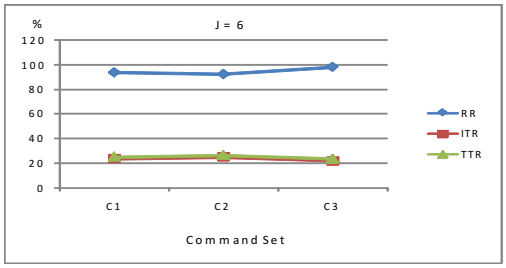


[Fig. 8] Performance comparison w. r. t various *F*'s (*S*=6)

또 다른 실험은 명령어 개수가 입력 신호의 단축과 인식률에 미치는 영향을 S=6, F=9로 설정한 후 관찰했다. 앞 실험들에서 사용한 명령어 집합, "앞으로, 뒤로, 아래로, 위로, 점프, 시작, 그만" 명령어들로 구성된 집합, "앞으로, 뒤로, 점프, 아래로" 명령어들로 구성한 집합에 대해 실험했다. 그림 9는 실험 결과를 보여주는데, C1, C2 과 C3은 이들 집합을 각각 나타낸다.

실험 결과, 명령어 개수가 줄면 인식률은 증가했다. 특히 그림 9에서처럼, J=6인 경우 C3는 인식률

이 C1에 비해 크게 높았다. C3는 플랫홈 게임에서 사용되는 대표적 명령어들로 구성되어 있다. 이들 대표적인 명령어들만 게임 플레이에서 사용한다면, 명령어 입력의 시간은 현저히 줄일 수면서도 높은 인식률을 달성할 수 있을 것이다. 실험에서 명령어 수가 줄면 그 만큼 인식 시간도 감소함을 확인할 수 있었다.



[Fig. 9] Performance comparison w. r. t the various sizes of command sets (S=6, F=9)

5. 결 론

대부분 컴퓨터 게임에서는 키보드, 마우스, 터치 스크린 등을 이용해 게임 진행을 위한 명령을 입력한다. 본 논문에서는 우리는 게임 명령어를 사운드 형태로 입력할 때, 명령어 신호의 일부만을 사용해 입력 시간을 단축하는 방법을 제안했다. 우리의 방법을 사용하면 명령어 처리 시간을 줄여 보다 빠른 사용자 반응을 가능하게 한다.

플랫홈 게임에서 많이 사용하는 11개 명령어를 대상으로 우리의 방법을 실험했다. 우리는 실험에서 명령어 인식은 HMM을 기반으로 동작한다. 실험 결과에 따르면, 명령어 신호가 일정한 길이 이상일 때 명령어 신호 전체를 사용한 경우에 비해 인식률은 거의 감소가 없고, 명령어 입력 시간과 처리 시간을 단축시킬 수 있었다. 또한 HMM의 주요 특징 변수인 상태 수와 특징 벡터의 길이는 인식률에 크게 영향을 미치지 않음을 알 수 있었다. 하지만 상태 수는 인식 시간에 미침을 알 수 있었다. 또한, 명령어 수를 줄면 인식률도 높아지고 인식 시간도 현저히 줄어들음을 알 수 있었다. 향후에는 우리의 방법이 보다 큰 명령어 집합을 필요로 하는 게임에도 적용가능한 지를 실험해 보고자 한다.

인간은 음성이나 기타 신체 일부를 이용한 사운드로 의사 및 감정을 편리하게 표현한다. 이런 점을 감안하면, 본 연구는 멀티 모달의 게임 사용자 인터페이스를 지원함으로써 게임 플레이의 흥미를 크게 높일 수 있을 것이다.

ACKNOWLEDGMENTS

This work was supported by Hankuk University of Foreign Studies Research Fund of 2015

REFERENCES

[1] Zhang Jie, Zhao Ji, Bai Shuanhu, and Huang Zhiyong, "Applying Speech Interface to Mahjong Game", Proceedings of 10th International Conference on Multimedia Modelling, 2004, pp.86-92.

[2] http: //en.wikipedia.org/wiki/Hidden_Markov_model

[3] Alexander Franz, Brian Milch, Searching the Web by voice, Proceeding of Proceedings of the 19th International Conference on Computational Linguistics, Vol. 2, 2002, pp.1-5.

[4] R. Rogoff, "Voice Activated GUI-the Next User Interface", Proceedings of Professional Communication Conference, 2001, pp.117-120.

[5] H Sakoe, R Isotani, K Yoshida, KI Iso, and T Watanabe, "Speaker-Independent Word Recognition Using Dynamic, Programming Neural Networks", Proceeding of International Conference on Acoustics, Speech, and Signal Processing, 1989, pp.29-32.

[6] J. -C. Bolot, S. Fosse-Parisis, "Adding Voice to Distributed Games on the Internet", Proceedings of Seventeenth Annual Joint Conference of the IEEE Computer and Communications Societies, 1998, Vol. 2, pp.480-487.
[7] Chi-Wen Fann, Jehn-Ruey Jiang, and Jih-Wei Wu, "Peer-to-Peer Immersive Voice Communication for Massively Multiplayer Online Games", International Conference on Parallel and Distributed Systems, 2011, pp.759-764.
[8] Jehn-Ruey Jiang, Hung-Shiang Chen, "Peer-to-Peer AOI voice chatting for massively multiplayer online games", International Conference on Parallel and Distributed Systems, 2007, Vol. 2, pp.1-8.
[9] Kiyhoshi Nosu, et. al, "Real Time Emotion-Diagnosis of Video Game Players from their Facial Expressions and its Applications to Voice Feed-Backing to Game Players", International Conference on Machine Learning and Cybernetics, 2007, Vol. 4, pp.2208-2212.
[10] XiaoJie Yuan, Jing Fan, "Design and implementation of voice controlled Tetris game based on Microsoft SDK", Proceedings of International Conference on Multimedia Technology, 2011, pp.275-278.
[11] Izaya Nishimuta, et. al, "A Robot Qquizmaster That Can Localize, Separate, and Recognize Simultaneous Utterances for a Fastest-voice-first Quiz Game", International Conference on Humanoid Robots (Humanoids), 2014, pp.967-972.
[12] Hiroaki Nanjo, et. al, "A Fundamental Study of Novel Speech Interface for Computer Games", Proceedings of 13th International Symposium on Consumer Electronics, 2009. pp.558-560.
[13] Y. Sriboonruang, P. Kumhom, and K. Chamnongthai, "Visual Hand Gesture Interface for Computer Board Game Control", IEEE Tenth International Symposium on Consumer Electronics, 2006, pp.1-5.
[14] J Payne, et. al, "Gameplay Issues in the Design of Spatial 3D Gestures for Video Ggames", Extended Abstracts on Human Factors in Computing Systems. 2006, pp.1217-1222.
[15] Simon Günter, Horst Bunke, "Optimizing the Number of States, Training Iterations and Gaussians in an HMM-based Handwritten Word Recognizer", Proceedings of the Seventh International Conference on Document Analysis and Recognition, Vol. 1, pp.472-496.
[16] Nilu Singh, R.A Khan, and Raj Shree, "MFCC and Prosodic Feature Extraction Techniques: A Comparative Study", International Journal of Computer Applications, 54(1), 2012, pp.9-13.

박 도 생 (Park, Dosaeng)

2013- 한국외국대학교 컴퓨터공학전공 석박사 통합과정

관심분야 : 기능성게임, 멀티미디어시스템, 임베디드시스템

김 상 철 (Kim, Sangchul)

1994 미시간주립대학교 컴퓨터공학과 박사
1983-1994 ETRI 연구원
1994- 한국외국대학교 컴퓨터공학과 교수

관심분야 : 기능성게임, 게임 AI, 멀티미디어시스템