CONTROL DE UN ROBOT "ROBOSAPIEN" MEDIANTE VOZ

Vicent Tamarit Ballester, Carlos D. Martínez Hinarejos

Departamento de Sistemas Informáticos y Computación - DSIC
Universidad Politécnica de Valencia
Camino de Vera s/n, 46071, Valencia
{vtamarit,cmartine}@dsic.upv.es

RESUMEN

Uno de los campos más interesantes donde aplicar el reconocimiento del habla es, sin duda, aquellos entornos donde el reconocimiento se transforma en órdenes y además la cantidad de órdenes que acepta el sistema es limitada. Es el caso de las tareas que puede plantear la domótica, el manejo de pequeños aparatos eléctricos. En este artículo se presenta una tarea de esta clase: el manejo de un robot de juguete, con forma humanoide (un RoboSapien) a través de la voz. Se aborda, principalmente, la creación del modelo de lenguaje, que define las frases aceptadas por el sistema, así como la semántica de cada una de ellas. También se experimenta la combinación óptima de parámetros de reconocimiento.

1. INTRODUCCIÓN

Este artículo describe la construcción de un sistema de reconocimiento del habla que permita el control a través de computador de un RoboSapien. Para ello se ha diseñado un modelo de lenguaje acorde a la naturaleza del problema, teniendo en cuenta que las órdenes de voz van a ser órdenes que dirijan el comportamiento del robot.

El reconocedor debe ser capaz no sólo de aceptar las órdenes simples, sino de aceptar órdenes complejas que involucren varias órdenes simples, e incluso de posibilitar la definición de nuevas órdenes que derivarán de las 67 básicas.

1.1. El RoboSapien

Este robot humanoide, fabricado por la empresa *Wow Wee Toys* y puesto a la venta en 2004, es, en esencia, un juguete. Sin embargo, además de un divertido juguete, se convirtió al poco de salir al mercado en una buena herramienta de aprendizaje para quien se acerca por primera vez a la robótica o para quien busca una vía barata de experimentación.

El robot se maneja con un mando a distancia por infrarrojos desde el cual se pueden mandar hasta 67 órdenes, que posibilitan un control casi total de los movimientos de los brazos, así como giros, desplazamientos o la programación de sus sensores de contacto y de su sensor de sonido.

Trabajo parcialmente financiado por el proyecto TIC2003-08681-C02-02.

1.2. Manejo desde el ordenador

Para poder interactuar con el robot desde un computador se ha utilizado un circuito de construcción propia basado en el diseño propuesto en [2]. Este circuito permite la emisión de señales infrarrojas, así como su recepción, y es compatible con el software de infrarrojos LIRC (Linux Infrarred Remote Control) [6], que permite configurar dispositivos emisores/receptores bajo los sistemas GNU/Linux.

La configuración de LIRC se basa en la definición del mando a distancia, describiendo sus características técnicas y las diferentes órdenes que pueden enviarse, asociando a cada una de ellas un código que las identifique y un nombre.

2. RECONOCIMIENTO DEL HABLA PARA EL CONTROL DEL "ROBOSAPIEN"

El software de reconocimiento utilizado, sobre el que se construye la parte de reconocimiento y comprensión del sistema es ATROS [4], un acrónimo de "Automatically Trainable Recognizer Of Speech". Para adaptar este software a nuevas tareas hace falta definir tres modelos: el modelo de lenguaje, el léxico y el acústico. El modelo léxico describe la pronunciación de cada palabra como una secuencia de símbolos que representan sonidos, definidos mediante los modelos acústicos. La parte más importante es la construcción del primer modelo, pues es propio de cada tarea y su construcción no es trivial. Para la tarea que nos ocupa se han realizado dos versiones: una muy simple y otra con agrupamiento de estados. Ambas se modelan como transductores de estados finitos (TEF).

2.1. El fichero de órdenes

A la hora de construir los modelos se ha buscado sobre todo flexibilidad. Dada la naturaleza de la tarea, es posible que las frases que activan cada una de las órdenes

```
anda #WALK_ hacia adelante. #FORWARD_
anda #WALK_ hacia atras. #BACKWARD_
corre #WALK_ hacia adelante. #FORWARD_
corre #WALK_ hacia atras. #BACKWARD_
sube #UP_ el brazo #ARM_ derecho. #RIGHT_
sube #UP_ el brazo #ARM_ izquierdo. #LEFT_
```

Figura 1. Extracto del fichero de órdenes a partir del cuál se obtiene el modelo de lenguaje. Cada línea equivale a una orden.

pueda cambiar, o incluso sean diferentes para cada usuario. Pensando en esto se definió el fichero de órdenes, a partir del cual se genera toda la información necesaria para el reconocedor, y se implementaron las herramientas para extraer esa información automáticamente. En la Figura 1 se presenta un extracto del fichero de órdenes.

Cada palabra lleva asociado un significado marcado por "#". De esta forma, al reconocer la frase se emiten esos significados, con los que posteriormente puede construirse la orden final que enviar al robot.

2.2. Órdenes complejas

Se podría hacer una primera distinción de las órdenes, en función de su duración, separándolas entre finitas y continuas. Las primeras son aquellas que producen un resultado en tiempo finito, como saludar o los movimientos de los brazos. Las segundas, en cambio, se siguen ejecutando hasta que no se le da una nueva orden; la orden de correr sería una de este segundo tipo.

Sin embargo, a efectos prácticos se distinguen otros dos tipos de órdenes: simples y complejas. Una orden es simple si está formada por una o varias órdenes del mando y es compleja si la orden no está incluida en la configuración del mando (es decir, está definida por el usuario).

Una orden simple puede ser cualquier orden equivalente a cada uno de los botones del mando original del RoboSapien o a una combinación de éstas. En cambio, las órdenes complejas están orientadas a representar un comportamiento elaborado; es la forma de agrupar varias órdenes que conllevan la consecución de un movimiento concreto. Un ejemplo podría ser la definición de bailes.

Las órdenes complejas han de ser definidas por el usuario. Una posible orden compleja que agrupa los movimientos necesarios para bailar una jota sería:

```
baila una jota. #fJOTA
```

El significado "#fJOTA" indica que se trata de una orden no definida en el control remoto, ya que empieza por "f" minúscula, y está compuesta por las órdenes del fichero de texto "fJOTA". En la Figura 2 aparece el contenido de este fichero de texto. Como se puede ver, es un sencillo fichero de texto en el que pueden especificarse órdenes simples, así como repeticiones de bloques de órdenes que pueden anidarse sin límite. Cada orden debe llevar un retardo asociado. En el caso de que sea una orden finita este

```
UP_ARM_RIGHT 3
UP_ARM_RIGHT 3
UP_ARM_LEFT 3
UP_ARM_LEFT 3
repeat 5
STEP_FORWARD 3
STEP_BACKWARD 3
STEP_LEFT 3
STEP_RIGHT 3
end
```

Figura 2. Contenido del fichero fJOTA

retardo indica el tiempo que, se espera, va a costar de ejecutar la orden; en el caso de que sea una orden continua indica el tiempo que va a estar ejecutándose.

2.3. Inclusión de números en las órdenes

Algo que se ha mantenido igual en ambos modelos ha sido la inclusión de números en el vocabulario. Conforme se ha planteado la construcción del modelo, basado todo en un fichero donde cada línea es una orden, si quisiéramos incluir números deberíamos repetir la frase para cada uno de éstos. Por ejemplo:

```
Da un #1 paso #STEP_ hacia adelante. #FORWARD_
Da dos #2 pasos #STEP_ hacia adelante. #FORWARD_
Da tres #3 pasos #STEP_ hacia adelante. #FORWARD_
. . .
```

De hecho, la generación del modelo final se hace a partir de un conjunto de frases como ésas. Pero para simplificar el manejo al usuario, la herramienta acepta frases más compactas, utilizando una pseudovariable "$N". Antes de generarse el autómata final, un preproceso inicial despliega estas frases.

```
Da $N pasos #STEP_ hacia adelante. #FORWARD_
```

Se podría haber incluido algo similar para otras partes del lenguaje, como las direcciones, pero mientras que los números pueden crecer con facilidad, las direcciones se mantienen siempre en cuatro (adelante, atrás, izquierda, derecha) y no todas las órdenes que las incluyen necesitan esas cuatro. Por ejemplo no tiene sentido que las órdenes de giro incluyan las direcciones "delante", "atrás".

2.4. Modelo simple

En este modelo se construye un TEF para cada orden, independientemente de que comparta palabras con otras. Cada palabra de la frase generará una transición. A partir de todas las frases se construye el modelo final generando un nodo inicial y otro final que engloba al resto. En la Figura 3 se puede ver el TEF resultante para las dos primeras órdenes del fichero mostrado en la Figura 1.

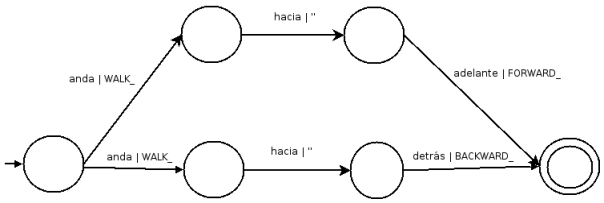

Figura 3. TEF de dos órdenes muy similares generadas siguiendo el planteamiento del modelo simple.

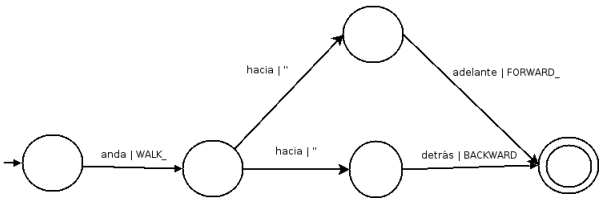

Figura 4. TEF de dos órdenes muy similares generadas siguiendo el planteamiento del modelo con agrupamiento.

Esta construcción es sencilla, pero puede llegar a ser bastante ineficiente, ya que el número de nodos y transiciones crece rápidamente con cada orden. Si suponemos una orden de n palabras, sabemos seguro que eso supondrán n nodos nuevos y $n+1$ transiciones, contando las internas entre las palabras de la orden más la transición del estado inicial al primer nodo de la frase.

2.5. Modelo con agrupamiento

El modelo anterior, por su sencillez, genera modelos con información redundante. Por ello se ha planteado otra forma de construir el modelo de lenguaje que busca obtener TEFs más compactos.

La construcción de este modelo parte también del fichero de órdenes comentado anteriormente y se divide en dos fases: un preproceso sobre las palabras y la construcción del TEF. Primero se realiza el preproceso que lee el fichero y almacena en memoria todas las palabras con su significado, si lo tienen, e información sobre si es estado inicial o final. Tras esto, se recorren todas las palabras y se van marcando como visitadas. Cada palabra nueva se compara con las ya visitadas y si coinciden sus significados, independientemente de que ellas no coincidan, se fusionan; de igual forma, aquellas palabras que no tienen significado pero son iguales también se fusionan. Con esto se pretenden agrupar sinónimos y palabras frecuentes como preposiciones.

Con toda esta información almacenada en memoria se construye el TEF definitivo que cuenta, además, con un estado inicial desde el que se transita con cada una de las palabras marcadas como iniciales, y de un estado final al que van todas las palabras marcadas como finales. El resto de palabras se transforman en transiciones internas del TEF. El resultado final se ilustra en la Figura 4. Aquí se puede apreciar como aquellas transiciones que eran sinónimos se han agrupado (ahora sólo hay un ”anda”) salvo en el caso de que ésto no fuera posible (los ”hacia” no se pueden fusionar porque preceden a una bifurcación necesaria para distinguir ”adelante” y ”detrás”).

A pesar de que existen otros algoritmos de aprendizaje para transductores mucho más potentes, como OMEGA [8] y OSTIA [7], no resultan convenientes para esta tarea debido a que suelen necesitar una muestra muy grande del lenguaje y para este caso se dispone, tan solo, de unas pocas frases.

Tabla 1. Comparación del tamaño de los dos modelos generados

MODELO	NODOS	ARISTAS
Simple	577	731
Agrupamiento	73	192

2.6. Comparación de los modelos generados

Estas dos maneras de crear el modelo de lenguaje producen salidas muy diferentes, con marcadas diferencias en el número de nodos y aristas del modelo resultante, como se puede ver en la Tabla 1. Ambos métodos de generación parten de un fichero con 56 órdenes. En total, el vocabulario consta de 207 palabras sólo para especificar las órdenes, más otras doce para los números, incluidos del uno al diez contando tres variantes para el uno: ”un”, ”uno” y ”una”. Este recuento no contempla las palabras de los significados, expresados con el símbolo ”#” en el fichero de órdenes.

3. MODELO LÉXICO

El modelo léxico consiste en un fichero que incluye todas las palabras del lenguaje reconocido e indican la secuencia de modelos acústicos que la forman. Generalmente se representa como un autómata lineal. Por ejemplo, la palabra ”gira” generaría el modelo de estados finitos de la Figura 5.

```
Name "gira"
State 0 i=1
0 1 "x" p=1
1 2 "i" p=1
2 3 "r" p=1
3 4 "a" p=1
State 4 f=1
```

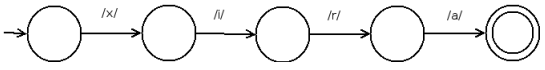

Figura 5. Autómata de estados finitos para la palabra ”gira”. Arriba, su representación en el fichero de configuración.

El modelo léxico se construye automáticamente desde el fichero de órdenes, incorporando los números que

allí no estaban definidios.

4. MODELOS ACÚSTICOS

Para los modelos acústicos se ha recurrido a un conjunto de modelos ya existente, entrenados a partir del corpus Albayzin [3].

Los modelos se representan como Modelos Ocultos de Markov (Hidden Markov Models, HMM) cuya topología es de tres estados, de izquierda a derecha, con bucles y sin saltos. Se utiliza una mixtura de 128 gaussianas (con matriz de covarianza diagonal) de 33 componentes y cada modelo es monofonema. Se utilizó HTK [5] para entrenar los modelos.

5. EXPERIMENTOS

Para poder ajustar correctamente los parámetros del reconocedor y poder evaluar el modelo de lenguaje se ha tomado un corpus de frases con el que se pudo preparar varios experimentos descritos más delante.

En la creación del corpus participaron diez locutores, cinco hombres y cinco mujeres, agrupados en parejas. Cada pareja pronunció diez frases, lo que resulta en un corpus de 50 frases pronunciadas por hombres y mujeres (en total 100 frases). El corpus está formado por 868 palabras y supone un total de cinco minutos y medio de grabación.

Las grabaciones se han realizado a 16KHz, la misma frecuencia a la que está grabado el material de entrenamiento del modelo acústico. No se buscaron en ningún momento condiciones óptimas del entorno para la grabación, a fin de poder disponer de un corpus cercano a las locuciones de un sistema en producción.

La adquisición se realizó utilizando las órdenes que para tal efecto tiene implementadas ATROS, por lo que el programa utilizado para la adquisición es el mismo con el que luego se reconoce.

Los batería de experimentos se llevó a cabo utilizando la versión off-line de ATROS. Los parámetros a variar son los comentados a continuación:

- GrammarBeam: Cada transición del modelo de lenguaje se despliega en un HMM sobre el que se aplica el algoritmo de Viterbi [1]. Si la transición recibe una puntuación mayor que la mejor puntuación más este factor se poda, dejan de explorarse los caminos que desde ella se pudieran generar. Con este factor se controla el número de caminos que hay que explorar. Para valores bajos muchos caminos se podarán, lo que previsiblemente repercutirá en un mayor error de reconocimiento pero en un menor tiempo de cálculo, dado que cada iteración requiere actualizar menos caminos.

- PhonemeBeam: Igual que el parámetro anterior pero a nivel del modelo acústico.

- GrammarScaleFactor: Este parámetro modifica el peso de las probabilidades del modelo de lenguaje frente al modelo acústico. Es útil para equilibrar la influencia del modelo acústico y la gramática. Valores altos dan más importancia a la gramática, mientras que valores bajos hacen que el modelo acústico guíe el reconocimiento.

- WordInsertionPenalty: Penaliza la inserción de nuevas palabras. Cuanto más alto es este factor más se premia a las frases cortas. Este parámetro es necesario porque frase largas, o incluso palabras, pueden decodificarse como palabras más cortas, rompiendo el significado original. Por ejemplo palabras compuestas como ”automóvil”, pueden ser reconocidas como ”auto” y ”móvil”. Con este parámetro se controla este efecto en el reconocimiento.

Se preparon dos baterías de experimentos, la primera para comparar los dos modelos de lenguaje y observar cuál se comporta mejor. La segunda buscaba afinar la primera búsqueda para encontrar los parámetros óptimos.

Los errores que se han tenido en cuenta son los relativos a la semántica de las órdenes, no a la orden en sí; es decir, no se ha tenido en cuenta si ATROS ha reconocido bien la orden, sino si ha reconocido bien el significado de la orden dada. Sobre el significado de las órdenes se han medido, a su vez, dos errores diferentes: el error a nivel de palabra (Word Error Rate), y el error a nivel de frase (Sentence Error Rate). Cada frase reconocida puede contener una o varias órdenes; así, en este caso, el WER se refiere al error de las órdenes a nivel individual, mientras que el SER se refiere al error de todas las órdenes reconocidas en una frase. Para el corpus adquirido la perplejidad es de $2{,}975$ sobre el modelo simple y de $3{,}004$ para el modelo con agrupamiento.

5.1. Comparación de modelos

La primera fue una búsqueda exhaustiva, haciendo un barrido por los cuatro parámetros: GrammarBeam, PhonemeBeam, GrammarScaleFactor (GSF) y WordInsertionPenalty (WIP). Para los dos primeros se han usado los mismos valores, por lo que, en la práctica, la búsqueda ha sido sobre tres parámetros, que en adelante se referirán como: BEAM, GSF y WIP.

En la Tabla 2 se muestran los resultados más destacados para el modelo de lenguaje simple. Se puede ver cómo los resultados mejoran al aumentar el BEAM, pero llegado un punto no merece la pena seguir aumentándolo. El BEAM establece el rango de búsqueda en el modelo de Markov final. Dado que el modelo de lenguaje no es demasiado complejo, al aumentar el parámetro llega un momento que el rango de búsqueda que se establece no elimina caminos óptimos y por ello aumentar este valor no mejora los resultados.

Los resultados para el modelo con agrupamiento están en la Tabla 3. Ocurre lo mismo con este modelo que con

Tabla 2. Tabla con los resultados para el modelo de lenguaje simple. En negrita el mejor resultado.

Error para la semántica (WER/SER)					
			BEAM		
		WIP	200	400	600
G S F	1	0	9.35/24	7.57/21	7.57/21
		1	9.79/24	8.01/21	8.01/21
		5	9.35/21	8.31/19	8.31/19
G S F	5	0	7.72/20	**6.38/17**	6.38/17
		1	7.72/20	6.97/18	6.97/18
		5	8.16/21	7.12/18	7.12/18
G S F	10	0	7.86/21	6.38/18	6.38/18
		1	7.86/21	6.38/18	6.38/18
		5	9.20/21	7.57/18	7.57/18

Tabla 3. Tabla con los resultados para el modelo de lenguaje con agrupamiento. En negrita el mejor resultado.

Error para la semántica (WER/SER)					
			BEAM		
		WIP	200	400	600
G S F	1	0	11.72/28	9.20/26	9.20/26
		1	10.83/27	8.61/25	8.61/25
		5	12.17/31	9.64/28	9.64/28
G S F	5	0	10.68/26	**8.31/24**	8.31/24
		1	10.83/27	8.46/25	8.46/25
		5	13.50/33	10.39/29	10.39/29
G S F	10	0	11.72/30	9.05/27	9.05/27
		1	12.17/31	9.50/28	9.50/28
		5	15.88/37	13.06/33	13.06/33

el anterior: llegado un punto, aumentar el BEAM, no mejora el reconocimiento. Sin embargo, a pesar de mantener el mismo comportamiento, los resultados generales son peores que con el modelo anterior. El culpable de este empeoramiento es la generalización que supone la fusión de estados. Ahora frases como "gira hacia la derecha" y "anda hacia adelante" se podrían reconocer como "gira hacia adelante", debido a que el agrupamiento ha unido todos los "hacia".

El modelo simple presenta, como se ha visto, mejores resultados. Para esta tarea concreta, con pocas órdenes, este modelo sería la elección más lógica. Para modelos mucho más grandes, donde el HMM de reconocimiento puede crecer en exceso generando peores tiempos de reconocimiento, podría plantearse el uso del modelo con agrupamientos, buscando un compromiso entre el tiempo de reconocimiento y el error. Pero para esta tarea la diferencia temporal entre ambos modelos, aunque notable, no es significativa para el reconocimiento. Como se puede ver en la Tabla 4, todos los tiempos, salvo para el BEAM de 1000, están por debajo de los 10 milisegundos. ATROS extrae características de fragmentos de señal de 10 milisegundos; por lo tanto puede considerarse como una tarea en tiempo real para cualquiera de los dos modelos, pues en ningún caso se supera ese tiempo.

Tabla 4. Tiempos de reconocimientos para diferentes BEAM. Se han tomado con GSF de 5 y WIP de 1.

Tiempos para el reconocimiento (segundos/frame)		
BEAM	Simple	Agrupamiento
100	0.002	0.002
200	0.005	0.004
400	0.008	0.006
600	0.009	0.007
1000	0.010	0.007

Tabla 5. Tabla con los resultados de los experimentos para la búsqueda de parámetros óptimos para el modelo normal, con un BEAM de 400. En negrita el mejor resultado.

Error para la semántica (WER/SER)			
	GSF		
WIP	2	5	7
0	7.12/20	**6.38/17**	6.68/18
1	6.53/18	6.97/18	6.68/18
2	6.97/18	7.42/18	7.12/18
3	7.12/18	7.42/18	7.12/18
4	7.86/18	7.42/18	6.82/18
5	7.86/18	7.12/18	6.82/18

5.2. Refinamiento de los parámetros

En las pruebas generales lanzadas se ve como el comportamiento no mejora al aumentar el BEAM, pero sí que hay pequeñas variaciones al modificar el resto de parámetros. Por ello se preparó un segundo bloque de experimentos donde el BEAM se fijó a 400 y se varió el GSF entre 0 y 10 y el WIP entre 0 y 5 en incrementos unitarios. Dado que la diferencia de error es tan sólo del 2 %, los experimentos se repitieron para ambos modelos.

En la Tabla 5 se muestran los resultados más significativos para el modelo simple. La columna central muestra los resultados del mejor GSF, las otras dos, una con un GSF superior y la otra inferior, confirman la tendencia de los errores a minimizarse en el valor óptimo. Para el modelo simple no se aprecia ningún tipo de mejora y no es previsible que una búsqueda aún más minuciosa vaya a proporcionar mejores resultados, por lo que la mejor configuración sigue siendo un BEAM de 400, un GSF de 5 y un WIP igual a 0.

En la Tabla 6 se muestran los resultados para el modelo con agrupamiento. Igual que en el caso anterior se han incluido los valores para tres GSF. Para este modelo, los nuevos experimentos sí que muestran una leve mejora del error.

5.3. Análisis de errores

Partiendo del mejor resultado para el modelo simple obtenido en el apartado 5.1, se ha realizado un estudio para descubrir cuáles son los errores más frecuentes y buscar una posible solución. El análisis se ha realizado sobre la semántica de las órdenes y a nivel de palabra.

Tabla 6. Tabla con los resultados de los experimentos para la búsqueda de parámetros óptimos para el modelo con agrupamiento, con un BEAM de 400. En negrita el mejor resultado.

Error para la semántica (WER/SER)			
	GSF		
WIP	5	7	9
0	8.31/24	**8.01/24**	8.61/26
1	8.46/25	8.16/25	9.20/28
2	9.05/28	9.50/29	10.39/30
3	9.35/28	10.09/29	11.13/30
4	10.09/29	10.39/29	11.72/30
5	10.39/29	10.98/29	12.02/31

Para poder ver los errores de reconocimiento se analizaron los significados de las 100 frases del experimento, una por una, comparándolas con el significado correcto. De esta forma se apreciaron tres problemas en el reconocimiento. El primero de ellos ya se veía con la matriz de confusión: algunos números como el 3 y el 10 o el 2 y el 6, se confunden entre ellos.

Los otros dos problemas son propios de la tarea y del modelo creado. Uno de ellos es la confusión entre palabras con fonética similar. Palabras como ”apágate”, ”ataca” y ”anda” se confunden con facilidad, igual que otras como ”levántate” y ”levanta”, dando lugar a varios errores. El tercer problema es también culpa de la definición del modelo de lenguaje: al crear el conjunto de órdenes se ha asociado varios significados a la misma palabra, por ejemplo ”anda” se utiliza tanto para mandar que dé un paso (STEP_) como para ordenar que ande (WALK_); la diferencia entre ambas órdenes radica en que la segunda orden es lo que en el apartado 2.2 se ha llamado orden continua, que se sigue ejecutando indefinidamente.

Resolver la confusión de los números es complicada, pero los errores causados por los otros dos problema podrían minimizarse actuando sobre el modelo de lenguaje: primero, buscando otras palabras para substituir a las que se confunden y segundo, evitando darle varios significados a la misma palabra.

6. CONCLUSIONES Y TRABAJO FUTURO

Se ha presentado en este artículo la solución a un problema real de reconocimiento del habla, como es el manejo de un robot. En este caso no pasa de ser un pequeño juguete, pero el fundamento puede ser el mismo para robots, no necesariamente humanoides, más complejos, así como también electrodomésticos y cualquier aparato electrónico con un número limitado de órdenes. Nuestra aproximación permite además generar a partir de esas órdenes simples otras más complejas, dotando al sistema, en este caso al RoboSapien, de más versatilidad.

Las posibles ampliaciones recaen sobre todo en el hardware del robot. Se podría mejorar la comunicación con él utilizando tecnologías como Bluetooth o incorporarle visión artificial. A nivel software se puede mejorar el sistema de órdenes complejas, permitiendo crear macros de comportamiento más elaboradas. También se puede revisar la construcción del modelo de lenguaje: partiendo de un gran fichero de órdenes y utilizando alguna modificación de los algoritmos de inferencia gramatical que existen se podrían obtener mejores modelos, con probabilidades más ajustadas. Otra opción más sencilla es construir un nuevo modelo teniendo en cuenta las consideraciones del análisis de errores.

7. BIBLIOGRAFíA

[1] G. Forney. The Viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278, March 1973.

[2] Iguanaworks. Emisor/receptor para lirc. `http://iguanaworks.net/ir/`.

[3] J.Diaz, A. Rubio, A. Peinado, E.Segarra, N.Prieto, and F.Casacuberta. Development of task-oriented spanish speech corpora. In *Proceedings of the First International Conference on Language Resources and Evaluation*, pages 497–501, Granada, May 1998. ESCA.

[4] D. Llorens, F. Casacuberta, E. Segarra, J. Sánchez, P. Aibar, and M. Castro. Acoustical and syntactical modeling in the atros system. In *International Conference on Acoustic, Speech and Signal Processing, volume 2*, pages 641–644. IEEE press, March 1999.

[5] HTK Team. HTK speech recognition toolkit, 2004. http://htk.eng.cam.ac.uk/.

[6] LIRC Team. Linux infrarred remote control - lirc. `http://www.lirc.org`.

[7] J. Oncina; P. García; E. Vidal. Learning subsequential transducers for pattern recognition interpretation tasks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):448–458, 1993.

[8] J. M. Vilar. Improve the learning of sub-sequential transducers by using alignments and dictionaries. In *Proceedings of the 5th International Colloquium on Grammatical Inference: Algorithms and Applications*, volume 1891 of *Lecture Notes in Computer Science*, pages 298–311. Springer-Verlag, 2000.

[9] M.A. Jack X.D. Huang, Y. Ariki. *Hidden Markov Models for speech recognition*. Edinburgh University Press, 1990.