Autolocalización inicial para robots móviles usando el método de K-NN

J. I. Alonso
Grupo SIMD
UCLM/i^{3}A
Dpto. Sist. Informáticos
Albacete
juanignacio.alonso@uclm.es

Jose A. Gámez
Grupo SIMD
UCLM/i^{3}A
Dpto. Sist. Informáticos
Albacete
jgamez@dsi.uclm.es

I. García-Varea
Grupo SIMD
UCLM/i^{3}A
Dpto. Sist. Informáticos
Albacete
ivarea@dsi.uclm.es

J. Martínez
Grupo SIMD
UCLM/i^{3}A
Dpto. Sist. Informáticos
Albacete
jesus_martinez@dsi.uclm.es

Resumen

A lo largo del artículo se presentará una propuesta para resolver un problema de especial interés en el mundo de la robótica móvil, como es el de la autolocalización. Para ello se hará uso del aprendizaje basado en casos, concretamente del algoritmo K vecinos más cercanos (K-NN), usando ponderación de diferencias. Mostraremos como el uso de un algoritmo genético nos puede permitir optimizar la selección de pesos, mejorando los resultados de nuestra propuesta. Por último, veremos la forma de estimar la posición ocupada a partir de los resultados obtenidos con nuestro algoritmo.

1. Introducción

El conocimiento que un robot móvil tenga de la posición que ocupa dentro de su entorno es uno de los principales problemas que se han estudiado en los últimos tiempos dentro del mundo de la robótica móvil[6][13].

Cualquier decisión que desee tomarse por parte de un robot móvil, debe tener en cuenta la posición que él mismo ocupe dentro de su entorno, en el momento de la decisión, por lo que realizar una correcta autolocalización es un punto clave dentro de cualquier desarrollo, y sobre todo en competiciones como la *Robocup*[1] .

[1] www.robocup.org

Para resolver este problema, la mayoría de propuestas realizadas hasta el momento, hacen uso de procesos de Markov[7], principalmente del método de Monte Carlo[11], así como de filtros de Kalman[9].

Estas técnicas obtienen la posición actual en función de las observaciones obtenidas hasta el momento y de las acciones realizadas, teniendo en cuenta las probabilidades de pasar de una posición a otra en función de los movimientos realizados. Estos métodos han demostrado obtener buenos resultados, tanto en la competición de fútbol de robots[8], como en aplicaciones de entornos interiores[5]. El principal problema encontrado reside en que la localización no se ejecuta de forma correcta hasta que no se han realizado un cierto número de movimientos[4], que quizá nos hayan alejado del objetivo perseguido por el comportamiento. Situaciones en las que la localización deba comenzar de nuevo, las podemos encontrar en numerosas ocasiones dentro de los partidos de la *Robocup*, como consecuencia de una penalización[2] o por diversas razones.

Para resolver este punto, a lo largo del artículo, se muestra una propuesta basada en realizar un mayor esfuerzo para obtener las primeras posiciones ocupadas dentro del entorno, con el objetivo de evitar realizar una serie de movimientos incorrectos antes de conocer la posición exacta ocupada.

Para ello se usa un método de aprendizaje basado en casos clásico, el método K-

NN[12][3], que tras una serie de variaciones que permitan mejorar sus prestaciones, incluyendo una ponderación de las distancias, y tras usar un algoritmo genético para seleccionar los pesos adecuados, nos permitirá obtener la posición esperada en función de las observaciones realizadas.

Este documento se organiza como sigue. La Sección 2 muestra el problema concreto que deseamos resolver y la forma de abordarlo. En la Sección 3 presentamos el algoritmo usado para la resolución del problema. La Sección 4 consta de las acciones realizadas mediante un algoritmo genético para la selección adecuada de los pesos. La Sección 5 muestra los resultados obtenidos y por último la Sección 6 incluye las conclusiones extraídas y el trabajo futuro.

2. Escenario

Los métodos de localización pueden dividirse según ciertos autores [1], en métodos de posicionamiento relativo, a partir de una posición inicial, o en métodos de posicionamiento absoluto.

Dentro de estos últimos se debe poseer un amplio conocimiento del entorno en el que nos encontramos, para lo que es necesario hacer una lectura de elementos de referencia, los cuales pueden ser elementos de localización activa (como el sistema GPS), marcas naturales dentro del entorno, marcas artificiales añadidas o mediante técnicas que realicen una correspondencia entre los valores obtenidos y el modelo almacenado del entorno.

Dado que se desea proponer una solución válida para el entorno *Robocup*, en particular para la categoría *four legged*, desarrollaremos una técnica de localización con posicionamiento absoluto, usando marcas adicionales, de forma que el conocimiento adquirido a través de la propuesta pueda ser extrapolado al entorno real *Robocup*.

El entorno en el que nos queremos posicionar consta de una superficie cuadrada de 1,50 x 2,00 metros de suelo color verde césped. En una de las zonas del entorno se han añadido 3 marcas artificiales, situadas todas a 50 cm del borde norte de la zona. Se trata de dos cilindros de 10 cm de altura y 10 cm de diámetro, situados a 30 cm de cada esquina, de colores amarillo y azul (similares a las balizas de la *Robocup*), y un tercer cilindro de 20 cm de altura y 10 de diámetro, de color blanco situado en el centro de la linea.

Almacenaremos la información obtenida por la cámara del robot relativa a 4 elementos: las 3 ayudas externas (balizas blanca, azul y amarilla) y el césped del campo.

Para conocer la zona donde el robot se encuentra situado, dividimos la superficie en 12 sectores de 50 x 50 cm, de forma que nuestro robot estará colocado en cada momento en una de las 12 casillas o sectores definidos, como vemos en la figura 1.

El entorno final, consta de una superficie de 2 x 2 metros, con los elementos de localización situados en la zona norte.

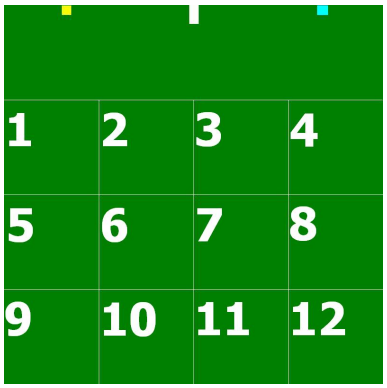

Figura 1: Escenario base

3. Resolución del problema

La solución aquí propuesta para la resolución del problema usa como base principal el algoritmo de los k-vecinos más cercanos, el cual es uno de los ejemplos de aprendizaje basado en casos o ejemplos más conocido.

El algoritmo consta de dos partes: entrenamiento y clasificación.
Para el entrenamiento únicamente almacenaremos instancias $\mathbf{x}_i$ representadas de la siguiente manera

$$< x_{i1}, x_{i2}, \ldots, x_{in}, C_i >$$

donde x_{ij} se corresponde con el atributo j de la instancia i y C_i contiene el valor de la clase para ésta instancia, que representa el cuadrante ocupado, según muestra la figura 1.

Para realizar la clasificación, seleccionaremos la clase de la instancia o instancias que mayor semejanza muestren con el caso a clasificar, de forma que a un caso $\mathbf{x}_p$ se le asignará la clase de la instancia $\mathbf{x}_i$ cuando $d(\mathbf{x}_p, \mathbf{x}_i)$ sea mínima.

Para calcular la distancia entre dos instancias, usaremos la ecuación 1, que hace uso de la distancia Euclídea estándar, y de una ponderación de la importancia de los distintos atributos, identificada por los pesos w_i

$$d(x_1, x_2) \equiv \sqrt{\sum_{i=0}^{n} w_i (x_{1i} - x_{2i})^2} \qquad (1)$$

Una vez se han obtenido los k-vecinos más cercanos, se puede ponderar la aportación de cada vecino en función de la distancia respecto al caso a clasificar. En nuestro caso, ponderaremos cada aportación multiplicándola por $(1 - d)$, donde d se corresponde con la distancia entre instancias.

El principal inconveniente de este clasificador es su coste computacional, dado que todo el coste se realiza en el momento de la clasificación. Dentro de las ventajas que muestra este algoritmo encontramos su simplicidad, su capacidad para utilizar representaciones complejas y su buen comportamiento ante la aparición de ruido.

3.1. Generación del conjunto de entrenamiento

Debemos decidir el tipo de información que se desea almacenar para cada muestra. En nuestro caso, almacenaremos la información del entorno captada en cada momento por nuestro robot, un Sony AIBO ERS-7[2], que posee una cámara frontal desde la cual tomaremos las medidas. Además almacenaremos el cuadrante en el cual se encuentra el robot en el momento de la medición.

Cada muestra almacenará la siguiente información:

```
Por color amarillo, azul, blanco y césped:
   Número de píxeles
   Pos. X del centroide de la distribución
   Pos. Y del centroide de la distribución
Número del cuadrante ocupado (Clase)
```

Las instancias serán tuplas de 13 elementos, y las posiciones X e Y de los centroides de cada distribución de color serán calculadas de la siguiente manera:

```
centroX=0;centroY=0;num_pixeles=0
para x=0 hasta x = ancho_imagen
   para y=0 hasta y = alto_imagen
       si pixel(x,y) ∈ color
           centroX+= x
           centroY+= y
           num_pixeles++
centroX=centroX / num_pixeles
centroY=centroY / num_pixeles
```

Algoritmo 1: Cálculo del centroide

Cuando el número de píxeles observados de un determinado color sea 0, como posición X e Y del centroide almacenaremos un valor que nos permita controlar esta situación durante el diseño del algoritmo.

3.2. Diseño del clasificador

Las características del algoritmo son las siguientes:

- Las diferencia entre cada par de atributos (x_{ai}, x_{bi}), varía entre 0 y un valor máximo común a todos estos valores (100).
- Cada una de las diferencias entre atributos está ponderada con un peso.
- Se usa la distancia Euclídea para acentuar diferencias.

[2] http://es.wikipedia.org/wiki/Aibo

- Para la diferencia entre posiciones X e Y, cuando una instancia almacene una posición válida y la otra en cambio no contenga esa información, debido a la ausencia de píxeles del elemento oservado, la diferencia entre las dos instancias será máxima.
- Una vez obtenidos los k vecinos más cercanos, se ponderará su cercanía usando la función diferencia $(1-d)$ para seleccionar la clase adecuada.

El resultado del algoritmo consiste en un número n de clases (n≤k), que puede usarse para aproximar de forma más exacta la posición ocupada por el robot en el entorno. Para conseguir esto, cada uno de los k vecinos realizará una aportación proporcional a su distancia al caso a clasificar, como veremos de forma detallada en la sección 5.1.

4. Optimización del algoritmo mediante el uso de un algoritmo genético

Una vez diseñado nuestro algoritmo k-vecinos más cercanos, haremos uso de un algoritmo genético[10] que seleccione los pesos w_i adecuados para realizar la mejor clasificación posible. Para su diseño seguimos el esquema típico de este tipo de algoritmos:

```
Inicializar la población()
Mientras condición no satisfecha
   Evaluar individuos()
   Seleccionar individuos como padres()
   Realizar cruce y mutación()
   Generar una nueva población()
```

Los individuos de nuestra población son cadenas de enteros, con valores discretos que van desde 0 hasta 12.

$$\mathbf{w}_i =< w_1, w_2, \ldots, w_{12} >$$

De entre todos los parámetros que podemos seleccionar a la hora de lanzar el algoritmo genético, usaremos los siguientes:

- Función de adecuación o *fitness*: porcentaje de acierto sobre el conjunto de test.
- Selección de padres elitista mediante probabilidades proporcionales al fitness.
- Tipos de cruce: por un punto.
- Probabilidad de mutación p=0.05.
- La nueva población se conforma con los mejores individuos de la población actual y los descendientes generados.

La justificación de usar un algoritmo genético viene dada por la complejidad del problema, ya que el espacio de búsqueda es elevado.

5. Resultados

El objetivo de esta sección no es realizar un estudio exhaustivo sobre el escenario propuesto, generando grandes cantidades de ejemplos y obteniendo un gran número de métricas, sino realizar un pequeño estudio en el que se muestre el resultado de aplicar el algoritmo sobre un escenario de pruebas, que permita estudiar la factibilidad del uso de este algoritmo en los momentos iniciales de la autolocalización.

Para exponer los resultados mostrados por el algoritmo, generamos un conjunto de muestras que separaremos de forma aleatoria en un conjunto de entrenamiento y otro de test, estratificados por clases. El conjunto de entrenamiento contiene 1143 muestras y el de test 168.

5.1. Selección uniforme de pesos

Comenzamos por una selección de pesos uniforme y estática, con valor uno para todos los elementos.Comprobaremos el resultado obtenido con valores pequeños de k.

K	1	2	3	4
% aciertos	76.19	75.59	76.78	74.40
K	5	8	12	20
% aciertos	74.40	73.21	70.83	67.26

Tabla 1: Resultados con pesos igual a 1

Como podemos comprobar, sobre el conjunto de test, la mejor tasa de acierto que podemos obtener es de un 76 %, y esta tasa se

obtiene con una valor de k igual a 1 y 3. Conforme el valor de k supera los valores bajos, el algoritmo comienza a perder precisión y el porcentaje de aciertos desciende hasta un 67.26 con un valor de k 20.

5.2. Selección de pesos usando WEKA

Antes de lanzar el algoritmo genético, intentaremos obtener toda la información posible de los atributos, usando para ello el software WEKA[14]. Haremos uso de las herramientas disponibles para selección de variables, en concreto evaluaremos los atributos usando *SymmetricalUncertAttributeEval*, tomando como método de búsqueda un *ranker*.

Este algoritmo evalúa la importancia de cada variable i por separado respecto a la variable clase, devolviendo una valor real entre 0 y 1, usando para ello la siguiente expresión,

$$S_i(C, v_i) = 2*(H(C)-H(C|v_i))/H(C)+H(v_i)$$

donde $H(\cdot)$ es la entropía de Shannon.

El resultado de su ejecución nos proporciona una lista de las variables ordenadas en función de su importancia respecto a la clase, acompañadas de la medida usada para realizar esta ordenación. Si usamos el valor proporcionado por el algoritmo a cada variable, para seleccionar una serie de pesos iniciales, transformando los valores reales del dominio [0,1] a enteros dentro del dominio [0,12], obtenemos la distribución de la tabla 2.

Como podemos apreciar, el número de píxels observado desde la cámara de los colores azul, blanco y amarillo es lo que mayor información nos aporta, seguido a continuación de la posición X del centroide del color césped. La clasificación que realiza nuestro algoritmo usando la anterior selección de pesos, sobre el mismo conjunto de pruebas y con los mismos valores de k, puede verse en la tabla 3.

Esta selección de pesos no ha incrementado el mayor porcentaje de aciertos obtenido con pesos uniformes (76.78), aunque mejora el porcentaje de aciertos para la mayoría de valores de k.

Variable	Posición	Peso
N Píxels azul	1	7
N Píxels blanco	2	6
N Píxels amarillo	3	5
X césped	4	5
X azul	5	5
N Píxels césped	6	4
Y amarillo	7	4
Y blanco	8	4
Y azul	9	4
Y cesped	10	4
X amarillo	11	3
X blanco	12	2

Tabla 2: Listado de variables ordenadas y sus pesos asociados

K	1	2	3	4
% aciertos	76.19	76.19	76.19	76.78
K	5	8	12	20
% aciertos	76.19	74.40	72.61	72.02

Tabla 3: Resultados con los pesos obtenidos con WEKA

5.3. Selección de pesos mediante el algoritmo genético

A pesar de buscar la mejor selección de pesos para este problema concreto, lo cual se podría realizar a través de otras técnicas, e incluso por fuerza bruta, también se busca obtener una algoritmo que, sin demasiado tiempo, y con cierto poder de generalización, permita realizar una selección acertada de pesos. Esto puede ser útil si partimos de una amplia base de casos, que nos confiere nuestro conocimiento del dominio, pero las condiciones en las que se realice la localización puedan variar ligeramente, situación en la que en lugar de generar un nuevo fichero de casos (con el tiempo que necesita), pueda ser suficiente con obtener una pequeña muestra de instancias bien clasificadas para ser usadas como conjunto de test dentro de nuestro algoritmo genético, y realizar un ajuste de pesos óptimo para esa situación.

La realización de nuestro algoritmo genético también nos puede aportar información adicional sobre el conjunto de casos almacenado,

permitiendo realizar una posterior selección de variables. Este planteamiento nos permite superar una de las desventajas del algoritmo K-NN clásico, en el cual todos los elementos son tenidos en cuenta para la clasificación.

En la tabla 4 mostraremos los porcentajes de acierto obtenidos, optimizando la selección de pesos para los mismos valores de k mostrados con anterioridad, y sobre el mismo conjunto de test. Se han realizado 1500 iteraciones sobre 20 individuos, y la tabla muestra el mejor porcentaje obtenido en su ejecución.

K	1	2	3	4
% aciertos	80.35	81.54	83.33	81.54
K	5	8	12	20
% aciertos	80.35	80.95	75.59	75.00

Tabla 4: Resultados con pesos optimizados

Como vemos, el mayor porcentaje de aciertos se corresponde con un 83.33 %, para un valor de k igual a 3.

Para las mejores clasificaciones realizadas, mostramos en la tabla 5 los pesos asignados a cada una de las variables y el valor de k con el que se obtuvo la clasificación.

Si observamos la selección de variables realizada con WEKA, podemos observar que no se corresponde con las selecciones de atributos realizadas por el algoritmo genético, ya que mientras WEKA ordenaba las variables de forma individual, el algoritmo genético tiene en cuenta el conjunto global de todas las variables.

5.4. Estimación de la posición ocupada

Una vez hemos visto los datos cuantitativos de acierto sobre el conjunto de test, vamos a realizar un esfuerzo posterior a la obtención de la casilla mas probable, para estimar de forma más exacta la posición ocupada en el entorno.

La motivación de esta sección proviene de las clasificaciones en las cuales dos o más vecinos poseen los mejores valores para representar a la clase, con unas diferencias mínimas. Desaprovechar la información que proporciona un vecino que obtenga un valor muy cercano al

Variable	w_i k=2	w_i k=3	w_i k=4
Pxls amarillo	9	5	7
X amarillo	3	3	4
Y amarillo	1	4	1
Pxls azul	8	8	5
X azul	0	7	5
Y azul	2	3	0
Pxls césped	9	3	2
X césped	7	5	10
Y césped	11	9	8
Pxls blanco	8	7	3
X blanco	8	7	10
Y blanco	3	5	2

Tabla 5: Listado de variables y sus pesos asociados para diferentes valores de k

vecino más próximo no parece la decisión más inteligente, ya que si para una determina lectura de los sensores de nuestro robot, tenemos dos posiciones posibles destacadas respecto al resto, lo más probable es que nuestro robot se encuentre en una posición intermedia.

Esta información puede usarse para realizar un estimador, que a partir de las posiciones obtenidas como más prometedoras, calcule la posición ocupada por el robot, tomando como información base las posiciones absolutas del centro de las casillas en la que hemos dividido nuestro entorno. El centro de una casilla i puede obtenerse mediante las siguientes fórmulas

$$\begin{aligned} X_i &= 25 + ((i-1)mod(4)) * 50 \\ Y_i &= 75 + ((i-1)div(4)) * 50 \end{aligned}$$

Cada vecino de los k seleccionados aporta para la posición X e Y final, la posición X e Y de su centro, multiplicada por el porcentaje de aportación como vecino, de forma que si tenemos n posiciones seleccionadas como vecinos ($n \leq k$), y cada posición aporta una proporción p_i con $\sum p_i = 1$, la posición X final del robot, quedará dada por la siguiente fórmula, (la posición Y se obtiene de forma análoga)

$$X_{final} \equiv \sum_{i=0}^{n} (p_i) X_i$$

Para comprobar esa aproximación, mostra-

mos en la imagen 2 una representación gráfica del resultado, que nos permite comprobar la calidad de la estimación realizada.

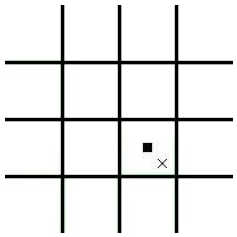

Figura 2: Posición estimada

Se muestra la posición del centro de la casilla predominante con un cuadrado, y la posición final con la intervención de todas las casillas seleccionadas dentro de los k vecinos más cercanos con una cruz.

A pesar de bajar el porcentaje de aciertos conforme el valor de k supera el valor de 3, seleccionar un alto valor para k nos puede permitir mejorar la estimación posterior de la posición ocupada.

6. Conclusiones y trabajo futuro

Podemos concluir que se ha conseguido el objetivo planteado, que consiste en mostrar la viabilidad de realizar una auto localización dentro de un escenario concreto, mediante un aprendizaje basado en casos, usando un algoritmo de vecinos más cercanos ponderado, gracias al cual se han obtenido unos resultados aceptables.

Los pesos seleccionados se obtienen gracias al uso de un algoritmo genético, que permite realizar una buena selección de los mismos.

Por último se ha mostrado cómo utilizar toda la información obtenida por el algoritmo, y no solamente la relativa al vecino más prometedor, para realizar una estimación más precisa de la posición ocupada por el robot dentro de su entorno.

El algoritmo ha demostrado obtener unos buenos resultados, a pesar de haber utilizado un conjunto de entrenamiento con un número de muestras reducido, en comparación con otros estudios. Se debe tener en cuenta que las condiciones en las que se ha realizado el estudio son óptimas, ya que el número de posiciones no es elevado, y las variaciones dentro de la orientación del robot son mínimas, casi siempre con algún tipo de intersección entre el campo de visión del robot y la zona donde los elementos de localización están situados.

Para desarrollos más complejos, como pueda ser la localización en un entorno *Robocup*, la cantidad de información a almacenar normalmente deberá ser superior a las 12 variables usadas en nuestro caso, y en lugar de tratar con los datos captados, puede ser necesario algún tipo de procesamiento previo. El número de casillas debe ser superior a las 12 mostradas en nuestro escenario, y también deben tenerse en cuenta distintas orientaciones, por lo que el volumen final de datos puede ser enorme, y dado que el algoritmo K-NN evalúa para cada nueva clasificación el conjunto completo de instancias almacenadas, el tiempo de cómputo realizado por un robot autónomo puede ser demasiado elevado para ejecutarse en tiempo real. En estas situaciones, una vez generado el fichero de casos, se deben utilizar técnicas de clustering, selección de instancias, selección de variables o de extracción de características para dejar en nuestro conjunto de entrenamiento un número elevado de las instancias más representativas, pero sin información redundante.

Debido al volumen de datos necesario para realizar una correcta clasificación, y a los buenos resultados mostrados algoritmos como el algoritmo de Monte Carlo, tras haber realizado una serie de movimientos, una de las principales conclusiones que podemos obtener es que el algoritmo aquí mostrado puede ser una buena solución para los primeros momentos de una localización, en los cuales se hayan realizado pocas acciones y por tanto, no se haya aprovechado la información de las posiciones anteriores y las probabilidades de haber migrado a una posición adyacente. Usando este algoritmo, puede requerirse una mayor cantidad de tiempo para realizar unas primeras localizaciones, aunque si éstas se realizan de forma correcta y los movimientos realizados por nuestro robot son los deseados (en lugar de

movimientos erróneos), nuestro planteamiento habrá demostrado su validez.

Una vez expuesto nuestras ideas, la principal línea de trabajo futuro pasa por comprobar el resultado de añadir una localización basada en nuestro algoritmo a las primeras fases de un proceso de localización que utilice el método de Monte Carlo.

Otra de las lineas para trabajar en un futuro están relacionadas con la mejora de la estimación de la posición del robot una vez se haya realizado el proceso y se hayan obtenido las posiciones probables.

Referencias

[1] J. Borenstein, H. Everestt, and L. Feng. *Where am I? Sensors and Methods for Mobile Robot Positioning.* 1996.

[2] R. T. Committee. Robocup four-legged league rule book. Technical report, RoboCup Technical Committee, 2007.

[3] T. Cover and P. Hart. Nearest neighbor pattern classification. 1967.

[4] M. A. Crespo, J. M. Cañas, and V. Matellan. Comparing bayesian and montecarlo localization for a robot with local vision, 2003.

[5] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE InternationalConference on Robotics*, 1999.

[6] G. Dudek, M. Jenkin, E. Milios, and D. Wilkes. Map validation and robot self-location in a graph-like world, 1997.

[7] J. Gutmann. Markov-kalman localization for mobile robots, 2002.

[8] R. A. Lastra, P. A. Vallejos, and J. R. del Solar. Integrated self-localization and ball tracking in the four-legged robot soccer league. 2004.

[9] P. S. Maybeck. *The Kalman filter: An introduction to concepts.* Springer-Verlag New York, Inc., 1990.

[10] M. Mitchell. *An Introduction to Genetic Algorithms.* 1996.

[11] T. Rofer and M. Jungel. *Vision-based fast and reactive Monte-Carlo localization.* 2003.

[12] B. Sierra. *Aprendizaje Automático: Conceptos básicos y avanzados.* Prentice Hall, 2006.

[13] R. Talluri and J. Aggarwal. Mobile robot self-location using model-image feature correspondence. 1996.

[14] I. Witten, E. Frank, L. Trigg, M. Hall, G. Holmes, and S. Cunningham. Weka: Practical machine learning tools and techniques with java implementations, 1999.