Instituto Tecnológico de Tuxtla Gutiérrez

Ingeniería Electrónica

Reporte de Residencia

Desarrollo de un Sistema Personalizado de Aprendizaje en Línea

Presentado por:

Ricardo Vega Ayora

Realizado en:

Intelligent Systems Research Laboratory

Asesor Interno:

Ing. Lester Acosa Maza

Asesor Externo:

Professor Atta Badii

Contenidos

Capítulo 1

"Generalidades"

Introducción

El reconocimiento de actividades humanas es importante para aplicaciones como la interacción humano-máquina y el tratamiento médico. Este proyecto busca diseñar e implementar una manera de reconocer cuando una persona está de pie y cuando está caminando. El reconocimiento será a través de sensores ubicados en el pecho y la espalda de los sujetos de prueba. Los sensores usados son acelerómetros integrados en Unidades de Medición Inercial (IMU, del inglés Inertial Measurment Unit). Aunque las IMUs también contienen giroscopios y magnetómetros, sólo los acelerómetros fueron usados. Información técnica de los sensores como el proceso de calibración, los requerimientos de energía y sus limitaciones no fueron incluidos en este documento. La forma en que los sensores envían sus lecturas a la computadora también fue descartada de este reporte.

Métodos de minería de datos e inteligencia artificial fueron usados para construir el reconocedor de actividades. El resultado final distingue entre estar de pie y caminar con una precisión superior a 96%.

El Sistema Operativo para Robots (ROS, del inglés Robot Operating System) fue usado por su capacidad de enviar mensajes entre procesos e hilos. Esto permite aislar cada tarea en un programa diferente, simplificando el desarrollo y organizándolo. Junto a ROS, la Interfaz de Programación de Aplicaciones (API, del inglés Application Programming Interface) de Qt fue usada para desarrollar el reconocedor de actividades. La API de Qt hizo posible desarrollar una aplicación robusta. Cabe mencionar que todo el software utilizado en este trabajo es de código abierto.

Acerca del ISR

El Laboratorio de Investigación de Sistemas Inteligentes (ISR, del inglés Intelligent Systems Research Laboratory) es un centro de investigación desarrollando proyectos en robótica y diferentes áreas de informática, como búsquedas semánticas, visión por computadora e interacción humano-máquina.

Ubicado en la Escuela de Ingeniería en Sistemas de la Universidad de Reading, el ISR es el Centro de Investigación Principal del Centro Europeo de Excelencia en Videoanálisis Socioético y de Preservación de Privacidad (Lead Research Centre for the European Socio-Ethical and Privacy-Preserving Video-Analytics Centre of Excellence). El ISR tiene un importante historial en investigación e innovación con contribuciones en varios proyectos nacionales e internacionales exitosos.

Más información acerca de los proyectos desarrollados en el ISR puede ser obtenida en su sitio web: http://www.isr.reading.ac.uk/.

Antecedentes

Ya se han realizado varios proyectos exitosos en el reconocimiento de actividades diarias como caminar y correr usando sensores ubicados en el cuerpo de las personas. Es importante saber que los sensores no son invasivos, se usan en accesorios como pecheras que los mantienen fijos al cuerpo. En [1], por ejemplo, se recopila información de los sensores y luego es clasificada con un clasificador de tipo bosque aleatorio y tiene una precisión de 94%. En [2] se usa selección de características para reducir el número de dimensiones usadas en la clasificación y con esto se mejora el desempeño. También desarrollaron un marco de trabajo multicapa que también mejora el desempeño del sistema al clasificar las actividades de manera jerárquica.

Otros tipos de actividades también son reconocidos usando sistemas similares, como en [3], donde una IMU se coloca en la muñeca de trabajadores para saber si están taladrando, martillando, atornillando o ninguna. El clasificador de ese trabajo está basado en el método del vecino más cercano (KNN, del inglés k-nearest neighbour), alcanzando una precisión promedio de 90%.

Planteamiento del Problema

La cantidad de problemas involucrados con actividades humana es considerablemente grande, algunos ejemplos son gente con vocaciones que dependen de la condición física. Estas personas necesitan mejorar su condición física para poder alcanzar velocidades más altas, dar saltos de mayor altura o resistir estrés físico por más tiempo. Actualmente estas mejoras requieren de un análisis exhaustivo por parte de los científicos y expertos en deporte para ser posibles.

Además, gente con problemas neurológicos que impiden o dificultan el movimiento de sus extremidades se ve limitada físicamente y en algunos casos socialmente también. Al igual que el caso de la mejora de deportistas, el método más común de rehabilitación exige mucha atención por parte de los terapistas.

Aun cuando se ha hecho investigación en el campo del reconocimiento de actividades, todavía queda mucho por hacer y entender para llegar a un punto en el que la gente pueda realmente hacer uso de esta tecnología.

Justificación

Un sistema de reconocimiento de actividades humanas ayudará a científicos a entender qué tan relacionadas están las actividades entre sí, cuáles son los cambios en velocidad y aceleración cuando, por ejemplo, una persona empieza a caminar después de haber estado detenido.

Diseñar e implementar un reconocedor de actividades también ayudará a que en el futuro se pueda entender y monitorear el estado físico de las personas para así poder mejorarlo. Esto puede ayudar a un deportista a ser mejor en el deporte que practique, o ayudar a alguien con un problema neurológico a superar completa o parcialmente su estado.

Objetivos

Objetivo General

Desarrollar con las herramientas de ROS un reconocedor de actividades capaz de recibir información de sensores a través de mensajes de ROS, reconocer si la persona que usa los sensores está de pie o está caminando y enviarlo a un servidor web.

Objetivos Específicos

- Analizar la información pregrabada que contiene lecturas de acelerómetros de gente estando de pie y gente caminando.

- Entrenar un clasificador que pueda distinguir si una persona está de pie o está caminando.

- Agregar el clasificador a un nodo de ROS para poder clasificar información proveniente de los sensores.

Alcances y Limitaciones del Proyecto

Por el momento, el sistema será capaz de reconocer sólo dos actividades: caminar y estar de pie. La precisión será de mínimo 90%. El sistema únicamente usará información de acelerómetros, descartando las lecturas de los giroscopios y los magnetómetros de las IMUs.

El clasificador no podrá volver a entrenarse automáticamente. Los parámetros de su entrenamiento serán escritos en el código y será necesario modificar directamente el código para mejorar el clasificador, ya se para incrementar la precisión o agregar más actividades.

El sistema será capaz de enviar su predicción a un servidor web para que la información pueda ser usada en algún otro lugar.

Metodología

Todas las áreas involucradas con el desarrollo del reconocedor de actividades están divididas en siete etapas de acuerdo al área con la que se relacionan. La figura 1 muestra el diagrama de flujo de las etapas.

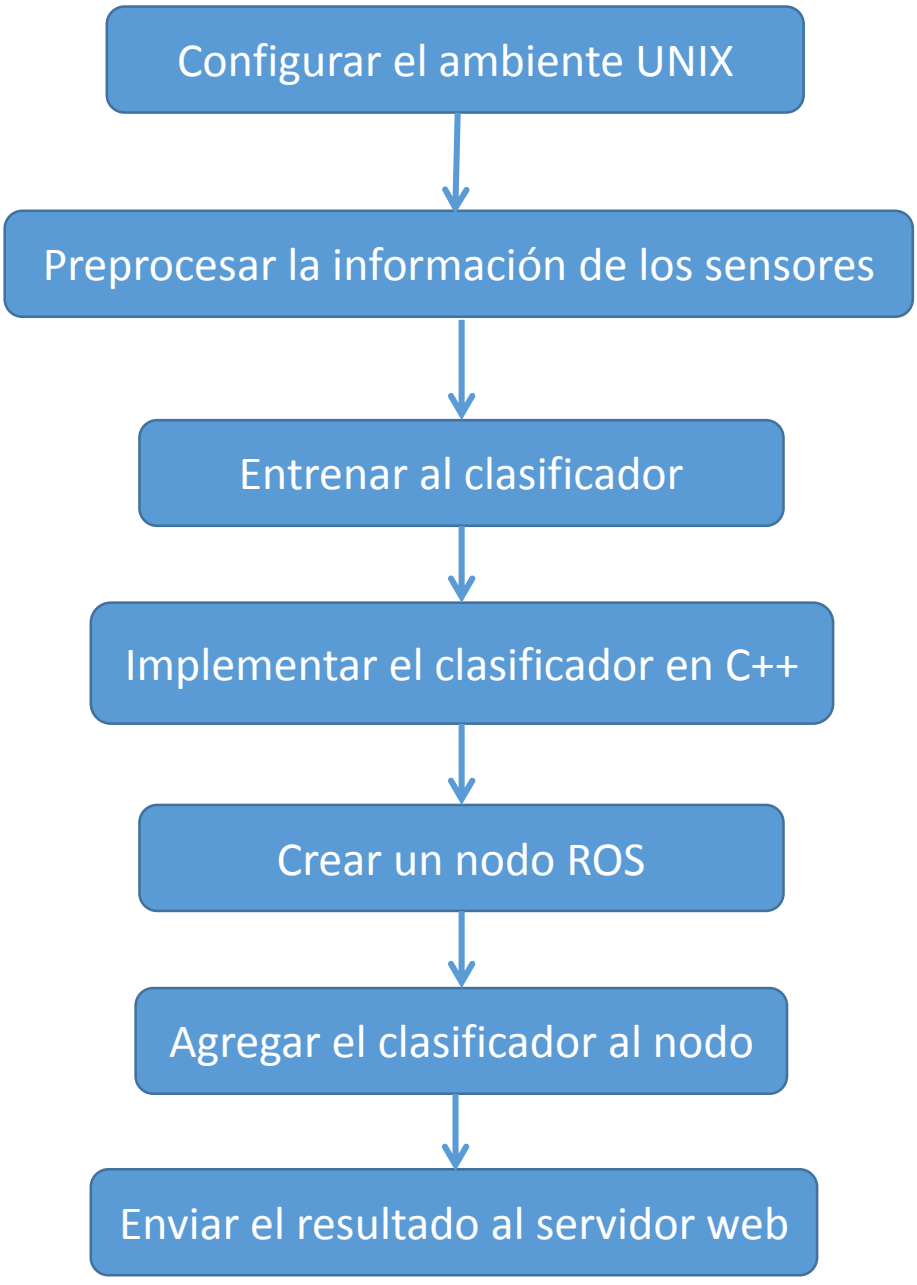

Figura 1 Diagrama de flujo del proyecto.

Etapa 1 – Configurar el ambiente UNIX

Ubuntu Linux 10.04 fue descargado e instalado en la estación de trabajo junto con el resto del software necesario. El administrador de paquetes de Ubuntu facilita la descarga e instalación de las aplicaciones y bibliotecas requeridas. Este administrador de paquetes puede ser usado ya sea a través de su interfaz gráfica o a través de la terminal con el comando:

```
sudo apt-get install 'package-name'
```

Donde **package-name** es reemplazado por el nombre del paquete deseado, en este caso Qt 4.7, ROS Electric and Weka 3.6.10.

Etapa 2 – Preprocesar la información de los sensores

Esta etapa se concentra en entender y preparar la información de los sensores para poder introducirla en el algoritmo de entrenamiento y producir el clasificador.

Información de muestra fue guardada en archivos CSV (comma separated vector) anteriormente. El primer archivo contiene sólo información de una persona estando de pie. El segundo archivo contiene información de ambas actividades, estar de pie y caminar.

A ambos archivos les hace falta la columna de la clase, esto significa que cada renglón debe ser clasificado previamente para poder usar los archivos en el entrenamiento del clasificador. Para el primer archivo es sencillo porque contiene sólo una actividad, por lo que basta con agregar *standing* al final de cada renglón.

Sin embargo, el segundo archivo contiene información de ambas actividades y no se puede saber a simple vista qué renglón pertenece a qué actividad, por esto fue necesario usar agrupamiento (o clustering), un método de aprendizaje no supervisado que divide información en clases. El agrupamiento se hizo con Weka.

Etapa 3 – Entrenar el clasificador

Ya se trabajó anteriormente con los archivos y los resultados muestran que un clasificador de Naive Bayes supera a las otras opciones de clasificación como árboles de decisiones y el método KNN. En esta etapa se entrena y prueba al clasificador de Naive Bayes con Weka, después se implementa en C++ para ser agregado al nodo ROS.

Etapa 4 – Implementar el clasificador en C++

Weka puede producir código fuente para varios clasificadores con algunas excepciones. El clasificador de Naive Bayes es una de las excepciones, por lo que el código deberá ser escrito manualmente.

Etapa 5 – Desarrollar un nodo ROS

Las lecturas de los sensores son publicadas en un tema ROS, esto quiere decir que se debe desarrollar un nodo ROS que se suscriba al tema y reciba la información.

Etapa 6 – Agregar el clasificador al nodo ROS

Una vez que el nodo esté trabajando y recibiendo la información del tema ROS, el clasificador de Naive Bayes debe ser agregado al nodo para clasificar las lecturas de los sensores.

Etapa 7 – Enviar el resultado de la clasificación al servidor web

El sistema debe ser capaz de enviar los resultados de la clasificación a un servidor web, por lo que es necesario implementar un cliente web en el sistema.

Capítulo 2

"Marco Teórico"

Ubuntu Linux

Linux es un clon del sistema operativo UNIX, escrito desde el inicio por Linus Torvalds con asistencia de un equipo de hackers de la red. Busca cumplir con las especificaciones Posix y Single UNIX.

Tiene todas las características que se buscan en un UNIX moderno, incluyendo la operación multitarea, memoria virtual, bibliotecas compartidas, carga en demanda, ejecutables compartidos, una correcta administración de memoria y trabajo multistack en red, incluyendo IPv4 y IPv6.

Aunque originalmente fue desarrollado para PCs de 32 bits, ahora Linux funciona en una multitud de arquitecturas, variantes tanto de 32 como de 64 bits.

Ubuntu es una de las distintas distribuciones de Linux. Fue elegido para este proyecto porque ROS no tiene problemas de compatibilidad con él.

GNU Compiler Collection (GCC)

GCC es un conjunto de herramientas de compilación que incluye compiladores para C, C++, Objective-C, Fortran, Java, Ada, y Go, también incluye bibliotecas para estos lenguajes. GCC fue escrito originalmente como el compilador del sistema operativo GNU, el cual fue desarrollado para ser 100% libre, es decir, que respeta la libertad del usuario.

ROS

El Sistema Operativo para Robots (ROS) es un marco de trabajo flexible para desarrollar software de robots. Es un conjunto de herramientas, bibliotecas y convenciones que busca simplificar la tarea de crear comportamientos complejos y robustos para una amplia variedad de robots.

ROS es multiplataforma y puede ser instalado en varios sistemas operativos y robots. También incluye controladores para varios sensores industriales.

Paquetes ROS

El software en ROS es organizado en paquetes. Un paquete puede contener nodos ROS, una biblioteca no relacionada con ROS, archivos de configuración, software de terceros y cualquier otra cosa que constituya un módulo útil. El propósito de estos paquetes es proveer funcionalidad en una forma fácil de entender para que el software pueda ser reusado fácilmente. En general, los paquetes ROS siguen el principio “Goldilocks”: suficiente funcionalidad para ser útil, pero no demasiada para evitar que el paquete se complique y sea difícil de usar desde software diferente.

Nodos ROS

Un nodo puede ser visto como un programa. Los nodos se comunican entre sí usando temas, servicios RPC y el Servidor de Parámetros. Los nodos fueron diseñados para operar a una escala de grano fino. Un sistema de control robótico usualmente tendrá varios nodos. Por ejemplo, un nodo controla un sensor láser, otro nodo controla las ruedas del robot, otro nodo hace localización, otro nodo hace planeación de rutas, etc.

El uso de nodos en ROS provee varios beneficios al sistema general. Hay una tolerancia a fallas ya que los problemas se aíslan en el nodo en el que ocurren. La complejidad del código se reduce en comparación con sistemas monolíticos.

Mensajes ROS

Los nodos se comunican entre sí publicando mensajes en temas, Un mensaje es una simple estructura de datos, incluye varios tipos campos de información, como enteros, números de punto flotante, booleanos, etc. Los arreglos también pueden ser usados en mensajes porque son tipos primitivos. Los mensajes pueden contener estructuras y arreglos anidados, como las estructuras de C.

Temas ROS

Los temas son buses con nombre por los cuales los nodos intercambian mensajes. Los temas tienen una semántica de publicación/suscripción anónima, lo que separa a la producción de información del consumo de la misma. En general, los nodos no saben con quién se comunican, sólo se interesan en la información publicada en los temas. Un tema puede tener varios publicistas y varios suscriptores.

Computación Concurrente

Se dice que un sistema es concurrente si puede realizar dos acciones al mismo tiempo. Una aplicación concurrente tendrá dos o más hilos en proceso en algún momento. Esto significa que la aplicación tiene dos hilos que están siendo trabajados por el sistema operativo en un procesador de un solo núcleo.

Qt

Qt es un marco de trabajo para desarrollo que tiene herramientas diseñadas para acelerar la creación de aplicaciones e interfaces de usuario para plataformas de escritorio, embebidas y móviles. Qt permite reusar código eficientemente y soportar múltiples plataformas con un solo código base.

Minería de Datos

La minería de datos se define como el proceso de encontrar patrones en información. El proceso debe ser automático o (más comúnmente) semiautomático. Los patrones encontrados deben ser significativos, es decir, deben representar una ventaja.

Patrones útiles permiten crear predicciones importantes en información nueva. Hay dos extremos en la expresión de un patrón: una caja negra de la cual no se puede comprender el interior, y una caja transparente cuya construcción revela la estructura del patrón. Amabas hacen buenas predicciones, la diferencia está en la capacidad de examinar y entender la estructura del patrón para poder informar nuevas decisiones. Tales patrones son llamados estructurales porque capturan la estructura de una manera explícita. En otras palabras, ayudan a explicar algo de la información.

Clasificación de Datos

La clasificación de datos es un proceso de dos pasos. En el primero, el clasificador es construido describiendo un conjunto predeterminado de clases y conceptos. Éste es el paso de aprendizaje (o etapa de entrenamiento), donde un algoritmo de clasificación construye el clasificador analizando o "aprendiendo de" un conjunto de datos de entrenamiento construido a partir de renglones de bases de datos con sus respectivas clases, que normalmente son el último valor en el renglón. Matemáticamente, un renglón es representado por un vector n atributos.

$$X = (a_1, a_2, \dots, a_n)$$

Un vector como este muestra n mediciones hechas en el renglón n atributos. Respectivamente, se asume que cada renglón X pertenece a una determinada clase determinada por uno de los atributos en el vector. La clase categoriza al vector dependiendo del valor que tenga, cada valor es una clase diferente. Los renglones que forman el conjunto de entrenamiento son llamados renglones de entrenamientos y son seleccionados analíticamente de la base de datos. En el contexto de la clasificación, renglones de datos pueden ser referidos como muestras, ejemplos, instancias, puntos de datos u objetos.

Debido a que la clase es proveída en todos los renglones, este paso se conoce también como aprendizaje supervisado. Contrasta con el aprendizaje no supervisado en donde el valor de la clase no se conoce y en donde el número de clases por aprender puede ser desconocido.

Weka

Weka es un conjunto de algoritmos de aprendizaje de máquina para tarea de minería de datos. Los algoritmos pueden ser aplicados directamente a un conjunto de datos o llamados desde código Java. Weka contiene herramientas para preprocesar datos, clasificación, regresión, agrupación, visualización y reglas de

asociación. También es adecuado para desarrollar nuevos esquemas de aprendizaje de máquina.

Capítulo 3

"Desarrollo"

Etapa 1

Ubuntu fue instalado en la computadora de trabajo en una partición secundaria, esto hizo posible usarlo como sistema operativo alternativo a Windows™. No fue instalado en una máquina virtual porque la virtualización puede no ser tan eficiente como el sistema físico.

Cuando la instalación de Ubuntu terminó, hizo falta descargar e instalar el controlador de red porque Ubuntu 10.04 no tiene el driver para la tarjeta de red del sistema. En situaciones como ésa es en donde tener dos sistemas operativos es una ventaja, el controlador fue descargado desde Windows™ y luego desde Ubuntu, el cual incluye a GCC, el controlador fue compilado e instalado. Era necesario habilitar la conexión a la red porque el administrador de paquetes **aptitude**, la necesitaba para descargar e instalar el resto del software.

El software que hacía falta fue instalando usando el administrador de paquetes a través de la terminal. Qt 4.7 y Weka son instalaciones directas, es decir, no se requieren pasos adicionales para configurarlos y usarlos. Sin embargo, hizo falta configurar ROS para que sus bibliotecas y ejecutables estuvieran disponibles para todo el sistema. La configuración consta de agregar la ruta de instalación de ROS a la variable de entorno PATH.

Etapa 2

Como se mencionó anteriormente, dos archivos con muestras pregrabadas fueron proveídos por el trabajo anterior para poder entrenar el clasificador. Durante el resto del documento el primer archivo será llamado *conjunto de pie* y el segundo *conjunto transición* para poder distinguirlos fácilmente y saber a qué actividad se refieren. El *conjunto de pie* contiene información sólo de la actividad de estar de pie. El *conjunto transición* contiene información tanto de estar de pie como de caminar.

Antes de implementar el clasificador la información debe ser preprocesada limpiándola y agrupándola. Como se mencionó antes, los archivos están en formato CSV en los cuales las columnas están separadas por tabuladores. Los archivos contienen nueve columnas de los cuales sólo 3 corresponden a los ejes del acelerómetro, las otras seis columnas fueron eliminadas con Weka y el símbolo decimal, que era una coma, fue reemplazado por un punto.

A todos los renglones del *conjunto de pie* se les agregó un último atributo con la palabra “standing”, este atributo es la clase y significa que la persona está de pie.

El *conjunto transición* no fue tan sencillo porque combina información de ambas actividades, hacía falta separar las dos clases pero primero había que saber qué renglones pertenecían a qué actividad.

Las herramientas de aprendizaje no supervisado de Weka simplificaron el trabajo de agrupación. Se utilizó la interfaz gráfica de Weka para acelerar el proceso. El explorador de Weka contiene herramientas para preoprocesar, clasificar y agrupar.

La figura 2 muestra la ventana principal de Weka y la figura 3 la ventana del explorador.

Figura 2 Ventana principal de Weka.

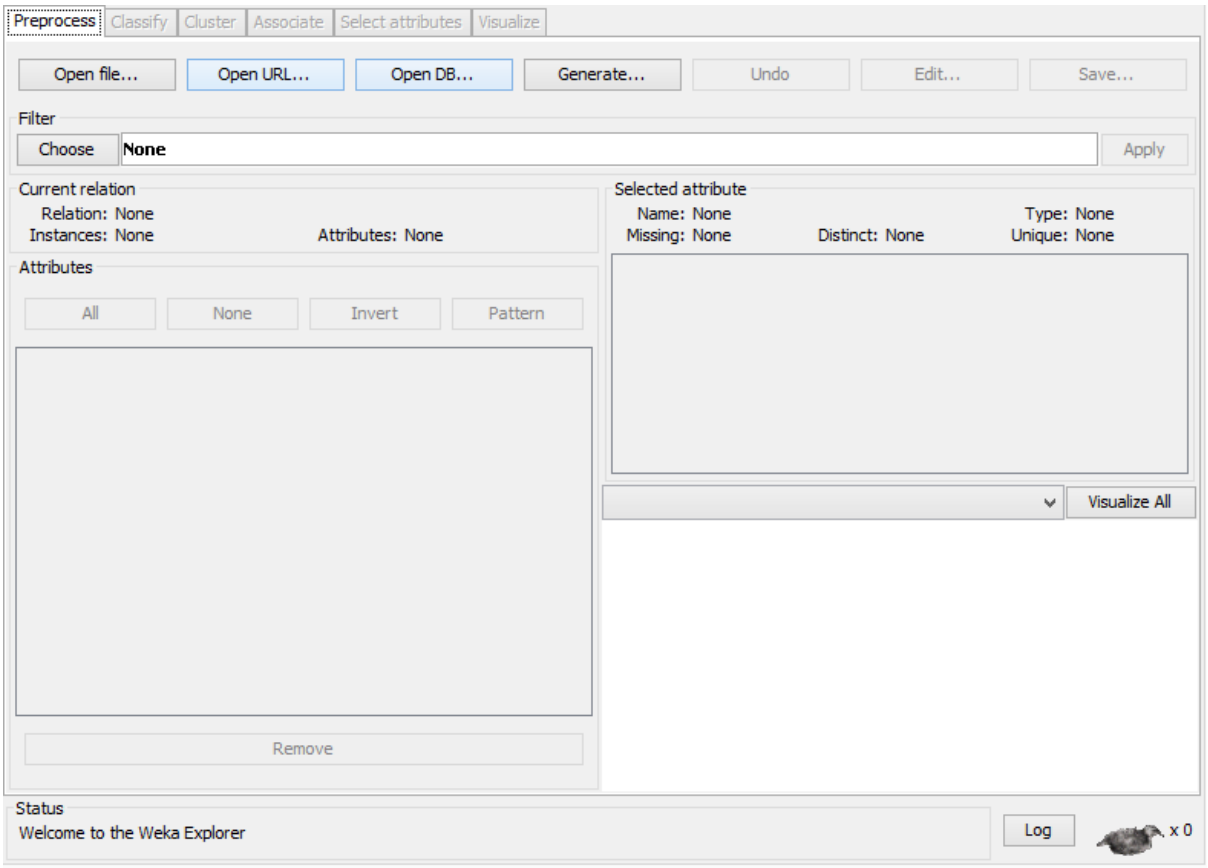

Figura 3 Ventana del explorer de Weka.

Se probaron todos los métodos de agrupación disponibles en Weka, cada uno con varias combinaciones de parámetros y los mejores resultados fueron dados por MakeDensityBaseClusterer con la siguiente configuración:

```
MakeDensityBaseClusterer 1.0E-6 EM 100 1.0E-6 2 100
```

MakeDensityBaseClusterer es un comando que envuelve a un agrupador y lo hace regresar una distribución y una densidad. 1.0e-6 es la desviación estándar mínima; EM es el agrupador envuelto, 100 es el número de iteraciones que hará el agrupador, 1.0e-6 es la desviación estándar mínima del agrupador, 2 es el número de clases a encontrar y el último 100 es la semilla para generar números aleatorios.

Los resultados de la agrupación pueden verse en la figura 4.

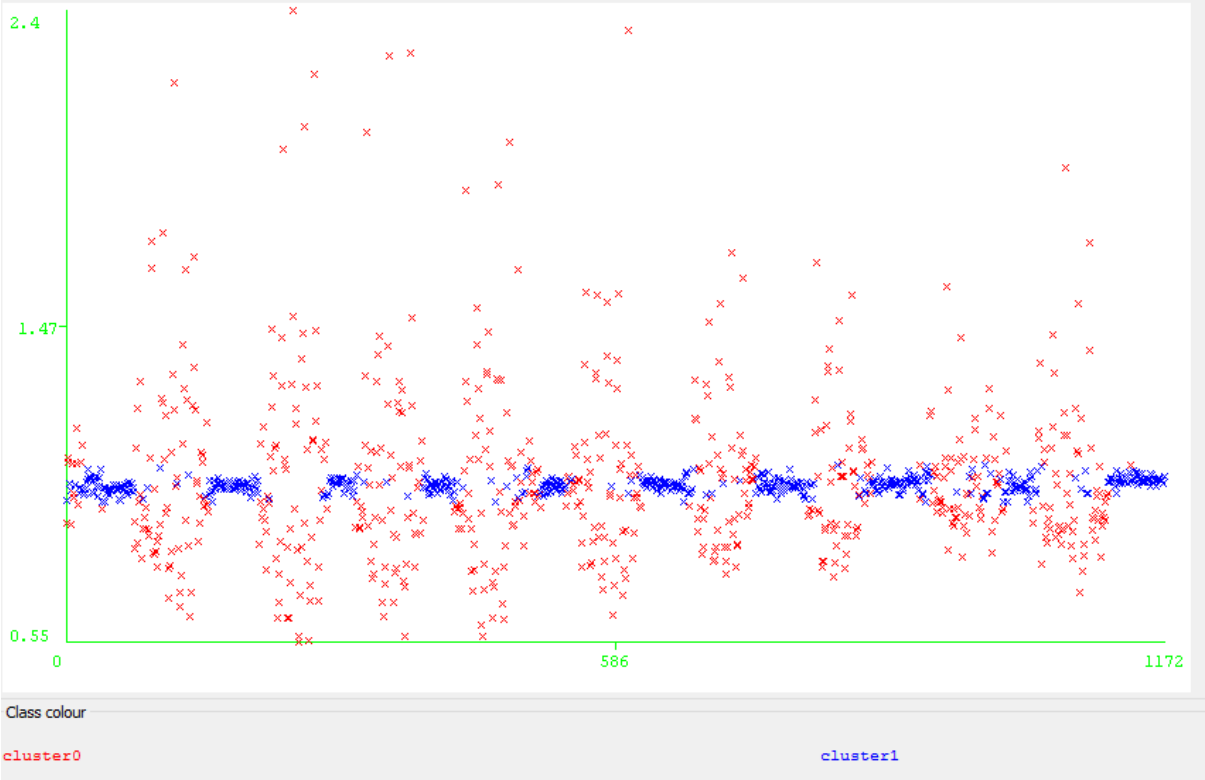

Figura 4 Resultados de la agrupación.

Se puede apreciar que la agrupación hecha por Weka es muy buena, los puntos rojos o cluster0 indican una magnitud mayor en las lecturas del acelerómetro, lo que significa que la persona está caminando. Los puntos azules o cluster1 se mantienen en un valor casi constante, por lo que se puede asumir que la persona está de pie sin moverse.

A continuación se presenta los resultados de Weka al agruparÑ

```
Instances:    1173
Attributes:   3
              AX
              AY
              AZ
Test mode:evaluate on training data

=== Model and evaluation on training set ===

MakeDensityBasedClusterer:

Wrapped clusterer:
EM
```

```
==

Number of clusters: 2


            Cluster
Attribute         0       1
              (0.56)  (0.44)
============================
AX
  mean        1.0319  1.0068
  std. dev.   0.2596  0.0213

AY
  mean       -0.0091  0.0188
  std. dev.    0.133  0.0505

AZ
  mean         0.055  0.1166
  std. dev.   0.1679  0.0471


Fitted estimators (with ML estimates of variance):

Cluster: 0 Prior probability: 0.5472

Attribute: AX
Normal Distribution. Mean = 1.032 StdDev = 0.2628
Attribute: AY
Normal Distribution. Mean = -0.0107 StdDev = 0.1345
Attribute: AZ
Normal Distribution. Mean = 0.0525 StdDev = 0.1693

Cluster: 1 Prior probability: 0.4528

Attribute: AX
Normal Distribution. Mean = 1.0074 StdDev = 0.0205
Attribute: AY
Normal Distribution. Mean = 0.0199 StdDev = 0.0494
Attribute: AZ
Normal Distribution. Mean = 0.1178 StdDev = 0.0465


Time taken to build model (full training data) : 0.53 seconds

=== Model and evaluation on training set ===

Clustered Instances

0       645 ( 55%)
1       528 ( 45%)


Log likelihood: 2.41887
```

Después de la agrupación, Weka permite guardar un archivo con el valor de la clase de en cada renglón, al guardar ese archivo y volverlo a abrir con Weka es posible cambiar el nombre de cluster0 a “walking” y cluster1 a “standing”. Después de esto se combinaron los dos archivos (*conjunto de pie* y *conjunto caminando*) en uno solo.

Etapa 3

El conjunto resultando fue introducido a Weka para entrenar al clasificador de Naïve Bayes. Weka muestra los valores de la media y la desviación estándar de cada atributo, pero no es capaz de producir código fuente del clasificador. El clasificador tuvo que ser implementado manualmente.

El siguiente texto es el registro del entrenamiento del clasificador de Naive Bayes:

```
Instances:    2426
Attributes:   4
              AX
              AY
              AZ
              Class
Test mode:10-fold cross-validation

=== Classifier model (full training set) ===

Naive Bayes Classifier

                  Class
Attribute      standing  walking
                 (0.73)   (0.27)
================================
AX
  mean            1.0047   1.0317
  std. dev.       0.0165   0.2622
  weight sum        1781      645
  precision       0.0083   0.0083

AY
  mean            0.0068   -0.011
  std. dev.       0.0295   0.1344
  weight sum        1781      645
  precision       0.0064   0.0064

AZ
  mean            0.1326   0.0528
  std. dev.       0.0347    0.169
  weight sum        1781      645
  precision       0.0063   0.0063



Time taken to build model: 0 seconds

=== Stratified cross-validation ===
=== Summary ===
```

```
Correctly Classified Instances        2349               96.8261 %
Incorrectly Classified Instances        77                3.1739 %
Kappa statistic                          0.9217
Mean absolute error                      0.0372
Root mean squared error                  0.1538
Relative absolute error                  9.5254 %
Root relative squared error             34.8029 %
Total Number of Instances             2426

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure
ROC Area  Class
                 0.957     0          1          0.957     0.978
1         standing
                 1         0.043      0.893      1         0.944
1         walking
Weighted Avg.    0.968     0.011      0.972      0.968     0.969
1

=== Confusion Matrix ===

    a    b   <-- classified as
 1704   77 |    a = standing
    0  645 |    b = walking
```

Una validación cruzada de 10 iteraciones fue usada para probar la precisión del clasificador y el resultado muestra que la clasificación tiene una precisión de 96.8261%. Este es un resultado satisfactorio que es superior al 90% esperado.

Etapa 4

De [4] se puede asumir la siguiente regla:

$$\hat{y} = arg\max_{y} P(y) \prod_{i=1}^{n} P(x_i|y)$$

Esto es el clasificador en sí. En este caso, los atributos son los tres ejes del acelerómetro y las clases son *standing* (de pie) y *walking* (caminando). Una expansión de la fórmula es:

$$\max(P(standing)P(ax, ay, az|standing), P(walking)P(ax, ay, az|walking))$$

Se asume que la probabilidad de los atributos es Gaussiana:

$$P(x_i|y) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\left(-\frac{(x_i-\mu)^2}{2\pi\sigma^2}\right)}$$

Hay una estructura y tres clases que conforman la mayor parte del reconocedor de actividades. *Attribute* (figura 5), *DataClass* (figura 6) y *NaiveBayesClassifier* (figura 7).

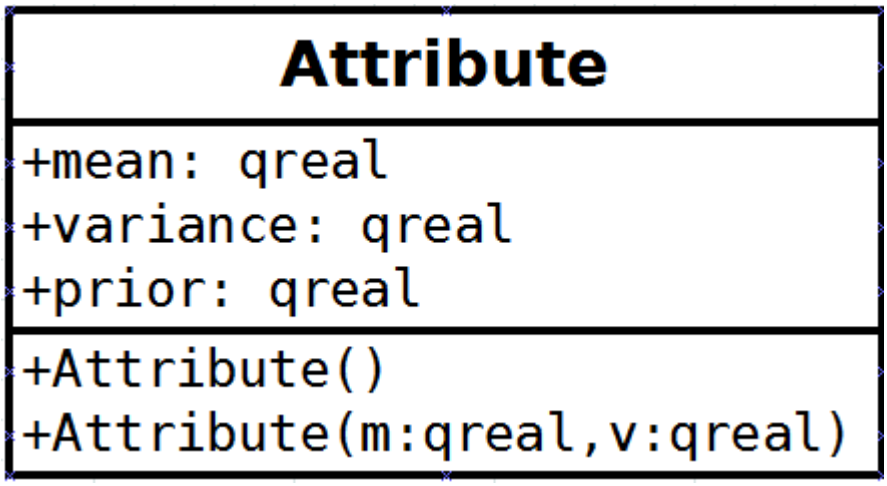

Figura 5 Estructura Attribute.

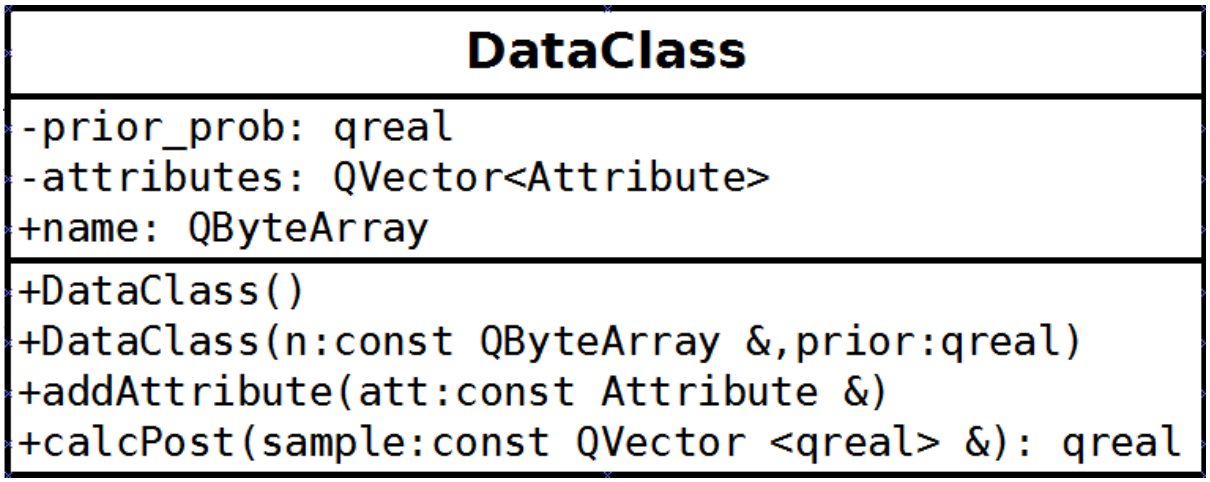

Figura 6 Clase DataClass.

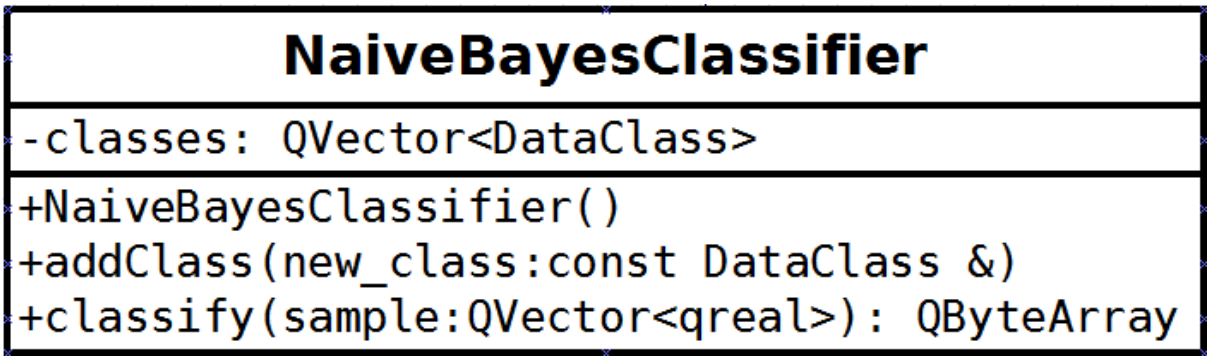

Figura 7 Clase del Clasificador de Naive Bayes.

Aun cuando el clasificador etará preentrenado, en el future será expandido para ser reentrenado automáticamente y para soportar más actividades (como dar vueltas) y más atributos (como las lecturas del giroscopio), por esto los atributos y las clases tienen su propia estructura y no están escritas directamente en el clasificador.

En la estructura *Attribute*, *mean* (media) y *variance* (varianza) se entienden directamente. Sin embargo, *prior* no. Este último miembro representa la fracción de la probabilidad Gaussiana:

$$\frac{1}{\sqrt{2\pi\sigma^2}}$$

El propósito de este miembro es reducir la complejidad de los cálculos matemáticos al momento de la clasificación. *Prior* es calculado automáticamente de la varianza cuando se crea una instancia de *Attribute*, esto significa que el usuario debe usar únicamente el constructor con parámetros en su lista. El constructor con la lista de parámetros vacía es usado por *QVector* y no debe ser usado por el usuario.

DataClass almacena información de una clase del conjunto de datos. Contiene un *QVector* de attributos y un número de doble precisión para almacenar la probabilidad a priori de la clase. Como con *Attribute*, sólo el constructor con parámetros en su lista debe ser usado.

Cada vez que un atributo es agregado a una instancia de *DataClass*, el valor de la probabilidad a prior es reemplazado por el producto de la probabilidad a priori actual y el miembro *prior* del atributo agregado, ayudando a reducir la complejidad matemática de la clasificación. El reemplazo de la probabilidad a priori puede ser expresado como:

$$p_i = p_{i-1} * prior$$

El método *calcPost(const QVector<qreal> sample)* toma una muestra (la cual debe tener el mismo número de elementos que los atributos en la clase) y calcula la parte exponencial de la probabilidad Gaussiana:

$$e^{\left(-\frac{(x_i-\mu)^2}{2\pi\sigma^2}\right)}$$

El resultado de esto es multiplicado por la probabilidad a priori de la clase y el producto es lo que el método regresa.

NaiveBayesClassifier contiene un QVector de clases, la muestra será clasificada en una de ellas. Cuando *classify(QVector<qreal> sample)* es llamado, *NaiveBayesClassifier* ejecuta *a*l método *calcPost(const QVector<qreal> sample)* de cada clase, este método regresará la probabilidad a posteriori de cada clase, la clase que regrese la probabilidad más alta será el resultado final.

Para usar el clasificador:

1. Crear las clases requeridas.
2. Llenar las clases de atributos.
3. Agregar las clases al clasificador.
4. Tomar una muestra.
5. Pasar la muestra al clasificador.

Etapa 5

El nodo ROS que contiene al clasificador de Naïve Bayes se subscribe al tema llamado **/AccData** ya que las lecturas del acelerómetro son publicadas ahí.

La wiki de ROS tiene un ejemplo de cómo programar un suscriptor, el nodo ROS del reconocer de actividades está basado en este ejemplo.

La arquitectura de ROS permite al programa trabajar en varias tareas concurrentemente. El suscriptor recibe mensajes en su propio hilo mientras el resto del programa trabaja en otras cosas. Cuando el suscriptor recibe un mensaje, éste llama al método *callBack* con el que esté relacionado con el mensaje recibido como argumento. El clasificador es usado en este método *callBack*.

Etapa 6

Una vez que el nodo ROS estuvo listo, el clasificador le fue agregado y el método *callBack* fue adaptado para enviar el resultado de la clasificación al servidor web.

La figura 6 muestra la estructura de la clase del reconocedor de actividades.

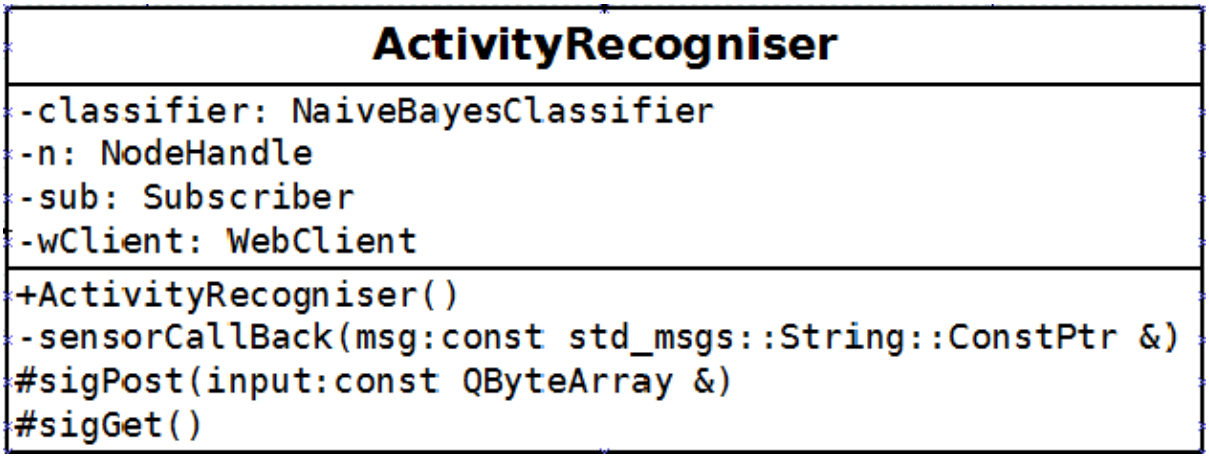

Figura 8 Estructura del reconocedor de actividad.

Los valores de la probabilidad a priori de cada clase junto con la media y la varianza obtenidos con Weka después de entrenar al clasificador fueron escritos como definiciones de preprocesador en el archivo cabecera de la clase del reconocedor de actividades, de esta forma basta con cambiar el valor de las definiciones sin tener que modificar el reconocedor en sí.

El constructor crea seis atributos, tres para los ejes X, Y y Z del acelerómetro para la actividad *standing* y otros tres para la actividad *walking*. Cada atributo es inicializado con sus valores de media y varianza correspondientes, la tabla 1 ilustra esto.

Attribute	Mean	Variance
s_ax	1.0047	0.00027225
s_ay	0.0068	0.00087025
s_az	0.1326	0.00120409
w_ax	1.0317	0.06874884
w_ay	-0.011	0.01806336
w_az	0.0528	0.028561

Tabla 1 Valores de la media y la varianza para los atributos.

Después de crear los objetos de tipo *Attribute*, el constructor crea dos objetos de tipo *DataClass*, uno para cada actividad. Estos son inicializados con el nombre que le corresponde (*standing* y *walking*) y el valor de su probabilidad a priori, luego sus respectivos atributos son agregados. La tabla 2 ilustra la inicialización de las clases.

Class	Prior probability	Attributes
Standing	0.73	s_ax, s_ay, s_az
Walking	0.27	w_ax, w_ay, w_az

Tabla 2 valores de la probabilidad a priori de cada actividad.

Cuando un mensaje es recibido, el reconocedor de actividades comprueba que la información provenga de las IMUs, si el mensaje tiene información de las IMU, los valores de X, Y y Z son extraídos del mensaje, colocados en un QVector de muestra y éste es pasado al clasificador para ser clasificado. El resultado es enviado al servidor web.

Etapa 7

El cliente web aprovecha las ventajas de varios de los componentes y herramientas principales de Qt, incluyendo a la clase *QThread*, la cual permite que un objeto sea movido a un hilo diferente, dividiendo así la carga de trabajo y convirtiendo en concurrente a la aplicación.

La figura 9 muestra la estructura del cliente web.

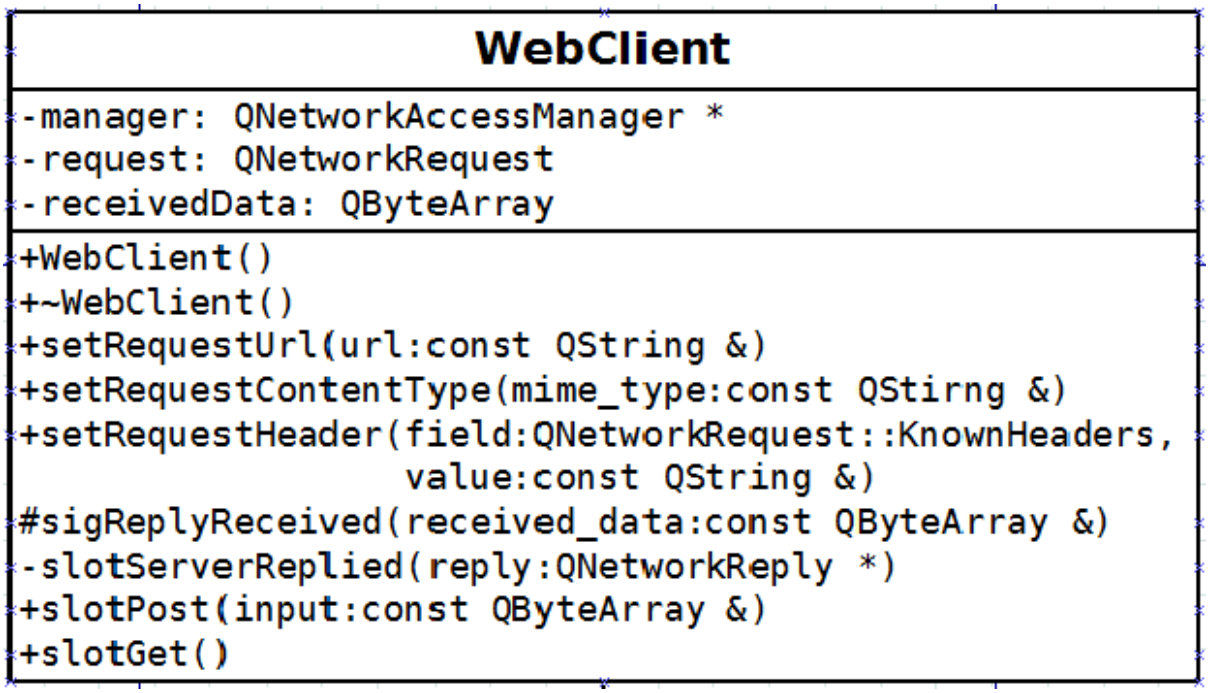

Figura 9 Estructura de la clase del cliente web.

Hizo falta desarrollar un algoritmo para poder usar la clase *QThread* correctamente.

Una implementación simple de lo que la documentación de Qt dice es explicada en [5]. El algoritmo presentado aquí es una extensión del método presentado en ese artículo.

Cada *QObject* debe ser creado en el hilo en el que su objeto padre está para después ser movido a un hilo diferente. Intentar crear al objeto directamente en otro hilo causará un error.

El algoritmo es:

1. Crear al objeto padre.
2. Crear un puntero al hilo en donde el objeto nuevo va a ser ejecutado.
3. Crear un apuntador al objeto nuevo.
4. Mover el objeto nuevo al nuevo hilo.
5. Conectar las señales del objeto padre con los slots del objeto nuevo y viceversa. Sólo las señales con las que se intercambiará información. La conexión debe ser *Qt::AutoConnection* o *Qt::QueuedConnection*.
6. Conectar la señal *finished()* del hilo con el slot deleteLater() del objeto nuevo. La conexión debe ser de tipo *Qt::AutoConnection* or *Qt::DirectConnection*.
7. Iniciar el hilo.
8. Intercambiar información con el objeto nuevo.
9. Llamar al método *exit()* del hilo.
10. Llamar al método *wait()* del hilo.
11. Borrar el hilo.

El siguiente código es un ejemplo que ilustra al algoritmo.

```
class Parent : public QObject {
   Q_OBJECT
   QThread *childThread;
   Child *child;
public:
   Parent() {
      childThread = new QThread;
      child = new Child(); // Hasta aquí el hijo y el padre están en
el mismo hilo.
      child->moveToThread(childThread);
      connect(this, SIGNAL(sigChildDoSomething()), child,
SLOT(slotDoSomething()), Qt::QueuedConnection);
      connect(child, SIGNAL(sigAnswer(int)), this,
SLOT(slotChildAnswered(int)), Qt::QueuedConnection);
      connect(childThread, SIGNAL(finished()), child,
SLOT(deleteLater()));
      childThread->start();
   }
   ~Parent() {
      childThread.exit();
      childThread.wait();
      delete childThread;
   }
   void method1() {
      emit sigChildDoSomething(); // Esto hará que el código del
slot al que esta señal esté conectada ejecute su código. El objeto
padre hará otras cosas.
      // Hacer otras cosas.
   }
signals:
   void sigChildDoSomething();
public slots:
   void slotChildAnswered();
};
```

Los pasos del 1 al 4 no serán detallados porque son conocimiento básico de C++.

En el paso 5 se establece un puente para intercambiar información entre hilo. Tanto conexiones encoladas (Queued connections) [6] como conexiones encoladas de bloqueo (blocking queued connections) pueden ser utilizadas para comunicar hilos diferentes. El tipo adecuado para el cliente web es conexión encolada porque permite que ambos hilos trabajen al mismo tiempo mientras la información es transferida. Una conexión encolada de bloqueo detendría al reconocedor de -actividades hasta que la información haya sido procesada por el cliente web.

Con usos como el anterior, *QThread* funciona como un envoltorio de clases para administrar los recursos del objeto dentro de él (el cliente web en este caso), entonces cuando una señal externa invoca a un slot del objeto envuelto, *QThread* agenda la ejecución del slot para cuando el método que esté en ejecución en ese momento regrese.

En el paso 6 se crea una conexión entre la señal *finished()* del hilo y el slot *deleteLater()* del objeto nuevo para que cuando el hilo vaya a ser terminado la memoria ocupada por el objeto nuevo sea liberada correctamente [7]. Esta conexión debe ser automática (Qt::AutoConnection) o directa (Qt::DirectConnection) porque de otra forma el objeto no será borrado porque *QThread* agendará la eliminación del objeto para cuando el método en ejecución termine, pero para ese momento el hilo ya habrá terminado y la eliminación del objeto quedará encolada, lo que requeriría reiniciar la computadora para liberar esa memoria.

En el paso 7 se inicia el ciclo de eventos del nuevo hilo. A partir de este momento se puede intercambiar información bidireccionalmente entre hilos. El intercambio debe ser a través de señales y slots para evitar que los hilos compartan memoria y se necesiten bloqueos mutex. Si se pretende llamar métodos del objeto nuevo directamente desde el padre, los mutex deben ser usados para evitar que la información se corrompa.

En el paso 8 se intercambia información, este paso se repite tantas veces como se desee.

Una vez que el hilo secundario ya no es necesario se pasa a los pasos 9 y 10. En el paso 9 se llama al método *exit()* para detener al ciclo de eventos y después, en el paso 10, se llama al método *wait()* para bloquear al hilo principal hasta que el secundario haya terminado correctamente.

Aun cuando el código anterior funciona correctamente, si el objeto nuevo tiene objetos anidados, estos deberán ser movidos al hilo nuevo también ya que son creados en el hilo principal originalmente. Una solución a esto es crear punteros a los objetos anidados e inicializarlos en NULL en el constructor del objeto nuevo. Luego en otro método se instancia al objeto con el operador new. El siguiente código ilustra la solución.

```
class Child : public QObject {
   Q_OBJECT
   GrandChild *grandchild;
public:
   Child() {grandchild = NULL;}
   ~Child() {delete grandchild;}
signals:
   sigAnswer();
public slots:
   slotDoSomething() {
      if (grandchild == NULL) {
         grandchild = new GrandChild();
         grandchild.moveToThread(this->thread());
      }
      // hacer algo.
      emit sigAnswer();
   }
};
```

El código anterior se asegura de que los objetos anidados sean instanciados sólo una vez.

Ya no es necesario comunicar al objeto nuevo con sus objetos anidados a través de señales y slots porque no hay necesidad para objetos en el mismo hilo.

La figura 10 muestra el diagrama de clases del sistema completo.

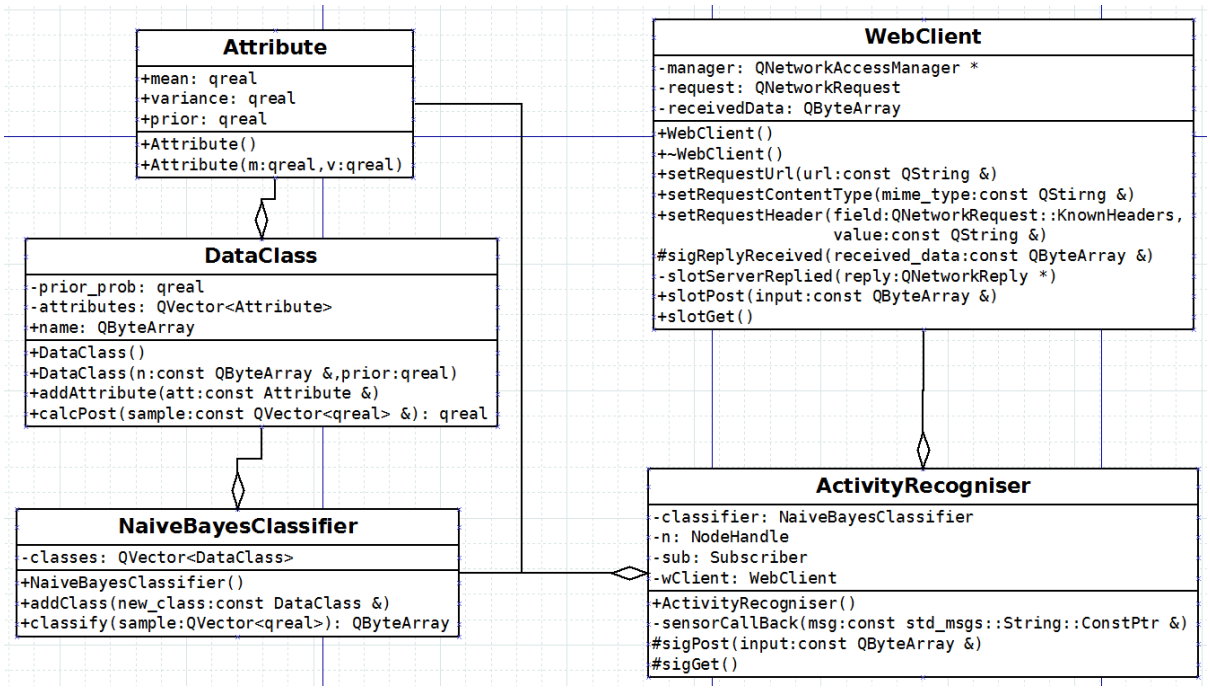

Figura 10 Diagrama de clases del sistema de reconocimiento de actividades.

Capítulo 4

"Resultados y Conclusiones"

Resultados

El producto final es un sistema modular capaz de distinguir cuando una persona camina de cuando está de pie a partir de información proveniente de acelerómetros en el pecho y la espalda de las personas para después enviar el resultado a un servidor web para que pueda ser utilizada de manera remota. El sistema tiene una precisión de 96.8%, la cual es mayor al 90% esperado.

Trabajo a Futuro

El siguiente paso en el futuro cercano es agregar más actividades al sistema, algunos ejemplos son correr, brincar, girar a la izquierda y a la derecha.

También, agregar las lecturas de los sensores restantes de la IMU al proceso de clasificación para incrementar la resolución y la precisión del sistema. Esto también hará al sistema más robusto.

En el futuro, se puede extender el conocimiento del comportamiento del cuerpo humano cuando realiza ciertas actividades agregando sensores de señales electromiográficas para obtener información de los músculos involucrados en cada actividad.

Conclusiones

Linux, independientemente del nombre de su distribución, es un sistema robusto, estable, completo y sobre todo, extensible. Es un sistema que ha estado involucrado en investigación científica desde hace mucho tiempo. Linux está en el 98% de las supercomputaduras del mundo así como en la mayoría de los servidores que le dan vida a la internet [8].

Las herramientas que ofrece Linux para desarrollar aplicaciones pueden tener una curva de aprendizaje lenta pero, una vez que se adquiere el conocimiento básico, el proceso de desarrollo se vuelve simple y el producto final es tan robusto y completo como el desarrollador desee.

Usar Ubuntu simplificó el desarrollo del nodo ROS porque es completamente con su arquitectura. Otras distribuciones de Linux cuentan únicamente con versiones experimentales de ROS.

Todas las características ofrecidas por ROS aceleran el desarrollo cuando sensores y otros tipos de elementos de robótica están involucrados. Las tareas de bajo nivel como el acceso a sensores y la transferencia de mensajes son administradas transparentemente por ROS. También hay características de alto nivel como la navegación que son realizadas tan bien como las de bajo nivel.

Qt eleva el nivel en el que el desarrollador trabaja al brindar clases y herramientas que de otra forma el desarrollador tendría que construir por su cuenta. Esto requeriría invertir tiempo que podría ser usado en la parte principal del desarrollo.

Weka es una herramienta excelente para la minería de datos, hace que la clasificación y la agrupación sean tareas simples así como la implementación de los clasificadores.

Todos los diagramas en Lenguaje Unificado de Modelado (UML, del inglés Unified Modelling Language) fueron hechos con Dia. Aplicaciones como ésta simplifican la tarea del modelado ya que las referencias gráficas organizan la estructura modular del sistema.

La programación orientada a objetos hizo posible desarrollar un Sistema modular que puede ser mantenido y extendido fácilmente en el futuro.

Referencias

Software

ROS. http://www.ros.org

Qt. http://qt.digia.com/, http://qt-project.org/

Linux. https://www.kernel.org/

Ubuntu. http://www.ubuntu.com/

GCC. http://gcc.gnu.org/

Dia. https://wiki.gnome.org/Apps/Dia

Bibliografía

[1] J. Vitrià, J.M. Sanches, and M. Hernández, “Human Activity Recognition from

Accelerometer Data Using a Wearable Device”: IbPRIA 2011, LNCS 6669, pp. 289–296, 2011.

[2] Mi Zhang, Alexander A. Sawchuk, "A Feature Selection-Based Framework for Human Activity Recognition Using Wearable Multimodal Sensors", International Conference on Body Area Networks (BodyNets), Beijing, China, November 2011.

[3] H Koskimaki, V Huikari, P Siirtola, P Laurinen, J Roning “Activity recognition using a wrist-worn inertial measurement unit: A case study for industrial assembly lines”, Control and Automation, 2009. MED'09. 17th Mediterranean Conference on, 401-405

[4] Scikit-Learn Developers (2013). *Naive Bayes*. [ONLINE] Available at: http://scikit-learn.org/stable/modules/naive_bayes.html. [Last Accessed 04 November 13].

[5] Maya Posch (2011). How To Really, Truly Use QThreads; The Full Explanation. [ONLINE] Available at: http://mayaposch.wordpress.com/2011/11/01/how-to-really-truly-use-qthreads-the-full-explanation/. [Last Accessed 14 October 2013].

[6] Digia. Threading Basics. [ONLINE] Available at: http://qt-project.org/doc/qt-5.0/qtcore/thread-basics.html. [Last Accessed 14 October 2013].

[7] Digia. Threads and QObjects. [ONLINE] Available at: http://qt-project.org/doc/qt-4.8/threads-qobject.html. [Last Accessed 14 October 2013].

[8] The Linux Foundation (2013). About Us. [ONLINE] Available at: http://www.linuxfoundation.org/. [Last Accessed 20 August 2013].

[9] Clay Breshears, (2009). The Art of Concurrency: A Thread Monkey's Guide to Writing Parallel Applications. 1st ed. USA: O'Reilly Media, Inc.

[10] Ian H. Witten, Eibe Frank, Mark A. Hall, (2011). Data Mining: Practical Machine Learning Tools and Techniques. 3rd ed. USA: Morgan Kaufmann.

[11] Jiawei Han, Micheline Kamber, (2006). Data Mining: Concepts and Techniques. 2nd ed. USA: Morgan Kaufmann.

[12] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, Ian H. Witten (2009); The WEKA Data Mining Software: An Update; SIGKDD Explorations, Volume 11, Issue 1.