Trabajo de Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Detección de Personas mediante AdaBoost usando Sensor de Escáner Láser

Autor: Gonzalo Mier Muñoz
Tutor: Fernando Caballero Benítez

Dep. Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Trabajo Fin de Grado
Grado en Ingeniería Electrónica, Robótica y Mecatrónica

Detección de Personas mediante AdaBoost usando Sensor de Escáner Láser

Autor:
Gonzalo Mier Muñoz

Tutor:
Fernando Caballero Benítez
Profesor Contratado Doctor

Dep. de Ingeniería de Sistemas y Automática
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2016

Autor: Gonzalo Mier Muñoz

Tutor: Fernando Caballero Benítez

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2016

El Secretario del Tribunal

A mi familia
A mi tutor del Proyecto
A mis compañeros y compañeras

Agradecimientos

Este Trabajo Fin de Grado es un esfuerzo en el cual, directo o indirectamente participaron distintas personas opinando, corrigiendo, teniéndome paciencia, dando ánimo, acompañando en los momentos de crisis y en los de felicidad. Este trabajo me ha permitido aprovechar la competencia y la experiencia de muchas personas que deseo agradecer en este apartado.

Querría expresar mi reconocimiento y agradecimiento a todas aquellas personas que, gracias a su colaboración, han contribuido a la realización de este Trabajo Fin de Grado.

En primer lugar, mi sincero agradecimiento a Fernando Caballero Benítez, tutor de este proyecto, por sus consejos, ayuda y dedicación durante el desarrollo de este trabajo.

A todos mis compañeros/as y amigos/as, ya que con ustedes he compartido horas de trabajo y buenos ratos, lo cual no tiene precio.

Además, todo esto nunca hubiera sido posible sin el amparo incondicional que me otorgaron y el cariño que me inspiraron mis padres, quienes me entendieron en mis malos momentos. Que, a pesar de la distancia, siempre estuvieron a mi lado para saber cómo iba mi trabajo.

A Pablo y Diana, que se han preocupado por el desarrollo del trabajo y de mi carrera durante todos estos años, apoyándome cuando era necesario.

Gonzalo Mier Muñoz
Sevilla, 2016

Resumen

De un tiempo a esta parte, la investigación en robótica está avanzando espectacularmente. La robótica de ámbito industrial está siendo desplazada por los investigadores científicos al ámbito de lo social. La necesidad que surge de mano de obra en el sector asistencial y de servicios, está provocando que cada vez sea mayor el uso de robots que interaccionen con las personas.

Hoy en día, los robots autónomos móviles pueden rastrear la ubicación de una persona y proporcionar información adecuada del entorno.

Gracias a un proyecto de investigación internacional de robótica social financiado por la Unión Europea, se ha desarrollado un robot que actúa como guía turístico. Este robot social ha sido llamado Frog, y hacia su evolución en cuanto a la detección de personas va enfocado el proyecto.

El funcionamiento adecuado del robot exige un trabajo cooperativo de los sensores láser, para una navegación correcta del mismo, evitando los obstáculos e interactuando con las personas que se encuentran en su entorno.

El Trabajo Fin de Grado está dirigido a seguir avanzando en la detección de personas mediante algoritmos de láser. El proyecto va enfocado, principalmente, en dar mayor precisión al robot Frog a la hora de esquivar a las personas que se encuentran a su paso, mediante la creación de un nuevo paquete de ROS. Este paquete trata de detectar a las personas utilizando la información de cada sensor por separado, y posteriormente, fusionar dichas detecciones.

Para ello, se ha necesitado el uso de sensores de escáner láser, que nos han permitido medir la distancia con las personas, la utilización del software libre ROS, y la aplicación de algoritmos que han hecho posible este proyecto.

Abstract

Nowadays, research in robotics is advancing dramatically. The industrial robotics field is being displaced by scientific researchers to social sphere. The need of manpower in the care sector and service is causing the use of robots that interact with people increases.

Today, mobile autonomous robots can track the location of a person and provide adequate environmental information.

Thanks to an international research project of social robotics funded by the European Union, it has been developed a robot that acts as a tour guide. This social robot has been called Frog, and to its evolution in terms of the people detection is focused this project.

The correct operation of the robot requires a cooperative work of laser sensors for a good navigation, avoiding obstacles and interacting with people in their environment.

The Final Degree Project is aimed at further progress in the detection of people using laser sensors and artificial intelligence algorithms. The project is mainly focused on greater precision on people detection for Frog robot, to dodge or get closer to people who are in their path, by creating a new ROS package. This package tries to detect people using information from each sensor separately and then merge these detections.

To this end, it has been necessary the use of laser-scan sensors, which have allowed us to measure the distance with people, the use of free software ROS, and the application of algorithms that have made this project possible.

Índice general

Índice de figuras

Índice de cuadros

Notación

Bag	—	Componente de ROS que permite guardar información obtenida de tópicos para poder ser reproducida posteriormente.
Falsos positivos	—	En el contexto del proyecto, son detecciones de objetos obtenidas por el sensor láser, las cuales tienen las mismas características que busca al detectar personas, pero no son verdaderamente personas.
Falsos negativos	—	En el contexto del proyecto, son no detecciones puntuales de personas reales que se encuentran dentro del alcance de los detectores del robot, pero que estos detectores no las detectan temporalmente.
Framework	—	Estructura formada por distintos programas/ controladores...
IA	—	Inteligencia Artificial
Nube de puntos	—	Vector de puntos en x e y
Robot Operating System (*ROS*)	—	Software utilizado en programación que esta explicado con más detalle en el capítulo 3 de la memoria.
Sistema embebido	—	Es un sistema de computación, normalmente en tiempo real, que ha sido diseñado para realizar una o algunas funciones dedicadas.

1. Introducción

1.1. Contexto del Proyecto

El hecho de que los investigadores científicos se estén especializando en los robots sociales con la intención de cubrir unas necesidades que a medida que el mundo evoluciona se hacen más evidentes, ha hecho que nos fijemos en cómo con nuestro trabajo se puede implementar la efectividad y la precisión de un robot que nació para actuar como guía turístico.

En el entorno de los museos y de las visitas guiadas, se vio que el que un robot hiciese las funciones de guía, podía ser un modo de dar una visión distinta de dichos espacios culturales. El robot en cuestión podía interactuar con las personas, enseñando y explicando todo lo interesante del lugar. Con esta intención se creó el robot sobre el que se ha basado el Trabajo Fin de Grado. Frog, que así se ha llamado, se diseñó para dar cobertura a esa necesidad.

Frog, a pesar de ser un robot de última generación y de tener unas características específicas muy útiles para su función, también cuenta con puntos de su mecanismo que pueden ser perfeccionados. Uno de estos puntos es sobre el que se ha desarrollado todo el proyecto, siendo este el perfeccionamiento en la detección de personas en el entorno, y como las esquiva sin tener contacto con ellas.

1.2. Estado del Arte

La detección de personas en entornos sociales se ha vuelto un problema complejo de solucionar, por lo que muchos autores han decidido proponer sus propios clasificadores.

En interiores, [1] propone un detector de personas mediante sensor de escáner láser en una oficina. Usa 13 características para el clasificador AdaBoost y realiza una segmentación basándose en distancias. En contraposición, se propone usar las intensidades devueltas por el sensor láser para segmentar, además de añadir dos características más relacionadas con las intensidades, en [2].

En exteriores, se usan 63 características en [3], las cuales son 13 iniciales y las mismas multiplicadas y divididas por distancia al origen y número de puntos. Estas 13 características iniciales son las mismas que en [1], eliminando las características que no están relacionadas exclusivamente con el propio segmento, y añadiendo dos nuevas características.

Tanto [4] como [2], proponen usar más de un láser colocadas a distintas alturas. [2] usa dos sensores láser, uno colocado por debajo de la cintura y otro por la altura del pecho. El objetivo de esto es ser capaz de detectar personas tras objetos de mediana altura. [4] además añade otros dos lásers intermedios para hacer tracking a las personas detectadas.

Continuando con la fusión sensorial, [3] mejora la detección fusionando los resultados del láser con una cámara. Propone usar estas detecciones para crear regiones de interés donde buscar personas, con el objetivo de reducir el coste computacional del algoritmo de detección en imagenes.

Por último, en [5] y [6] prefieren usar un sensor LIDAR del cual extraer características de movimiento y de texturas. [7] lo complementa usando una cámara RGB para mejorar la detección.

1.3. Motivación

La motivación de esta investigación viene en parte provocada por la necesidad de crear una robótica móvil social que sea capaz de adaptarse a las normas sociales humanas, facilitando la relación robot-humano. Cuando se busca que un robot de estas dimensiones se mueva en espacios poblados de manera autónoma, hay que plantear no solo un generador de trayectorias que evite obstáculos, si no también, un detector de personas que sea capaz de discernir si el camino trazado invade el espacio vital de una persona o no.

Este proyecto tiene como motivación principal crear un paquete que cumpla tal función en el robot Frog, para ser usado como guía en el Alcázar de Sevilla. Por otro lado, esto presenta una doble dificultad añadida, ser capaz de usar el módulo en entornos con características de luz muy distintas, y realizar detecciones con unos sensores a los cuales se le añade como ruido la vibración de un suelo empedrado. Además, al estar en un sistema embebido, se requiere que la detección sea rápida y poco compleja.

Mi motivación personal es la de introducirme en el campo de la IA, el cual me ha llamado la atención por la gran cantidad de logros que ha conseguido en un periodo relativamente corto de tiempo. Ser capaz de fusionar la información de distintos sensores, predecir o extraer conclusiones de un set de datos, buscar para un problema soluciones no lineales óptimas o entender un texto son algunas cosas prácticamente impensables si no contamos con los avances de esta área en los últimos 50 años. Por todo ello, considero que es un campo muy interesante al que dedicar mi tiempo como investigador en pos de desarrollar nuevos avances con los que solucionar problemas, los cuales de otra forma se volverían imposibles.

1.4. Objetivos

La implementación de un detector de personas es el objetivo principal de este proyecto. Utilizando algoritmos de Inteligencia Artificial ya creados, se pretende que sea capaz de identificar qué segmentos de la nube de puntos proporcionada por el sensor corresponden a la proyección de una pierna humana usando un sensor de escáner láser. Con ello, se quiere hacer un estudio comparativo de estos algoritmos para mejorar su comportamiento en sistemas autónomos, y publicarlo posteriormente en internet para que pueda ser usado en nuevos proyectos como base para avanzar en la investigación de la robótica social.

En este proyecto, se pretende que el detector cumpla los siguientes objetivos:

- Segmentar correctamente la nube de puntos.
- Detectar personas en interiores.
- Detectar personas en exteriores.
- Reducir el coste computacional de aplicar el algoritmo.

- Reducir el número de Falsos Positivos y Falsos Negativos en la detección.

1.5. Estructura del proyecto

El primer paso fue implementar el algoritmo de IA, AdaBoost, ya que el paquete que viene en el framework ROS realiza una clasificación multivariable, la cual se ha demostrado ser mucho menos eficiente comparada con clasificadores monovariables en cuanto a ratio de aciertos para la misma capacidad de computación.

Con AdaBoost implementado, y habiendo comprobado que funcionaba correctamente, se generaron varios set de datos para entrenamiento y predicción en interiores, en distintos entornos poblados y no poblados. La segmentación en estas situaciones no supuso ningún inconveniente, ya que este problema es causado por la luz natural. Posteriormente se realizó el mismo proceso en exteriores, creando algoritmos que añadiesen robustez a la segmentación.

Con sendas bases de datos, se dispuso a implementar un nodo que extrajese las características propuestas de cada segmento y los almacenase en otra base de datos, siendo asociadas estas características con su correspondencia a un humano o no. A partir de ahí, se entrenó el algoritmo de AdaBoost con una base de datos que incluyese a personas y no personas tanto en interiores como en exteriores, dejando una base de datos únicamente de interiores y otra únicamente de exteriores para comprobar, por último, las virtudes y defectos del propio paquete.

2. Hardware y Software

2.1. Hardware

En este apartado del proyecto vamos a estudiar el equipo utilizado durante el transcurso de la investigación, describiendo el funcionamiento de la aplicación y como se implementa en el robot Frog. Lo que se pretende es mejorar y completar las funciones en detección de personas mediante algoritmos de IA que usen la información obtenida de un sensor de escáner láser en Frog. Esto le permitirá reconocer que hay personas en el espacio donde se encuentre, distinguiéndolas de los objetos inanimados y mediante los sensores incorporados, se moverá a su alrededor sin que usurpe el espacio vital de dichas personas.

Vamos a empezar explicando las características del robot Frog, y a continuación seguiremos viendo los sensores láser utilizados para poner en práctica este proyecto.

2.1.1. Robot Frog: Fun Robot Outdoor Guide

FROG (2.1) es el resultado de un proyecto de investigación internacional de robótica social financiado por la Unión Europea. Se ha desarrollado un robot de última generación que actúa como guía turístico. Ha sido probado ya como guía en el zoo de Lisboa y en el Alcázar de Sevilla.

Habla tres idiomas, inglés, español y holandés. La principal característica del robot FROG, es la de ser capaz de interactuar con las personas. A través del reconocimiento facial, puede detectar si el turista está atendiendo o si se está aburriendo. En estos casos, FROG es capaz de cambiar su exposición, reducirla e incluso hacerla más amena llegando a poder bromear con las personas.

Figura 2.1: Frog real

Dispone de una pantalla táctil (2.2), y además es capaz de proyectar imágenes en las paredes. Su autonomía es de 3 horas aproximadamente, y cuenta con los medios necesarios para detectar su nivel de batería, de manera que él de forma autónoma se retira para cargarse de nuevo. A pesar de los avances con los que cuenta este robot, se echa en falta el que todavía no cuenta con un programa de reconocimiento de voz.

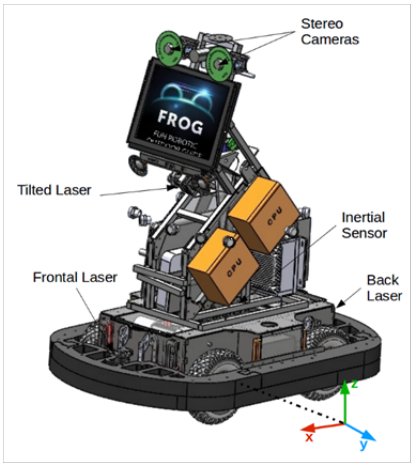

Figura 2.2: Interior de Frog

2.1.2. Sensores

Sensor HOKUYO URG-04LX-UG01

Figura 2.3: Sensor HOKUYO URG-04LX-UG01

El láser URG-04LX (2.3) es un sensor para escaneado de áreas. La luz del sensor es infrarroja, siendo el área de escaneado un semicírculo de 240º (2.4). El principio de medición de la distancia se basa en el cálculo de la diferencia de fase, debido a lo cual es posible obtener la medición estable con influencia mínima desde el color del objeto y la reflectancia.

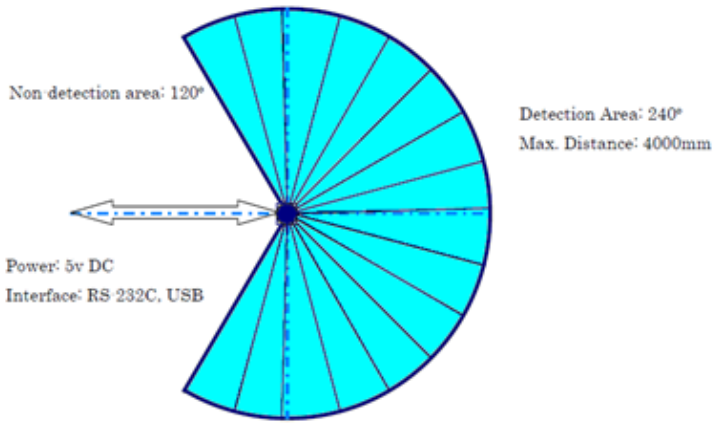

Figura 2.4: Campo de visión del sensor

Este láser esta colocado en los laterales del robot FROG a una altura de 50 centímetros de altura (2.5).

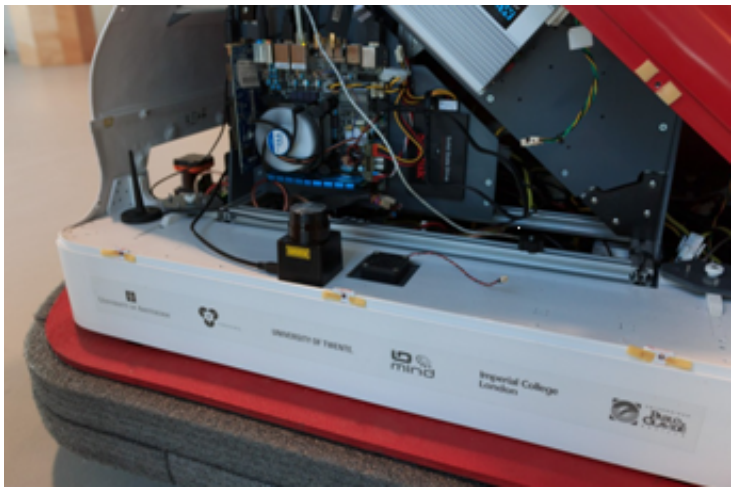

Figura 2.5: Posición del sensor

Datos técnicos:

N ° de Modelo	URG-04LX-UG01
Fuente de alimentación	5VDC ± 5 % (USB Bus de potencia)
Fuente de luz	diodo láser semiconductor (λ = 785nm), Protección de láser clase 1
Área que mide	20 a 5600mm (papel blanco con 70 mm × 70 mm), 240°
Exactitud	60 a 1.000 mm: ± 30 mm, 1,000 a 4,095mm: ± 3 % de la medición
Resolución angular	Ángulo de paso: aprox. 0.36° (360° / 1.024 pasos)
El tiempo de escaneado	100 ms / escaneado
Ruido	25 dB o menos
Interfaz	USB 2.0 / 1.1 [Mini B] (velocidad total)
Sistema de comando	SCIP versión 2.0

Iluminación ambiental	Lámpara / mercurio halógeno: 10,000Lux o menos, fluorescente: 6000Lux (Max)
Temperatura ambiente / humedad	-10 A +50 ºC, 85 % o menos (sin condensación, ni la formación de hielo)
Resistencia de vibración	10 a 55 Hz, de 1,5 mm de amplitud doble cada 2 horas en X, Y y Z
Resistencia al impacto	196m / s 2, cada 10 vez en X, Y y Z
Peso	Aprox. 160g

Cuadro 2.1: Datos técnicos del sensor URG-04LX-UG01

Sensor HOKUYO UTM-30LX

Figura 2.6: Sensor HOKUYO UTM-30LX

Este sensor (2.6) es un dispositivo de alta velocidad para aplicaciones robóticas. Este modelo utiliza la interfaz USB 2.0 para la comunicación y puede obtener datos de medición en un campo de visión más amplio, hasta una distancia de 30 metros con una resolución milimétrica en un arco de

270º (2.7). Debido a la filtración interna y carcasa de protección, este dispositivo puede ser utilizado al aire libre y es menos susceptible a la luz ambiente.

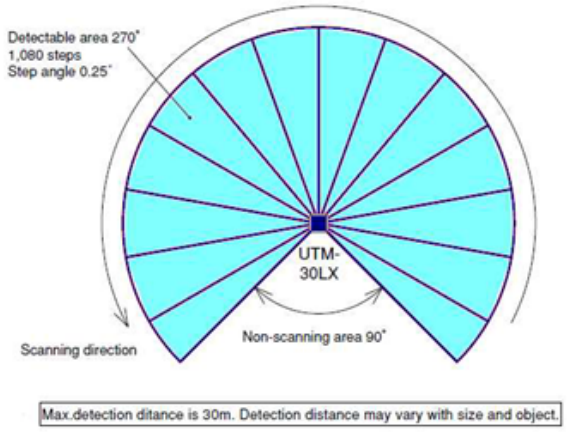

Figura 2.7: Campo de visión del sensor

Datos técnicos:

Nº de Modelo	UTM-30LX
Fuente de alimentación	12 V CC ± 10 % (consumo actual: Max: 1A, normal: 0.7A)
Fuente de luz	diodo láser de semiconductor (λ = 905 nm) Clase de seguridad del láser 1 (FDA)
Área que mide	0,1 a 30 m (casilla blanca Kent Hoja de 500 mm o más), Max.60m en 270°
Exactitud	0,1 a 10 m: ± 30 mm, 10 a 30 m: ± 50 mm * 1
Resolución angular	Ángulo de paso: aprox. 0,25° (360º / 1.440 pasos)
El tiempo de escaneado	25msec / escaneado
Ruido	Menos de 25 dB

Interfaz	USB 2.0 (Full Speed)
Salida sincrónica	NPN colector abierto
Sistema de mando	Diseñado exclusivamente comando SCIP versión 2.0
Conexión	Alimentación y la emisión síncrona: 2m volar alambre de plomo USB: 2 m de cable con conector tipo A
Temperatura ambiente / humedad	-10 A +50 °C, a menos de 85 % de humedad relativa (sin rocío y escarcha)
Resistencia de vibración	Doble amplitud de 1,5 mm de 10 a 55 Hz, 2 horas en las direcciones X, Y y Z
Resistencia al impacto	196m/s^2, 10 veces en X, Y y Z
Peso	Aprox. 370g (con el accesorio de cable)

Cuadro 2.2: Datos técnicos del sensor UTM-30LX

2.2. Software

Para el trabajo fin de grado se ha utilizado el software libre ROS (Robot Operating System) en el que se han implementado todos los algoritmos del proyecto.

Estos algoritmos han sido implementados en C++, creando el nodo en ROS. Para comprobar que el robot funcionase correctamente, se hicieron diferentes experimentos reales.

A continuación, se detalla el software ROS que se ha utilizado para el proyecto, describiendo en un primer punto las características principales de ROS, así como su funcionamiento.

2.2.1. ROS: Robot Operating System

ROS es un meta-sistema operativo muy utilizado en robótica. Proporciona servicios que se esperan de un sistema operativo, como abstracción hardware, control de dispositivos a bajo nivel, implementación de funcionalidades comúnmente utilizadas, paso de mensajes entre procesos y gestión de paquetes. También aporta herramientas y librerías para obtener, construir, escribir y ejecutar códigos entre múltiples ordenadores. Una de las principales ventajas de este framework es la capacidad de desarrollar nodos software reutilizables, además de proporcionar herramientas y algoritmos de uso típico en robótica.

Su principal característica es la capacidad de dividir las acciones complejas en funciones más pequeñas que implementan una parte concreta del proceso, encapsuladas en ficheros ejecutables llamados nodos, que se comunican entre ellos usando buses de datos para llevar a cabo la función encomendada. Decir que dentro del sistema cada nodo es único, diferenciándose unos de otros y pudiéndose comunicar entre ellos, publicando o subscribiéndose a un tópico, proveyendo o usando un servicio, o usando acciones. Todo este complejo entramado hace que cuando se detecta algún error, sea más fácil su corrección, ahorrando mucho tiempo de búsqueda de errores.

ROS es un software de código abierto y libre tanto para uso comercial como de investigación, bajo licencia BSD. Está disponible desde Enero de 2010 con la versión 1.0 y hasta la actualidad se han lanzado otras versiones: Box Turtle (2010), C Turtle (2010), Diamondback (2011), Electric Emys (2011), Fuerte Turtle (2012), Groovy Galapagos (2012), Hydro Medusa(2013), Indigo Igloo (2014), Jade Turtle (2015) y la última versión Kinetic Kame (2016).

En ROS hay implementados varios estilos de comunicación, como comunicación síncrona de tipo RPC (llamada a procedimiento remoto, del inglés Remote Procedure Call) a través de servicios, comunicación asíncrona por medio de topics (o tópicos) y almacenamiento de datos en un servidor de parámetros que se pueden consultar desde cualquier nodo.

Para mayor comprensión se describen a continuación los sistemas de ficheros de ROS.

Sistema de ficheros de ROS

En el sistema de ficheros de ROS, se pueden distinguir:

- Paquetes (packages): Los paquetes es la principal unidad para organizar el software en ROS. Un paquete puede contener nodos (la unidad de ejecución en ROS), librerías, data sets o ficheros de configuración, entre otros.
- Ficheros de manifiesto (manifest.xml): Los manifiestos proporcionan metadatos de un paquete, incluyendo información de licencia, dependencias con otros paquetes y opciones de compilación.
- Stacks: Un stack es un conjunto de paquetes que proporcionan una funcionalidad. Un ejemplo de stack es la "*navigation_stack*" disponible en los repositorios de ROS.
- Ficheros de manifiesto de un stack (stack.xml): De forma similar al manifiesto de un paquete, este tipo de ficheros proporcionan información del stack, como la versión, la licencia y las dependencias.
- Descripción de mensajes: Estos ficheros describen los mensajes a utilizar por los nodos, definiendo la estructura de datos del mismo. Se suelen ubicar dentro de la carpeta msg de un paquete y tienen la extensión "*.msg*".
- Descripción de servicio: Se utilizan para describir los servicios, definiendo la estructura de datos para la petición y la respuesta del servicio. Se suelen ubicar dentro de la carpeta srv de un paquete y tienen la extensión "*.srv*".

Protocolos de comunicación entre nodos en ROS

A continuación se pasa a describir los distintos elementos que forman parte del proceso de comunicación entre nodos:

- Nodo: Los nodos son procesos que realizan la computación. Un nodo es un archivo ejecutable dentro de un paquete de ROS. Mediante los nodos se puede realizar un complejo modular en el que los nodos se comunican entre sí. Un sistema de control del robot comprende por lo general muchos nodos.

- Máster: El ROS Máster proporciona el registro de nombre y consulta el resto del grafo de computación. Almacena información de registro de topics y servicios de nodos ROS. El Máster coordina los nodos realizando una función similar a la de un servidor DNS en internet.
- Servidor de parámetros: Este permite almacenar datos, de forma que se pueden actualizar y consultar por cualquier nodo. Actualmente es parte del Máster.
- Mensajes: Es la forma de comunicación entre los nodos. Un mensaje es una estructura simple con campos tipados. Los tipos que se permiten son desde tipos básicos como enteros o reales a estructuras C más complejas y vectores.
- Tópico o topic: El topic es el canal de comunicación entre nodos. El intercambio de mensajes entre nodos lo hacen a través de los topics que son buses de datos. El topic es un nombre que se usa para identificar el contenido del mensaje. Un nodo publica un mensaje en un topic al que se ha suscrito un segundo nodo para leer los mensajes allí depositados. Pueden existir muchos publicadores concurrentemente para un solo topic y un nodo puede suscribirse y/o publicar en varios topics. El nodo suscriptor o publicador no tiene porque conocer si existen otros nodos utilizando el mismo topic.
- Servicios: Cuando no es posible la comunicación entre nodos mediante mensajes, ROS permite la comunicación a través del servicio. Este está definido por dos mensajes, uno para la petición y otro para la respuesta. De esta manera el nodo ofrece un servicio con un nombre específico y otro nodo pude solicitar dicho servicio mediante un mensaje de petición.
- Acciones: Se diferencian de los servicios, en que se envía una petición y se recibe una respuesta, pero en esta existe la posibilidad de cancelar el servicio, y por tanto no es necesario esperar hasta obtener la respuesta.
- Rosbags: Son archivos donde se guardan los datos de los mensajes publicados por los nodos publicadores que interesan para una posterior utilización. Los rosbags son muy útiles para almacenar datos de sensores para después probar los algoritmos de forma offline en nuestro ordenador.
- Rviz: Es un entorno gráfico de visualización en 3D para ROS (2.8).

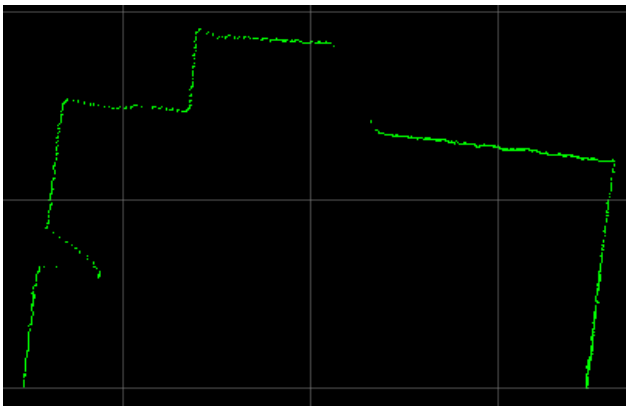

Figura 2.8: Nube de puntos en Rviz

2.2.2. Paquete HOKUYO NODE

Es el controlador que se encarga de leer la información que proporciona el láser. Esta información se compone de una sucesión de distancias a las que se detectan objetos. Para visualizar los datos de forma legible y sea comprensible para las personas y para el robot, es necesaria una herramienta que interprete esta información. La herramienta rviz de ROS representa de forma gráfica esos datos de distancias.

Este paquete devuelve como información un tipo de variable llamado "LaserScan" al topic "\scan". La variable "LaserScan" es una estructura de datos compuesto por:

- Float32 angle_min, es el ángulo de inicio de la exploración.
- Float32 angle_max, es el ángulo final de la exploración.
- Float32 angle_increment, es la distancia angular entre las mediciones.
- Float32 range_min, es el valor de rango mínimo.
- Float32 range_max, es el valor máximo del rango.
- Float32[] ranges, es el vector que almacena la distancia a la que encuentra un obstáculo.
- Float32[] intensities, es el vector que almacena la intensidad con la que se recibe el láser.

3. Adaboost

3.1. Concepto

AdaBoost (Adaptative Boosting) es un algoritmo de IA que fusiona clasificadores binarios para crear un clasificador mejor. A cada uno de estos clasificadores binarios se les denomina "*weak learners*" o "*clasificadores débiles*" y al clasificador completo "*strong learner*" o "*clasificador fuerte*".

El único requisito que tienen que cumplir los *weak learners* para formar un *strong learner* es que el error de clasificación sea menor de 0.5, lo cual significa que acierte más de la mitad de las veces. Adaboost se caracteriza porque es capaz de lidiar con un gran número de variables durante el entrenamiento, eligiendo unicamente aquellas que realmente mejoran la detección. Gracias a esto, se pueden conocer las variables que no intervienen, permitiendo reducir el tiempo de predicción, no calculándolas.

Se ha decidido no usar el paquete de ROS donde esta implementado AdaBoost, ya que es una versión del algoritmo que considera que cada uno de los *weak learners* es un clasificador multivariable, dando como frontera de clasificación un hiperplano de dimensión idéntica a variables de entrada. Esta aproximación clasifica mejor con pocos *weak learners*, pero añade complejidad tanto en el entrenamiento como en la predicción, siendo computacionálmente más costosa, e imposibilitando eliminar las variables que menos intervengan.

En su contra, se ha implementado el algoritmo haciendo que los *weak learners* sean monovariables, lo que significa que cada clasificador se verá caracterizado unicamente por una variable, el valor de la frontera, el sentido de esta y el peso con el que intervienen.

3.2. Segmentación

El algoritmo de clasificación AdaBoost se caracteriza por ser entrenado "*out-of-the-box*", o fuera de línea. Por ello, se necesita un set de datos que relacione la salida deseada con las entradas del algoritmo.

Como ya se ha mencionado anteriormente, la información proporcionada por el sensor de escáner láser es un vector de distancias a las que el haz de luz rebota para cada ángulo. Si se quiere detectar donde se sitúan las personas en ese vector, hace falta ser capaz de segmentarlo aplicando algún criterio que separe adecuadamente en distintos objetos. Este paso es completamente necesario y va a condicionar todo el experimento, ya que una mala segmentación puede hacer que el segmento de una persona se fusione con segmentos de pared, por lo que la clasificación se volvería imposible (tanto para crear la base de datos de entrenamiento como para clasificar con el algoritmo entrenado).

Lo primero es filtrar las medidas obtenidas. Si la distancia es menor de la mínima o mayor que la máxima que devuelve el sensor, es que se ha producido un error en la medida provocada por una superficie reflectante o en mal ángulo, que no haya obstáculo donde colisionar o que la luz solar provoque interferencias.

Como criterio de segmentación se ha aplicado el de distancia de puntos consecutivos. Si entre dos puntos consecutivos existe una distancia euclidiana mayor de un umbral determinado, se supondrá que estos puntos pertenecen a segmentos distintos. En caso contrario, serán del mismo segmento. Para este trabajo se ha usado una distancia de umbral de 0'3 metros.

Por último, aquellos segmentos con menos de 4 puntos se supone que son debido a un objeto con radio excesivamente pequeño, o una persona suficientemente lejos como para no tenerla en cuenta, por lo que estos segmentos se consideran no personas.

Aunque esto sería suficiente para detección en interiores, cuando se usa este sensor en exteriores se tiene que lidiar con problemas asociados a la luz solar. Esto es debido a que mientras que la luz artificial se concentra en frecuencias de luz visible, la luz solar abarca todo el espectro. Como el haz de luz del láser tiene una frecuencia de luz no visible, la luz artificial no interfiere mientras que la luz solar sí. Estos problemas son las ya comentadas *medidas fuera de rango* y las medidas falsas dentro de rango. Estas segundas son más

perjudiciales porque no se pueden eliminar fácilmente.

La solución propuesta en este trabajo ha sido volver a comprobar las distancias entre el punto final de un segmento y el punto inicial de los 4 segmentos posteriores, aplicando el mismo umbral. Si la distancia es menor, se fusionan los segmentos. Al hacer esto, solucionamos el problema de obtener una multisegmentación para una misma pared o una pierna.

3.3. Entradas del algoritmo

Una vez segmentado, hay que extraer características de cada segmento para usarlas como entradas del algoritmo de clasificación. Estas características se seleccionan pensando en variables que pueden diferenciar un segmento producido por una pierna de uno que no.

3.3.1. Detección en Interiores

Para hacer clasificación en interiores se suele usar las características propuestas por [1]. Son 13 variables estáticas y una dinámica, pero para este trabajo solo se usarán las variables estáticas, ya que en el propio documento concluye con que esta última característica no es relevante.

Estas características son:

1. Número de puntos del segmento.

$$n = |S_i| \tag{3.1}$$

2. Desviación típica del centroide. Siendo $\bar{x}$ el centro de gravedad:

$$\sigma = \sqrt{\frac{1}{n-1}\sum_j \|x_j - \bar{x}\|^2} \tag{3.2}$$

3. Desviación media de la mediana. Definiendo la mediana como:

$$\tilde{x} = \begin{cases} x_{(n+1)/2} & \text{si n es impar} \\ \frac{1}{2}(x_{n/2} + x_{(n+1)/2}) & \text{si n es par} \end{cases} \tag{3.3}$$

La desviación media de la mediana es:

$$\zeta = \frac{1}{n}\sum_j \|x_j - \tilde{x}\| \tag{3.4}$$

4. Distancia al segmento precedente. Distancia entre el primer punto de este segmento y el último punto del segmento anterior.

5. Distancia al segmento siguiente. Distancia entre el último punto del segmento y el primer punto del siguiente.

6. Anchura. Distancia entre el primer y el último punto del segmento.

7. Linealidad. Calculada la recta que mejor se aproxima al segmento por mínimos cuadrados, la linealidad se define como el media del error cuadrático entre los puntos del segmento y la recta aproximada.

8. Circularidad. Se puede calcular la circunferencia que mejor se aproxima al segmento definiendo $x = (x_c, y_c, x_c^2 + y_c^2 - r_c^2)^T$ siendo x_c, y_c el centro de la circunferencia y r_c el radio:

$$A = \begin{pmatrix} -2x_1 & -2y_1 & 1 \\ -2x_2 & -2y_2 & 1 \\ \vdots & \vdots & \vdots \\ -2x_n & -2y_n & 1 \end{pmatrix} \tag{3.5}$$

$$b = \begin{pmatrix} -x_1^2 - y_1^2 \\ -x_2^2 - y_2^2 \\ \vdots \\ -x_n^2 - y_n^2 \end{pmatrix} \tag{3.6}$$

$$A * x = b \tag{3.7}$$

Despejando de (3.8) con (3.5) y (3.6):

$$x = (A^T A)^{-1} A^T * b \tag{3.8}$$

Se define la circularidad como la media del error cuadrático:

$$s_c = \frac{\sum_{i=1}^{n} (r_c - \sqrt{(x_c - x_i)^2 + (y_c - y_i)^2})^2}{n} \tag{3.9}$$

9. Radio. Anteriormente calculado como r_c

10. Longitud del contorno. Suma de la distancia entre los puntos consecutivos de un segmento:

$$l = \sum_j d_{j,j-1} \tag{3.10}$$

11. Desviación típica del contorno.

12. Curvatura. Dados 3 puntos consecutivos x_A, x_B y x_C, definimos A como el área del triángulo $x_A x_B x_C$ y d_A, d_B, d_C como la distancia entre los puntos, la curvatura de esos tres puntos se calcula como:

$$\overline{k} = \frac{4A}{d_A d_B d_C} \tag{3.11}$$

La curvatura del segmento es la suma de curvaturas del segmento:

$$\overline{k} = \sum \widehat{k} \tag{3.12}$$

13. Diferencia media del ángulo del segmento. Es una característica que mide la concavidad/convexidad del segmento. Para ello, se calcula la media de los ángulos formados por cada tres puntos consecutivos.

3.3.2. Detección en Interiores y Exteriores

A diferencia de la detección hecha exclusivamente en interiores, en exteriores se presentan unas condiciones geométricas completamente distintas. Por ejemplo, en exteriores la mayoría de las veces los puntos siguientes y anteriores a una persona puede que estén fuera de rango, limitando las características de *Distancia al segmento siguiente* y *Distancia al segmento precedente* de [1].

En [3], se propone eliminar todas las características geométricas que no dependan exclusivamente del propio segmento. Además, debido a que en exteriores las personas pueden estar mucho más lejos del láser (por lo que son detectadas como menos puntos), también se propone aplicar una normalización de los parámetros usando el número de puntos y la distancia al origen.

Se usan 63 características, las cuales son:

1. Número de puntos.
2. Desviación típica del centroide.
3. Desviación media de la mediana.
4. Anchura.
5. Linealidad.
6. Circularidad.
7. Radio.
8. Longitud del contorno.
9. Desviación típica del contorno.
10. Curvatura.
11. Ángulo del segmento.
12. Curtosis.
13. Ratio de aspecto.

$14 - 26$. Las 13 primeras características dividida por la distancia al origen.

$27 - 39$. Las 13 primeras características multiplicadas por la distancia al origen.

$40 - 51$. Las características 2 a la 13 dividida por el número de puntos.

$52 - 63$. Las características 2 a la 13 multiplicadas por el número de puntos.

3.4. Entrenamiento

Para entrenar el algoritmo de clasificación, se ha visto la necesidad de hacer un set de datos que reuniese información tanto de segmento de piernas de personas como de segmentos sin piernas.

Para ello se ha grabado un bag frente a una pared situada a una distancia conocida, mientras cruzaban personas entre el láser y la propia pared. Posteriormente, se ha trasladado el láser a un entorno sin personas y se ha grabado otro bag. Sabiendo que del primer bag, era persona todo lo que estuviese a menos de la distancia de la pared y que en el segundo bag no había personas, se extrae de cada uno de los segmentos todas las características y se guardan con el formato: P o N según si es persona o no, y seguido del resto de valores de las características separados por una ",".

El algoritmo de Adaboost se entrena de la siguiente manera:

- Teniendo un set de ejemplos $(e_1, l1), \ldots, (e_N, l_N)$ y siendo a el número de muestras positivas $(l_n = +1)$ y b el número de negativas $(l_n = -1)$, se inicializan los pesos tal que:

  - Para muestras positivas

$$D_1(n) = \frac{1}{2a} \tag{3.13}$$

  - Para muestras negativas

$$D_1(n) = \frac{1}{2b} \tag{3.14}$$

- Desde $t = 1$ hasta T, siendo T el número de **weak learners** que se quieren usar:

  - Se normalizan los pesos:

$$D_t(n) = \frac{D_t(n)}{\sum_{i=1}^{N} D_t(i)} \tag{3.15}$$

  - Para cada característica se entrena un clasificador débil o **weak learner** h_j usando los pesos D_t

  - Para cada h_j se calcula:

$$r_j = \sum_{n=1}^{N} D_t(n) l_n h_j(e_n) \tag{3.16}$$

    Donde

$$h_j(e_n) \in \{+1, -1\} \tag{3.17}$$

  - Elegir h_j que maximice $|r_j|$ y guardarlo como

$$(h_t, r_t) = (h_j, r_j) \tag{3.18}$$

  - Actualizar los pesos:

$$D_{t+1}(n) = D_t(n) exp(-a_t l_n h_t(e_n)) \tag{3.19}$$

    Donde

$$\alpha = \frac{1}{2} log(\frac{1 + r_t}{1 - r_t}) \tag{3.20}$$

Por último, los datos obtenidos se guardarán en una base de datos, para que se puedan usar estos valores sin tener que reentrenar el clasificador cada vez que se quiera usar.

3.5. Predicción

Una vez guardados los valores del clasificador, la predicción se hace de manera trivial aplicando la fórmula:

$$H(e) = sign(\sum_{t=1}^{T} \alpha_t h_t(e)) \tag{3.21}$$

Donde $H(e)$ es, para los valores de un experimento e, si la clasificación es positiva o no.

Viendo este último cálculo, podemos asegurar que AdaBoost es muy poco costoso computacionálmente hablando. Sin embargo, la elección de T va a suponer el compromiso entre precisión y computación. Cuantos más **weak learners** se usen, mejor será la clasificación, pero a cambio habrá que realizar n (siendo el número de segmentos) multiplicaciones y sumas más por cada nube de puntos en la que se quieran localizar nuevas personas.

4. Paquete implementado

En este capítulo, se explicará la funcionalidad de cada nodo implementado en el paquete para la predicción de personas mediante láser. En cada sección se comentará el código de cada nodo haciendo referencia a los apéndices, donde se encuentran los textos completos. Estas referencias se harán siempre a los programas usados en interiores, ya que, debido al paralelismo entre ambos programas, se diferencian en pocos detalles. Sin embargo, cuando estas existan se especificará.

Las diferencias principales que se ha realizado en este trabajo para cada uno de los dos programas son las características del segmento que se usen para entrenar y predecir. Las técnicas para añadir robustez a la segmentación se han aplicado en ambos casos, tanto para interiores como para exteriores, porque se considera que una mala segmentación imposibilita una futura predicción.

Por ello, aunque constantemente se realice esta distinción, ambos programas se pueden usar tanto en interiores como en exteriores, solo que, a priori, no están pensados para ello.

4.1. Programas

4.1.1. Creación de base de datos

Versiones del nodo: interiores ("$laser\_features\_extractor\_indoor.cpp$") y exteriores ("$laser\_features\_extractor\_outdoor.cpp$"). El código completo se encuentra en los apéndices A y B, respectivamente. En este programa se busca crear una base de datos que segmente las muestras en personas o no personas, seguidas de cada una de las características propuestas.

Para ello, el nodo tiene un parámetro con el cual se puede seleccionar a partir de qué distancia los segmentos no son personas. El objetivo de esto

es grabar una primera vez frente a una pared mientras pasan personas por delante, siendo el parámetro distancia un poco menor que la distancia a la pared, para almacenar muestras positivas, y una segunda vez con el parámetro distancia negativo en un entorno no poblado, para aumentar esa primera base de datos.

Como modificación posterior, la base de datos tiene que empezar con una muestra positiva. Esto se debe a que la librería de AdaBoost se ha programado de manera genérica, por lo que la división realizada en este trabajo entre "P" y "N" podría ser modificada por otros dos caracteres. Sin embargo, se ha tomado por convenio que la primera muestra será de persona, por lo que si esto no se cumple, las futuras predicciones se darán invertidas.

Código: Apéndice A líneas 1-16

En las primeras líneas se definen las librerías que se van a usar. "*cstdlib*", "*iostream*" y "*vector*" son librerías propias de C/C++ que añaden funciones como el trabajar con ficheros o vectores dinámicos.

"*ros/ros.h*" incluye las funciones propias de ROS, necesarias para crear un nodo, un publicador o un suscriptor.

"*sensor_msgs/LaserScan.h*" añade el tipo de variable "*LaserScan*", el cual es el usado por "*hokuyo_node*" para mandar la información capturada por el sensor láser.

Por último, "*laser_features.hpp*" (H) es una librería propia donde se han implementado todas las funciones relacionadas con el cálculo y manejo de las características de nubes de puntos. El objetivo de esto ha sido aportar limpieza al texto y rapidez a la hora de añadir o eliminar futuras características al algoritmo.

Luego, siguen unas variables globales asociadas a los parámetros del nodo. Las dos primeras tienen relación con la distancia de segmentación. Si dos puntos consecutivos superan la distancia "*g_segmentTh*" significará que son parte de segmentos diferentes.

"*g_distPerson*" es la distancia a la que están las personas. A menor distancia de ese valor serán personas y, en caso contrario, no lo serán.

"*g_minRange*" y "*g_maxRange*" son la distancia máxima y mínima a la que puede medir el sensor. Fuera de ese rango las medidas son errores.

"*g_minBeams*" es el número mínimo de puntos que debe tener un segmento para poderlo considerar como posible persona. Se considera que un

segmento con bajo número de puntos se debe a que es muy estrecho o a que esta lejos. En cualquiera de los dos casos, suponemos que no es persona.

"g_pFile" será el archivo donde se guardará la base de datos. Si no existe, se creará el archivo. En caso contrario, se continuará.

Por último, "*using namespace std*" sirve para ahorrar escribir "$std::$" en todas las funciones que lo requieran.

Código: Apéndice A líneas 18-23

"*scanCallback*" es la función principal de este nodo. Cada vez que se reciba una nueva nube de puntos ("$sensor\_msgs::LaserScan$"), se realizará la función completa.

"$LaserFeatures$", clase programada en la librería "$laser\_features.hpp$" (H), la cual tiene asociadas todos los campos y funciones necesarias para cumplir la función de la librería, comentada anteriormente.

"$features$" es un vector de esta clase, la cual servirá para almacenar la información de cada segmento.

En los vectores "px" , "py" , "$angles$" y "$ranges$" se almacenará la información proveniente del láser para ser manejado con mayor comodidad. "px" y "py" serán las componentes xy del vector de puntos, y "$angles$" y "$ranges$" son el ángulo y radio a la que se encuentran los puntos. En ambos casos, se toma como referencia la posición del láser como origen de referencia. En ambos casos se tiene la misma información solo que haciendo uso de transformaciones diferentes.

"i" y "j" son variables auxiliares, y "$isPerson$" es una variable que almacenará "P" o "N" dependiendo de si el segmento es persona o no.

Código: Apéndice A líneas 26-35

Para cada punto de la nube de puntos obtenida por el láser, se filtra entre la distancia máxima y la mínima las medidas para eliminar posibles errores de medidas fuera de rango, y se les aplican las transformaciones comentadas

anteriormente para obtener "px", "py", "$angles$" y "$ranges$".

Estas transformaciones son:

$$ranges = rango_{laser} \quad (4.1)$$
$$angles = \text{ángulo}_{min} + \Delta\text{ángulo}_{laser} \quad (4.2)$$
$$px = rango_{laser} + \cos(angles) \quad (4.3)$$
$$py = rango_{laser} + \sin(angles) \quad (4.4)$$

Código: Apéndice A líneas 39-51

Se genera una variable "$LaserFeatures$" y se inicializa el primer punto del segmento a 0, lo que indica que el primer segmento empieza en el punto 0 de la nube de puntos. Después se entra en un bucle "for" que recorre todos los puntos de la nube calculando la distancia del punto i con su anterior. En el caso de que la distancia sea mayor que la distancia de segmentación, guarda el punto anterior como el último del segmento actual, almacena el segmento en el vector "$features$" y guarda el punto actual como primero del siguiente segmento. Cuando acaba el bucle, se guarda el punto como último del último segmento y se almacena en "$features$".

La comprobación de la distancia se realiza elevando al cuadrado los dos términos de la inecuación. Ya que la operación "raíz cuadrada" es muy costosa y hay que realizarla continuamente, se opta por elevar al cuadrado "$g\_segmentTh$" una vez al inicio del programa y calcular el cuadrado de la distancia entre dos puntos, ya que:

$$g\_segmentTh^2 \equiv g\_segmentTh2 \quad (4.5)$$
$$g\_segmentTh < \sqrt{(px_i - px_{i-1})^2 + (py_i - py_{i-1})^2} \quad (4.6)$$
$$g\_segmentTh2 < (px_i - px_{i-1})^2 + (py_i - py_{i-1})^2 \quad (4.7)$$

Código: Apéndice A líneas 53-65

Para cada segmento detectado se comprueba si la distancia entre su punto final y el punto inicial de los cuatro segmentos siguientes es menor de la distancia "*g_segmentTh*". En principio, se ha de suponer que este paso no sirve, ya que es una comprobación que se presupone realizada en el paso anterior, pero debido a que puede haber falsas medidas que se encuentren dentro del rango del sensor, puede que una mala medida provoque que un segmento real sea etiquetado como varios. Si se cumple esta condición, se unen los segmentos y se borran todos los intermedios.

En el bucle principal se ha utilizado una cuenta regresiva para evitar los problemas a la hora de borrar segmentos intermedios. Si no se hubiese tomado esta consideración, en el caso de que se hubiese borrado un solo segmento, al final del bucle se hubiese intentado acceder a una posición que ya no existe, por lo que hubiese dado un error en el programa.

Interiores/ Código: Apéndice A líneas 67-87

Todas las acciones se harán para cada uno de los segmentos.

Primero, se indica el segmento que es y el número de segmentos que hay en esa nube de puntos. Se calcula el número de puntos restando el punto final del segmento al inicial. Si no se superan el número de puntos impuesto por "*g_minBeams*", se desecha el segmento.

Después, se extraen cada una de las características propuestas en [1] para interiores. A cada función se le dan los vectores "*px*" y "*py*" para realizar los cálculos correspondientes.

En la función "*Matrix_operations*" se realizan las cuentas matriciales necesarias para calcular la linealidad y la circularidad.

Interiores/ Código: Apéndice A líneas 90-91

Con esta linea eliminamos los errores de cálculo que se hayan podido producir. En el caso de que se produzca algún error, se descarta el segmento.

Interiores/ Código: Apéndice A líneas 94-104

Se presupone que el segmento no pertenece a una persona a priori. Si su primer punto se encuentra a menos de "$g\_distPerson$" es porque es una persona y se la etiqueta con una "P". Por último, se escribe en el documento asociado a "g_pFile" la información extraída.

El primer carácter corresponde a si es persona o no, seguido de cada una de las características. Estas características deben ser de tipo float, por lo que hay que realizar la conversión de int a float en la primera característica.

Exteriores/ Código: Apéndice B líneas 67-88

A diferencia del caso anterior, se han eliminado las características relacionadas con distancia a objetos cercanos, como se propone en [3], y se ha añadido la extracción de la curtosis y el ratio de aspecto.

Se añade una linea con una función llamada "$extract\_Seg\_Norm$". Esta función traza una recta entre el primer y último punto del vector y calcula un nuevo vector, de tantas componentes como el primer parámetro de la función, con las distancias equiespaciadas y entre la recta trazada y los puntos del contorno del segmento. Además, guarda las distancias máximas hacia ambos lados de la recta. El número 13 se debe a que se ha considerado que 13 puntos son suficientes para poder aproximar casi cualquier superficie que exista tanto en interiores como en exteriores. Esta función es necesaria para el cálculo del ratio de aspecto.

Exteriores/ Código: Apéndice B líneas 95-137

Al igual que en interiores, se vuelve a etiquetar el segmento como persona o no, basándose en la distancia "$g\_distPerson$".

Posteriormente, se escribe en el archivo el carácter "P" o "N" seguido por cada una de las características tipo float. En este caso, hay 63 características diferenciadas por:

- Las 13 características básicas.
- Las 13 multiplicadas por la distancia al origen.
- Las 13 divididas entre la distancia al origen.
- Las características 2-13 multiplicado por n.
- Las características 2-13 divido por n.

Por último, se añade un retorno de carro para separar las diferentes muestras.

Código: Apéndice A líneas 112-117

"*main*" es la función principal del programa que se ejecutará una vez. Se inicia el nodo de ROS "$laser\_features\_extractor\_indoor$", se declara el manejador del nodo "n" y un suscriptor "*predict_sub*" que se asocia al topic "$/scanfront$"· Cada vez que se recibe un mensaje se ejecuta la función "*scanCallback*".

Código: Apéndice A líneas 120-131

Se definen las variables globales como parámetros del nuevo manejador de nodo "*lnh*", a los cuales se les añade el valor por defecto. "$g\_segmentTh$" se inicia con $0'3$ m y se calcula "$g\_segmentTh2$".

"$g\_distPerson$" se inicializa a "-1" para crear una base de datos solo con muestras negativas. En el caso de que se quiera crear una base con muestras positivas, se varia este valor por la distancia en metros a la que se sitúa la pared.

"$g\_minRange$" y "$g\_maxRange$" tienen una distancia menor que el rango que el sensor puede medir, ya que se ha considerado que una persona a más de 15 metros no tiene sentido ser detectada y a menos de 10 cm esta tan cerca que la detección es imposible.

"$g\_minBeams$" o mínimo número de un segmento para ser detectado, se elige por defecto con un valor de 3 puntos.

Código: Apéndice A líneas 133-142

Se abre el archivo "$features.csv$", en el cual se van a guardar la base de datos con la cual se entrenará AdaBoost. Si el archivo no existe, lo crea,

en caso contrario, añade tras la última linea la información nueva.

Luego se ejecuta el bucle de ros hasta que se pulse "$Ctrl+C$" o se termine la ejecución del nodo de otra forma.

Por último, cierra el archivo y acaba el programa.

4.1.2. Entrenador

Código: Apéndice C

En la función "*main*" se declara la variable "*boost*" como una clase del tipo "*Adaboost*", implementada en la librería "*adaboost.hpp*".

Se lee la base de datos "$features.csv$" con la función "$read\_features$", y con la función "$weak\_learners$" se indica cuantos *clasificadores débiles* se calcularán. Se ha decidido el valor 20 como primera estimación para tener un número alto de clasificadores, y poder realizar posteriores pruebas variando este valor.

"$train$" entrena el algoritmo con los datos ya introducidos. Por último, se guarda la información del Adaboost ya calculada en el archivo "*boost.txt*" y se acaba el programa.

4.1.3. Matriz de confusión

Código: Apéndice D

La información de la base de datos "$features.csv$" se guarda en la variable "*boost*", y se carga la información del clasificador entrenado anteriormente.

La base de datos "$features.csv$" debe ser distinta de la base de datos que se haya usado para entrenar el algoritmo, para obtener medidas reales de las virtudes del clasificador.

Por último, se usa la función "$confusion\_matrix$" para calcular la matriz de confusión aplicando el algoritmo de predicción a la base de datos.

4.1.4. Predictor láser

Versiones del nodo: interiores y exteriores ("*predictor_laser_indoor.cpp*" y "*predictor_laser_outdoor.cpp*", E y F). Como el código de este nodo es muy parecido al usado para la creación de base de datos, se comentarán solo las partes que difieran.

Código: Apéndice E líneas 18-21

Se declara como variables globales la clase "*Adaboost*" llamada "*boosted*", un publicador llamado "*result_pub*" y una variable tipo "*LaserScan*" llamada "*scan_output_*".

Código: Apéndice E líneas 26-46

Dentro de la función "*scanCallback*" se comienza inicializando las variables que se van a usar. Después, se copia la variable "*scan_in*", donde se recibe la información del láser a "*scan_output_*", excepto las intensidades, las cuales se borran y se crea un vector vacío con el mismo tamaño que el campo "*ranges*" en "*scan_in*".

Luego, se revisa que los puntos están entre los rangos del sensor, y los que no, se descartan. Si están en el rango, se guarda el número del punto correspondiente en el vector "*ind*" y se aplican las transformaciones correspondientes para obtener "*ranges*", "*angles*", "*px*" y "*py*".

El objetivo del vector "*ind*" es mantener la relación entre los vectores transformados y la información láser inicial. Ya que se descartan puntos intermedios, el tamaño de estos vectores transformados será menor que el de "*scan_in*", por lo que si se quiere hacer la transformación inversa es necesario saber que puntos son validos y cuales no.

Código: Apéndice E línea 101

Después de extraer las características, se realiza la predicción usando la función "*predict*". La función "*get_vector_indoor*" devuelve un vector formado por el número "1" seguido de las características propuestas.

Para exteriores se usa la función "*get_vector_outdoor*".

Código: Apéndice E líneas 102-117

En el caso de que el segmento sea persona, guarda como "1" en el vector "*intensities*" de la variable "*scan_output_*" en las componentes correspondiente al punto real. En caso contrario, se guarda como "0".

Por último, en la función "*scanCallback*", se publica "*scan_output_*" .

Código: Apéndice E líneas 123-130

En la función "*main*", se inicia el nodo con su manejador, y se crea un suscriptor al tópico "*/scanfront*" y un publicador que envía mensajes tipo "*LaserScan*" al tópico "*/result*".
Después, carga los datos del clasificador de "*boost.txt*" y elige cuantos clasificadores débiles se van a usar.

4.2. Librerías

En este trabajo se han desarrollado 2 librerías. Debido a su extensión, se explicará el código comentándolo directamente, en vez de hacer un primer apartado con el texto completo. Con esto se busca añadir claridad al texto y facilitar la búsqueda de funciones en la documentación.

4.2.1. Librería adaboost.hpp

Clase Adaboost

Código: Apéndice G líneas 13-43

La clase "*Adaboost*" esta formada por variables privadas, las cuales solo se pueden acceder desde dentro de la clase, y por variables públicas a las que se pueden llamar desde programas exteriores.

Las características privadas son:

- "*weak_learner*" es el número de clasificadores débiles que se usarán.

- "*datos*" guarda la información de la base de datos cargada. Esta formada por un vector de las muestras, las cuales se definen, a su vez, como un vector indicando si es positiva o negativa la muestra seguida de sus características separado por ", ".

- "*H*" es un vector de la estructura "*h_j*". Estas estructuras parametrizan a un clasificador débil mediante los campos: "*signo*" (indica el sentido de la frontera de clasificación), "$features$" (la variable que se esta tomando en cuenta), "*wall*" (valor en el que se sitúa la frontera de clasificación), "*r*" (ratio de aciertos entre 0 y 1) y "$alfa$" (Peso asociado a ese clasificador débil en el clasificador fuerte).

- "*RP*", "*FP*", "*RN*", "*FN*" corresponde a real y falso positivo, y real y falso negativo respectivamente. Normalmente se suele presentar en el siguiente formato: 4.1. "*Clase actual*" es la clase a la que pertenece cada muestra y "*Predicción*" como es clasificada esas muestras.

		Predicción	
Población total		Real	Falso
Clase actual	Real	RP	FN
	Falso	FP	RN

Cuadro 4.1: Matriz de confusión

- "TPR" y "FPR" es el ratio de verdaderos y falsos positivos respectivamente.

- "$f\_weak\_learner$" es una función que devuelve el mejor clasificador débil, usando como parámetros el vector de los pesos de las muestras y la característica que se quiere clasificar.

Las funciones públicas son:

- "*weak_learners*" actualiza el valor de "*weak_learner*".
- "*read_features*" carga la base de datos de muestras de experimentos.
- "*train*" entrena el clasificador.
- "*save*" guarda la información del clasificador ya entrenado en un archivo con el nombre que se le pase como parámetro.
- "*load*" carga la información del clasificador ya entrenado del archivo con el nombre que se le pase como parámetro.
- "*predict*" predice usando el clasificador valorando las características pasadas por el vector.
- "*confusion_matrix*" calcula la matriz de confusión (4.1).

Función weak_learners

Código: Apéndice G líneas 45-48

Se guarda el valor en "*weak_learner*".

Función read_features

Código: Apéndice G líneas 50-117

Primero, se abre el archivo donde se guarda la base de datos, cuyo nombre ha sido pasado como parámetro. Si no se puede, acaba la función. En caso contrario, borra los datos guardados anteriormente en "*datos*" y comienza a leer el archivo.

Entra en un bucle while que tiene como condición no haber acabado el archivo. Se lee una línea completa y se utiliza el primer carácter para: discernir qué dos caracteres van a ser los representativos de muestras positivas y negativas; clasificar las muestras en positivas (1), negativas (-1), nulas (no hace nada) o otro carácter (Aparece un error y acaba la función). Esta clasificación se guarda en el vector "*segmento*" como primera componente.

Después, se separa la línea mediante las distintas ", ". Cada característica es leída usando "$strtof$" (función que se usa para leer números tipo float) y se apila en el vector segmento. Por último, se guarda el vector "$segmento$" en "$datos$".

Cuando se sale de los bucles, se cierra el archivo de la base de datos y se acaba la función.

Función f_weak_learner

Código: Apéndice G líneas 119-206

Los parámetros de entrada de esta función son "D", que son los pesos de cada muestra de la base de datos cargada y "$feature$", que es la característica en la cual nos estamos centrando. Devuelve de clasificador débil de la característica indicada para los pesos actuales.

Se guarda en "D_size" el número de muestras que había en la base de datos cargada y se redimensionan los vectores "$datos\_ord$", "D_ord" y "l_ord" con este tamaño.

Para cada punto, se guarda en su respectiva componente de "l_ord" si es una muestra positiva o negativa, en "D_ord" el peso de esa muestra y en "$datos\_ord$" el valor de la característica elegida como parámetro.

Después, se ordenan estos tres vectores de menor a mayor atendiendo al valor de "$datos\_ord$". Una vez ordenados, para cada muestra, exceptuando la última, se buscará cual es la mejor frontera de clasificación.

Para ello, se comprueba si la muestra actual y la siguiente son ambas positivas o negativas. Si se da esta condición, se supone que la frontera de decisión no se va a situar entre estos valores, ya que la clasificación es mejor cuanto mayor sea el peso que clasifique bien, por lo que colocarla entre dos valores positivos o negativos no tiene sentido. En caso contrario, se sitúa una frontera entre el valor actual y el siguiente de "$datos\_ord$". Para cada muestra, si es menor que esta frontera ficticia, se iguala "$pred$" a 1, y si no, a -1. Si este valor coincide con el que se le ha puesto en la base de datos, se suma el peso de esa muestra a la "r" del clasificador débil.

Como los valores de los pesos están normalizados, la suma de todos estos pesos será igual a la unidad, por lo que si "r" es menor de 0,5, es porque la

frontera esta al revés, por lo que se pone el signo en negativo ("$signo$" $= -1$) y se aplica $r = 1 - r$. Si no, se iguala "$signo$" $= 1$.

Si el clasificador débil propuesto tiene mayor "r" que el mejor clasificador propuesto hasta entonces, este será el nuevo clasificador débil mejor.

Cuando acaba con todas las posibles fronteras, guarda la característica que ha sido usada y devuelve el mejor clasificador débil.

Función train

Código: Apéndice G líneas 208-308

Se empieza borrando el clasificador fuerte que hubiese entrenado (si es que lo había) y guardando el número de muestras en "D_size". Se cuenta cuantas muestras positivas y negativas hay en la base de datos y se reparten los pesos, 0,5 entre todas las muestras positivas y 0,5 entre todas las negativas. Además, se guarda en "$n\_features$" el número de características que se usan.

Para cada uno de los clasificadores débiles (guardado en "$weak\_learner$"):

- Se normalizan los pesos de las muestras, sumándolos todos y dividiendo cada peso por este valor total.
- Se entrena un clasificador débil para cada característica con esos pesos, llamando a la función "$f\_weak\_learner$", y lo guardamos en un vector de clasificadores.
- Se guarda el primer clasificador débil como la mejor variable, y luego, se comprueba para cada característica si su clasificador es mejor que los anteriores, y si es así lo guarda como el mejor de este "$weak\_learner$".
- Se calcula la α del mejor clasificador débil de todos.
- Se vuelven a repartir los pesos, haciendo una predicción por muestra con el último clasificador débil. Si la predicción es cierta, se le baja el peso, y si no, se sube aplicando la siguiente fórmula:

$$D_j = D_j * \exp(-\alpha * clase * prediccion) \tag{4.8}$$

si la clase y la predicción son iguales, la exponencial estará elevada a un número negativo, por lo que será menor que 1. En otro caso, este factor será mayor que 1, siendo el peso mayor.

Cuando se acabe el bucle, habrá clasificado los "*weak_learner*" clasificadores débiles del clasificador Adaboost.

Función predict

Código: Apéndice G líneas 310-344

En "*F*" se guarda la suma de las predicciones de los clasificadores débiles multiplicadas por su peso. Si "*F*" es positiva al final de la función, la muestra del vector "*f*" será clasificada como una muestra positiva, y en caso contrario, como una negativa.

Para cada "*weak_learner*", se comprueba que "*F*" tiene valores lógicos y se aplica la siguiente inecuación:

$$f_{caracteristica\ de\ i} * signo_i < frontera_i * signo_i \tag{4.9}$$

donde "*f*" es el vector que se quiere clasificar, "*i*" es el "*weak_learner*" que se esta calculando, y "*signo*", "*frontera*" y "*caracteristica*" corresponden a los campos de la estructura "*h_j*": "*signo*", "*wall*" y "*feature*". Si la inecuación se cumple significa que ese clasificador débil lo considera una muestra positiva, por lo que se suma el α del clasificador a "*F*". En caso contrario, se le resta.

Por último, se mira si "*F*" es mayor que 0, haciendo que la función devuelva un "1", y si no, devuelve "-1".

Función save

Código: Apéndice G líneas 346-356

Se guarda en el archivo pasado por parámetro el clasificador entrenado. Para ello se abre el archivo en modo escritura, destruyendo lo que hubiese anteriormente, y se escribe por filas cada uno de los clasificadores débiles como: la característica que usa ("*features*"), la frontera de clasificación ("*wall*"), el signo ("*signo*") y la α ("*alfa*").

Por último, cierra el archivo y se acaba la función.

Función load

Código: Apéndice G líneas 358-384

Se abre el archivo pasado por parámetro como lectura y se limpian los pesos del clasificador que estuviese guardado, y si no se puede abrir el archivo, se sale de la función.

Para cada línea del archivo, guarda los parámetros leídos y separados por ", " como la característica del clasificador débil "i", la frontera, el signo y el alfa, y se guarda en el vector "H", donde se guarda el clasificador fuerte entrenado.

Por último, se cierra el archivo y se termina la función.

Función confusion_matrix

Código: Apéndice G líneas 386-440

Usando la base de datos ya guardada y el clasificador ya entrenado, se calcula la matriz de confusión. Se aconseja calcular la matriz de confusión con otra base de datos distinta a la que se haya usado para entrenar.

Para cada muestra, se predice con el clasificador entrenado y se etiqueta como:

- Si la predicción y la muestra es positiva, se suma 1 a la categoría "$Real\ Positivo$".
- Si la predicción es positiva y la muestra negativa, se suma 1 a "$Falso\ Positivo$".
- Si la predicción es negativa y la muestra positiva, se suma 1 a "$Falso\ Negativo$".
- Si la predicción y la muestra son negativas, se suma 1 a "$Real\ Negativo$"

Finalmente, se muestra por pantalla la matriz (4.1) gráficamente junto con el número de muestras.

Se calculan otros parámetros como la "$Precisión$", "$Exactitud$", "$Sensibilidad$", "$Especificidad$", "TPR" y "FPR" aunque no se usen, para facilitar su acceso en el caso de que se necesiten en el futuro.

4.2.2. Librería laser_features.hpp

Clase LaserFeatures

Código: Apéndice H líneas 9-98

La clase "*LaserFeatures*" es la encargada de extraer las características de cada segmento.

Al igual que la clase "*Adaboost*", tiene miembros privados que solo se pueden acceder desde dentro de la clase, y miembros públicos que se pueden acceder desde dentro y fuera de la misma clase.

Los campos privados son:

- Las 15 características básicas de segmentos propuestas en este trabajo (13 características de interiores + Curtósis y Ratio de aspecto).
- "*segment*" guarda el número del segmento y "*scan_size*" el número de segmentos en total.
- Algunas variables relacionadas con el cálculo matricial.
- "*Long*" es la distancia máxima entre la recta que une los puntos extremos del segmento y el contorno.
- "*cx*" y "*cy*" son el centro de gravedad del segmento.

Mientras que los campos públicos son:

- "*first_point*" y "*last_point*" es el índice del primer y último punto del segmento en referencia a la nube de puntos completa.
- "*seg_norm*" es un vector normalizado que caracteriza el contorno del segmento mediante distancias en perpendicular a la recta que une primer y último punto del segmento.
- "*set_points*" es una función que permite introducir los valores de "*segment*" y "*scan_size*".
- Un conjunto de funciones llamadas "*extract_*" seguido del nombre de las características junto con "*Matrix_operations*", son funciones para calcular cada una de las características.

- Un conjunto de funciones llamadas "*get_*" seguido del nombre de las características, que devuelven el valor de esa característica.
- "*get_vector_indoor*" y "*get_vector_outdoor*" devuelve un vector formado por un 1 seguido de las características propuestas para interiores y exteriores respectivamente.
- Un conjunto de funciones llamadas "*set_*" seguido del nombre de las características, que fuerzan a que la característica tenga el valor pasado por parámetros.

Función set_points

Código: Apéndice H líneas 101-105

Con la función "*set_points*", se introduce los valores de "*segment*" y "*scan_size*".

Función extract_n

Código: Apéndice H líneas 107-110

Con la función "*extract_n*", se calcula el valor de n restando el indice del punto final al inicial y sumándole 1.

Función extract_stand_desv

Código: Apéndice H líneas 112-131

Primero se calcula el centro de gravedad sumando los puntos que componen el segmento y dividiéndolo entre n.

Con ello, se calcula la desviación estándar aplicando la fórmula 3.2

Función extract_madfm

Código: Apéndice H líneas 133-156

En esta función se extrae la desviación media de la mediana.
Para ello, se calcula el punto medio como 3.3:

- Si "n" es impar (resto igual a 1), la mediana de la posición es el punto con el indice igual al del primero más $\frac{n}{2}$, o lo que es lo mismo, n desplazado un bit a la derecha ($n >> 1$).
- Si "n" es par, la mediana de la posición es el punto intermedio entre los dos puntos intermedios del segmento.

Una vez con la mediana de la posición, la desviación media de la mediana se consigue sustituyendo en la fórmula 3.4.

Función extract_Jump_p_seg

Código: Apéndice H líneas 158-172

Esta función calcula la distancia al segmento anterior. Si es el primer segmento, la distancia es 0, mientras que si no, es la distancia del primer punto del segmento al punto anterior.

Función extract_Jump_s_seg

Código: Apéndice H líneas 174-189

Para calcular la distancia al segmento siguiente, se comprueba si es el último segmento. Si es así, la distancia es 0. En caso contrario, es la distancia entre el último punto del segmento y el siguiente.

Función extract_Width

Código: Apéndice H líneas 191-198

Para calcular el ancho del segmento, se calcula la distancia entre el primer y el último punto del mismo.

Función Matrix_operations

Código: Apéndice H líneas 200-273

La función "*Matrix_operations*" reúne los cálculos matriciales para calcular la linealidad, circularidad y radio del segmento.

Primero, se inicializan todos los valores a 0. Para cada punto del segmento se calculan los valores "x" e "y" como el sumatorio de las componentes "x" e "y" de los puntos del segmento respectivamente, "xy" como el

sumatorio de las componentes "x" multiplicadas por la "y", y "$x2$" e "$y2$" como el sumatorio de las componentes "x"^{2} e "y"^{2}.

Después, se calculan las matrices "A" y "B". Estas matrices no son las mismas que las definidas en 3.5 y 3.6. Si la fórmula de 3.8 es :

$$x = (A^T A)^{-1} A^T * b \tag{4.10}$$

En esta librería se ha definido:

$$A_{prog} = A^T * A \tag{4.11}$$
$$B_{prog} = A^T * b \tag{4.12}$$

Por lo que se han reducido las cuentas a hacer. Se invierte "A", calculando su inversa. Si el determinante de la matriz "A" es 0, es un punto singular y se dan unos valores del vector "C" por defecto. Si no, se calcula:

$$x \equiv C = A_{prog}^{-1} * B_{prog} \tag{4.13}$$

Función extract_Linearity

Código: Apéndice H líneas 275-286

Para calcular la linealidad, se ha calculado la recta que mejor se aproxima a el segmento como:

$$b = \frac{n * xy - x * y}{n * x2 - x * x} \tag{4.14}$$

$$a = \frac{x2 * y - x * xy}{n * x2 - x * x} \tag{4.15}$$

Se ha calculado la linealidad como el error cuadrático medio de aproximar el segmento a esta recta.

Función extract_Circularity

Código: Apéndice H líneas 288-303

Se calcula el radio despejándolo del vector "x". Con el radio y el punto del centro del segmento, se calcula la circularidad como el error cuadrático medio de aproximar el segmento por la circunferencia con esos valores.

Función extract_Boundary

Código: Apéndice H líneas 305-334

La distancia del contorno se calcula como la distancia entre puntos consecutivos sumados, desde el primer punto hasta el último del segmento. Luego, se calcula la mediana del contorno, y con ello se calcula la desviación típica del contorno.

Función extract_Curv

Código: Apéndice H líneas 336-360

La función "$extract\_Curv$" calcula la curvatura aplicando la fórmula 3.12. "$dist1$" es la distancia entre el punto actual y el anterior, "$dist2$" es la distancia entre ese y su anterior (o "$dist1$" en el punto anterior) y "$dist3$" entre el actual y dos anteriores.

"$Area$" es el área entre el punto actual y los dos anteriores, que se ha calculado aplicando la regla de Sarrus:

$$Area = \frac{1}{2} \begin{Vmatrix} p1_x & p1_y & 1 \\ p2_x & p2_y & 1 \\ p3_x & p3_y & 1 \end{Vmatrix} \tag{4.16}$$

Por último, se calcula la curvatura aplicando la fórmula para cada 3 puntos consecutivos dentro del segmento.

Función extract_Mean_ang_diff

Código: Apéndice H líneas 363-380

Para calcular la media de la diferencia de ángulos, primero se ha calculado el sumatorio de los ángulos y luego se ha dividido entre el número de ángulos sumados.

Cada ángulo se ha calculado a su vez como:

$$\cos\alpha = \frac{v_1 * v_2}{\sqrt{|v_1| * |v_2|}} \tag{4.17}$$

Siendo "v_1" el vector entre el punto anterior y su anterior, y "v_2" entre el anterior y el actual.

Función extract_Seg_Norm

Código: Apéndice H líneas 383-420

En esta función se calcula la variable global "*Long*", así como un vector formado por dos primeras componentes que hacen referencia a las distancias máximas y mínimas del contorno, seguido de "*num*" componentes, que son las distancias de la recta que une los dos puntos extremos al contorno normalizados con respecto a "*Long*".

Primero, se parametriza la recta que pasa por los puntos inicial y final del segmento mediante la ecuación general. Para "*num*" puntos, se calcula a que distancia por unidad tiene que estar el punto desde el cual se medirá la distancia al contorno, para que estos sean equidistantes. Se calcula el punto intermedio del contorno al que correspondería esa distancia, aproximando el contorno entre dos puntos consecutivos como rectas. Después, se guardan las distancias multiplicadas por un factor "*aux_div*" en un vector llamado "*seg*" y se calcula la máxima distancia y la mínima (o mejor dicho, máxima hacia el otro lado de la recta). Eliminamos el factor por el que estaban multiplicadas de estas distancias y se guardan en el vector "*seg_norm*" como sus dos primeras componentes.

"*Long*" se calcula como la distancia absoluta entre el máximo y el mínimo del segmento. Por último, se normaliza todo el vector guardado en "*seg*" y se pone en "*seg_norm*".

Función extract_Kurtosis

Código: Apéndice H líneas 422-433

La función para calcular la curtosis se hace aplicando la fórmula:

$$Kurtosis(X) = \frac{\sum_{i=1}^{N} (x_i - \bar{x})^4}{(N-1)s^4} \tag{4.18}$$

Función extract_Aspect_ratio

Código: Apéndice H líneas 436-439

El ratio de aspecto se define como la proporción entre su ancho y su altura.

Función get_vector_indoor

Código: Apéndice H líneas 442-462

Esta función devuelve un vector formado por un 1 seguido de las 13 características propuestas en el paper [1] para interiores.

Función get_vector_outdoor

Código: Apéndice H líneas 464-543

Esta función devuelve un vector formado por un 1 seguido de las 63 características propuestas en el paper [3] para exteriores.

5. Resultados experimentales

5.1. Ejemplo práctico

Para realizar las pruebas al detector se han grabado, tanto en interiores como en exteriores, dos *bags* con personas y dos sin ellas, sumando un total de 8 *bags*. 4 *bags* (una de cada tipo) han sido usadas para hacer la base de datos de entrenamiento, mientras que para hacer la matriz de confusión se han separado entre interiores y exteriores para ver como se comportaba el detector en estos entornos.

Los resultados obtenidos han sido las matrices de confusión (4.1):

2770		Predicción	
		Real	Falso
Personas	Real	1290	168
	Falso	96	1216

Cuadro 5.1: 13 características en interiores

- La matriz de confusión 5.1 corresponde a bases de datos tomados en interiores con el clasificador propuesto por [1].

  Del total de segmentos correspondiente a personas, 1290 han sido clasificados correctamente, y 168 incorrectamente. De los segmentos de no

personas, 1216 han sido bien etiquetados, mientras que 96 no lo han sido.

La detección en interiores con 13 características es muy buena, ya que la probabilidad de error en la clasificación esta en torno al 10 %, tanto para personas como para no personas.

3921		Predicción	
		Real	Falso
Personas	Real	1532	1416
	Falso	77	896

Cuadro 5.2: 13 características en exteriores

- Usando el mismo clasificador, pero con "*bags*" grabadas en exteriores se ha obtenido 5.2.

  Las detecciones correctas de personas han sido 1532, mientras que las falsas detecciones de personas 1416. Los segmentos no correspondientes a personas han sido mal clasificados 77 veces, y bien clasificados 896 veces.

  Las conclusiones que se pueden extraer es que si el clasificador detecta una persona, hay una probabilidad del 87 % de que así sea, pero si dice que el segmento no corresponde a una, el ratio de acierto baja a un 65 %.

2915		Predicción	
		Real	Falso
Personas	Real	1436	165
	Falso	238	1076

Cuadro 5.3: 63 características en interiores

- Se entrena un clasificador de 63 características propuesto en [3] y se realiza la matriz de confusión en interiores (5.3).

  Se han detectado 1436 segmentos de personas correctamente, 165 incorrectamente, 1076 segmentos de no personas correctamente y 238 incorrectamente.

  Con este clasificador se vuelven a obtener resultados muy buenos, ya que las detecciones tanto de personas como de no personas superan el 85 % de aciertos.

4068		Predicción	
		Real	Falso
Personas	Real	1875	1217
	Falso	148	828

Cuadro 5.4: 63 características en exteriores

- Un clasificador con 63 variables con los *bags* de exteriores (5.4).

  Se ha clasificado correctamente a 1875 segmentos de personas y 828 de no personas. Se ha errado en la clasificación de 1217 segmentos de personas y 148 no personas.

  El ratio de acierto de una clasificación de personas es de 80 %, mientras que de no persona esta en torno al 68 %.

Se puede ver como los mejores resultados se obtienen en interiores con 13 características (5.1), pero que al usar el mismo detector en exteriores (5.2), el número de $Falsos\ Negativos$ se dispara.

Con el detector de 63 características, como era de esperar, en interiores (5.3) empeora el resultado con respecto a 13 características (5.1) dando más $Falsos\_Positivos$, mientras que en exteriores (5.4) reduce considerablemente el número de $Falsos\_Negativos$.

6. Conclusiones

6.1. Lista de conclusiones

Como conclusiones de este trabajo:

- Se ha recibido nubes de puntos de sensores de escáner láser.
- Se ha segmentado la nube de puntos correctamente en interiores.
- Se ha aumentado la robustez de la segmentación para que también se realizase de manera correcta en exteriores.
- Se ha implementado un nodo para extraer las características del clasificador propuesto para interiores y otro para exteriores.
- Se ha implementado un nodo para entrenar el clasificador con una base de datos dada.
- Se ha implementado un nodo para calcular la matriz de confusión.
- Se ha implementado el clasificador láser propuesto en [1].
- Se ha implementado el clasificador láser propuesto en [3].
- Se ha creado una librería propia que implementa las funciones necesarias para poder entrenar y usar el algoritmo de clasificación "*AdaBoost*".
- Se ha creado una librería propia que reúne todos los cálculos de características propuestas en [1] y [3].
- Se han grabado "*bags*" de datos en interiores y exteriores con el sensor láser.
- Se han realizado las pruebas necesarias para validar el buen funcionamiento de los clasificadores.

6.2. Futuro trabajo

Como mejoras para un futuro trabajo se propone:

- Buscar nuevas características en interiores y exteriores para mejorar la detección de personas.
- Añadir un filtro de Kalman para hacer "*tracking*" o seguimiento de las personas detectadas para reducir $FalsosPositivos$ y $FalsosNegativos$.
- Añadir un sensor para diferenciar el entorno entre interiores y exteriores, y aplicar el algoritmo de detección selectivamente.
- Añadir una cámara con la que detectar personas por vídeo y fusionar ambas detecciones.

Bibliografía

[1] K. O. Arras, Ó. M. Mozos, and W. Burgard, "Using boosted features for the detection of people in 2D range data," *Proceedings - IEEE International Conference on Robotics and Automation*, pp. 3402–3407, 2007.

[2] a. Carballo, a. Ohya, and S. Yuta, "People detection using range and intensity data from multi-layered Laser Range Finders," *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5849–5854, 2010. [Online]. Available: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=5649769

[3] E. P. Fotiadis, M. Garzón, and A. Barrientos, "Human detection from a mobile robot using fusion of laser and vision information." *Sensors (Basel, Switzerland)*, vol. 13, no. 9, pp. 11 603–11 635, 2013.

[4] S. Gidel, P. Checchin, C. Blanc, T. Chateau, L. Trassoudaine, and U. B. Pascal, "Conf. Pedestrian Detection Method using a Multilayer Laserscanner; Application in Urban Environment, 2008.pdf."

[5] L. E. Navarro-Serment, C. Mertz, N. Vandapel, and M. Hebert, "LADAR-based Pedestrian Detection and Tracking," *Proc. 1st. Workshop on Human Detection from Mobile Robot Platforms, IEEE ICRA 2008*, 2008.

[6] L. E. Navarro-Serment, C. Mertz, and M. Hebert, "Pedestrian Detection and Tracking Using Three-dimensional LADAR Data," *Ijrr*, vol. 29, pp. 1516–1528, 2010. [Online]. Available: http://ijr.sagepub.com/cgi/doi/10.1177/0278364910370216

[7] C. Premebida, J. Batista, and U. Nunes, "Pedestrian Detection Combining RGB and Dense LIDAR Data."

Apéndices

A. Apéndice I

Script	laser_features_extractor_indoor.cpp
Descripción	Saca las 13 características de interiores de cada segmento de nubes de puntos y las guardan en una base de datos.
Entradas	Tópico tipo "*LaserScan*": "*/scanfront*"
Salidas	Crea un archivo llamado "*features.csv*" donde se guarda la base de datos.

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "laser_features.hpp"

double g_segmentTh;
double g_segmentTh2;
double g_distPerson;
double g_minRange;
double g_maxRange;
int g_minBeams;
FILE *g_pFile;

using namespace std;

```

```
18 void scanCallback (const sensor_msgs::LaserScan::
      ConstPtr& scan_in)
19 {
20         vector<float> px, py, angles, ranges;
21         vector<LaserFeatures> features;
22         unsigned int i, j;
23         char isPerson;
24 
25         // Filter out laser information balow and over
              the range thresholds
26         for(i=0;i<scan_in->ranges.size();i++)
27         {
28                 if(scan_in->ranges[i] > g_minRange &&
                      scan_in->ranges[i] < g_maxRange)
29                 {
30                         ranges.push_back(scan_in->ranges
                              [i]);
31                         angles.push_back(scan_in->
                              angle_min + i * scan_in->
                              angle_increment);
32                         px.push_back(ranges[(int)ranges.
                              size()-1]*cos(angles[(int)
                              ranges.size()-1]));
33                         py.push_back(ranges[(int)ranges.
                              size()-1]*sin(angles[(int)
                              ranges.size()-1]));
34 
35                 }
36         }
37 
38         // Divide laser scan into segments separated
              more than g_segmentTh meters
39         LaserFeatures f;
40         f.first_point = 0;
41         for(i=1;i<ranges.size();i++)
42         {
43                 if((px[i-1]-px[i])*(px[i-1]-px[i]) + (py
                      [i-1]-py[i])*(py[i-1]-py[i]) >
                      g_segmentTh2)
44                 {
45                         f.last_point = i-1;
46                         features.push_back(f);
47                         f.first_point = f.last_point +
```

```
                                1;
48              }
49          }
50          f.last_point = i-1;
51          features.push_back(f);
52
53          for(i=features.size()-2; i > -1 ;i--)
54          {
55
56              for( j = i+1; j<i+5 && j<features.size()
                    ; j++)
57              {
58                  if((px[features[i].last_point]-
                       px[features[j].first_point])
                       *(px[features[i].last_point]-
                       px[features[j].first_point])
                       + (py[features[i].last_point
                       ]-py[features[j].first_point
                       ])*(py[features[i].last_point
                       ]-py[features[j].first_point
                       ]) < g_segmentTh2)
59                  {
60                      features[i].last_point =
                            features[j].
                           last_point;
61
62                      features.erase(features.
                           begin()+i+1,features.
                           begin()+j);
63                  }
64              }
65          }
66          // Extract features of each laser segment save
               on file
67          for(i=0;i<features.size();i++)
68          {
69              features[i].set_points( i, features.size
                   ());
70              features[i].extract_n();
71
72              // Remove too small segments
73              if(features[i].get_n() < g_minBeams)
74                  continue;
```

```
75
76                  // Extract the rest of features
77                  features[i].extract_stand_desv( px, py);
78                  features[i].extract_madfm( px, py);
79                  features[i].extract_Jump_p_seg( px, py);
80                  features[i].extract_Jump_s_seg( px, py);
81                  features[i].extract_Width( px, py);
82                  features[i].Matrix_operations( px, py);
83                  features[i].extract_Linearity( px, py);
84                  features[i].extract_Circularity( px, py)
                        ;
85                  features[i].extract_Boundary( px, py);
86                  features[i].extract_Curv( px, py);
87                  features[i].extract_Mean_ang_diff( px,
                        py);
88
89                  // Remove non-sense features
90                  if( fabs( features[i].get_madfm() ) >
                        100000 || fabs( features[i].
                        get_stand_desv() ) > 100000)
91                        continue;
92
93                  // Save features on file
94                  isPerson = 'N';
95                  if( px[ features[i].first_point ] <
                        g_distPerson )
96                        isPerson = 'P';
97                  fprintf (g_pFile, "%c,%f,%f,%f,%f,%f,%f
                        ,%f,%f,%f,%f,%f,%f,%f", isPerson,
                              \
98                        (float) features[i].get_n(),
                              features[i].get_stand_desv(),
                               features[i].get_madfm(),
                                    \
99                        features[i].get_Jump_p_seg(),
                              features[i].get_Jump_s_seg(),
                               features[i].get_Width(),
                                    \
100                       features[i].get_Linearity(),
```

```
                                features[i].get_Circularity()
                                , features[i].get_Radius(),

                                                \
101                             features[i].get_Boundary_length
                                ()), features[i].
                                get_Boundary_reg(),


                                                                \
102                             features[i].get_Curv(), features
                                [i].get_Mean_ang_diff() );
103
104                     fprintf (g_pFile, "\n");
105
106
107             }
108
109     }
110
111
112     int main(int argc, char** argv)
113     {
114             // Setup ROS
115             ros::init(argc, argv, "
                    laser_features_extractor_indoor");
116             ros::NodeHandle n;
117             ros::Subscriber predict_sub = n.subscribe("/
                    scanfront", 1, scanCallback);
118
119             // Read node parameters
120             ros::NodeHandle lnh("~");
121             if(!lnh.getParam("segment_threshold",
                    g_segmentTh))
122                     g_segmentTh = 0.3;
123             g_segmentTh2 = g_segmentTh*g_segmentTh;
124             if(!lnh.getParam("min_distance_person",
                    g_distPerson))
125                     g_distPerson = -1.0;
126             if(!lnh.getParam("min_detection_range",
                    g_minRange))
127                     g_minRange = 0.1;
128             if(!lnh.getParam("max_detection_range",
```

```
            g_maxRange))
129             g_maxRange = 15.0;
130         if(!lnh.getParam("min_segment_beams", g_minBeams
            ))
131             g_minBeams = 3;
132
133         // Open file to save laser features
134         g_pFile = fopen ("features.csv", "a");
135
136         // Spin for ever
137         ros::spin();
138
139         // Close features database
140         fclose (g_pFile);
141
142         return 0;
143 }
```

B. Apéndice II

Script	laser_features_extractor_outdoor.cpp
Descripción	Saca las 63 características de exteriores de cada segmento de nubes de puntos y las guardan en una base de datos.
Entradas	Tópico tipo "*LaserScan*": "*/scanfront*"
Salidas	Crea un archivo llamado "*features.csv*" donde se guarda la base de datos.

```
#include <cstdlib>
#include <iostream>
#include <vector>
#include "ros/ros.h"
#include "sensor_msgs/LaserScan.h"
#include "laser_features.hpp"

double g_segmentTh;
double g_segmentTh2;
double g_distPerson;
double g_minRange;
double g_maxRange;
int g_minBeams;
FILE *g_pFile;

using namespace std;
```

```
17
18  void scanCallback (const sensor_msgs::LaserScan::
        ConstPtr& scan_in)
19  {
20          vector<float> px, py, angles, ranges;
21          vector<LaserFeatures> features;
22          unsigned int i, j;
23          char isPerson;
24
25          // Filter out laser information balow and over
               the range thresholds
26          for(i=0;i<scan_in->ranges.size();i++)
27          {
28                  if(scan_in->ranges[i] > g_minRange &&
                       scan_in->ranges[i] < g_maxRange)
29                  {
30                          ranges.push_back(scan_in->ranges
                               [i]);
31                          angles.push_back(scan_in->
                               angle_min + i * scan_in->
                               angle_increment);
32                          px.push_back(ranges[(int)ranges.
                               size()-1]*cos(angles[(int)
                               ranges.size()-1]));
33                          py.push_back(ranges[(int)ranges.
                               size()-1]*sin(angles[(int)
                               ranges.size()-1]));
34
35                  }
36          }
37
38          // Divide laser scan into segments separated
               more than g_segmentTh meters
39          LaserFeatures f;
40          f.first_point = 0;
41          for(i=1;i<ranges.size();i++)
42          {
43                  if((px[i-1]-px[i])*(px[i-1]-px[i]) + (py
                       [i-1]-py[i])*(py[i-1]-py[i]) >
                       g_segmentTh2)
44                  {
45                          f.last_point = i-1;
46                          features.push_back(f);
```

```
47                          f.first_point = f.last_point +
                               1;
48                  }
49          }
50          f.last_point = i-1;
51          features.push_back(f);
52
53          for(i=features.size()-2;i > -1;i--)
54          {
55                  for( j = i+1; j<i+5 && j<features.size()
                        ; j++)
56                  {
57                          if((px[features[i].last_point]-
                             px[features[j].first_point])
                             *(px[features[i].last_point]-
                             px[features[j].first_point])
                             + (py[features[i].last_point
                             ]-py[features[j].first_point
                             ])*(py[features[i].last_point
                             ]-py[features[j].first_point
                             ]) < g_segmentTh2)
58                          {
59                                  features[i].last_point =
                                       features[j].
                                     last_point;
60
61                                  features.erase(features.
                                     begin()+i+1,features.
                                     begin()+j);
62                          }
63                  }
64          }
65
66          // Extract features of each laser segment save
               on file
67          for(i=0;i<features.size();i++)
68          {
69                  features[i].set_points( i, features.size
                       ());
70                  features[i].extract_n();
71
72                  // Remove too small segments
73                  if(features[i].get_n() < g_minBeams)
```

```
74                          continue;
75
76                  // Extract the rest of features
77                  features[i].extract_stand_desv( px, py);
78                  features[i].extract_madfm( px, py);
79                  features[i].extract_Width( px, py);
80                  features[i].Matrix_operations( px, py);
81                  features[i].extract_Linearity( px, py);
82                  features[i].extract_Circularity( px, py)
                        ;
83                  features[i].extract_Boundary( px, py);
84                  features[i].extract_Curv( px, py);
85                  features[i].extract_Mean_ang_diff( px,
                        py);
86                  features[i].extract_Seg_Norm(13, px, py)
                        ;
87                  features[i].extract_Kurtosis(px,py);
88                  features[i].extract_Aspect_ratio(px,py);
89
90                  // Remove non-sense features
91                  if( fabs( features[i].get_madfm() ) >
                        100000 || fabs( features[i].
                        get_stand_desv() ) > 100000)
92                          continue;
93
94                  // Save features on file
95                  isPerson = 'N';
96                  if( px[ features[i].first_point ] <
                        g_distPerson )
97                          isPerson = 'P';
98                  fprintf (g_pFile, "%c,%f,%f,%f,%f,%f,%f
                        ,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
                        ,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
                        ,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
                        ,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f,%f
                        ,%f,%f,%f,%f,%f,%f,%f,%f,%f",\
99                          isPerson, \
100                         // 13 basic features
101                         (float) features[i].get_n(),
                                features[i].get_stand_desv(),
                                 features[i].get_madfm(),
```

```
                                \
102                         features[i].get_Width(),
                               features[i].get_Linearity(),
                               features[i].get_Circularity()
                               ,


                                \
103                         features[i].get_Radius(),
                               features[i].
                               get_Boundary_length(),
                               features[i].get_Boundary_reg
                               (),

                                                        \
104                         features[i].get_Curv(), features
                               [i].get_Mean_ang_diff(),
                               features[i].get_Kurtosis(),



                                          \
105                         features[i].get_Aspect_ratio(),









                                                        \
106                         // 13 features * dist origin
107                         (float) features[i].get_n()*
                               features[i].get_dist_origin()
                               , features[i].get_stand_desv
                               ()*features[i].
                               get_dist_origin(),
                                \
108                         features[i].get_madfm()*features
                               [i].get_dist_origin(),
                               features[i].get_Width()*
                               features[i].get_dist_origin()
```

```
                                ,
                                        \
109                             features[i].get_Linearity()*
                                   features[i].get_dist_origin()
                                   , features[i].get_Circularity
                                   ()*features[i].
                                   get_dist_origin(),
                                        \
110                             features[i].get_Radius()*
                                   features[i].get_dist_origin()
                                   , features[i].
                                   get_Boundary_length()*
                                   features[i].get_dist_origin()
                                   ,                    \
111                             features[i].get_Boundary_reg()*
                                   features[i].get_dist_origin()
                                   ,features[i].get_Curv()*
                                   features[i].get_dist_origin()
                                   ,
                                        \
112                             features[i].get_Mean_ang_diff()*
                                   features[i].get_dist_origin()
                                   ,features[i].get_Kurtosis()*
                                   features[i].get_dist_origin()
                                   ,                     \
113                             features[i].get_Aspect_ratio()*
                                   features[i].get_dist_origin()
                                   ,
                                        \
114                             // 13 features / dist origin
115                             (float) features[i].get_n()/
                                   features[i].get_dist_origin()
                                   , features[i].get_stand_desv
                                   ()/features[i].
                                   get_dist_origin(),
                                        \
116                             features[i].get_madfm()/features
                                   [i].get_dist_origin(),
                                   features[i].get_Width()/
```

```
                                features[i].get_dist_origin()
                                ,
                                        \
117                             features[i].get_Linearity()/
                                features[i].get_dist_origin()
                                , features[i].get_Circularity
                                ()/features[i].
                                get_dist_origin(),
                                        \
118                             features[i].get_Radius()/
                                features[i].get_dist_origin()
                                , features[i].
                                get_Boundary_length()/
                                features[i].get_dist_origin()
                                ,                       \
119                             features[i].get_Boundary_reg()/
                                features[i].get_dist_origin()
                                ,features[i].get_Curv()/
                                features[i].get_dist_origin()
                                ,
                                        \
120                             features[i].get_Mean_ang_diff()/
                                features[i].get_dist_origin()
                                ,features[i].get_Kurtosis()/
                                features[i].get_dist_origin()
                                ,                         \
121                             features[i].get_Aspect_ratio()/
                                features[i].get_dist_origin()
                                ,
                                        \
122                             // 12 basic features * n
123                             features[i].get_stand_desv()*
                                features[i].get_n(), features
                                [i].get_madfm()*features[i].
                                get_n(),
                                                          \
124                             features[i].get_Width()*features
                                [i].get_n(), features[i].
```

```
                                get_Linearity()*features[i].
                                get_n(),

                                                          \
125                             features[i].get_Circularity()*
                                features[i].get_n(),features[
                                i].get_Radius()*features[i].
                                get_n(),

                                                          \
126                             features[i].get_Boundary_length
                                ()*features[i].get_n(),
                                features[i].get_Boundary_reg
                                ()*features[i].get_n(),

                                                    \
127                             features[i].get_Curv()*features[
                                i].get_n(), features[i].
                                get_Mean_ang_diff()*features[
                                i].get_n(),

                                                          \
128                             features[i].get_Kurtosis()*
                                features[i].get_n(),features[
                                i].get_Aspect_ratio()*
                                features[i].get_n(),


                                                \
129                             // 12 basic features / n
130                             features[i].get_stand_desv()/
                                features[i].get_n(), features
                                [i].get_madfm()/features[i].
                                get_n(),

                                                          \
131                             features[i].get_Width()/features
                                [i].get_n(), features[i].
                                get_Linearity()/features[i].
                                get_n(),

                                                          \
132                             features[i].get_Circularity()/
```

```
                                features[i].get_n(),features[
                                i].get_Radius()/features[i].
                                get_n(),

                                                                \
133                             features[i].get_Boundary_length
                                ()/features[i].get_n(),
                                features[i].get_Boundary_reg
                                ()/features[i].get_n(),

                                                        \
134                             features[i].get_Curv()/features[
                                i].get_n(), features[i].
                                get_Mean_ang_diff()/features[
                                i].get_n(),

                                                                \
135                             features[i].get_Kurtosis()/
                                features[i].get_n(),features[
                                i].get_Aspect_ratio()/
                                features[i].get_n());
136
137
138
139
140                     fprintf (g_pFile, "\n");
141
142             }
143
144     }
145
146
147     int main(int argc, char** argv)
148     {
149             // Setup ROS
150             ros::init(argc, argv, "
                    laser_features_extractor_outdoor");
151             ros::NodeHandle n;
152             ros::Subscriber predict_sub = n.subscribe("/
                    scanfront", 1, scanCallback);
153
154             // Read node parameters
155             ros::NodeHandle lnh("~");
```

```
156             if(!lnh.getParam("segment_threshold",
                    g_segmentTh))
157                     g_segmentTh = 0.3;
158             g_segmentTh2 = g_segmentTh*g_segmentTh;
159             if(!lnh.getParam("min_distance_person",
                    g_distPerson))
160                     g_distPerson = -1.0;
161             if(!lnh.getParam("min_detection_range",
                    g_minRange))
162                     g_minRange = 0.1;
163             if(!lnh.getParam("max_detection_range",
                    g_maxRange))
164                     g_maxRange = 15.0;
165             if(!lnh.getParam("min_segment_beams", g_minBeams
                    ))
166                     g_minBeams = 3;
167
168             // Open file to save laser features
169             g_pFile = fopen ("features.csv", "a");
170
171             // Spin for ever
172             ros::spin();
173
174             // Close features database
175             fclose (g_pFile);
176
177             return 0;
178     }
```

C. Apéndice III

Script	trainer.cpp
Descripción	Entrena el algoritmo de clasificación
Entradas	Necesita que en la carpeta donde se esta ejecutando este el archivo "*features.csv*" obtenida antes.
Salidas	Crea un archivo llamado "*boost.txt*" con la información del clasificador entrenado.

```
#include "adaboost.hpp"

int main(int argc, char** argv)
{
        Adaboost boost;
        // Primer valor de la base de datos tiene que
           ser positivo
        boost.read_features("features.csv");

        boost.weak_learners(20);
        boost.train();
        boost.save("boost.txt");

        return EXIT_SUCCESS;
}
```

D. Apéndice IV

Script	confusion_matrix.cpp
Descripción	Calcula la matriz de confusión de una base de datos con un clasificador entrenado.
Entradas	Archivos "*features.csv*" y "*boost.txt*"
Salidas	Muestra por pantalla la matriz de confusión

```
#include "adaboost.hpp"

int main(int argc, char** argv)
{
        Adaboost boost;
        // Primer valor de la base de datos tiene que
           ser positivo
        boost.read_features("features.csv");

        boost.load("boost.txt");
        boost.weak_learners(20);

        boost.confusion_matrix();

        return EXIT_SUCCESS;
}
```

E. Apéndice V

Script	predictor_laser_indoor.cpp
Descripción	Se suscribe a un tópico que manda nube de puntos donde detecta que segmentos (mediante 13 características) son personas.
Entradas	Tópico tipo "*LaserScan*": "*/scanfront*"
Salidas	Tópico tipo "*LaserScan*": "*/result*"

```
#include "ros/ros.h"
#include "opencv/cv.h"

#include "sensor_msgs/LaserScan.h"

#include "adaboost.hpp"
#include "laser_features.hpp"

double g_segmentTh;
double g_segmentTh2;
double g_minRange;
double g_maxRange;
int g_minBeams;

using namespace std;
using namespace cv;

```

```
18  Adaboost boosted;
19
20  ros::Publisher result_pub;
21  sensor_msgs::LaserScan scan_ouput_;
22
23
24  void scanCallback (const sensor_msgs::LaserScan::
       ConstPtr& scan_in)
25  {
26          vector<float> px, py, angles, ranges;
27          vector<int> ind;
28          vector<LaserFeatures> features;
29          int isPerson;
30          int i, j;
31
32          scan_ouput_ = *scan_in;
33          scan_ouput_.intensities.clear();
34          scan_ouput_.intensities.resize(scan_in->ranges.
               size());
35
36          for(i=0;i<scan_in->ranges.size();i++)
37          {
38                  if(scan_in->ranges[i] >= g_minRange &&
                       scan_in->ranges[i] <= g_maxRange)
39                  {
40                          ind.push_back( i );
41                          ranges.push_back(scan_in->ranges
                               [i]);
42                          angles.push_back(scan_in->
                               angle_min + i * scan_in->
                               angle_increment);
43                          px.push_back(ranges[(int)ranges.
                               size()-1]*cos(angles[(int)
                               ranges.size()-1]));
44                          py.push_back(ranges[(int)ranges.
                               size()-1]*sin(angles[(int)
                               ranges.size()-1]));
45                  }
46          }
47
48          LaserFeatures f;
49          f.first_point = 0;
50          for(i=1;i<ranges.size();i++)
```

```
51          {
52                  float dist = (px[i-1]-px[i])*(px[i-1]-px
                        [i]) + (py[i-1]-py[i])*(py[i-1]-py[i
                        ]);
53                  if( dist > g_segmentTh2 && dist <
                        g_maxRange)
54                  {
55                          f.last_point = i-1;
56                          features.push_back(f);
57                          f.first_point = f.last_point +
                                1;
58                  }
59          }
60          f.last_point = i-1;
61          features.push_back(f);
62
63          for(i=features.size()-2;i>=0;i--)
64          {
65                  for( j = i+1; j<i+5 && j<features.size()
                        ; j++)
66                  {
67                          if((px[features[i].last_point]-
                                px[features[j].first_point])
                                *(px[features[i].last_point]-
                                px[features[j].first_point])
                                + (py[features[i].last_point
                                ]-py[features[j].first_point
                                ])*(py[features[i].last_point
                                ]-py[features[j].first_point
                                ]) < g_segmentTh2)
68                          {
69                                  features[i].last_point =
                                        features[j].
                                        last_point;
70
71                                  features.erase(features.
                                        begin()+i+1,features.
                                        begin()+j);
72                          }
73                  }
74          }
75
76          for(i=0;i<features.size();i++)
```

```
77          {
78                  features[i].set_points( i, features.size
                       ());
79                  features[i].extract_n();
80
81                  if(features[i].get_n() < g_minBeams)
82                  {
83                          features.erase(features.begin()+
                               i);
84                          i--;
85                          continue;
86                  }
87
88                  features[i].extract_stand_desv( px, py);
89                  features[i].extract_madfm( px, py);
90                  features[i].extract_Jump_p_seg( px, py);
91                  features[i].extract_Jump_s_seg( px, py);
92                  features[i].extract_Width( px, py);
93                  features[i].Matrix_operations( px, py);
94                  features[i].extract_Linearity( px, py);
95                  features[i].extract_Circularity( px, py)
                       ;
96                  features[i].extract_Boundary( px, py);
97                  features[i].extract_Curv( px, py);
98                  features[i].extract_Mean_ang_diff( px,
                       py);
99
100                 // Prediction
101                 isPerson = boosted.predict( features[i].
                       get_vector_indoor() );
102
103                 if(isPerson == 1)
104                 {
105                         for(j=features[i].first_point; j
                               <=features[i].last_point; j
                               ++)
106                         {
107                                 scan_ouput_.intensities[
                                       ind[j]] = 1;
108                         }
109                 }
110                 else
111                 {
```

```
112                         for(j=features[i].first_point; j
                               <=features[i].last_point; j
                               ++)
113                         {
114                                 scan_ouput_.intensities[
                                       ind[j]] = 0;
115                         }
116                 }
117         }
118         result_pub.publish(scan_ouput_);
119 }
120
121 int main(int argc, char** argv)
122 {
123         ros::init(argc, argv, "predictor_laser_indoor");
124
125         ros::NodeHandle n;
126         ros::Subscriber predict_sub = n.subscribe("/
               scanfront", 1, scanCallback);
127         result_pub = n.advertise<sensor_msgs::LaserScan
               >("/result", 1);
128
129         boosted.load("boost.txt");
130         boosted.weak_learners(20);
131
132         // Read node parameters
133         ros::NodeHandle lnh("~");
134         if(!lnh.getParam("segment_threshold",
               g_segmentTh))
135                 g_segmentTh = 0.3;
136         g_segmentTh2 = g_segmentTh*g_segmentTh;
137         if(!lnh.getParam("min_detection_range",
               g_minRange))
138                 g_minRange = 0.15;
139         if(!lnh.getParam("max_detection_range",
               g_maxRange))
140                 g_maxRange = 15.0;
141         if(!lnh.getParam("min_segment_beams", g_minBeams
               ))
142                 g_minBeams = 3;
143
144         // Spin for ever
145         ros::spin();
```

```
146
147         return EXIT_SUCCESS;
148 }
```

F. Apéndice VI

Script	predictor_laser_outdoor.cpp
Descripción	Se suscribe a un tópico que manda nube de puntos donde detecta que segmentos (mediante 63 características) son personas .
Entradas	Tópico tipo "*LaserScan*": "*/scanfront*"
Salidas	Tópico tipo "*LaserScan*": "*/result*"

```
1  #include "ros/ros.h"
2  #include "opencv/cv.h"
3
4  #include "sensor_msgs/LaserScan.h"
5
6  #include "adaboost.hpp"
7  #include "laser_features.hpp"
8
9  double g_segmentTh;
10 double g_segmentTh2;
11 double g_minRange;
12 double g_maxRange;
13 int g_minBeams;
14
15 using namespace std;
16 using namespace cv;
17
```

```
18  Adaboost boosted;
19
20  ros::Publisher result_pub;
21  sensor_msgs::LaserScan scan_ouput_;
22
23
24  void scanCallback (const sensor_msgs::LaserScan::
        ConstPtr& scan_in)
25  {
26          vector<float> px, py, angles, ranges;
27          vector<int> ind;
28          vector<LaserFeatures> features;
29          int isPerson;
30          int i, j;
31
32          scan_ouput_ = *scan_in;
33          scan_ouput_.intensities.clear();
34          scan_ouput_.intensities.resize(scan_in->ranges.
               size());
35
36          for(i=0;i<scan_in->ranges.size();i++)
37          {
38                  if(scan_in->ranges[i] >= g_minRange &&
                       scan_in->ranges[i] <= g_maxRange)
39                  {
40                          ind.push_back( i );
41                          ranges.push_back(scan_in->ranges
                               [i]);
42                          angles.push_back(scan_in->
                               angle_min + i * scan_in->
                               angle_increment);
43                          px.push_back(ranges[(int)ranges.
                               size()-1]*cos(angles[(int)
                               ranges.size()-1]));
44                          py.push_back(ranges[(int)ranges.
                               size()-1]*sin(angles[(int)
                               ranges.size()-1]));
45                  }
46          }
47
48          LaserFeatures f;
49          f.first_point = 0;
50          for(i=1;i<ranges.size();i++)
```

```
51          {
52                  float dist = (px[i-1]-px[i])*(px[i-1]-px
                        [i]) + (py[i-1]-py[i])*(py[i-1]-py[i
                        ]);
53                  if( dist > g_segmentTh2 && dist <
                        g_maxRange)
54                  {
55                          f.last_point = i-1;
56                          features.push_back(f);
57                          f.first_point = f.last_point +
                                1;
58                  }
59          }
60          f.last_point = i-1;
61          features.push_back(f);
62
63          for(i=features.size()-2;i>=0;i--)
64          {
65                  for( j = i+1; j<i+5 && j<features.size()
                        ; j++)
66                  {
67                          if((px[features[i].last_point]-
                                px[features[j].first_point])
                                *(px[features[i].last_point]-
                                px[features[j].first_point])
                                + (py[features[i].last_point
                                ]-py[features[j].first_point
                                ])*(py[features[i].last_point
                                ]-py[features[j].first_point
                                ]) < g_segmentTh2)
68                          {
69                                  features[i].last_point =
                                        features[j].
                                        last_point;
70
71                                  features.erase(features.
                                        begin()+i+1,features.
                                        begin()+j);
72                          }
73                  }
74          }
75
76          for(i=0;i<features.size();i++)
```

```
77          {
78                  features[i].set_points( i, features.size
                       ());
79                  features[i].extract_n();
80
81                  if(features[i].get_n() < g_minBeams)
82                  {
83                          features.erase(features.begin()+
                               i);
84                          i--;
85                          continue;
86                  }
87
88                  features[i].extract_stand_desv( px, py);
89                  features[i].extract_madfm( px, py);
90                  features[i].extract_Width( px, py);
91                  features[i].Matrix_operations( px, py);
92                  features[i].extract_Linearity( px, py);
93                  features[i].extract_Circularity( px, py)
                       ;
94                  features[i].extract_Boundary( px, py);
95                  features[i].extract_Curv( px, py);
96                  features[i].extract_Mean_ang_diff( px,
                       py);
97                  features[i].extract_Seg_Norm(13, px, py)
                       ;
98                  features[i].extract_Kurtosis( px, py);
99                  features[i].extract_Aspect_ratio( px, py
                       );
100
101                 // Predecir
102                 isPerson = boosted.predict( features[i].
                       get_vector_outdoor() );
103
104                 if(isPerson == 1)
105                 {
106                         for(j=features[i].first_point; j
                               <=features[i].last_point; j
                               ++)
107                         {
108                                 scan_ouput_.intensities[
                                       ind[j]] = 1;
109                         }
```

```
110                     }
111                     else
112                     {
113                             for(j=features[i].first_point; j
                                    <=features[i].last_point; j
                                    ++)
114                             {
115                                     scan_ouput_.intensities[
                                            ind[j]] = 0;
116                             }
117                     }
118             }
119             result_pub.publish(scan_ouput_);
120     }
121
122     int main(int argc, char** argv)
123     {
124             ros::init(argc, argv, "predictor_laser_outdoor")
                    ;
125
126             ros::NodeHandle n;
127             ros::Subscriber predict_sub = n.subscribe("/
                    scanfront", 1, scanCallback);
128             result_pub = n.advertise<sensor_msgs::LaserScan
                    >("/result", 1);
129
130             boosted.load("boost.txt");
131             boosted.weak_learners(20);
132
133             // Read node parameters
134             ros::NodeHandle lnh("~");
135             if(!lnh.getParam("segment_threshold",
                    g_segmentTh))
136                     g_segmentTh = 0.3;
137             g_segmentTh2 = g_segmentTh*g_segmentTh;
138             if(!lnh.getParam("min_detection_range",
                    g_minRange))
139                     g_minRange = 0.15;
140             if(!lnh.getParam("max_detection_range",
                    g_maxRange))
141                     g_maxRange = 15.0;
142             if(!lnh.getParam("min_segment_beams", g_minBeams
                    ))
```

```
143                     g_minBeams = 3;
144
145             // Spin for ever
146             ros::spin();
147
148             return EXIT_SUCCESS;
149     }
```

G. Apéndice VII

Script	adaboost.hpp
Descripción	Librería con todas las funciones referentes al clasificador Adaboost.

```
#include "ros/ros.h"
#include <fstream>
#include <vector>

#include "opencv/cv.h"
#include <opencv2/opencv.hpp>

#include "sensor_msgs/LaserScan.h"

using namespace std;
using namespace cv;

struct h_j
{
        int signo;
        int features;
        float wall;
        float r;
        float alfa;
};

class Adaboost
{
```

```
24          private:
25                  int weak_learner;
26                  vector< vector <float> >  datos;
27                  vector< h_j > H;
28                  int RP;
29                  int FP;
30                  int RN;
31                  int FN;
32                  float TPR, FPR;
33                  h_j f_weak_learner( vector<float> D, int
                        feature );
34
35          public:
36                  void weak_learners( int learners);
37                  void read_features(char* data_base);
38                  void train(void);
39                  void save(char* trained_algorith);
40                  void load(char* trained_algorith);
41                  int predict( vector<float> f );
42                  void confusion_matrix();
43  };
44
45  void Adaboost::weak_learners( int learners)
46  {
47          weak_learner = learners;
48  }
49
50  void Adaboost::read_features(char* data_base)
51  {
52          ifstream fp;
53          char linea[1024];
54          char pos = 0;
55          char neg = 0;
56
57          fp.open( data_base, ios::in );
58          if (fp.fail())
59                  exit(1);
60          else
61          {
62                  datos.erase( datos.begin(), datos.end()
                        );
63                  while(!fp.eof())
64                  {
```

```
65                              vector<float> segmento;
66
67                              fp.getline(linea,sizeof(linea),'
                                    \n');
68                              char * pch;
69
70                              // Declaramos en que dos
                                    variables se divide la base
                                    de datos
71                              if ( !neg )
72                              {
73                                      neg = linea[0];
74                              }
75                              else if( pos == 0 && neg !=
                                    linea[0] )
76                              {
77                                      pos = linea[0];
78                              }
79
80                              // Decimos si el segmento es una
                                     muestra positiva o negativa
81                              if ( pos == linea[0] )
82                              {
83                                      segmento.push_back( 1.0
                                            );
84                              }
85                              else if ( neg == linea[0] )
86                              {
87                                      segmento.push_back( -1.0
                                             );
88                              }
89                              else if( NULL == linea[0] )
90                              {
91                                      continue;
92                              }
93                              else
94                              {
95                                      ROS_ERROR("clasificador
                                            de dos variables, has
                                             usado mas: %d",
                                            linea[0]);
96                                      ROS_ERROR("pos: %c  &&
                                            neg: %c",pos,neg);
```

```
97
98                                  exit(1);
99                          }
100
101                         // Despues de cada ',' guardamos
                                el valor numerico que hay
                               detras
102                         pch = strchr(linea,',');
103
104                         while (pch != NULL)
105                         {
106                                 // Cada char array de
                                       numeros la
                                       transformamos en un
                                       float
107                                 float aux = 0;
108                                 aux = strtof( pch + 1,
                                       NULL);
109                                 segmento.push_back( aux
                                       );
110
111                                 pch = strchr(pch+1,',');
112                         }
113                         datos.push_back(segmento);
114                 }
115         }
116         fp.close();
117 }
118
119 h_j Adaboost::f_weak_learner( vector<float> D, int
       feature )
120 {
121         int D_size = datos.size();
122         vector<float> datos_ord, D_ord, l_ord;
123         float aux;
124         h_j h_max;
125         h_j h;
126
127         h_max.r = 0.0;
128
129         datos_ord.resize( D_size );
130         D_ord.resize( D_size );
131         l_ord.resize( D_size );
```

```
132
133         for( int i = 0; i < D_size; i++ )
134         {
135                 datos_ord[i] = datos[i].at(feature);
136                 D_ord[i] = D[i];
137                 l_ord[i] = datos[i].at(0);
138         }
139
140         for( int i = 0; i < D_size; i++ )
141         {
142                 for( int j = 0; j < (D_size - i - 1) ; j
                        ++ )
143                 {
144                         if( datos_ord[j] > datos_ord[ j
                                + 1 ] )
145                         {
146                                 aux = datos_ord[j];
147                                 datos_ord[j] = datos_ord
                                        [j + 1];
148                                 datos_ord[j + 1] = aux;
149
150                                 aux = D_ord[j];
151                                 D_ord[j] = D_ord[j + 1];
152                                 D_ord[j + 1] = aux;
153
154                                 aux = l_ord[j];
155                                 l_ord[j] = l_ord[j + 1];
156                                 l_ord[j + 1] = aux;
157                         }
158                 }
159         }
160
161         for( int i = 0; i < D_size - 1; i++ )
162         {
163                 int pred;
164
165                 if( l_ord[i] != l_ord[i + 1] )
166                 {
167                         h.wall = (datos_ord[i] +
                                datos_ord[i + 1]) / 2;
168                         h.r = 0.0;
169                         for( int j = 0; j < D_size; j++
                                )
```

```
170                             {
171                                     if( datos_ord[j] < h.
                                           wall)
172                                     {
173                                             pred = 1;
174                                     }
175                                     else
176                                     {
177                                             pred = -1;
178                                     }
179
180                                     if( pred == l_ord[j] )
181                                     {
182                                             h.r += D_ord[j];
183                                     }
184                             }
185                             if( h.r < 0.5 )
186                             {
187                                     h.r = 1 - h.r;
188                                     h.signo = -1;
189                             }
190                             else
191                             {
192                                     h.signo = 1;
193                             }
194
195                             if( h_max.r < h.r )
196                             {
197                                     h_max.signo = h.signo;
198                                     h_max.r = h.r;
199                                     h_max.wall = h.wall;
200                             }
201                     }
202             }
203
204             h_max.features = feature;
205             return (h_max);
206     }
207
208     void Adaboost::train(void)
209     {
210             float D_pos_ini = 0.0, D_neg_ini = 0.0;
211             int n_pos = 0, n_neg = 0;
```

```
212             vector<float> D;
213             int D_size = datos.size();
214             H.erase( H.begin(), H.end() );
215
216             //Contamos cuantas muestras positivas tenemos y
                    cuantas negativas
217             for( int i = 0; i < D_size; i++)
218             {
219                     if( datos[i].at(0) == -1)
220                     {
221                             n_neg++;
222                     }
223                     if ( datos[i].at(0) == 1)
224                     {
225                             n_pos++;
226                     }
227             }
228
229             //Repartimos pesos
230             D_pos_ini = 0.5 / n_pos;
231             D_neg_ini = 0.5 / n_neg;
232
233             for( int i = 0; i < D_size; i++)
234             {
235                     if( datos[i].at(0) == -1)
236                     {
237                             D.push_back( D_neg_ini );
238                     }
239                     else if ( datos[i].at(0) == 1)
240                     {
241                             D.push_back( D_pos_ini );
242                     }
243             }
244             int n_features = datos[0].size();
245
246             // Iniciamos el bucle de entrenamiento para cada
                    weak_learner
247             for( int i = 0; i < weak_learner; i++)
248             {
249                     // Normalizamos los pesos
250                     vector<h_j> h_r;
251                     h_r.clear();
252
```

```
253                     float Sum_D = 0;
254
255                     for( int j = 0; j < D_size; j++)
256                     {
257                             Sum_D += D[j];
258                     }
259                     for( int j = 0; j < D_size; j++)
260                     {
261                             D[j] /= Sum_D;
262                     }
263
264                     // Entrenamos clasificador para cada
                           variable
265                     for( int j = 1; j < n_features; j++)
266                     {
267                             h_j h;
268
269                             h.r = 0.0;
270                             h.wall = 0.0;
271                             h.features = 0;
272
273                             // Elegimos el weak learner para
                                   hallar h.wall
274                             h = f_weak_learner( D, j );
275
276                             h_r.push_back( h );
277                     }
278
279                     H.push_back( h_r[0] );
280
281                     for( int j = 1; j < n_features ; j++)
282                     {
283                             if( h_r[j].r > H[i].r )
284                             {
285                                     H[i].signo = h_r[j].
                                           signo;
286                                     H[i].r = h_r[j].r;
287                                     H[i].wall = h_r[j].wall
                                           ;
288                                     H[i].features = h_r[j].
                                           features;
289                             }
290                     }
```

```
291
292                     H[i].alfa = log( ( 1 + H[i].r ) / ( 1 -
                           H[i].r ) ) / 2;
293
294                     // Volvemos a repartir los pesos
295                     for( int j = 0; j < D_size; j++)
296                     {
297                             int pred;
298
299                             if( datos[j].at( H[i].features )
                                    * H[i].signo <= H[i].wall *
                                   H[i].signo )
300                             {
301                                     pred = 1;
302                             }
303                             else pred = -1;
304
305                             D[j] = D[j] * exp( -H[i].alfa *
                                   datos[j].at(0) * pred );
306                     }
307             }
308     }
309
310     int Adaboost::predict( vector<float> f )
311     {
312             float F = 0.0;
313
314             for( int i = 0; i < weak_learner; i++)
315             {
316                     int h;
317
318                     if( fabs(F) > 10000000 )
319                             continue;
320
321                     if( f[ H[i].features  ] * H[i].signo < H
                           [i].wall * H[i].signo )
322                     {
323                             h = 1;
324                     }
325                     else
326                     {
327                             h = -1;
328                     }
```

```
329
330                 F += H[i].alfa * h;
331         }
332
333         int predict = 0;
334         if ( F >= 0 )
335         {
336                 predict = 1;
337         }
338         else
339         {
340                 predict = -1;
341         }
342
343         return predict;
344 }
345
346 void Adaboost::save(char* trained_algorith)
347 {
348         FILE * pFile;
349
350         pFile = fopen ( trained_algorith, "w" );
351         for(int i = 0; i < H.size(); i++)
352         {
353                 fprintf (pFile, "%f %f %f %f\n", (float)
                        H[i].features, H[i].wall, (float) H[
                       i].signo, H[i].alfa ); // borrar r al
                        acabar pruebas
354         }
355         fclose (pFile);
356 }
357
358 void Adaboost::load(char* trained_algorith)
359 {
360         ifstream fp;
361         char linea[128];
362         fp.open( trained_algorith, ios::in );
363         H.clear();
364
365         if (fp.fail())
366                 exit(1);
367         else
368         {
```

```
369                     while(!fp.eof())
370                     {
371                             fp.getline(linea,sizeof(linea),'
                                   \n');
372                             char * pch;
373                             h_j h_lect;
374
375                             h_lect.features = (int) strtof(
                                   linea, &pch);
376                             if( h_lect.features == 0 )
                                   continue;
377                             h_lect.wall = strtof( pch, &pch)
                                   ;
378                             h_lect.signo = (int) strtof( pch
                                   , &pch);
379                             h_lect.alfa = strtof( pch, NULL)
                                   ;
380                             H.push_back( h_lect );
381                     }
382             }
383             fp.close();
384     }
385
386     void Adaboost::confusion_matrix()
387     {
388             int datos_size = datos.size();
389             int prediction;
390
391             RP = 0;
392             FP = 0;
393             RN = 0;
394             FN = 0;
395
396             for( int i = 0; i < datos_size; i++)
397             {
398                     prediction = predict( datos[i] );
399
400                     if( prediction == 1 )
401                     {
402                             if( datos[i].at(0) == 1)
403                             {
404                                     RP++;
405                             }
```

```
406                             else
407                             {
408                                     FP++;
409                             }
410                     }
411                     else
412                     {
413                             if( datos[i].at(0) == 1)
414                             {
415                                     FN++;
416                             }
417                             else
418                             {
419                                     RN++;
420                             }
421                     }
422             }
423
424             ROS_INFO("Muestras: %d\n\n", datos_size);
425
426             ROS_INFO("
                    |-----------------|-----------------|");
427             ROS_INFO("                 |     PERSONA     |    NO
                     PERSONA    |");
428             ROS_INFO("
                    |---------------|-----------------|-----------------|
                    ");
429             ROS_INFO("|   PERSONA     |       %d\t|       %d \t
                     |", RP, FN);
430             ROS_INFO("
                    |---------------|-----------------|-----------------|
                    ");
431             ROS_INFO("| NO PERSONA    |       %d\t|       %d \t
                     |", FP, RN);
432             ROS_INFO("
                    |---------------|-----------------|-----------------|
                    ");
433
434             float Precision = RP / (RP + FP);
435             float Exactitud = (float) (RP + RN) / (RP + RN +
                     FP + FN);
436             float Sensibilidad = (float) RP / (RP + FN);
437             float Especificidad = (float) RN / (RN + FP);
```

```
438             TPR = Especificidad;
439             FPR = 1 - Sensibilidad;
440     }
```

H. Apéndice VIII

Script	laser_features.hpp
Descripción	Librería con todas las funciones referentes al cálculo de las características de segmento.

```
#ifndef __LASER_FEATURES_H__
#define __LASER_FEATURES_H__

#include <math.h>
#include <vector>

using namespace std;

class LaserFeatures
{
        private:
                int n;
                float stand_desv;
                float madfm;
                float Jump_p_seg;
                float Jump_s_seg;
                float Width;
                float Linearity;
                float Circularity;
                float Radius;
                float Boundary_length;
                float Boundary_reg;
                float Curv;
```

```
24                  float Mean_ang_diff;
25                  float Kurtosis;
26                  float Aspect_Ratio;
27
28                  int segment;            // segment
                       number
29                  int scan_size;          // laser.size()
30
31                  float x;
32                  float y;
33                  float xy;
34                  float x2;
35                  float y2;
36                  float C[3];
37
38                  float Long;
39                  float cx;
40                  float cy;
41
42          public:
43                  int first_point;
44                  int last_point;
45                  vector<float> seg_norm;
46
47                  void set_points( int segment_number, int
                        laser_size);
48                  void extract_n();
49                  void extract_stand_desv(const vector<
                       float>& px, const vector<float>& py);
50                  void extract_madfm(const vector<float>&
                       px, const vector<float>& py);
51                  void extract_Jump_p_seg(const vector<
                       float>& px, const vector<float>& py);
52                  void extract_Jump_s_seg(const vector<
                       float>& px, const vector<float>& py);
53                  void extract_Width(const vector<float>&
                       px, const vector<float>& py);
54                  void Matrix_operations(const vector<
                       float>& px, const vector<float>& py);
55                  void extract_Linearity(const vector<
                       float>& px, const vector<float>& py);
56                  void extract_Circularity(const vector<
                       float>& px, const vector<float>& py);
```

```
57                  void extract_Boundary(const vector<float
                       >& px, const vector<float>& py);
58                  void extract_Curv(const vector<float>&
                       px, const vector<float>& py);
59                  void extract_Mean_ang_diff(const vector<
                       float>& px, const vector<float>& py);
60                  void extract_Kurtosis(const vector<float
                       >& px, const vector<float>& py);
61                  void extract_Aspect_ratio(const vector<
                       float>& px, const vector<float>& py);
62                  void extract_Seg_Norm(int num, const
                       vector<float>& px, const vector<float
                       >& py);
63
64                  int get_n()
                       { return n ;};
65                  float get_stand_desv()          { return
                        stand_desv ;};
66                  float get_madfm()
                       { return madfm ;};
67                  float get_Jump_p_seg()          { return
                        Jump_p_seg ;};
68                  float get_Jump_s_seg()          { return
                        Jump_s_seg ;};
69                  float get_Width()
                       { return Width ;};
70                  float get_Linearity()           { return
                        Linearity ;};
71                  float get_Circularity()         { return
                        Circularity ;};
72                  float get_Radius()
                       { return Radius ;};
73                  float get_Boundary_length() { return
                       Boundary_length ;};
74                  float get_Boundary_reg()        { return
                        Boundary_reg ;};
75                  float get_Curv()
                       { return Curv ;};
76                  float get_Mean_ang_diff()       { return
                        Mean_ang_diff ;};
77                  float get_Kurtosis()            { return
                        Kurtosis ;};
78                  float get_Aspect_ratio()        { return
```

```
                    Aspect_Ratio ;};
79                  float get_dist_origin()         { return
                    sqrt(cx*cx+cy*cy) ;};
80                  vector<float> get_vector_indoor();
81                  vector<float> get_vector_outdoor();
82
83                  void set_n(int a)
                              { n = a ;} ;
84                  void set_stand_desv(float a)
                       { stand_desv = a ;};
85                  void set_madfm(float a)
                              { madfm = a ;} ;
86                  void set_Jump_p_seg(float a)
                       { Jump_p_seg = a ;} ;
87                  void set_Jump_s_seg(float a)
                       { Jump_s_seg = a ;} ;
88                  void set_Width(float a)
                              { Width = a ;} ;
89                  void set_Linearity(float a)
                       { Linearity = a ;} ;
90                  void set_Circularity(float a)
                       { Circularity = a ;} ;
91                  void set_Radius(float a)
                              { Radius = a ;} ;
92                  void set_Boundary_length(float a)   {
                       Boundary_length = a ;} ;
93                  void set_Boundary_reg(float a)
                       { Boundary_reg = a ;} ;
94                  void set_Curv(float a)
                              { Curv = a ;} ;
95                  void set_Mean_ang_diff(float a)
                       { Mean_ang_diff = a ;} ;
96                  void set_Kurtosis(float a)
                              { Kurtosis = a ;} ;
97                  void set_Aspect_ratio(float a )
                       { Aspect_Ratio = a ;} ;
98  };
99
100
101 void LaserFeatures::set_points( int segment_number, int
       laser_size)
102 {
103         segment = segment_number;
```

```
104         scan_size = laser_size;
105 }
106
107 void LaserFeatures::extract_n()
108 {
109         n = last_point - first_point + 1;
110 }
111
112 void LaserFeatures::extract_stand_desv(const vector<
    float>& px, const vector<float>& py)
113 {
114         // Centroide
115         cx = 0.0, cy = 0.0;
116         for(int j = first_point; j <= last_point; j++)
117         {
118                 cx += px[j];
119                 cy += py[j];
120         }
121         cx /= n;
122         cy /= n;
123
124         // Stand dev
125         float dev = 0.0;
126         for(int j = first_point; j <= last_point; j++)
127         {
128                 dev += (cx - px[j]) * (cx - px[j]) + (cy
                    - py[j]) * (cy - py[j]);
129         }
130         stand_desv = sqrt( dev / (n-1) );
131 }
132
133 void LaserFeatures::extract_madfm(const vector<float>&
    px, const vector<float>& py)
134 {
135         // Median
136         float x_median = 0.0;
137         float y_median = 0.0;
138
139         if (n % 2 == 1)
140         {
141                 x_median = px[ first_point + ( n >> 1 )
                    ];
142                 y_median = py[ first_point + ( n >> 1 )
```

```
                        ];
143         }
144         else
145         {
146                 x_median = (px[ first_point + (( n - 1)
                       >> 1 )] + px[ first_point + (( n - 1
                       ) >> 1 ) + 1]) / 2;
147                 y_median = (py[ first_point + (( n - 1)
                       >> 1 )] + py[ first_point + (( n - 1
                       ) >> 1 ) + 1]) / 2;
148         }
149
150         // Mean average deviation from median
151         for( int j = first_point; j <= last_point; j++)
152         {
153                 madfm += sqrt( (px[j] - x_median) * (px[
                       j] - x_median) + (py[j] - y_median) *
                        (py[j] - y_median) );
154         }
155         madfm /= n;
156 }
157
158 void LaserFeatures::extract_Jump_p_seg(const vector<
       float>& px, const vector<float>& py)
159 {
160         float var_x, var_y;
161         // Jump precedure segment
162         if(segment)
163         {
164                 var_x = px[ first_point ] - px[
                       first_point - 1 ];
165                 var_y = py[ first_point ] - py[
                       first_point - 1 ];
166                 Jump_p_seg = sqrt( var_x * var_x + var_y
                        * var_y );
167         }
168         else
169         {
170                 Jump_p_seg = 0;
171         }
172 }
173
174 void LaserFeatures::extract_Jump_s_seg(const vector<
```

```
    float>& px, const vector<float>& py)
175 {
176         float var_x, var_y;
177         // Jump succeeding segment
178         if (segment != scan_size - 1)
179         {
180                 var_x = px[ last_point ] - px[
                        last_point + 1 ];
181                 var_y = py[ last_point ] - py[
                        last_point + 1 ];
182                 Jump_s_seg = sqrt( var_x * var_x + var_y
                        * var_y );
183
184         }
185         else
186         {
187                 Jump_s_seg = 0;
188         }
189 }
190
191 void LaserFeatures::extract_Width(const vector<float>&
    px, const vector<float>& py)
192 {
193         float var_x, var_y;
194         // Width
195         var_x = px[ first_point ] - px[ last_point ];
196         var_y = py[ first_point ] - py[ last_point ];
197         Width = sqrt( var_x * var_x + var_y * var_y );
198 }
199
200 void LaserFeatures::Matrix_operations(const vector<float
    >& px, const vector<float>& py)
201 {
202         float A[3][3], B[3], A_inv[3][3], Det;
203
204         x = 0.0;
205         y = 0.0;
206         xy = 0.0;
207         x2 = 0.0;
208         y2 = 0.0;
209         C[0] = 0.0;
210         C[1] = 0.0;
211         C[2] = 0.0;
```

```
212             B[0] = 0.0;
213             B[1] = 0.0;
214             B[2] = 0.0;
215
216             for(int j = first_point; j <= last_point; j++)
217             {
218                     x += px[j];
219                     y += py[j];
220                     xy += px[j] * py[j];
221                     x2 += px[j] * px[j];
222                     y2 += py[j] * py[j];
223             }
224
225             for(int j = first_point; j <= last_point; j++)
226             {
227                     B[0] += (px[j] * px[j] + py[j] * py[j])
                           * px[j];
228                     B[1] += (px[j] * px[j] + py[j] * py[j])
                           * py[j];
229                     B[2] += (px[j] * px[j] + py[j] * py[j]);
230             }
231             B[0] *= 2;
232             B[1] *= 2;
233             B[2] *= -1;
234             A[0][0] = 4 * x2;
235             A[0][1] = 4 * xy;
236             A[0][2] = -2 * x;
237             A[1][0] = 4 * xy;
238             A[1][1] = 4 * y2;
239             A[1][2] = -2 * y;
240             A[2][0] = -2 * x;
241             A[2][1] = -2 * y;
242             A[2][2] = n;
243
244             A_inv[0][0] = A[1][1] * A[2][2] - A[1][2] * A
                   [2][1];
245             A_inv[0][1] = A[2][1] * A[0][2] - A[0][1] * A
                   [2][2];
246             A_inv[0][2] = A[0][1] * A[1][2] - A[1][1] * A
                   [0][2];
247
248             Det = A_inv[0][0] * A[0][0] + A_inv[0][1] * A
                   [1][0] + A_inv[0][2] * A[2][0];
```

```
249
250             if(Det)
251             {
252                     A_inv[0][0] /= Det;
253                     A_inv[0][1] /= Det;
254                     A_inv[0][2] /= Det;
255                     A_inv[1][0] = (A[2][0] * A[1][2] - A
                           [1][0] * A[2][2]) / Det;
256                     A_inv[1][1] = (A[0][0] * A[2][2] - A
                           [2][0] * A[0][2]) / Det;
257                     A_inv[1][2] = (A[0][2] * A[1][0] - A
                           [0][0] * A[1][2]) / Det;
258                     A_inv[2][0] = (A[1][0] * A[2][1] - A
                           [2][0] * A[1][1]) / Det;
259                     A_inv[2][1] = (A[2][0] * A[0][1] - A
                           [0][0] * A[2][1]) / Det;
260                     A_inv[2][2] = (A[0][0] * A[1][1] - A
                           [1][0] * A[0][1]) / Det;
261
262                     C[0] = A_inv[0][0] * B[0] + A_inv[0][1]
                           * B[1] + A_inv[0][2] * B[2];
263                     C[1] = A_inv[1][0] * B[0] + A_inv[1][1]
                           * B[1] + A_inv[1][2] * B[2];
264                     C[2] = A_inv[2][0] * B[0] + A_inv[2][1]
                           * B[1] + A_inv[2][2] * B[2];
265             }
266             else
267             {
268                     C[0] = 10000;
269                     C[1] = 0;
270                     C[2] = 0;
271                     //ROS_INFO("Punto singular");
272             }
273     }
274
275     void LaserFeatures::extract_Linearity(const vector<float
           >& px, const vector<float>& py)
276     {
277             float a = 0, b = 0;
278             // Linearity
279             b = (n * xy - x * y) / (n * x2 - x * x);
280             a = ( x2 * y - x * xy) / (n * x2 - x * x);
281
```

```
282         for(int j = first_point; j <= last_point; j++)
283                 Linearity += (a + b * px[j] - py[j]) * (
                        a + b * px[j] - py[j]);
284
285         Linearity /= n;
286 }
287
288 void LaserFeatures::extract_Circularity(const vector<
        float>& px, const vector<float>& py)
289 {
290         float aux = ( C[0] * C[0] + C[1] * C[1] - C[2] )
                ;
291         // Radius
292         Radius = sqrt( (aux>0)?aux:(-aux) );
293         // Circularity
294         Circularity = 0;
295         float var_x, var_y;
296         for( int j = first_point; j <= last_point; j++)
297         {
298                 var_x = (C[0] - px[j]) * (C[0] - px[j]);
299                 var_y = (C[1] - py[j]) * (C[1] - py[j]);
300                 Circularity += (Radius - sqrt(var_x +
                        var_y)) * (Radius - sqrt(var_x +
                        var_y));
301         }
302         Circularity /= n;
303 }
304
305 void LaserFeatures::extract_Boundary(const vector<float
        >& px, const vector<float>& py)
306 {
307         //Boundary length
308         for( int j = first_point; j <= last_point; j++)
309         {
310                 if( j - first_point )
311                 {
312                         Boundary_length += sqrt( (px[j]
                                - px[j-1]) * (px[j] - px[j
                                -1]) + (py[j] - py[j-1]) * (
                                py[j] - py[j-1]) );
313                 }
314         }
315
```

```
316             // Boundary median
317             float Boundary_median;
318             float dist;
319             float aux = 0;
320
321             Boundary_median = Boundary_length / (n - 1);
322
323             // Boundary regularity
324             for(int j = first_point; j <= last_point; j++)
325             {
326                     if( j - first_point )
327                     {
328                             dist = sqrt( (px[j] - px[j-1]) *
                                    (px[j] - px[j-1]) + (py[j] -
                                    py[j-1]) * (py[j] - py[j-1])
                                    );
329                             aux += (dist - Boundary_median)
                                   * (dist - Boundary_median);
330                     }
331             }
332
333             Boundary_reg = sqrt( aux / (n - 2));
334     }
335
336     void LaserFeatures::extract_Curv(const vector<float>& px
        , const vector<float>& py)
337     {
338             // Mean Curvature
339
340             float dist1 = 0, dist2 = 0, dist3 = 0, Area = 0;
341             Curv = 0;
342
343             for(int j = first_point; j <= last_point; j++)
344             {
345                     dist2 = dist1;
346                     if( j - first_point > 2)
347                     {
348                             dist1 = sqrt( (px[j] - px[j-1])
                                   * (px[j] - px[j-1]) + (py[j]
                                   - py[j-1]) * (py[j] - py[j
                                   -1]) );
349                             dist3 = sqrt( (px[j] - px[j-2])
                                   * (px[j] - px[j-2]) + (py[j]
```

```
                                - py[j-2]) * (py[j] - py[j
                                -2]) );
350                         // Regla de Sarrus
351                         Area = (px[j] * (py[j-1] - py[j
                                -2]) + py[j] * (px[j-2] - px[
                                j-1]) + px[j-1] * py[j-2] -
                                py[j-1] * px[j-2]) / 2.0;
352                         Area = (Area>0)?Area:(-Area);
353                         Curv += 4 * Area / (dist1 *
                                dist2 * dist3) ;
354                 }
355                 else if( j - first_point > 1)
356                 {
357                         dist1 = sqrt( (px[j] - px[j-1])
                                * (px[j] - px[j-1]) + (py[j]
                                - py[j-1]) * (py[j] - py[j
                                -1]) );
358                 }
359         }
360 }
361
362
363 void LaserFeatures::extract_Mean_ang_diff(const vector<
    float>& px, const vector<float>& py)
364 {
365         // Mean angular difference
366         float cos_ang;
367
368         for(int j = first_point+2; j <= last_point; j++)
369         {
370                 // Cos Ang = v1 * v2 / sqrt(|v1| * |v2|)
371                 cos_ang = ((px[j-2] - px[j-1]) * (px[j]
                        - px[j-1]) +    \
372                         (py[j-2] - py[j-1]) * (py[j] -
                                py[j-1])) /
                                        \
373                         sqrt( (px[j-2] - px[j-1]) * (px[
                                j-2] - px[j-1]) +       \
374                         (py[j-2] - py[j-1]) * (py[j-2] -
                                 py[j-1]) +             \
375                         (px[j] - px[j-1]) * (px[j] - px[
                                j-1]) +                 \
376                         (py[j] - py[j-1]) * (py[j] - py[
```

```
                                j-1]) );
377                     Mean_ang_diff += acos( cos_ang );
378             }
379             Mean_ang_diff /= n-2;
380     }
381
382
383     void LaserFeatures::extract_Seg_Norm(int num, const
           vector<float>& px, const vector<float>& py)
384     {
385             float A, C, aux, aux_div, x_p, y_p, max_s = 0,
                   min_s = 0, max;
386             vector<float> seg;
387
388             A = ( py[last_point] - py[first_point] ) / ( px[
                   last_point] - px[first_point] );
389             C =  py[first_point] - A * px[first_point];
390             aux_div = sqrt( A * A + 1);
391
392             for( int i = 0; i < num; i++)
393             {
394                     aux = ( n - 1.0 ) / ( i + 2.0 );
395                     x_p = (px[(int) (first_point + aux + 1)]
                           - px[(int) (first_point + aux)]) * (
                           aux - (int) aux) + px[(int) (
                           first_point +  aux)];
396                     y_p = (py[(int) (first_point + aux + 1)]
                           - py[(int) (first_point + aux)]) * (
                           aux - (int) aux) + py[(int) (
                           first_point +  aux)];
397                     seg.push_back( -1 * (A * x_p + y_p + C))
                           ;
398
399                     if( seg[i] > max_s )
400                     {
401                             max_s = seg[i];
402                     }
403                     if( seg[i] < min_s )
404                     {
405                             min_s = seg[i];
406                     }
407             }
408
```

```
409             seg_norm.push_back( max_s / aux_div );
410             seg_norm.push_back( min_s / aux_div );
411
412             max = (max_s > -min_s)?max_s:-min_s;
413
414             Long = fabs(max_s - min_s) / aux_div;
415
416             for( int i = 0; i < seg.size(); i++)
417             {
418                     seg_norm.push_back( seg[i] / max );
419             }
420     }
421
422     void LaserFeatures::extract_Kurtosis(const vector<float
        >& px, const vector<float>& py)
423     {
424             //Kurtosis(X) ={\sum_{i=1}^N{(x_i-\bar{x})^4} \
                over (N-1)s^4}
425             float aux = 0.0;
426
427             for(int j = first_point; j <= last_point; j++)
428             {
429                     aux += ((px[j]-cx)*(px[j]-cx) + (py[j]-
                        cy)*(py[j]-cy))*((px[j]-cx)*(px[j]-cx
                        ) + (py[j]-cy)*(py[j]-cy));
430             }
431
432             Kurtosis = aux/((n-1)*stand_desv*stand_desv*
                stand_desv*stand_desv);
433     }
434
435
436     void LaserFeatures::extract_Aspect_ratio(const vector<
        float>& px, const vector<float>& py)
437     {
438             Aspect_Ratio = Long/Width;
439     }
440
441
442     vector<float> LaserFeatures::get_vector_indoor()
443     {
444             vector<float> v;
445
```

```
446             v.push_back(1);
447             v.push_back( (float) n );
448             v.push_back( stand_desv );
449             v.push_back( madfm );
450             v.push_back( Jump_p_seg );
451             v.push_back( Jump_s_seg );
452             v.push_back( Width );
453             v.push_back( Linearity );
454             v.push_back( Circularity );
455             v.push_back( Radius );
456             v.push_back( Boundary_length );
457             v.push_back( Boundary_reg );
458             v.push_back( Curv );
459             v.push_back( Mean_ang_diff );
460
461             return v;
462     }
463
464     vector<float> LaserFeatures::get_vector_outdoor()
465     {
466             vector<float> v;
467
468             // 13 basic features
469             v.push_back(1);
470             v.push_back( (float) n );
471             v.push_back( stand_desv );
472             v.push_back( madfm );
473             v.push_back( Width );
474             v.push_back( Linearity );
475             v.push_back( Circularity );
476             v.push_back( Radius );
477             v.push_back( Boundary_length );
478             v.push_back( Boundary_reg );
479             v.push_back( Curv );
480             v.push_back( Mean_ang_diff );
481             v.push_back( Kurtosis );
482             v.push_back( Aspect_Ratio );
483
484             // 13 features * dist origin
485             v.push_back( (float) n *get_dist_origin() );
486             v.push_back( stand_desv *get_dist_origin());
487             v.push_back( madfm *get_dist_origin());
488             v.push_back( Width *get_dist_origin());
```

```
489             v.push_back( Linearity *get_dist_origin());
490             v.push_back( Circularity *get_dist_origin());
491             v.push_back( Radius *get_dist_origin());
492             v.push_back( Boundary_length *get_dist_origin())
                    ;
493             v.push_back( Boundary_reg *get_dist_origin());
494             v.push_back( Curv *get_dist_origin());
495             v.push_back( Mean_ang_diff *get_dist_origin());
496             v.push_back( Kurtosis *get_dist_origin());
497             v.push_back( Aspect_Ratio *get_dist_origin());
498
499             // 13 features / dist origin
500             v.push_back( (float) n /get_dist_origin() );
501             v.push_back( stand_desv /get_dist_origin());
502             v.push_back( madfm /get_dist_origin());
503             v.push_back( Width /get_dist_origin());
504             v.push_back( Linearity /get_dist_origin());
505             v.push_back( Circularity /get_dist_origin());
506             v.push_back( Radius /get_dist_origin());
507             v.push_back( Boundary_length /get_dist_origin())
                    ;
508             v.push_back( Boundary_reg /get_dist_origin());
509             v.push_back( Curv /get_dist_origin());
510             v.push_back( Mean_ang_diff /get_dist_origin());
511             v.push_back( Kurtosis /get_dist_origin());
512             v.push_back( Aspect_Ratio /get_dist_origin());
513
514             // 12 basic features * n
515             v.push_back( stand_desv *n);
516             v.push_back( madfm *n);
517             v.push_back( Width *n);
518             v.push_back( Linearity *n);
519             v.push_back( Circularity *n);
520             v.push_back( Radius *n);
521             v.push_back( Boundary_length *n);
522             v.push_back( Boundary_reg *n);
523             v.push_back( Curv *n);
524             v.push_back( Mean_ang_diff *n);
525             v.push_back( Kurtosis *n);
526             v.push_back( Aspect_Ratio *n);
527
528             // 12 basic features / n
529             v.push_back( stand_desv /n);
```

```
530             v.push_back( madfm /n);
531             v.push_back( Width /n);
532             v.push_back( Linearity /n);
533             v.push_back( Circularity /n);
534             v.push_back( Radius /n);
535             v.push_back( Boundary_length /n);
536             v.push_back( Boundary_reg /n);
537             v.push_back( Curv /n);
538             v.push_back( Mean_ang_diff /n);
539             v.push_back( Kurtosis /n);
540             v.push_back( Aspect_Ratio /n);
541
542             return v;
543     }
544
545     #endif
```