



UNIVERSIDAD DE BURGOS  
ESCUELA POLITÉCNICA SUPERIOR  
Grado en Ingeniería Informática



**TFG del Grado en Ingeniería  
Informática**

**Detección de palets mediante  
LIDAR  
Documentación Técnica**



Presentado por Mario Flores Espiga  
en Universidad de Burgos — 1 de julio  
de 2019

Tutor: Jesús Enrique Sierra García



---

# Índice general

---

<b>Índice general</b>	<b>I</b>
<b>Índice de figuras</b>	<b>III</b>
<b>Índice de tablas</b>	<b>IV</b>
<b>Apéndice A Plan de Proyecto Software</b>	<b>1</b>
A.1. Introducción . . . . .	1
A.2. Planificación temporal . . . . .	2
A.3. Estudio de viabilidad . . . . .	12
<b>Apéndice B Especificación de Requisitos</b>	<b>17</b>
B.1. Introducción . . . . .	17
B.2. Objetivos generales . . . . .	17
B.3. Catálogo de requisitos . . . . .	17
B.4. Especificación de requisitos . . . . .	18
<b>Apéndice C Especificación de diseño</b>	<b>25</b>
C.1. Introducción . . . . .	25
C.2. Diseño de datos . . . . .	25
C.3. Diseño procedimental . . . . .	26
C.4. Diseño arquitectónico . . . . .	27
<b>Apéndice D Documentación técnica de programación</b>	<b>29</b>
D.1. Introducción . . . . .	29
D.2. Estructura de directorios . . . . .	29
D.3. Manual del programador . . . . .	30

D.4. Compilación, instalación y ejecución del proyecto . . . . .	34
D.5. Pruebas del sistema . . . . .	34
<b>Apéndice E Documentación de usuario</b>	<b>37</b>
E.1. Introducción . . . . .	37
E.2. Requisitos de usuarios . . . . .	37
E.3. Instalación . . . . .	38
E.4. Manual del usuario . . . . .	38
<b>Bibliografía</b>	<b>41</b>

---

# Índice de figuras

---

A.1. Burndown del sprint 0 . . . . .	3
A.2. Burndown del sprint 1 . . . . .	4
A.3. Burndown del sprint 2 - Se puede observar que se invirtieron más horas de las previstas inicialmente . . . . .	5
A.4. Burndown del sprint 3 . . . . .	6
A.5. Burndown del sprint 4 . . . . .	7
A.6. Burndown del sprint 5 . . . . .	8
A.7. Burndown del sprint 6 . . . . .	9
A.8. Burndown del sprint 7 . . . . .	10
A.9. Burndown del sprint 8 . . . . .	11
A.10. Burndown del sprint 9 . . . . .	12
 B.1. Diagrama de casos de uso. . . . .	 24
 C.1. Diagrama de secuencia de la ejecución del programa . . . . .	 27
C.2. Diagrama de clases del sistema . . . . .	28
 D.1. Menú 'File' . . . . .	 31
D.2. 'Clone Repository' . . . . .	32
D.3. URL del repositorio a clonar . . . . .	32
D.4. Descarga del repositorio . . . . .	33
D.5. Búsqueda del directorio . . . . .	33
 E.1. Navegación al directorio 'source' . . . . .	 39
E.2. Ejecución del archivo . . . . .	39
E.3. Apertura de los archivos . . . . .	40
E.4. Ejecución de la aplicación . . . . .	40

---

# Índice de tablas

---

A.1. Costes de personal . . . . .	13
A.2. Costes de hardware . . . . .	13
A.3. Costes de software . . . . .	13
A.4. Coste total . . . . .	14

## *Apéndice A*

---

# **Plan de Proyecto Software**

---

### **A.1. Introducción**

Este apartado del proyecto se centra en estimar de la manera más precisa posible los costes del proyecto. Estos se desglosan en costes de tiempo, costes en trabajo y costes monetarios.

Este análisis nos permite conocer los recursos que van a ser necesarios para la realización del proyecto. Al tratarse este de un proyecto de investigación, es de especial dificultad predecir el marco temporal puesto que los plazos son complicados de cumplir cuando surgen dificultades, e incluso complicados también de predecir en un primer momento al no conocer parte de los procedimientos a realizar.

El plan de proyecto software se divide en:

- Planificación temporal
- Estudio de viabilidad

La planificación temporal trata de ajustar los tiempos que va a llevar cada parte del proyecto. Se establecen las fechas en las que se inicia cada tarea y también en las que termina, basándose en la complejidad de cada una de ellas y los requisitos asociados.

En el estudio de viabilidad se tratan de estimar los costes monetarios del proyecto por un lado, y por otro adecuar el proyecto al marco legal actual, estudiando todas las leyes que podrían afectar, a nivel de licencias del software, protección de datos etc.

## A.2. Planificación temporal

En el proyecto se intenta seguir una metodología ágil conocida como SCRUM (Ya explicado en otras partes del proyecto) de la manera más fiel posible, teniendo en cuenta el carácter investigativo del proyecto y los contratiempos que han ido surgiendo a lo largo del desarrollo.

A continuación se procede a desglosar el desarrollo del proyecto en los distintos sprints:

### Sprint 0 (4/02/2019 - 20/02/2019)

Este es el sprint inicial, donde se plantearon los primeros pasos a seguir. Tras una reunión se decidió que los primeros pasos iban a ser la investigación del estado del arte para obtener conocimientos acerca de otras aproximaciones al problema. También se escogió el lenguaje de programación, el IDE en el que se trabajaría, siendo Python y Spyder los elegidos. Por último, se planteó la búsqueda de una fuente de alimentación para el láser.

La duración fue superior a la planificada hasta que la documentación sobre otros proyectos resultó suficiente.

Se estimó inicialmente una semana para el sprint pero finalmente se invirtió 16 días

#### Tareas

- Búsqueda y configuración de las herramientas para el desarrollo del proyecto (Spyder, LaTeX, Python, RealTerm etc.)
- Comienzo de la documentación del estado del arte sobre otros proyectos que abordaran el mismo problema.
- Búsqueda de una fuente de alimentación adecuada para el equipo láser.
- Estudio y comprensión del código previo.



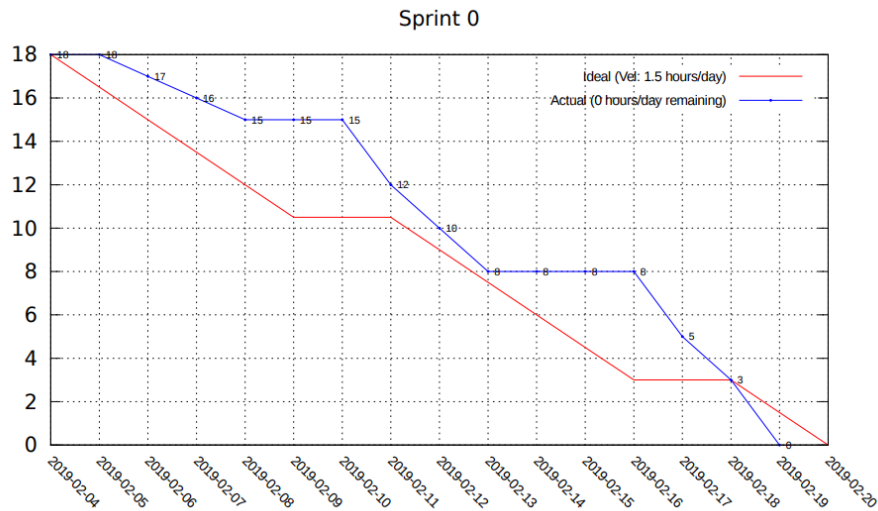


Figura A.1: Burndown del sprint 0

### Sprint 1 (20/02/2019 - 6/03/2019)

En este segundo sprint se comienza a aprender a utilizar LaTeX para el desarrollo de la memoria, se comparan diferentes alternativas para la alimentación del láser, se realiza la búsqueda de un palet de tipo europeo para tener un entorno de pruebas y se desglosa el programa em los diferentes requisitos funcionales.

Se estudia el funcionamiento interno del láser a través de los manuales de usuario y también se estudia el formato de las tramas de datos que se envían. En este sprint se sigue con la investigación del estado del arte recopilando y analizando más artículos.

En este sprint se cumplieron los plazos establecidos.

#### Tareas

- Aprendizaje y primeros pasos con LaTeX.
- Continuación de la documentación de proyectos similares y estudios científicos.
- Búsqueda de palet para el entorno de pruebas.
- Desglose de requisitos funcionales del programa.
- Estudio del funcionamiento del láser y las tramas de datos.

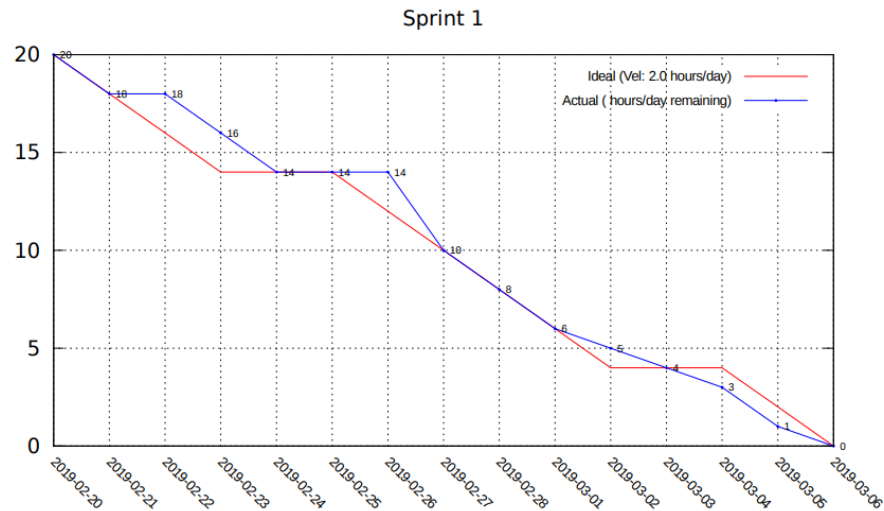


Figura A.2: Burndown del sprint 1

## Sprint 2 (11/03/2019 - 26/03/2019)

El objetivo principal de este sprint es plantear la estructura del código que se va a implementar en iteraciones futuras. Se ha planteado la división en tres capas utilizando el patrón de diseño modelo-vista-controlador puesto que se adecúa a la estructura planteada. Una vez entendido ya el código previo, se realiza una prueba de ejecución para comprobar su funcionamiento. En materia de investigación, se han seleccionado los estudios que comparten la fuente de datos (sensor láser) para analizar sus algoritmos y poder aplicar los conocimientos observados.

Inicialmente se estimó una semana para la duración del sprint pero finalmente se alargó hasta ocupar dos semanas aproximadamente y se superaron las horas de trabajo que se habían previsto para el sprint.

### Tareas

- Planteamiento de la estructura del código a implementar.
- Clasificación de artículos en función de la tecnología utilizada.
- Prueba de funcionamiento del código previo.

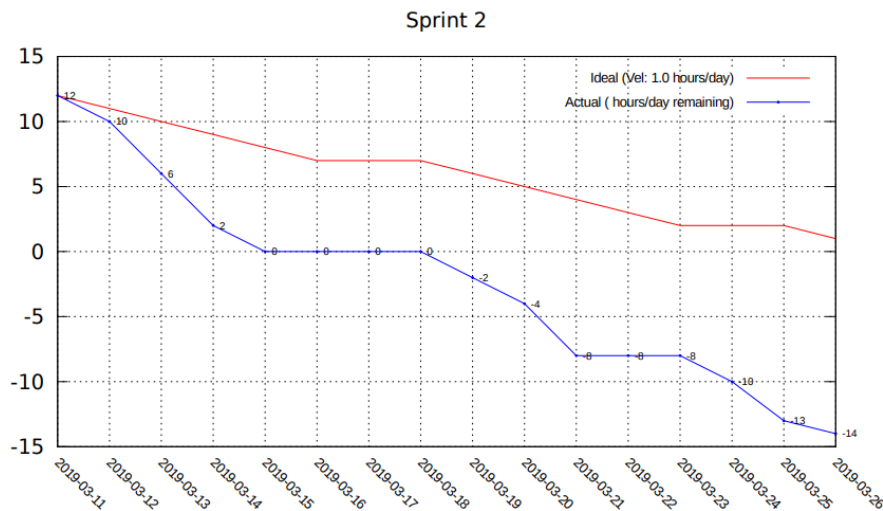


Figura A.3: Burndown del sprint 2 - Se puede observar que se invirtieron más horas de las previstas inicialmente

### Sprint 3 (28/03/2019 - 19/04/2019)

En este sprint se han realizado los primeros pasos a nivel de programación del proyecto.

Se ha estudiado la biblioteca TKinter de Python a través de la documentación y tutoriales online para realizar un apartado visual en el proyecto. Se ha modificado el código para posibilitar el procesamiento de las tramas de manera continua. También se ha desarrollado una interfaz gráfica básica con TKinter. Respecto al apartado investigativo, se ha generado una tabla comparativa entre los distintos estudios en función de la tecnología de los sensores de detección que utiliza cada uno.

Se ha creado el repositorio en GitHub en el que se irán publicando los cambios y variaciones del proyecto.

Ha sido uno de los sprints más largos debido a la semana santa y a distintos problemas surgidos a la hora de implementar la interfaz gráfica con TKinter, surgiendo errores que congelaron el desarrollo durante varios días.

#### Tareas

- Investigación interfaces gráficas TKinter y alternativas.
- Comienzo de la codificación.

- Generar tabla comparativa entre los estudios.
- Procesar tramas de forma continua.
- Generar repositorio en GitHub.
- Creación de interfaz gráfica básica.

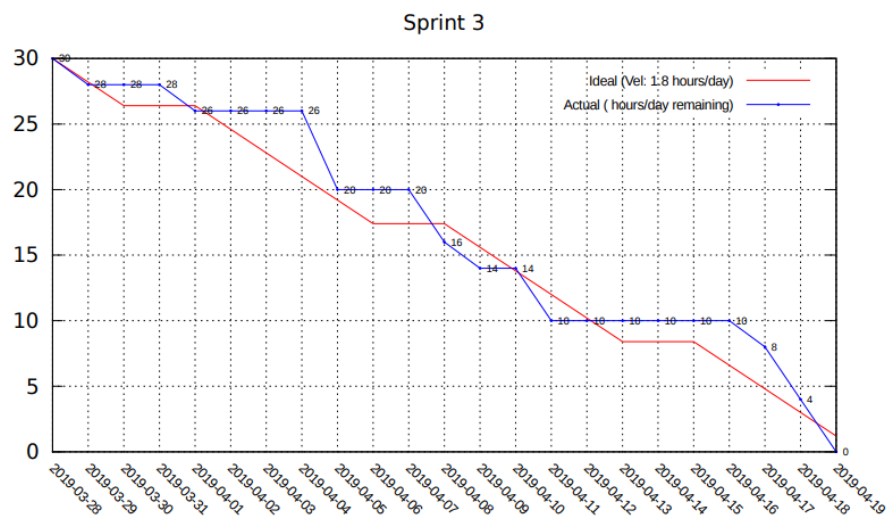


Figura A.4: Burndown del sprint 3

### Sprint 4 (22/04/2019 - 06/05/2019)

En este sprint se comienzan a investigar técnicas para el tratamiento de la nube de puntos. Entre las estudiadas están la segmentación, clustering, filtrado mediante erosión y dilatación etc. Se recopilan tramas con el palet en diferentes posiciones y distancias para estudiarlas.

En adición, se ha seguido mejorando la interfaz gráfica, que hasta ahora solo incluía control del programa pero no visualizado, y se han intentado añadir las gráficas de la nube de puntos dentro de la interfaz.

Así mismo se ha continuado desarrollando algunos apartados de la memoria del proyecto.

Parte del trabajo de la inclusión de la nube de puntos se arrastra hasta el siguiente sprint por no haber podido solucionarlo dentro del plazo previsto.

### Tareas

- Investigación de técnicas para el procesamiento de la nube de puntos.
- Investigación de la morfología de las tramas en las que se observa un palet para adecuar las comprobaciones pertinentes.
- Incluir gráficos de la nube de puntos en la interfaz.
- Continuación del desarrollo de la memoria.

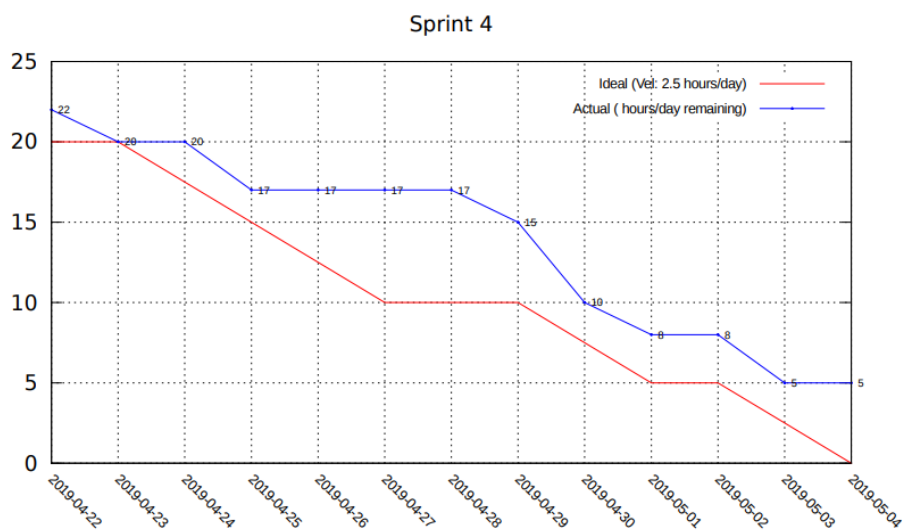


Figura A.5: Burndown del sprint 4

### Sprint 5 (07/05/2019 - 21/05/2019)

Tras dedicar tiempo y esfuerzo en este sprint a la inclusión de la nube de puntos en la interfaz gráfica, se ha considerado que es un gasto de tiempo y esfuerzo demasiado grande para lo que está aportando al proyecto, por lo tanto se ha descartado dicho camino. En su lugar se intenta desarrollar un gráfico de puntos que se actualice en tiempo real con la librería **Matplotlib**. Se ha investigado sobre los métodos necesarios y su codificación para hacer esto posible.

También se han hecho avances significativos en la memoria del proyecto. Se ha creado un archivo de log con los puntos de cada trama para actualizar así la gráfica en tiempo real.

Se realizan los primeros pasos con el algoritmo de clustering escogido por ser un algoritmo sencillo pero potente y adecuado al problema (**KMEANS**).

## Tareas

- Investigar gráficos actualizables en matplotlib.
- Codificar gráficos actualizables.
- Continuación del desarrollo de la memoria.
- Creación de archivo de log para mantener el gráfico actualizado.

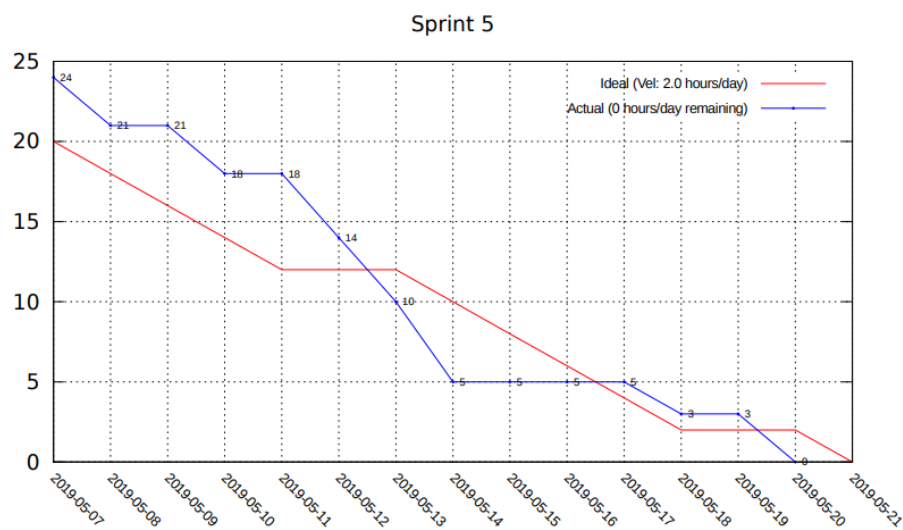


Figura A.6: Burndown del sprint 5

## Sprint 6 (22/05/2019 - 05/06/2019)

En este sprint se sigue con la actualización de la memoria.

Se ha solucionado un problema por el cual las nubes de puntos se acumulaban en el fichero de log y hacían que la impresión de la gráfica se bloqueara. Esto último ha consumido la mayor parte del sprint al no poder localizar el origen del problema y bloquear el desarrollo.

Se descarta el archivo de log como medio para actualizar las gráficas, dejando el archivo para labores de depuración. Se ha implementado la primera de las medidas en el algoritmo de reconocimiento de los palets. Consiste en un método que compara el tamaño de cada uno de los clusters. Las horas reales en este sprint fueron notablemente superiores a las estimadas en consecuencia de la depuración para encontrar el *bug* que bloqueaba la ejecución del programa.

### Tareas

- Actualización de la memoria.
- Solucionar *bug* por el que se bloquea el programa a la hora de imprimir las gráficas.
- Eliminar archivo intermedio y destinarlo a log de las tramas.
- Añadir comprobaciones al algoritmo de detección.

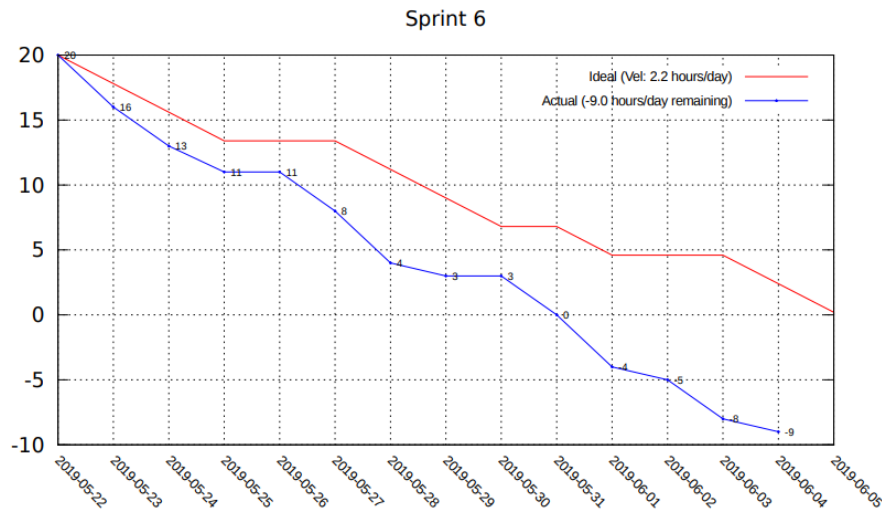


Figura A.7: Burndown del sprint 6

### Sprint 7 (6/05/2019 - 12/06/2019)

Este sprint tiene como objetivo principal añadir las comprobaciones a la detección para aumentar su robustez y precisión. Se ha codificado dos métodos para conocer la distancia y el ángulo exacto de cada cluster. Para ello se utiliza el centro geométrico de los clusters. Se comienza alguna prueba con datos reales del palet para comprobar que estas adiciones al algoritmo funcionen adecuadamente.

### Tareas

- Mejorar algoritmo de detección.

- Probar algoritmo de detección.

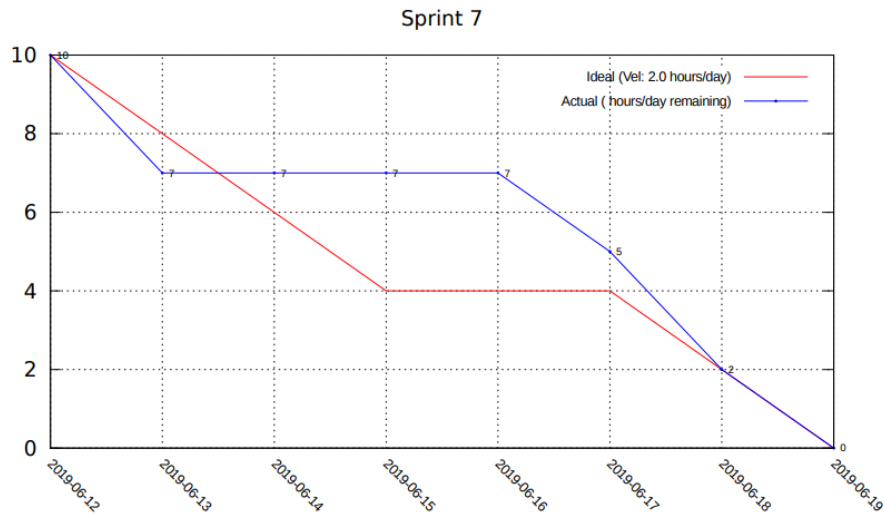


Figura A.8: Burndown del sprint 7

### Sprint 8 (13/05/2019 - 27/06/2019)

Durante este sprint se han corregido fallos en el algoritmo que impedían que este funcionara correctamente:

Tras realizar las pruebas con los datos reales del palet, se observó que el centro geométrico de los clusters en los que se basaba el algoritmo para realizar los cálculos nunca iba a coincidir con ningún punto real de la nube de puntos que devolvía el láser, con lo cual calcular la separación entre los clusters era imposible. Para solucionar este contratiempo se ha optado por calcular la distancia media entre todos los puntos de cada cluster, así como el ángulo medio de todos los puntos, para hallar el punto más representativo de cada cluster pero esta vez obteniendo los datos de distancia y ángulo. De esta manera aplicando el teorema del coseno se ha podido hallar la distancia real que separa los clusters y compararla con las medidas reales de separación entre las patas del palet. Esta es la comprobación final que confirma si se reconoce o no un palet.

En otra línea de trabajo se han realizado actualizaciones importantes en la memoria así como la realización de gran parte de los anexos.



### Tareas

- Arreglar detección de punto representativo de los clusters.
- Continuar con la realización de la memoria y los anexos.

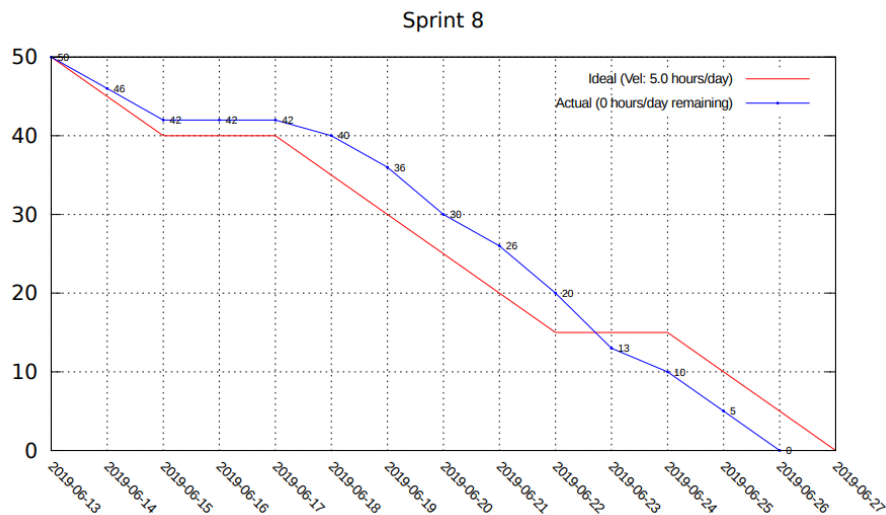


Figura A.9: Burndown del sprint 8

### Sprint 9 (28/05/2019 - 03/07/2019)

En este sprint se ultiman los preparativos de entrega del proyecto. Se ha actualizado la memoria y los anexos, generado los documentos entregables y añadida documentación dentro del código.

### Tareas

- Generar entregables del proyecto.
- Terminar la realización de la memoria y los anexos.

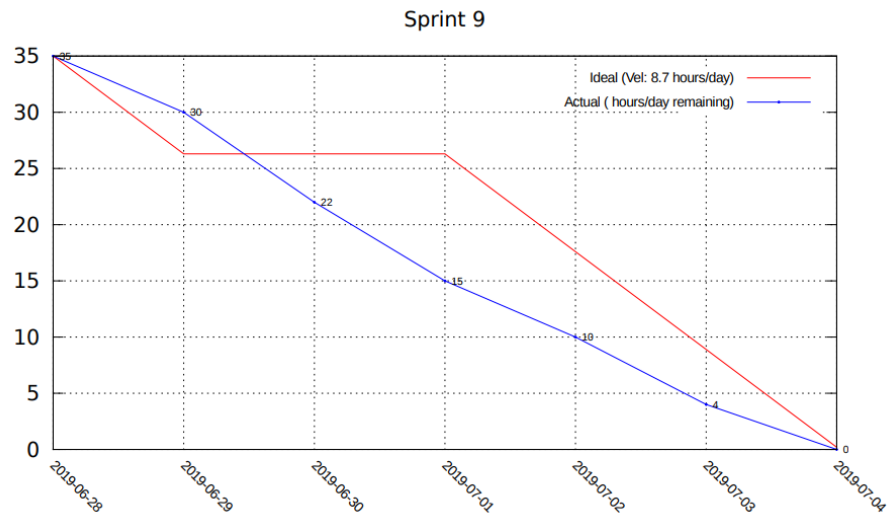


Figura A.10: Burndown del sprint 9

### A.3. Estudio de viabilidad

#### Viabilidad económica

En este apartado se estudiará los costes y beneficios del proyecto en el caso de que se hubiera desarrollado en el mundo laboral en lugar del académico.

Los costes se desglosan en los siguientes:

- Costes de personal
- Costes de infraestructura hardware
- Costes de infraestructura software

#### Costes de personal

Se atribuyen al salario de un desarrollador a tiempo parcial durante 4 meses, para el que se considera un salario neto de €800. Los porcentajes de cotización a la seguridad social del régimen de 2019 corresponden a 23.6 % y 4.7 % para empresas y trabajadores respectivamente.

Concepto	Coste (€)
Salario neto	800 €
Retención IRPF (15 %)	120 €
Seguridad social (23.60 %)	189 €
Salario bruto (mensual)	1109 €
Total 4 meses	4436 €

Tabla A.1: Costes de personal

Concepto	Coste (€)	Amortización
Ordenador portátil	650 €	43.35 €
Equipo láser	400 €	27.29 €
Cableado y fuente de alimentación	40 €	2.7 €
Alquiler mensual laboratorio	350 € mensual	1400 €
Total 4 meses		1446.05 €

Tabla A.2: Costes de hardware

Concepto	Coste (€)	Amortización
Windows 10 Home	145 €	36.25 €

Tabla A.3: Costes de software

### Costes de infraestructura hardware

Estos costes se componen de los costes del ordenador utilizado, el equipo láser y el cableado de alimentación y conexión con el láser. Además se considera el entorno de pruebas necesario para el desarrollo como alquiler durante los cuatro meses de duracion del desarrollo del proyecto.

### Costes de infraestructura software

En estos costes se incluye el coste de todas las licencias del software necesario para la realizacion del proyecto. Más alla del coste de la licencia del sistema operativo Windows 10 no se ha incurrido en ningún otro coste de software

Coste total de software 4 meses 36.25 €.

Concepto	Coste (€)
Personal	4436 €
Hardware	1446.05 €
Software	36.25 €
Total	5918.30 €

Tabla A.4: Coste total

Concluyendo todos los costes mencionados anteriormente, suman un total de 5918.30 €.

### Beneficios

Para la distribución del software creado en el proyecto se considera la implantación tanto a nivel hardware con el equipo láser como a nivel software en AGVs. Se calcula la implantación en unos 200 AGV's con un precio de 400 € por el láser (Precio de coste 225 €), sumado a 50 € en costes de software de carácter anual y 35 € en costes de instalación y puesta en marcha. El total asciende a 52000 € en el primer año y 10000 € anualmente a partir del segundo año.

### Viabilidad legal

Para comprobar la viabilidad del proyecto se ha estudiado las licencias de todas las herramientas utilizadas:

Librerías:

- Numpy: licencia BSD.
- Matplotlib: licencia BSD.

Software:

- Spyder: licencia BSD.

- RealTerm: licencia BSD.
- LaTeX: licencia LPPL Versión 1.3c

Ninguna de las licencias anteriores dificulta la distribucion del proyecto ni ninguno de sus contenidos.

Para el proyecto se ha optado por la licencia GNU-3.0 [\[1\]](#). De esta manera se permite que el software sea usado, estudiado, compartido y modificado siempre y cuando la licencia se mantenga.



## *Apéndice B*

---

# **Especificación de Requisitos**

---

### **B.1. Introducción**

En este anexo se detallan los objetivos de la aplicación y se describe la especificación de requisitos del sistema a nivel funcional y no funcional.

### **B.2. Objetivos generales**

Los objetivos generales del proyecto son los siguientes:

- Crear y mantener una conexión estable con el equipo láser que permita el intercambio de mensajes.
- Recoger los datos que el láser retorne.
- Realizar una traducción y tratamiento adecuados sobre los datos recibidos
- Ejecutar una toma de decisión para confirmar si hay existencia o no de un palet en las lecturas.

### **B.3. Catálogo de requisitos**

En esta sección se exponen los requisitos que los objetivos generales del proyecto requieren.

Se puede hacer la distinción entre requisitos funcionales y no funcionales:

### Requisitos funcionales

- RF-1 Conectar con el láser
- RF-2 Enviar mensaje al láser
- RF-3 Recibir tramas del láser
- RF-4 Traducir las tramas
- RF-5 Procesar las tramas
- RF-6 Tomar decisión palet
- RF-7 Imprimir las tramas

### Requisitos no funcionales

- RNF-1 Precisión: El sistema debe concluir con la máxima precisión posible si ha detectado un palet y en caso afirmativo su distancia y su ángulo respecto del origen.
- RNF-2 Rendimiento: La aplicación debe mostrar un rendimiento adecuado para que pueda procesar las tramas en tiempo real.
- RNF-3 Modularidad: La aplicación debe permitir de manera sencilla mediante modularidad su extensión con nuevas funcionalidades
- RNF-4 Mantenibilidad: La aplicación debe incorporar algún patrón arquitectónico que facilite su mantenimiento, comprensión y escalabilidad.
- RNF-5 Conexión: Debe existir una conexión con el láser para poder ejecutar la aplicación.

## B.4. Especificación de requisitos

### Casos de uso

A continuación se detalla la especificación de casos de uso derivados de los requisitos funcionales del proyecto:



**C1: Conectar con el láser.**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** El programa debe crear una conexión con el equipo láser para poder intercambiar información a través de ella.

**Precondición** Se ha lanzado el programa principal

**Secuencia Normal:**

1. Se ejecuta el programa principal.
2. Se crea el objeto de tipo Socket.
3. Se establece la dirección IP y el número de puerto.
4. Se abre la conexión TCP a la dirección y el puerto especificados

**Postcondición** Se ha recibido una respuesta por parte del láser tras haber enviado un mensaje petición.

**Excepciones** El láser no está conectado o el la dirección está mal especificada.

**Importancia** Alta

**Comentarios** En el caso de no estar disponible la conexión, el sistema permite trabajar de manera transparente al usuario con un servidor local a modo de pruebas que suministra lecturas reales del láser. La única condición es que dicho servidor debe ser previamente puesto en marcha.

**C2: Enviar mensaje al láser.**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** El sistema crea, codifica y envía un mensaje al láser solicitando el comienzo del envío de tramas

**Precondición** La conexión con el láser se ha realizado con éxito. Requiere el cumplimiento del caso de uso C1.

**Secuencia Normal:**

1. Se genera una lista con los caracteres adecuados para solicitar el envío de información continua desde el láser.
2. Se condifica la lista en un array de bytes.
3. Se envía la lista al láser a través del objeto de tipo socket.

**Postcondición** Se recibe la información de lectura o en su defecto de error del láser tras haber recibido la petición

**Excepciones** Se recibe un mensaje de error debido a la codificación inadecuada del mensaje peticionario.

**Importancia** Alta

**Comentarios**

### **C3: Recibir tramas del láser (de manera continua).**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** Se reciben los datos de lectura devueltos por el láser de manera continua e ininterrumpida y se almacenan para su posterior tratamiento más adelante en flujo del programa

**Precondición** El mensaje peticionario al láser debe estar formulado adecuadamente así como la conexión con el mismo. Deben cumplirse los casos de uso C1 y C2.

**Secuencia Normal:**

1. Se escucha la dirección especificada en el programa para recibir la información.
2. El láser envía las tramas de manera continua.
3. Se lleva un conteo de los bytes que se van recibiendo para separar en tramas la información.
4. El conteo de bytes recibido se resetea cuando se llega a la información máxima de cada trama para comenzar a procesar la trama siguiente.

**Postcondición** La información recibida se almacena hasta su posterior tratamiento.

**Excepciones** El sistema se bloquea en caso de desconexión del láser.

**Importancia** Alta

**Comentarios**

### **C4: Traducir las tramas.**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** Se realiza una traducción de cada trama recibida para facilitar su tratamiento posterior.

**Precondición** Se ha recibido la trama correctamente. Deben cumplirse los casos de uso C1, C2, y C3.

**Secuencia Normal:**

1. Los datos de lectura son particionados en función de los caracteres de inicio y fin de cada trama
2. Se descartan las divisiones vacías quedándose solo con la información relevante.
3. El sistema se queda con el segundo grupo de datos que son los que albergan la información de las lecturas, dado que el primer grupo se usa para datos de confirmación.
4. Se divide la información en grupos de 4 caracteres correspondiéndose con cada punto de cada trama enviada por el láser.
5. Dado que los puntos han sido devueltos ordenados en función su ángulo de captura de menor a mayor, se crea una lista de igual tamaño que la de los puntos con ángulos crecientes.
6. Se realiza una conversión de coordenadas polares a coordenadas cartesianas.

**Postcondición** Ha sido creada una lista de puntos en coordenadas cartesianas con su distancia al emisor en el eje de las X y su posición lateral respecto del láser en el eje de las Y

**Excepciones** En el caso de no realizarse correctamente la traducción los datos del sistema no guardarían relación con los percibidos por el láser.

**Importancia** Alta

**Comentarios** Se ha añadido la funcionalidad de limitar el ángulo que procesa el programa para eliminar ruido innecesario del entorno y centrarse en la parte frontal del AGV a la hora de buscar la detección de un palet.

#### **C5: Procesar los datos.**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** Se realiza un proceso de tratamiento mediante el algoritmo de detección sobre la nube de puntos de cada trama.

**Precondición** Se ha recibido la trama correctamente. Se ha traducido la trama correctamente. Deben cumplirse los casos de uso C3 y C4.

**Secuencia Normal:**

1. Se procesa la nube de puntos con un algoritmo de clustering y  $nclusters = 3$ .
2. Se calculan los puntos que componen cada cluster
3. Se calcula el punto que representa mejor a cada cluster (distancia y ángulo medio de los puntos del cluster)
4. Se aplica la ley del coseno para conocer la distancia real que separa cada cluster

**Postcondición** El algoritmo de clustering se ha ejecutado con éxito y se han realizado los cálculos de cantidad de puntos y distancia que separa los clusters

**Excepciones** En el caso de no haber seleccionado la zona en la que se encuentra el palet o obtener una lectura con mucho ruido los clusters pueden no corresponder a las patas del palet y efectuarse una detección errónea

**Importancia** Alta

**Comentarios** Para desarrollos futuros se pueden añadir comprobaciones más rigurosas a la hora de tratar la nube de puntos.

#### **C6: Tomar decisión de confirmación de palet.**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** En base a las mediciones realizadas en el caso de uso C5 se comparan con unas medidas fijas y entre si mismas para concluir la existencia o no de un palet en la nube de puntos

**Precondición** Se han ejecutado las mediciones sobre los clusters correctamente

**Secuencia Normal:**

1. Se compara el número de puntos que compone cada cluster para comprobar que sean de tamaño similar.
2. Se comprueba que la distancia que separa ambas patas concuerde con la del tipo de palet que se esté utilizando y sea similar entre sí.
3. Se emite el veredicto de la detección del palet.

**Postcondición** Se ha informado de la existencia o no existencia del palet en la trama procesada

**Excepciones** En condiciones desfavorables la detección de palets puede no ser adecuada.

**Importancia** Alta

**Comentarios**

#### **C7: Imprimir las tramas.**

**Versión** 1.0

**Autor** Mario Flores Espiga

**Descripción** Se muestran de manera gráfica y continua las tramas que el láser esta observando se detecte o no un palet.

**Precondición** Se ha obtenido la trama y procesado adecuadamente. Requiere el cumplimiento de los casos de uso C1 hasta C5.

**Secuencia Normal:**

1. Se particionan los puntos en dos listas correspondientes a cada eje de coordenadas.
2. Se introduce en la gráfica la nube de puntos.
3. Se actualiza de forma periódica los puntos de cada nueva trama.

**Postcondición** Se imprimen las gráficas de manera periódica con los puntos de cada trama.

**Excepciones** La función encargada de actualizar los datos de cada trama puede fallar.

**Importancia** Baja

**Comentarios**

#### **Actores**

Como único actor se plantea el usuario final de la aplicación, en concreto, la persona encargada de poner en marcha el sistema en un AGV. Interactúa directamente con el sistema a través de línea de comandos o un IDE.

#### **Diagrama de casos de uso**

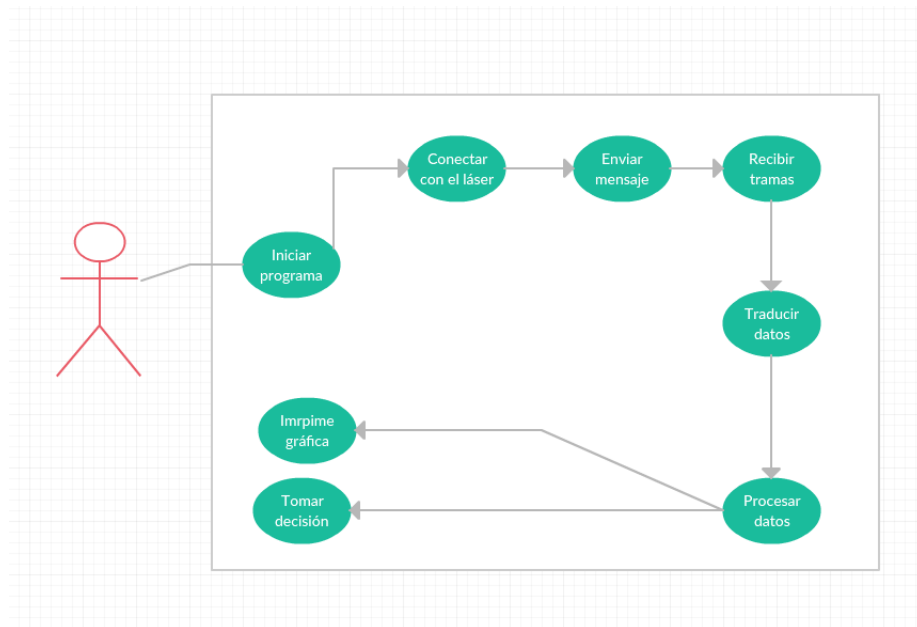


Figura B.1: Diagrama de casos de uso.

## *Apéndice C*

---

# Especificación de diseño

---

### C.1. Introducción

En este anexo se incluyen todos los aspectos de diseño que componen el proyecto.

Define los datos que va a utilizar el sistema, el diseño de la arquitectura, etc. Se divide en diseño de datos, diseño procedimental y diseño arquitectónico.

### C.2. Diseño de datos

La aplicación está compuesta por las siguientes entidades:

- **objectDetectorViaLaser:** En este archivo se encuentra la clase raíz del sistema (clase Operations).  
En ella se maneja la conexión con el láser, la traducción de los datos y el procesamiento de los mismos. Se compone de los siguientes métodos:
  - **mainFunction:** Función principal que codifica la conexión con el láser y envío y recepción de mensajes.
  - **procesadoYMuestra:** Esta función agrupa las técnicas de traducción, clustering y procesamiento de las tramas del láser.
  - **realClusterAngles:** Calcula el ángulo medio de todos los puntos de cada cluster.
  - **realClusterDistances:** Calcula la distancia media a todos los puntos de cada cluster.

- `betweenClusterDistances`: Calcula la distancia real que separa a los agrupamientos.
  - `findIndex`: Encuentra el índice del punto que se le pasa como parámetro en la lista que también recibe como parámetro.
  - `clusterSize`: Computa el total de puntos que pertenecen a cada cluster.
  - `creaAngulos`: Función para generar una lista de ángulos que se utiliza en el proceso de traducción de coordenadas.
  - `aCartesianos`: Función encargada de la traducción de coordenadas polares a cartesianas.
  - `toDecimal`: Encargada de realizar el cambio de base hexadecimal a decimal de las tramas enviadas por el láser.
- **Punto**: Esta es la clase que codifica los puntos. Las lecturas del láser son creadas como puntos en la etapa de traducción, encapsulando la información mediante las dos coordenadas que representan a un punto en el espacio 2D. Sus funciones son simplemente getters y setters para las variables de la clase, las cuales codifican las dos coordenadas que identifican a un punto 2D en las coordenadas cartesianas.
  - **DummySV**: Servidor de pruebas para poder usar el programa en momentos de desconexión con el láser.
  - **GUI**: Aquí se encuentra la clase encargada del apartado gráfico del sistema (clase `MyGUI`).
    - `animate`: Función encargada de realizar la actualización en tiempo real de valores en la gráfica.
    - `start`: Esta función es la encargada de iniciar la representación de los gráficos por primera vez.

Los datos que se tratan en todo el programa son coordenadas cartesianas, encapsuladas como ya se ha explicado con la clase `Punto`, y los ángulos, los cuales se almacenan en una lista.

### C.3. Diseño procedimental

El sistema sigue el proceso que se detalla en el siguiente diagrama de secuencia:



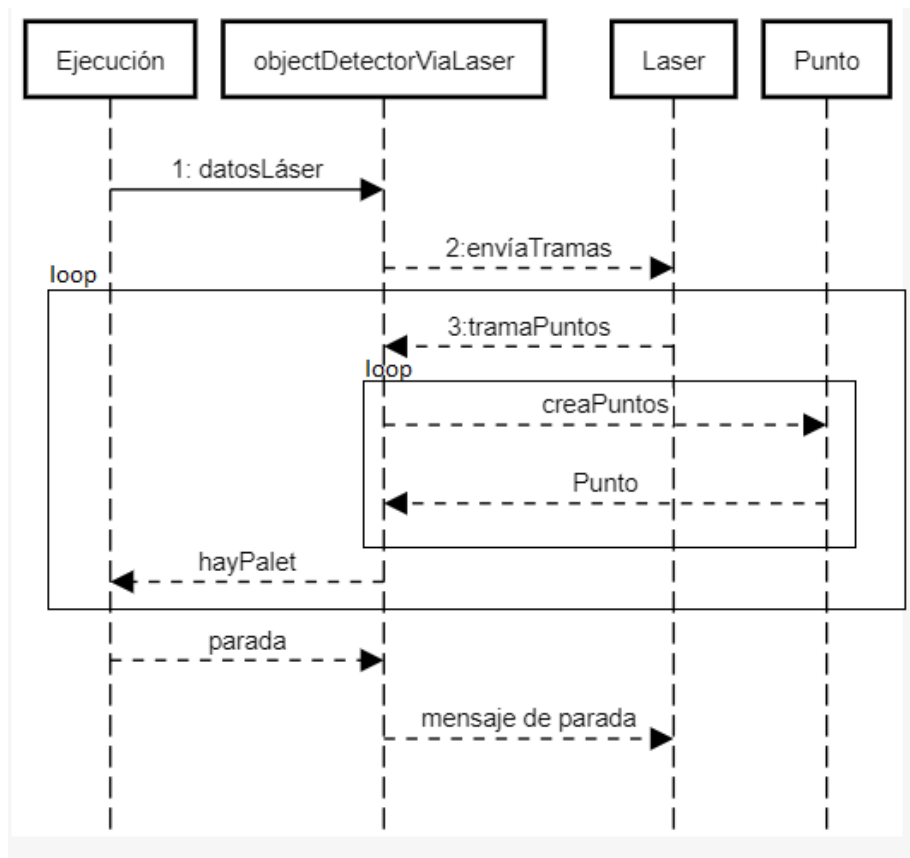


Figura C.1: Diagrama de secuencia de la ejecución del programa

El usuario final se encarga de lanzar el programa el cual después de eso no necesita ninguna otra intervención por parte del usuario.

## C.4. Diseño arquitectónico

El diseño arquitectónico del proyecto es sencillo al tener como objetivo crear un proceso por el cual se transforman las lecturas del láser en un veredicto sobre la existencia de un palet, por lo tanto, no se ha necesitado hacer uso de interfaces, jerarquías de herencia u otros métodos.

Se sigue un diseño como ya se ha explicado en la memoria utilizando un patron MVC modelo vista controlador [2]. Se decidió que era un modelo bastante apropiado al sistema que se iba a diseñar, pudiendo separar el programa en las tres capas claramente diferenciadas:

- Modelo: Clase punto que codifica las coordenadas de los puntos que transmite el láser.
- Vista: Clase MyGUI encargada de encapsular todo lo relacionado con la impresión de las gráficas.
- Controlador: Clase Operations delegada de todos los procesos de conexión, traducción y transformación de los datos.

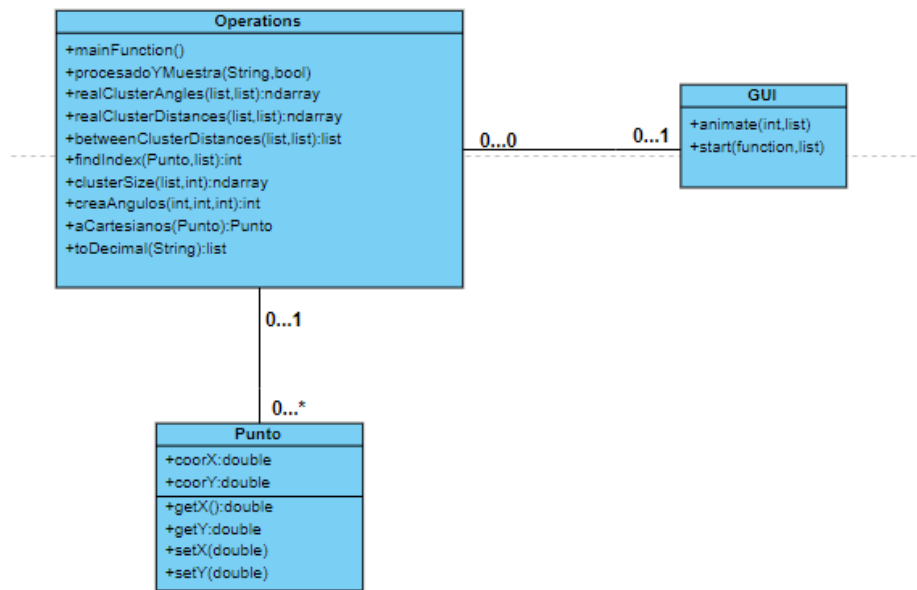


Figura C.2: Diagrama de clases del sistema

Existe la clase como ya se ha visto DummySV a parte de las representadas en el diagrama que se usa para realizar pruebas.

## *Apéndice D*

---

### Documentación técnica de programación

---

#### D.1. Introducción

En este anexo se explica los pasos necesarios para poder modificar el código de cara a futuros programadores.

Este proyecto está disponible en GitHub [aquí](#).

#### D.2. Estructura de directorios

La estructura de directorios es bastante sencilla y se desglosa a continuación:

- memoria: En este directorio se encuentran todos los archivos e imágenes que conforman la memoria y sus anexos.
- source: Carpeta donde se encuentra el código fuente. Los archivos están codificados en Python y representan la versión final del código.
- diagramas: Como su nombre indica, aquí se guardan los diagramas que se han utilizado para desarrollar el proyecto.
- Drivers: Contiene el driver utilizado para la conexión con el láser vía USB.

- Manuales del equipo laser: Directorio que alberga los manuales de uso y especificaciones del laser y su conexión que se han usado para comprender su funcionamiento.

## D.3. Manual del programador

En este apartado se explican los pasos a seguir para desplegar el entorno de desarrollo del proyecto y su configuración.

### Python

Se debe descargar el lenguaje de programación python desde su [página web](#).

Se ha utilizado la última versión disponible hasta la fecha, la 3.6.3.

Una vez se tiene instalado el lenguaje, se procede a descargar las librerías que se utilizan a lo largo del proyecto en el caso de ejecución desde línea de comandos, puesto que en el IDE Spyder vienen instaladas por defecto. Haciendo uso de los siguientes comandos se pueden instalar las diferentes librerías:

- `pip install matplotlib`
- `pip install sklearn`
- `pip install numpy`

### IDE

Con el fin de modificar y depurar el código de una manera más sencilla e intuitiva, se puede descargar un IDE que nos ayude con estas tareas. El IDE escogido para el desarrollo del proyecto por razones ya explicadas en el apartado 3 de la memoria es Spyder

Para descargar Spyder instalamos la aplicación de Anaconda Navigator, la cual incorpora a ambos contenedora del IDE Spyder [aquí](#)

## Git

El control de versiones del proyecto se ha realizado con este sistema. Aunque no se ha utilizado en el desarrollo mediante la línea de comandos, para poder usarlo de este modo se requiere su instalación. La descarga se puede realizar desde la siguiente [URL](#)

## GitHub Desktop

Se ha elegido este cliente de Git desde el que se han realizado las subidas al repositorio. Es un entorno fácil de aprender y suficiente para los cometidos del proyecto. Se descarga desde su [página web](#)

Una vez instalado, nos logueamos con nuestra cuenta de GitHub y clonamos el proyecto en nuestro ordenador.

Esto se puede realizar de varias maneras:

Introduciendo el enlace del repositorio en nuestro cliente de GitHub, lo cual luego nos permitirá clonarlo eligiendo un directorio donde queremos que se copie. Primero hacemos click en 'File'

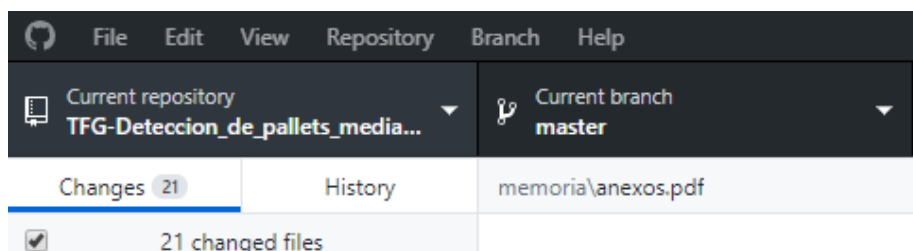


Figura D.1: Menú 'File'

Después en 'clone repository'

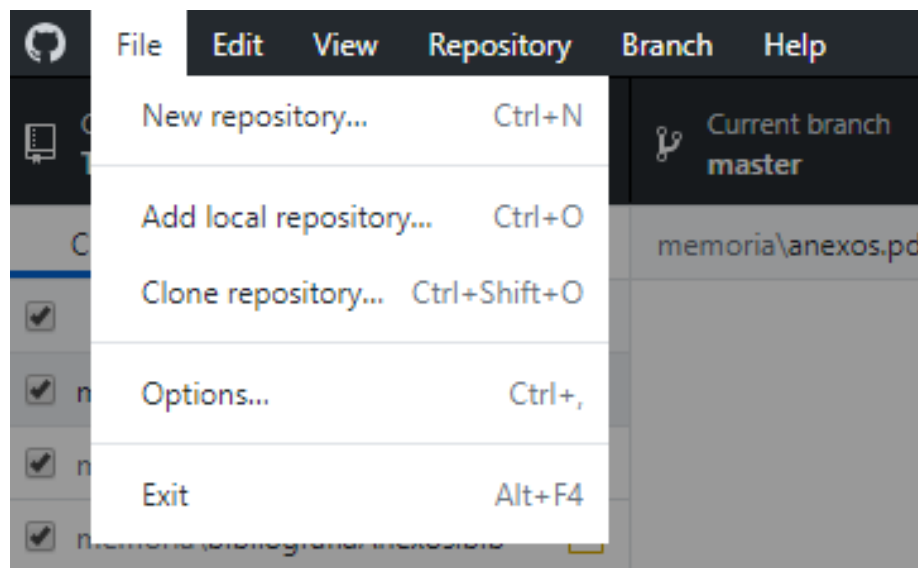


Figura D.2: 'Clone Repository'

Y por último vamos a la pestaña URL, copiamos el enlace del repositorio y elegimos la ubicación para que sea clonado.

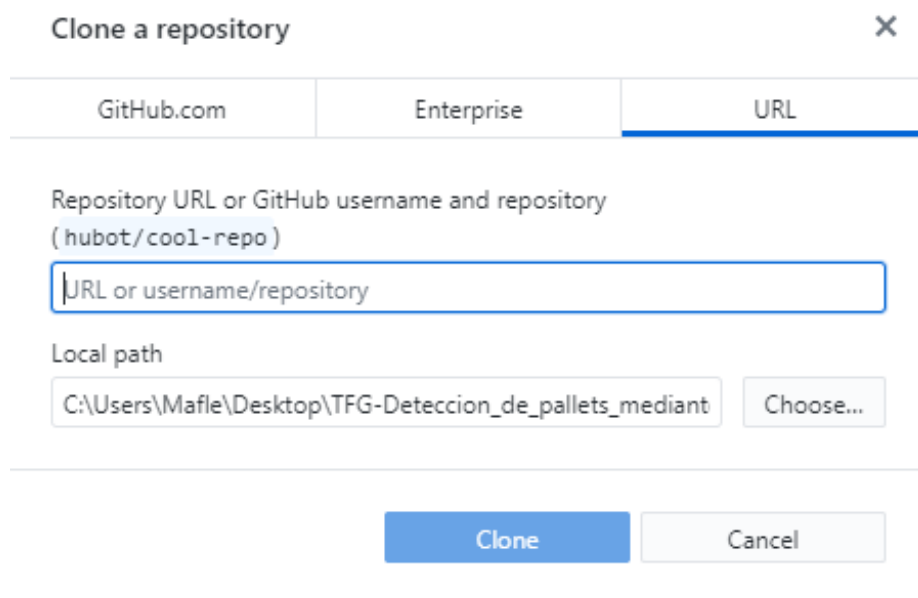


Figura D.3: URL del repositorio a clonar

La otra forma es descargando el repositorio desde la página web de GitHub y seleccionando la opción en el cliente 'Add local repository' como se ve en las siguientes imagenes.

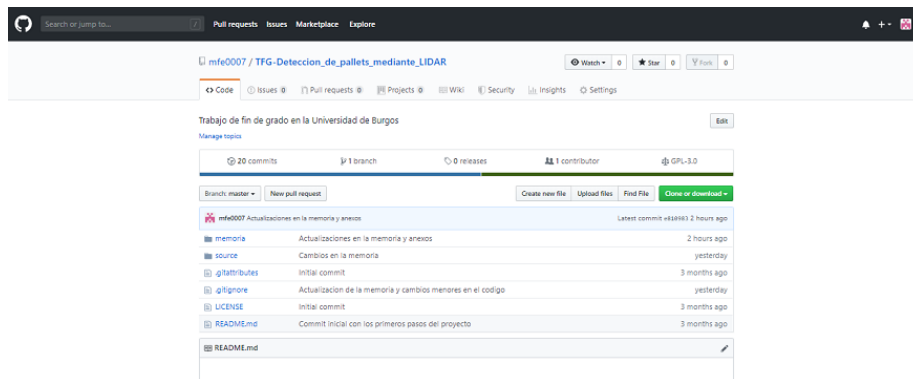


Figura D.4: Descarga del repositorio

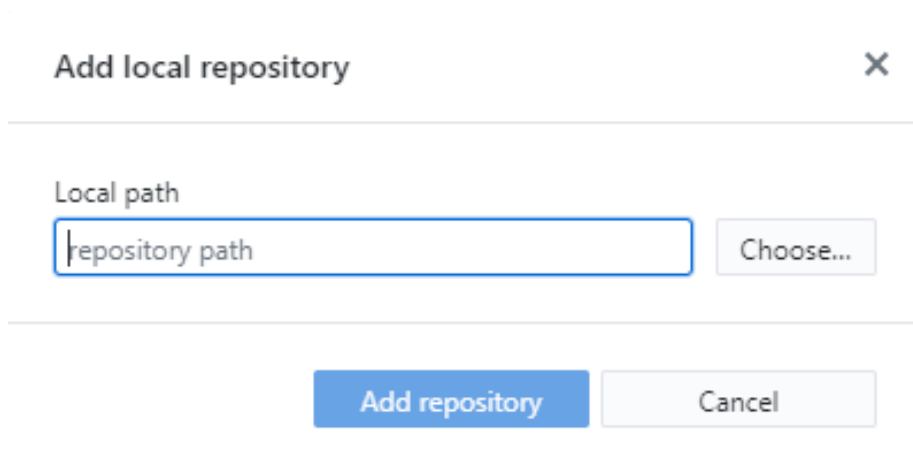


Figura D.5: Búsqueda del directorio

Después de realizar cualquiera de estos dos pasos ya es posible el uso del proyecto.

## D.4. Compilación, instalación y ejecución del proyecto

Una vez se han realizado los pasos anteriores, ya podemos compilar y ejecutar el proyecto, puesto que no es necesaria la instalación de ningún paquete ni dependencia del propio proyecto.

Se puede ejecutar de dos maneras distintas:

1. Mediante el CMD en Windows: Abrimos el *Símbolo del sistema*, nos movemos al directorio en el que se encuentran los ficheros fuentes y escribimos lo siguiente: **objectDetectorViaLaser.py**
2. Mediante Spyder: Una vez en el IDE, abrimos el archivo `objectDetectorViaLaser.py` con el explorador de archivos y lo ejecutmos mediante el menú.

## D.5. Pruebas del sistema

Durante el proyecto se han realizado diferentes pruebas para ayudar a detectar fallos y solucionarlos, avanzar en el desarrollo del programa o incluso comprender el funcionamiento de diferentes partes del láser. En esta sección se explican las pruebas que se han ido realizando y los resultados de las mismas.

En esre apartado no se incluyen las pruebas realizadas con el servidor de testeo puesto que han sido pruebas rápidas para la depuración del código y su formalización habría supuesto una notable carga de trabajo.

### Prueba 1: Uso de RealTerm para visualizar la perspectiva del láser

**Descripción** Estas pruebas han sido de utilidad para escoger la ubicación óptima del láser respecto del palet en las etapas tempranas donde el código no era operativo. Se crea un mensaje para que el láser comience a enviar informacion y se le envía: *2,48,48,48,69,65,82,48,50,0x00,0x83,3*. **Entorno** Se realiza la conexión entre el láser y el ordenador utilizando un cable ethernet. Se conecta el láser a la fuente de alimentación para su funcionamiento. Se inicia el programa RealTerm.



**Resultado esperado** Se reciben tramas en bucle de lo que el láser este visualizando. Se espera que funcione puesto que ya ha sido probado por otro alumno previamente en su TFG.

**Resultado obtenido** Se reciben las tramas y se almacenan para su posterior análisis

**Conclusiones** Es necesaria la traducción para poder comprender las tramas

**Acciones derivadas** Se codifican las tramas con el formato necesario para más adelante poder ser interpretadas por el código.

## Prueba 2: Comprobación de las tramas en las que aparece un palet

**Descripción** Para poder desarrollar el algoritmo de la mejor manera posible, se despliega un entorno de pruebas con un palet real de tipo europeo en el laboratorio. Se coloca el láser en el mismo plano que el palet y enfrentándose directamente a este separado por una distancia de 2 metros aproximadamente. Se realizan varias pruebas a diferentes distancias y variaciones del ángulo del palet respecto del láser, pero no se incluyen por ser de la misma índole que la aquí descrita. **Entorno** Se realiza la conexión entre el láser y el ordenador utilizando un cable ethernet. Se conecta el láser a la fuente de alimentación para su funcionamiento. Se crea un pequeño programa para hacer logs de las tramas y separarlas entre sí e imprimir después las gráficas de las nubes de puntos obtenidas

**Resultado esperado** Se visualiza claramente el palet en las gráficas

**Resultado obtenido** Se observa gran cantidad de ruido, aunque se aprecia el palet centrado y a la distancia concreta a la que se ha ido separando del láser. También se comprueba las medidas entre los distintos elementos del palet.

**Conclusiones** Se debe buscar una forma de eliminar el ruido que aparece en la nube de puntos y hace la tarea de detección mucho más compleja

**Acciones derivadas** Se codifica una limitación del ángulo que se procesa en el programa para quedarse con la parte frontal de la captura y poder aislar los puntos que representan al palet. Esto se ha realizado en base a que el algoritmo no debe detectar un palet sin ningún conocimiento previo, sino que se apoya en el conocimiento de la existencia del palet en un área concreta y debe verificar su posición y distancia.

## Apéndice *E*

---

# Documentación de usuario

---

## E.1. Introducción

En este manual se explica cómo instalar la aplicación y cómo utilizarla correctamente.

## E.2. Requisitos de usuarios

Para poder utilizar la aplicación, se debe tener un hardware específico si lo que se quiere es realizar lecturas con el equipo láser. Existe la posibilidad de trabajar en modo local sin necesidad de la conexión con el láser, utilizando el servidor de prueba que se suministra con varias tramas, no obstante, no es el objetivo final de la aplicación sino un paso intermedio para su correcto desarrollo.

Se necesita el siguiente material:

- Un ordenador en el que poder ejecutar el programa. Es necesario que disponga de un puerto ethernet. Debe cumplir ciertos requerimientos software que se detallan en el apartado instalación ???. Para la ejecución del IDE Spyder contenido en el entorno Anaconda se recomienda cómo **mínimo**:
  - Sistema operativo: Windows 7 or superior, 64-bit macOS 10.10+, o Linux, incluyendo Ubuntu, RedHat, CentOS 6+, y otros.
  - Disco duro: Mínimo 5GB de espacio libre para la descarga e instalación.

- Memoria: 4GB de memoria RAM
- El equipo láser para realizar las lecturas de puntos. El modelo utilizado para el desarrollo (Hokuyo Safety Laser Scanner /UAM-05LP-T301).
- Fuente de alimentación DC de 24v necesaria para suministrar al láser de energía eléctrica continua.
- Cable de red Ethernet para efectuar el intercambio de mensajes con el equipo láser.

### E.3. Instalación

Para poder ejecutar el programa es necesario haber instalado antes los siguientes elementos software:

- Python
- Bibliotecas necesarias (Sklearn, matplotlib, numpy)
- Opcional: Un entorno de desarrollo integrado. Se recomienda Spyder.

Las instrucciones de instalación de estos componentes se encuentran detalladas en el Anexo D - Manual del programador ??

### E.4. Manual del usuario

Para comenzar la ejecución de la aplicación, una vez completados los pasos de instalación, tenemos dos opciones disponibles: la ejecución desde línea de comandos o la ejecución desde un IDE compatible con Python.

No es necesaria la modificación de ningún parámetro por parte del usuario, en sistema viene configurado automáticamente.

El sistema genera una gráfica que se va actualizando conforme recibe las tramas del láser en tiempo real y devolviendo la decisión sobre la existencia o no de palet en la trama actual en pantalla.

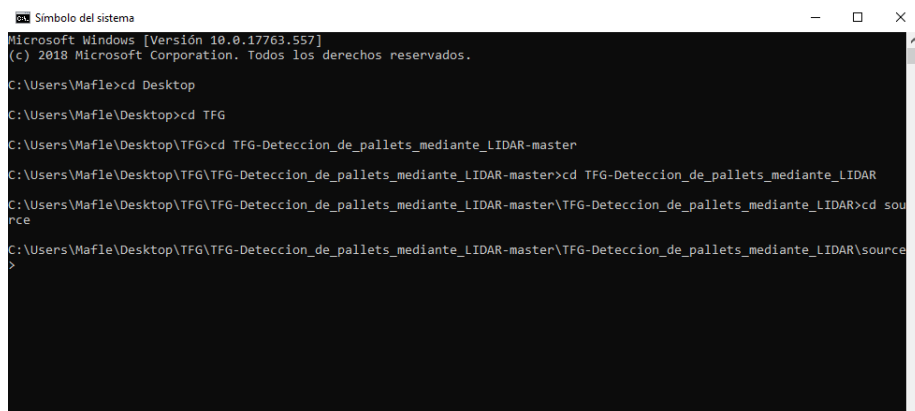
#### Ejecución desde la línea de comandos

Primero debemos abrir una terminal (cmd) y navegar hasta la ruta que contiene los ficheros Python (source).

Una vez estemos en ese directorio y asegurándose de que el equipo láser esté

encendido y conectado, introducir la siguiente línea en el cmd y ejecutarla: *objectDetectorViaLaser.py*.

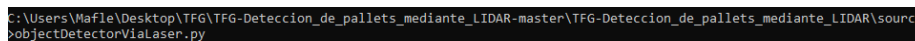
En el caso de querer trabajar con el servidor local, es necesario previo al anterior paso ejecutar la siguiente línea en otra terminal a parte: *DummySV.py*.



```
Símbolo del sistema
Microsoft Windows [Versión 10.0.17763.557]
(c) 2018 Microsoft Corporation. Todos los derechos reservados.

C:\Users\Mafle>cd Desktop
C:\Users\Mafle\Desktop>cd TFG
C:\Users\Mafle\Desktop\TFG>cd TFG-Deteccion_de_pallets_mediante_LIDAR-master
C:\Users\Mafle\Desktop\TFG\TFG-Deteccion_de_pallets_mediante_LIDAR-master>cd TFG-Deteccion_de_pallets_mediante_LIDAR
C:\Users\Mafle\Desktop\TFG\TFG-Deteccion_de_pallets_mediante_LIDAR-master\TFG-Deteccion_de_pallets_mediante_LIDAR>cd source
C:\Users\Mafle\Desktop\TFG\TFG-Deteccion_de_pallets_mediante_LIDAR-master\TFG-Deteccion_de_pallets_mediante_LIDAR\source>
```

Figura E.1: Navegación al directorio 'source'



```
C:\Users\Mafle\Desktop\TFG\TFG-Deteccion_de_pallets_mediante_LIDAR-master\TFG-Deteccion_de_pallets_mediante_LIDAR\source>objectDetectorViaLaser.py
```

Figura E.2: Ejecución del archivo

## Ejecución desde un IDE

Existen multitud de aplicaciones compatibles con Python. En este proyecto se ha trabajado con Spyder.

Una vez instalada la aplicación deseada siempre y cuando sea compatible, abrimos en el IDE los archivos necesarios (*objectDetectorViaLaser.py* y *DummySV.py* en el caso de querer trabajar en local) y lanzamos el primero de ellos.

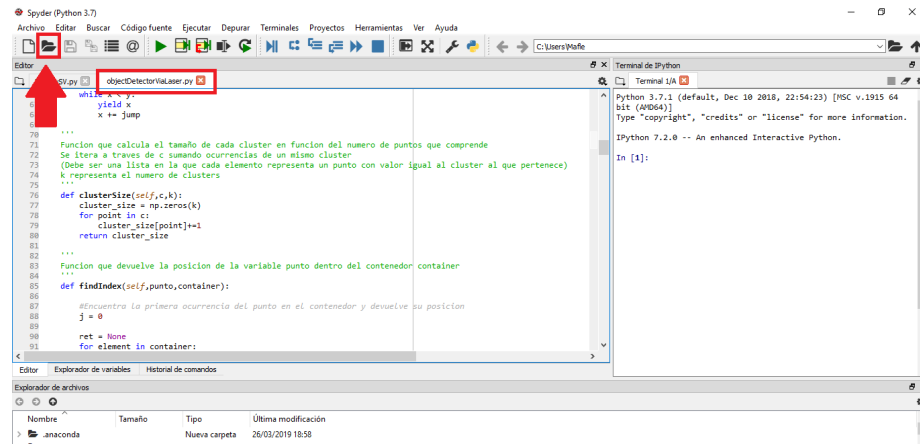


Figura E.3: Apertura de los archivos

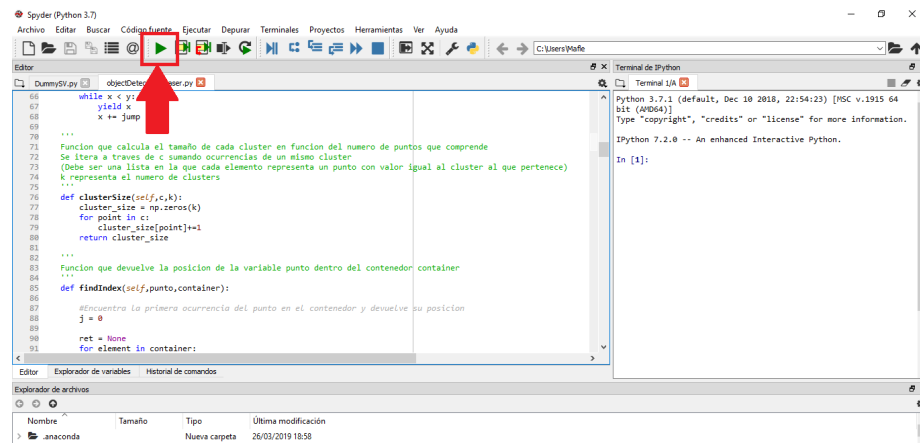


Figura E.4: Ejecución de la aplicación

---

## Bibliografía

---

- [1] Wikipedia. Gnu general public license — wikipedia, la enciclopedia libre, 2005. [Internet; descargado Junio-2019].
- [2] Wikipedia. Modelo-vista-controlador — wikipedia, la enciclopedia libre, 2006. [Internet; consultado Abril-2019].