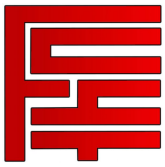
UNIVERSIDAD MAYOR DE SAN SIMÓN
FACULTAD DE CIENCIAS Y TECNOLOGÍA
CARRERA DE INGENIERÍA DE SISTEMAS



DESARROLLO DE UN SISTEMA ...

PROYECTO DE GRADO, PRESENTADO PARA OPTAR
AL DIPLOMA ACADÉMICO DE LICENCIATURA
EN INGENIERÍA DE SISTEMAS.

PRESENTADO POR: Ronald Alejandro Oquendo Muñoz

TUTOR: Lic. Fernando ...

COCHABAMBA - BOLIVIA
OCTUBRE, 2015

II

Dedicado a
...

La realización de este proyecto de grado
blah blah blah blah blah blah blah blah
blah blah blah blah blah blah blah blah
blah blah blah blah blah blah blah blah
blah blah blah blah blah blah blah blah

Ficha resumen

El presente proyecto de grado trata del blah.

Índice general

Índice de figuras

Índice de cuadros

Capítulo 1

Introducción

1.1. Antecedentes

La visión artificial es un ...

1.2. Identificación del problema

blah blah.

1.2.1. Definición del problema

blah blah.

1.3. Objetivos

1.3.1. Objetivos de la aplicación

- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.

1.3.2. Objetivos metodológicos

Los objetivos a cumplir durante el desarrollo del proyecto son:

- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.

1.4. Alcance

El proyecto tendrá el siguiente alcance:

- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.
- blah blah blah blah.

1.5. Justificación

blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

Capítulo 2

Visión Artificial

2.1. Introducción

blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah.

2.2. Componentes de un sistema de visión

La visión artificial es un tema complejo, por lo tanto es necesario dividirlo en varios componentes:

- **Una fuente de radiación** puede ser la luz visible. La luz visible esta formada por radiación electromagnética cuyas longitudes de onda están comprendidas entre 390 y 700 nm. [18] Si la radiación no es emitida desde la escena o del objeto interés, nada puede ser observado o procesado. Por lo tanto es necesaria una apropiada iluminación para objetos que por si mismos no son luminosos.

- **Una cámara** obtiene la proyección plana de una imagen externa sobre el sensor.[17] En el caso mas simple la cámara solo estaría compuesto de lentes ópticos.

- **El sensor** convierte la radiación colectada en señales eléctricas para luego procesarlas.

- **La unidad** de procesamiento procesa las señales colectadas, extrayendo características adecuadas que pueden ser usadas para medir las propiedades de un objeto y categorizarlo en clases.

- **Los actores** reaccionan al resultado de la observación visual. Ellos llegan a ser una parte integral del sistema de visión cuando el sistema de visión esta respondiendo activamente a la observación, por ejemplo, rastreando un objeto de interés o usando navegación guiada por visión.

2.3. Aplicaciones de la visión artificial

2.4. Problemas típicos en la visión artificial

2.5. Conclusiones

blah blah.

Capítulo 3

Herramientas

3.1. Raspberry Pi

Raspberry Pi es una computadora de placa de bajo costo que corre GNU/Linux y otros sistemas operativos. Existen dos modelos el A y B, el modelo B tiene la revisión 1 y 2, y además de una nueva versión que es B+.

3.1.1. Componentes

- Broadcom BCM2835, SoC que contiene memoria, microprocesador y procesador gráfico.
- Conector GPIO (Entradas y salidas de propósito general).
- Conector salida de video compuesto
- Conector salida de video HDMI
- LEDs de actividad de LAN y Power
- Conector de salida de audio
- Conector microusb para alimentación
- Circuito integrado para LAN
- Conectores de expansión (para cámara de video y salida de video)
- Conectores USB
- Conector RJ45 para Ethernet
- Reguladores de voltaje de 1.8v, 2.8v y 3.3v

3.2. Conclusiones

blah blah.

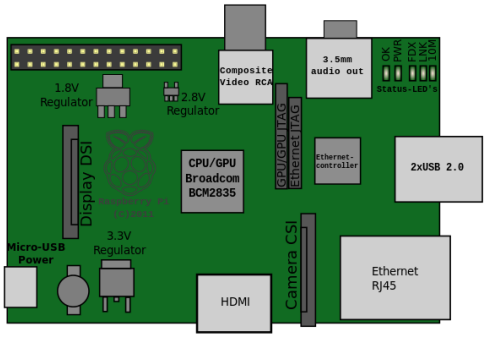


Figura 3.1: Ubicación de los componentes de Raspberry Pi. [29]

Pin	Descripción	Pin	Descripción
1	3.3v	2	5v
3	SDA0*	4	5v
5	SCL0*	6	GND
7	GPIO_GCLK	8	TXD0*
9	GND	10	RXD0*
11	GPIO_GEN0	12	GPIO_GEN1
13	GPIO_GEN2	14	GND
15	GPIO_GEN3	16	GPIO_GEN4
17	3.3v	18	GPIO_GEN5
19	SPI_MOSI*	20	GND
21	SPI_MISO*	22	GPIO_GEN6
23	SPI_SCLK*	24	SPI_CE0_N*
25	GND	26	SPI_CE1_N*

Cuadro 3.1: Descripción de los pines de GPIO

Capítulo 4

Metodología

4.1. Pila del producto (product backlog)

En el presente proyecto se utilizará la metodología de desarrollo de software SCRUM [26].

La siguiente tabla es una lista priorizada de requisitos (features) para ser implementadas en el proyecto. Cada una de estas caracteristicas se escribieron de acuerdo a los objetivos específicos, que fueron definidos en el Capitulo 1.

N°	Características (features)	Prioridad
1	Como robot quiero ser capaz de reconocer objetos existentes en un fotograma, con el objetivo de tener estos datos para calcular su posición en el plano.	Alta
2	Como robot quiero saber a que profundidad en el plano se encuentran los obstáculos, con el objetivo de poder esquivarlos sabiendo su posición.	Alta
3	Como robot debo ser capaz de esquivar los obstáculos que se presentan en el camino del robot, con el objetivo de hacerlo de forma autónoma.	Alta
4	Como estudiante debo implementar una plataforma de hardware que sirva de base para el robot, con el objetivo de tener un robot operativo.	Alta
5	Como estudiante debo escribir la documentación del proyecto de grado, con el objetivo de tenerlo completado.	Media
6	Como robot quiero saber mi inclinación actual, con el objetivo de saber si estoy en peligro de volcarme.	Baja

4.2. Estimación de historias de usuario

La estimación de esfuerzo de las historias de usuario se realizaran utilizando la técnica de Planning Poker. Se utilizara el rango de 1, 2, 3, 5, 8, 13.

4.3. Sprint A

El sprint A empieza el día Viernes 26 de Septiembre del 2014. Este sprint consta de 2 semanas para completar las historias que serán definidas en la planificación del sprint. El sprint concluirá el Jueves 9 de Octubre del 2014.

4.3.1. Sprint backlog

En este sprint se eligió trabajar en los features 1 y 2 del product backlog, por el hecho de tener alta prioridad. Cada feature se divide en historias de usuario, y estas son estimadas con puntos de historia. En los Cuadros 4.2 y 4.3 se definen las estimaciones. Para cada historia de usuario se definen criterios de aceptación, como se muestra en el Cuadro 4.4.

Cada historia de usuario se divide en tareas, y se definen en los Cuadros 4.5, 4.6 y 4.7.

No.	Feature	Id.	Historia de usuario	Estimación
1	Como robot quiero ser capaz de reconocer objetos existentes en un fotograma, con el objetivo de tener estos datos para calcular su posición en el plano.	US1	Obtener coordenadas de objetos reconocidos.	3

Cuadro 4.2: Estimación de US1

No.	Feature	Id.	Historia de usuario	Estimación
2	Como robot quiero saber a que profundidad en el plano se encuentran los obstáculos, con el objetivo de poder esquivarlos sabiendo su posición.	US2	Controlar la posición de la cámara.	3
		US3	Obtener distancia aproximada de los objetos reconocidos.	8

Cuadro 4.3: Estimación de US2 y US3

4.3.2. Reconocimiento de objetos utilizando OpenCV

El sistema de reconocimiento de objetos ejecuta una serie de pasos antes de enviar un fotograma al reconocimiento de blobs. Primero se ejecuta `dilate()`, para remover pequeños puntos en la imagen, que no son relevantes para este caso. Segundo se ejecuta `Canny()`, que devuelve una imagen con bordes delgados y finos. Una vez ejecutados estos pasos se envía la imagen resultado a `CBlobresult()`, que devuelve los blobs detectados en la imagen. Todo este proceso se puede apreciar en la Figura 4.1.

El resultado de `CBlobresult()` es un conjunto de blobs. Cada elemento del conjunto contiene datos como ser: el área, coordenadas de la posición, dimensiones, etc.

Id.	Criterios de aceptación
US1	El software a implementar debe ser capaz de reconocer las coordenadas de un objeto en tiempo real, pueden ser de uno o varios objetos captados en el fotograma.
US2	Dando como entrada el ángulo con respecto al eje vertical, la cámara debe ser capaz de posicionarse al ángulo de entrada establecido.
US3	Ya teniendo las coordenadas del objeto, el software a implementar debe ser capaz de devolver la distancia que existe entre un objeto y el robot.

Cuadro 4.4: Criterios de aceptación del sprint A

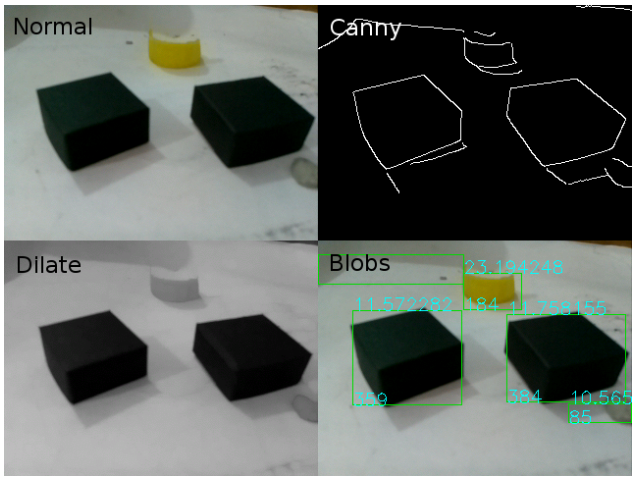


Figura 4.1: Reconocimiento de objetos utilizando OpenCV. Elaboración propia.

4.3.3. Control del servomotor de la cámara

La cámara estará montada a un servomotor y este al robot, de manera que será posible mover la cámara en el eje vertical como se muestra en la Figura 4.2.

Ya teniendo al servomotor y a la cámara montados, es necesario poder controlarlos. El servomotor recibe números para poder moverlo, la forma de poder enviar números es mediante un archivo especial llamado `/dev/servoblaster`. ServoBlaster es una interfaz para controlar servomotores mediante los pines de GPIO. Los números que recibe ServoBlaster son ancho de pulsos, luego estos datos son transmitidos mediante el GPIO hasta el servomotor. Para poder mover la cámara a un cierto ángulo con respecto al eje vertical, es necesario transformar los anchos de pulsos a grados, para esta transformación se utilizará la técnica de interpolación de Lagrange [23].

$$p(x) = \sum_{i=0}^{n} \left(\prod_{\substack{0 \le j \le n \\ j \neq i}} \frac{x - x_j}{x_i - x_j} \right) * y_i \tag{4.1}$$

Se utilizaran tres puntos, que cada uno es el ángulo en grados (con respecto al eje vertical como se muestra en la Figura 4.2) con su correspondiente ancho de pulso. Estos puntos se muestran en el Cuadro 4.8.

US1	Obtener coordenadas de objetos reconocidos
Id.	**Tareas**
T1	Preparar estructura simple de archivos para el sistema
T2	Investigar acerca OpenCV
T3	Configurar e instalar OpenCV en Raspberry Pi y en la laptop de trabajo
T4	Investigar acerca de OpenCVBlobsLib
T5	Investigar acerca de filtros de OpenCV para el POC
T6	Implementar POC usando OpenCVBlobsLib
T7	Escribir función que devuelva coordenadas de los blobs identificados
T8	Hacer pruebas de la implementación en Raspberry Pi

Cuadro 4.5: Tareas del US1

Luego reemplazando en 4.1 se obtiene la Ecuación 4.2.

$$p(x) = \frac{(x - x_1) * (x - x_2) * y_0}{(x_0 - x_1) * (x_0 - x_2)} + \frac{(x - x_0) * (x - x_2) * y_1}{(x_1 - x_0) * (x_1 - x_2)} + \frac{(x - x_0) * (x - x_1) * y_2}{(x_2 - x_0) * (x_2 - x_1)} \tag{4.2}$$

En la Ecuación 4.2, x representa el ángulo de entrada, y el resultado de $p(x)$ es el ancho de pulso. Con este dato ya es posible indicar al servomotor cuanto se debe mover para posicionar la cámara al ángulo x, que seria α en la Figura 4.2.

4.3.4. Casos de prueba

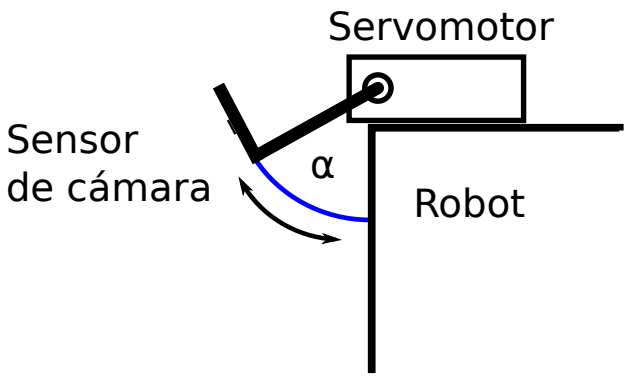


Figura 4.2: Cámara y servomotor montados al Robot. Elaboración propia.

US2	**Controlar la posición de la cámara.**
Id.	**Tareas**
T1	Fabricar soporte motorizado para cámara
T2	Implementar función que gira la cámara a una posición especifica
T3	Escribir y ejecutar prueba de unidad para el método que gira la cámara

Cuadro 4.6: Tareas del US2

US3	**Obtener distancia aproximada de los objetos reconocidos**
Id.	**Tareas**
T1	Deducir fórmulas para obtener la distancia de los objetos en la pantalla, de acuerdo con el ángulo de inclinación de la cámara.
T2	Implementar función que devuelva distancia de objetos.

Cuadro 4.7: Tareas del US3

Id.	CP1
Historia	US1
Nombre	Prueba de funcionamiento de ambiente de desarrollo
Descripción	Se realizarán pruebas de OpenCV de los ambientes de desarrollo de Raspberry Pi, para verificar que OpenCV y las herramientas de desarrollo estén correctamente instaladas
Ambiente de prueba	Raspberry Pi con Raspbian y OpenCV

	x	**y**
0	180	89
1	90	189
2	45	242

Cuadro 4.8: Conversión de grados a modulación por ancho de pulso

Inicialización	Encender Raspberry Pi
Acciones	■ Copiar `opencv-2.4.9.zip` al Raspberry Pi ■ Descomprimir `opencv-2.4.9.zip` ■ Entrar a opencv-2.4.9/samples/cpp ■ Ejecutar `$ g++ ‘pkg-config -libs -cflags opencv‘ houghlines.cpp -o houghlines` ■ Ejecutar `./houghlines`
Salida esperada	Dos ventanas gráficas con títulos “source” y “detected lines”. La segunda ventana tiene los bordes marcados con rojo.
Salida obtenida	En la Figura 4.3 se muestra la salida obtenida de Houghlines, que es una transformada usada para detectar lineas rectas.
Resultado	**Correcto**

Id.	CP2
Historia	US1
Nombre	Prueba de funcionamiento de ambiente de desarrollo en laptop de trabajo
Descripción	Se realizarán pruebas de OpenCV de los ambientes de desarrollo de la laptop de trabajo, para verificar que OpenCV y las herramientas de desarrollo estén correctamente instaladas.
Ambiente de prueba	Laptop de trabajo y OpenCV
Inicialización	Ninguna

Acciones	■ Encender laptop y entrar a la terminal. ■ Copiar `opencv-2.4.9.zip` a directorio. ■ Descomprimir `opencv-2.4.9.zip` ■ Entrar a opencv-2.4.9/samples/cpp ■ Ejecutar `$ g++ ‘pkg-config -libs -cflags opencv‘ houghlines.cpp -o houghlines` ■ Ejecutar `./houghlines`
Salida esperada	Dos ventanas gráficas con títulos “source” y “detected lines”. La segunda ventana tiene los bordes marcados con rojo.
Salida obtenida	En la Figura 4.4 se muestra la salida obtenida de Houghlines, que es una transformada usada para detectar lineas rectas.
Resultado	**Correcto**

Id.	CP3
Historia	US1
Nombre	Prueba de funcionamiento de `Blobs::findBlobs`
Descripción	Se realizarán pruebas de `Blobs::findBlobs`, para verificar que este método encuentra regiones (blobs) en una imagen dada.
Ambiente de prueba	Computadora con OpenCV, herramientas de desarrollo y código fuente del proyecto.
Inicialización	Ninguna
Acciones	■ Ingresar a la computadora de desarrollo ■ Ir al directorio sistema/unit_tests ■ Ejecutar la compilación de la prueba de unidad: `$ make test_find_blobs` ■ El anterior paso debería generar test_find_blobs ■ Ejecutar `./test_find_blobs`

Salida esperada	Una ventana gráfica con una foto de tres objetos negros, cada objeto con un recuadro verde.
Salida obtenida	En la Figura 4.5 se muestra la salida obtenida. El método `GetNumBlobs()` usado aquí, se ocupa de detectar objetos en una imagen dada.
Resultado	**Correcto**

Id.	CP4
Historia	US1
Nombre	Prueba de funcionamiento de coordenadas de `Blobs::findBlobs`
Descripción	Se realizarán pruebas para verificar que las coordenadas que devuelve `Blobs::findBlobs` sean correctas
Ambiente de prueba	Computadora con OpenCV, herramientas de desarrollo, código fuente del proyecto, y GIMP
Inicialización	Ninguna
Acciones	▪ Ingresar a la computadora de desarrollo ▪ Ir al directorio sistema/unit_tests ▪ Ejecutar la compilación de la prueba de unidad: `$ make test_find_blobs` ▪ El anterior paso debería generar test_find_blobs ▪ Ejecutar `./test_find_blobs` ▪ Verificar que todas las coordenadas listadas en la salida del programa sean las mismas que los recuadros verdes. Esto se puede realizar con la ayuda de GIMP.
Salida esperada	Salida en la interfaz de linea de comandos con las coordenadas marcadas en la ventana gráfica. En esta ventana debe mostrar una foto de tres objetos negros, cada objeto con un recuadro verde. (Es posible que se muestren otros recuadros verdes en otras regiones, esto es completamente normal.)

Salida obtenida	En la Figura 4.6 se muestra la salida obtenida. El método `GetNumBlobs()` también devuelve las coordenadas de los objetos detectados (blobs).
Resultado	**Correcto**

Id.	CP5
Historia	US2
Nombre	Prueba de posicionamiento de la cámara
Descripción	Se realizarán pruebas para verificar que la cámara este correctamente posicionada según el software implementado.
Ambiente de prueba	Raspberry Pi con OpenCV, cámara de Raspberry Pi, herramientas de desarrollo, código fuente del proyecto, y un transportador.
Inicialización	Conectar la cámara a Raspberry Pi. Marcar una hoja desde 40 grados hasta 180 grados.
Acciones	▪ Ingresar al Raspberry Pi ▪ Ir al directorio sistema/unit_tests ▪ Ejecutar la compilación de la prueba de unidad: `$ make test_servo` ▪ Verificar que el anterior paso genero test_servo ▪ Ejecutar `./test_servo` ▪ Con la hoja marcada, verificar que la cámara este correctamente alineada según los grados que se muestran en la pantalla.
Salida esperada	Los grados de la hoja deben coincidir con los grados mostrados en la salida del programa.
Salida obtenida	En la Figura 4.7 se muestra la salida obtenida. En la figura se puede obsevar a la cámara alineada a 90 grados.
Resultado	**Correcto**

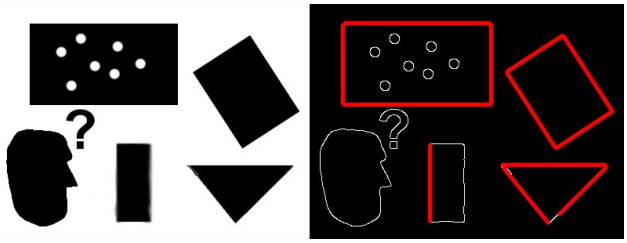


Figura 4.3: Salida de ejecución de Houghlines. Elaboración propia.

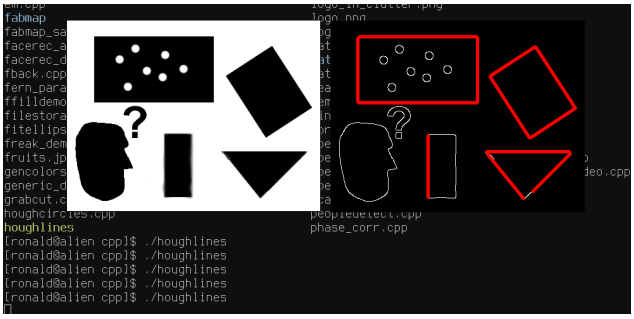


Figura 4.4: Salida de ejecución de Houghlines. Elaboración propia.

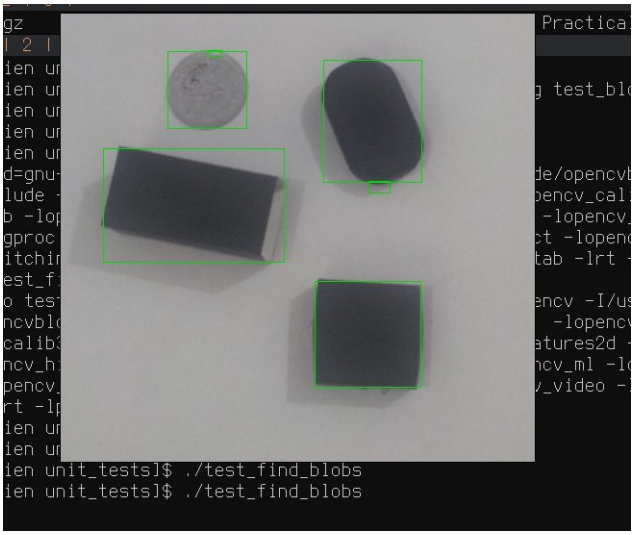


Figura 4.5: Salida de ejecución de `./test_find_blobs`. Elaboración propia.

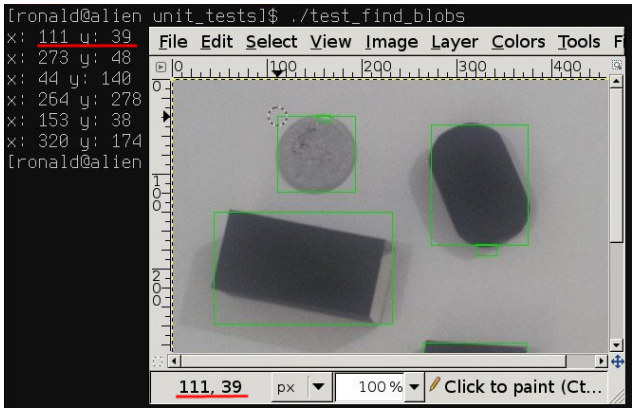


Figura 4.6: Salida de `./test_find_blobs` mostrando las coordenadas de los blobs. Elaboración propia.

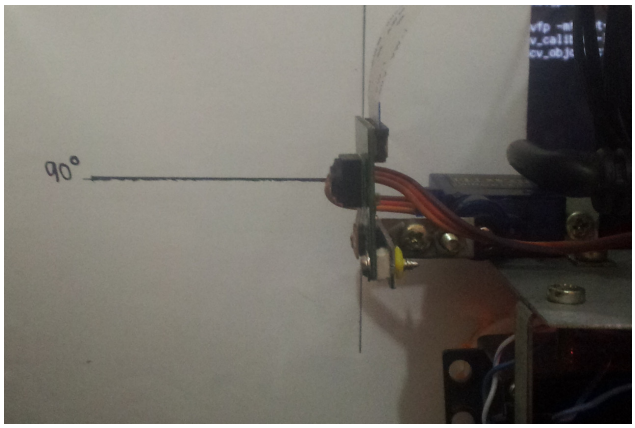


Figura 4.7: Cámara alineada a 90 grados. Elaboración propia.

4.3.5. Demostración de fin de sprint

El método `findBlobs()` encuentra blobs en una imagen dada (`im`). En la Figura 4.6 se observa las coordenadas de los objetos reconocidos.

Listado 4.1: Método para encontrar blobs en una imagen

```
CBlobResult Blobs::findBlobs(Mat im){
    Mat img, imgc, imgd;

    cvtColor(im,img,CV_BGR2GRAY);
    Mat st_elem = getStructuringElement(MORPH_RECT, Size(size_slider, size_slider));
    dilate(img, imgd, st_elem);

    Canny(imgd, imgc, canny_th1, canny_th2);

    return CBlobResult(imgc,Mat(),NUMCORES);
}
```

El método `addBlobToImg()` añade un rectángulo al objeto reconocido (blob). Este método recibe como parámetros de entrada, una imagen `im` y un blob `t2`. Se obtienen las coordenadas de `t2` y se utiliza la función `rectangle` para añadir el rectángulo a la imagen `im`. En la Figura 4.5 se observa que cuatro objetos son reconocidos y encuadrados.

Listado 4.2: Método para añadir rectangulos a blobs

```
Mat Blobs::addBlobToImg(Mat im, CBlob t2){

    Scalar mean, stddev;

    Rect bbox = t2.GetBoundingBox();
    rectangle(im,bbox,Scalar(0,220,0),1);

    return im;
}
```

El método `mover()` mueve el servomotor a un `ángulo` de entrada especificado.

Listado 4.3: Método para mover un servomotor

```
void Servo::mover(double angulo){
    FILE * pFile;

    int servo_a = (int) interpolacion(angulo);

    string s = std::to_string(servo_a);
    s = SERVO_NUM + s + "\n";
    const char *cstr = s.c_str();

    cout << "Servo:␣" << s << endl;

    if(enable){
        pFile = fopen (SERVO_FILE, "wb");
        if (pFile != NULL){
            fwrite (cstr , sizeof(char), s.size(), pFile) ;
            fclose (pFile);
        }else{
            cout << "No␣se␣puede␣abrir␣" << SERVO_FILE << endl;
        }
    }
```

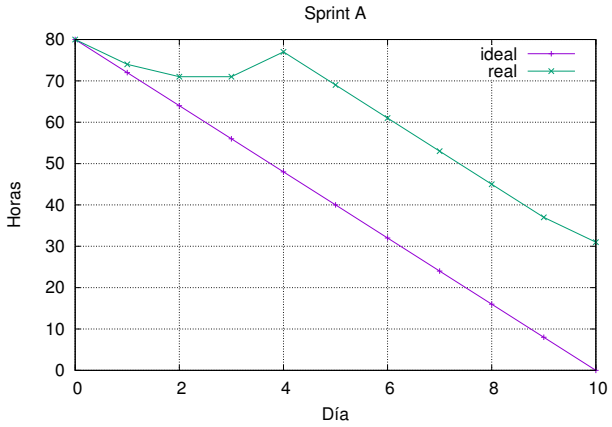


Figura 4.8: Gráfico burn down del sprint. Elaboración propia.

```
}
```

Utilizando la Ecuación 4.2 se implementa el método de `interpolacion()`.

Listado 4.4: Método para transformar de grados a modulación por ancho de pulso.

```
double Servo::interpolacion(double x){

    double x0 = 180, y0 = 89;
    double x1 = 90, y1 = 189;
    double x2 = 45, y2 = 242;

    double y;

    y = ((x - x1) * (x - x2) * y0)/((x0 - x1) * (x0 - x2)) +
        ((x - x0) * (x - x2) * y1)/((x1 - x0) * (x1 - x2)) +
        ((x - x0) * (x - x1) * y2)/((x2 - x0) * (x2 - x1));

    return y;
}
```

4.3.6. Gráfico burn down del sprint

La Figura 4.8 muestra el gráfico *burn down* del presente sprint.

4.3.7. Retrospectiva del sprint

- ¿Qué salió bien?
  - Se completaron la mayoria de las tareas de la iteración.
  - Se escribieron pruebas de unidad para el código desarrollado.
- ¿Qué podría haber sido mejor?
  - La estimación de la historia US3 no fue buena.
- ¿Qué se puede mejorar en el futuro?
  - Mejorar las estimaciones de las historias, si la historia es muy larga es mejor dividirla.

Capítulo 5

Conclusiones y Recomendaciones

5.1. Conclusiones

blah blah

5.2. Recomendaciones

blah blah

5.3. Posibles extensiones

blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah blah

blah blah

Bibliografía

[1] Laganière, Robert.OpenCV 2 Computer Vision Application Programming Cookbook.Packt Publishing.2011.Página 1

[2] Norris, Donald.Raspberry Pi Projects for the Evil Genius.Mc Graw Hill.2014 . Página 22

[3] Sahin, Ferat; Kachroo, Pushkin . Practical and Experimental Robotics . CRC Press . 2008 . Página 43

[4] STMicroelectronics . L298 DUAL FULL-BRIDGE DRIVER Datasheet. STMicroelectronics . 2000 . Página 2

[5] Jähne, Bernd; Haußecker, Horst . Computer Vision and Applications . Academic Press . 2000 . Página 1

[6] E. R. Davies . Computer and Machine Vision: Theory, Algorithms, Practicalities . Academic Press . 2012 . Página 10

[7] Brahmbhatt, Samarth . Practical OpenCV . APress . 2013 . Página 22

[8] Nixon, Mark; Aguado, Alberto . Feature Extraction and Image Processing . Newnes . 2002 . Página 2

[9] Davies, E. Roy . Machine Vision: Theory, Algorithms, Practicalities . Morgan Kaufmann . 2005 . Página 53

[10] Cyganek, Boguslaw; Siebert, J. Paul; An introduction to 3D computer vision techniques and algorithms . Wiley . 2009 . Página 412

[11] E. R. Davies . Computer and Machine Vision: Theory, Algorithms, Practicalities . Academic Press . 2012 . Página 40

[12] Nixon, Mark; Aguado, Alberto . Feature Extraction and Image Processing . Newnes . 2002 . Página 99

[13] Burger, Wilhelm; Bhanu, Bir . Estimating 3-D Egomotion from Perspective Image Sequences . IEEE . 2002 . Página 1

[14] Lyudmila MIHAYLOVA; Paul BRASNETT; Nishan CANAGARAJAH; David BULL . Object Tracking by Particle Filtering Techniques in Video Sequences . Department of Electrical and Electronic Engineering, University of Bristol, UK . . Página 1

[15] E. R. Davies . Computer and Machine Vision: Theory, Algorithms, Practicalities . Academic Press . 2012 . Página 112

[16] E. R. Davies . Computer and Machine Vision: Theory, Algorithms, Practicalities . Academic Press . 2012 . Página 149

[17] Wenczel, Norma . Inside the Camera Obscura ¿ Optics and Art under the Spell of the Projected Image . Max Planck Institute for the History of Science . 2007 . Página 13-30

[18] Starr, Cecie . Biology: Concepts and Applications . Thomson Brooks/Cole . 2005 . Página 94

[19] Jähne, Bernd y Haußecker, Horst . Computer Vision and Applications, A Guide for Students and Practitioners . Academic Press . 2000 . Página 564

[20] Jähne, Bernd y Haußecker, Horst . Computer Vision and Applications, A Guide for Students and Practitioners . Academic Press . 2000 . Página 7

[21] Bhat, Pravin . Gradientshop: A gradient-domain optimization framework for image and video filtering . ACM Transactions on Graphics . 2010 . Página 10

[22] M. Bertalmío, G. Sapiro, V. Caselles and C. Ballester. Image Inpainting . Proceedings of SIGGRAPH 2000 . 2000 . Página 1

[23] Agarwal, R.P.; Wong, J.Y. Patricia . Error Inequalities in Polynomial Interpolation and Their Applications . Springer Science+Bussiness Media, B.V. 1993 . Página 217

[24] Cohn, Mike . User Stories Applied, for Agile Software Development . Addison-Wesley . 2004 . Página 121

[25] Bermejo, Sergi . Desarrollo de robots basados en el comportamiento . Ediciones UPC . 2003 . Páginas 26, 27

[26] Sutherland, Jeff . The Scrum Handbook . Scrum, Inc . 2001 . Página 6

[27] Página oficial de OpenCV: http://opencv.org/. 27 de Marzo de 2014

[28] Página oficial de Raspberry Pi: http://www.raspberrypi.org/. 27 de Marzo de 2014

[29] Componentes de Raspberry Pi: https://www.raspberrypi.org/blog/new-graphic/. 8 de Septiembre de 2015

[30] Raspberry Pi camera module: http://www.raspberrypi.org/products/camera-module/. 11 de Septiembre de 2014

[31] Raspberry Pi camera module stock lens characteristics http://www.truetex.com/raspberrypi . 9 de Octubre de 2014

[32] Introducing turbo mode: up to 50 % more performance for free http://www.raspberrypi.org/introducing-turbo-mode-up-to-50-more-performance-for-free/ . 30 de octubre de 2014

[33] The Official Raspberry Pi Camera Module http://www.raspberrypi-spy.co.uk/2013/05/the-official-raspberry-pi-camera-module/ . 5 de noviembre de 2014

[34] Tamiya Track and Wheel Set http://www.superdroidrobots.com/shop/item.aspx/tamiya-track-and-wheel-set/409/ . 5 de noviembre de 2014

[35] CCD and CMOS sensors http://www.issibern.ch/forads/sr-009-23.pdf . 09 de Septiembre del 2015

ANEXOS

Anexo A: Código fuente

Clase para permitir movimiento autónomo al robot

BaseCon.cpp

```
#include "BaseCon.hpp"

BaseCon::BaseCon(bool w, TrackbarCallback tc, bool robotEnable){
    if(!robotEnable){
        camara = new Camara(false);
    }else{
        camara = new Camara(true);
    }

    robot = new Robot(!robotEnable);
    initCW(robot, ENC1_1, ENC1_2, ENC2_1, ENC2_2);

    window = w;
    if(window)
        createWindow(tc);
}

/**
  * Recibe blobs y los procesa en el motor de inferencia
  */
void BaseCon::kb_inference (){
    bool avanzarR = true;
    bool girarD = false;
    bool girarI = false;
    bool cont;
    bool inclinado = false;
    double distancia;
    double distancia_v;
    float *angle;

    Mat frame;
    if(camara->cap.isOpened() || !frame.empty()){
        camara->cap >> frame;
    }else{
        cout << "No se puede obtener frame" << endl;
        return;
    }

    CBlobResult blobs = camara->findBlobs(frame);


        for(int i=0;i<blobs.GetNumBlobs();i++){
        cont = false;
                CBlob blob = blobs.GetBlob(i);
```

```
        Rect bbox = blob.GetBoundingBox();
        //si esta dentro de un rango de area (no muy grande o muy pequeno),
        //entonces continua
        if( blob.Area(PIXELWISE) < camara->min_blob_area ||
            blob.Area(PIXELWISE) > camara->max_blob_area){
            continue;
        }

        //para hallar si el objeto esta al fondo
        int fondo = bbox.y + bbox.height;

        if(window){
            //if (!cont)
            camara->addBlobToImg(frame, blob);
                imshow("blobs", frame);
        }


        distancia = utils::calcular_distancia(camara->grados,
                    SCREEN_SIZE_Y - bbox.br().y);
        cout << "distancia␣" << distancia << "␣";

        if(bbox.x < (camara->width / 2)){
            //blob al lado izquierdo de la pantalla
            //distancia medida desde el borde izquierdo de la pantalla hasta el
            //borde derecho del blob
            distancia_v = utils::calcular_distancia_horizontal(distancia,
                          bbox.x + bbox.width);
            cout << "Caso␣1:␣" << bbox.x + bbox.width << "␣";
            if(distancia < 20){
                girarD = true;
            }
        }else{
            //blob al lado derecho de la pantalla
            //distancia medida desde el borde izquierdo de la pantalla hasta el
            //borde izquierdo del blob
            distancia_v = utils::calcular_distancia_horizontal(distancia, bbox.x);
            cout << "Caso␣2:␣" << bbox.x << "␣";
            if(distancia < 20){
                girarI = true;
            }
        }
        cout << "distancia_v:␣" <<distancia_v << "␣";

        if(window){
            camara->addDistanciaBlob(distancia, blob, frame);
            camara->addAreaBlob(blob.Area(PIXELWISE), blob, frame);
        }
    }

    if(robot->getAngles(angle)){
        float inclinacion = angle[1] * 180/M_PI;
        if(inclinacion < -20){
            inclinado = true;
        }
    }

    if(inclinado){
        robot->retroceder(1000);
        robot->girarI(500);
        robot->avanzar();
```

```
    }else{
        if(girarI && !girarD){
            robot->girar(20);
        }else if (girarD && !girarI){
            robot->girar(-20);
        }else if(!girarI && !girarD){
            robot->avanzar();
        }else if(girarD && girarI){
            //numero aleatorio
            std::srand(std::time(0));
            int random_variable = std::rand();

            //preguntar si es par o impar y girar a un sentido segun el
            //resultado
            if((random_variable % 2) == 0){
                robot->girar(50);
            }else{
                robot->girar(-50);
            }
        }
    }
}
void BaseCon::on_slider(int, void *) { }
void BaseCon::on_grados(int, void *) {
    camara->mover(camara->grados);
}

/**
  * Ventana para hacer pruebas
  */
void BaseCon::createWindow(TrackbarCallback tc){
    namedWindow("blobs");

    createTrackbar("min␣blobs␣area", "blobs", &camara->min_blob_area, 2000, tc);
    createTrackbar("max␣blobs␣area", "blobs", &camara->max_blob_area, 2000, tc);
    createTrackbar("grados␣camara", "blobs", &camara->grados, 180, tc);
}
```

Clase con la función de reconocer obstáculos en el plano

Blobs.cpp

```
#include "Blobs.hpp"

Blobs::Blobs() : NUMCORES(4)
{
    canny_th1 = 50;
    canny_th2 = 100;
    size_slider = 5;
}

Mat Blobs::addBlobToImg(Mat im, CBlob t2){

    Scalar mean, stddev;

    Rect bbox = t2.GetBoundingBox();
    rectangle(im,bbox,Scalar(0,220,0),1);

    return im;
}
```

```
CBlobResult Blobs::findBlobs(Mat im){
        Mat img, imgc, imgd;

        cvtColor(im,img,CV_BGR2GRAY);
    Mat st_elem = getStructuringElement(MORPH_RECT, Size(size_slider, size_slider));
    dilate(img, imgd, st_elem);

    Canny(imgd, imgc, canny_th1, canny_th2);

        return CBlobResult(imgc,Mat(),NUMCORES);
}
```

Blobs.hpp

```
#ifndef BLOBS_HPP
#define BLOBS_HPP

#include <opencv2/opencv.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/imgproc/imgproc.hpp>
#include <iostream>
#include "blob.h"
#include "BlobResult.h"
#include "Utils.hpp"

using namespace std;
using namespace cv;

class Blobs{
    private:
        const int NUMCORES;
        CBlobResult res;
    public:
        int canny_th1;
        int canny_th2;
        int size_slider;
        int grados;

        Blobs();
        Mat addBlobToImg(Mat im, CBlob t2);
        CBlobResult findBlobs(Mat im);
};

#endif
```

Clase que representa al sensor MPU6050

Esta clase se ocupa de inicializar al sensor y obtener los ángulos de inclinación del Robot.

Giro.cpp

```
#include "MPU6050_6Axis_MotionApps20.h"
#include "Giro.hpp"

MPU6050 mpu;
using namespace std;

Giro::Giro(){
    dmpReady = false;
    setup();
}
```

```
void Giro::setup() {

    uint8_t devStatus;
    uint8_t mpuIntStatus;

    printf("Initializing␣I2C␣devices...\n");
    mpu.initialize();

    printf("Testing␣device␣connections...\n");
    printf(mpu.testConnection() ? "MPU6050␣connection␣successful\n"
                                : "MPU6050␣connection␣failed\n");

    printf("Initializing␣DMP...\n");
    devStatus = mpu.dmpInitialize();

    if (devStatus == 0) {
        printf("Enabling␣DMP...\n");
        mpu.setDMPEnabled(true);

        mpuIntStatus = mpu.getIntStatus();

        printf("DMP␣ready!\n");
        dmpReady = true;

        packetSize = mpu.dmpGetFIFOPacketSize();
    } else {
        printf("DMP␣Initialization␣failed␣(code␣%d)\n", devStatus);
    }
}


bool Giro::getAngles(float *&angle_out) {

    uint8_t fifoBuffer[64];
    Quaternion q;
    float angle[3] = {0, 0, 0};
    angle_out = angle;
    bool ret = false;
    VectorFloat gravity;

    if (!dmpReady) return false;
    uint16_t fifoCount = mpu.getFIFOCount();

    if (fifoCount == 1024) {
        mpu.resetFIFO();
        printf("FIFO␣overflow!\n");
        ret = false;

    } else if (fifoCount >= 42) {
        mpu.getFIFOBytes(fifoBuffer, packetSize);

        mpu.dmpGetQuaternion(&q, fifoBuffer);
        mpu.dmpGetGravity(&gravity, &q);
        mpu.dmpGetYawPitchRoll(angle, &q, &gravity);

        angle_out = angle;
        ret = true;
    }
    return ret;
}
```

Utilitarios

En este archivo se encuentran las funciones `calcular_distancia()` y `calcular_distancia_horizontal()`, que calculan la distancia vertical y horizontal de un objeto detectado en el plano. También se encuentra la función `leerConfigFile()`, que sirve para leer archivos de configuración.

Utils.cpp

```
#include "Utils.hpp"


namespace utils{

    int leerConfigFile(std::string key1){
        std::ifstream is_file("file.cfg");
        std::string line;
        int value1 = 0;
        while( std::getline(is_file, line) )
        {
            std::istringstream is_line(line);
            std::string key;
            if( std::getline(is_line, key, '=') )
            {
                std::string value;
                if( std::getline(is_line, value) ) {
                    if(key1.compare(key) == 0){
                        value1 = std::stoi(value);
                        std::cout << "key:␣" << key <<
                        "␣value:␣" << value1 << std::endl;
                        break;
                    }
                }
            }
        }
        return value1;
    }

    int showText(Point pos, string text, Mat img){
            putText(img, text, pos, FONT_HERSHEY_SIMPLEX, 0.6, Scalar(255, 255, 0));
    }

    double calcular_distancia_horizontal(double distancia_v, double a_px){
        double f = (2 * distancia_v * sin(DEGTORAD(ALPHA/2))) / sin(DEGTORAD(BETA));
        double distancia_h =  a_px * f / SCREEN_SIZE_X;
        return distancia_h;
    }

    double calcular_distancia(double alpha, double b_px){

        double largo_b_cam = 2.6;
        double altura_eje_motor = 7.2;
        double omega = 41; //angulo de vision camara
        double theta = (180 - omega)/2;
        double mi = 90 + alpha - omega / 2;
        double a, c, d, e, f, g, i, x;
        double beta;
        double kappa;
        double b_cm, j;


        a = largo_b_cam * sin(DEGTORAD(alpha));
```

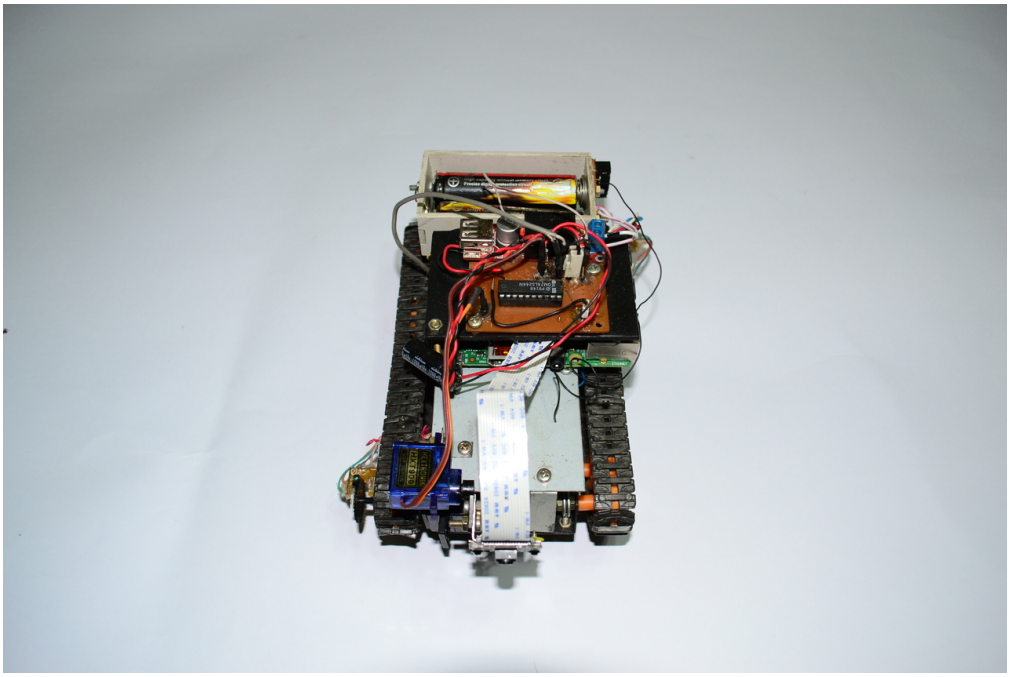


Figura 5.1: Vista frontal del robot. Se observa: el distribuidor de energía y regulador de voltaje al centro.

```
        c = largo_b_cam * sin(DEGTORAD(90-alpha));
        d = altura_eje_motor - c;
        e = (d * sin(DEGTORAD(alpha - omega/2)))
            /sin(DEGTORAD(90-alpha+omega/2));
        f = e / (sin(DEGTORAD(alpha - omega / 2)));
        j = f * sin(DEGTORAD(omega)) / sin(DEGTORAD(theta));
        b_cm = b_px * j / SCREEN_SIZE_Y;
        i = sqrt(b_cm * b_cm + f * f - 2 * b_cm * f * cos(DEGTORAD(theta)));
        beta = RADTODEG(asin(b_cm * sin(DEGTORAD(theta)) / i));
        kappa = 180 - mi - beta;
        g = f * sin(DEGTORAD(beta)) / sin(DEGTORAD(kappa));
        x = g + e + a;
        return x;
    }
    void printToCoordinates(int x, int y, string text)
    {
        printf("\033[%d;%dH%s\n", x, x, text.c_str());
    }
}
```

Anexo B: Fotos del Robot

Anexo C: Planilllas de seguimiento de sprint

				V	L	M	X	J	V	L	M	X	J
				26-sep	29-sep	30-sep	01-oct	02-oct	03-oct	06-oct	07-oct	08-oct	09-oct
			Tareas pendientes	9	8	8	8	8	5	4	2	2	2
			Horas pendientes	74	71	71	77	69	61	53	45	37	31
US	**Tarea**	**Tipo**	**Estado**	**Esfuerzo**									
US1	T1	Implementación	Completada	0	0	0	0	0	0	0	0	0	0
US1	T2	Investigación	Completada	0	0	0	0	0	0	0	0	0	0
US1	T3	Configuración	Completada	2	0	0	0	0	0	0	0	0	0
US1	T4	Investigación	Completada	0	0	0	0	0	0	0	0	0	0
US1	T5	Investigación	Completada	0	0	0	0	0	0	0	0	0	0
US1	T6	Implementación	Completada	5	4	4	10	2	0	0	0	0	0
US1	T7	Implementación	Completada	2	2	2	2	2	0	0	0	0	0
US1	T8	Pruebas	Completada	2	2	2	2	2	0	0	0	0	0
US2	T1	Implementación	Completada	8	8	8	8	8	6	0	0	0	0
US2	T2	Implementación	Completada	8	8	8	8	8	8	6	0	0	0
US2	T3	Pruebas	Completada	2	2	2	2	2	2	2	0	0	0
US3	T1	Implementación	Pendiente	20	20	20	20	20	20	20	20	12	6
US3	T2	Implementación	Pendiente	25	25	25	25	25	25	25	25	25	25

Cuadro 5.1: Planilla de seguimiento del sprint A