ESCUELA SUPERIOR DE INGENIERÍA Y TECNOLOGÍA

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo Fin de Grado

Diseño de Dron Submarino: Desarrollo del sistema de sensores y visión gestionado mediante software basado en ROS

Autor: Iván Jesús Torres Rodríguez

Tutores:

Jonay Tomás Toledo Carrillo

Oswaldo Bernabé González Hernández

Junio 2019

Agradecimientos

Muchísimas gracias a todas las personas que se han involucrado en el proyecto. En especial, a mis tutores, Jonay Toledo que siempre me ha resuelto los problemas que le iba planteando, y por prestarnos las instalaciones que nos permitieron realizar las pruebas de presión con el dron; y a Oswaldo González que, desde el primer momento, me apoyó en el desarrollo del proyecto y ha estado siempre disponible cuando he necesitado de su colaboración.

También agradecer a Rafael Arnay, que me ha brindado una gran ayuda especialmente en el ámbito de visión por ordenador.

Y a Leopoldo Acosta y Fernando Rosa por la ayuda que nos han prestado antes y durante todo el proyecto.

Además, agradecer a Carmelo Militello por colaborar con el diseño del dron y por el interés que ha mostrado en el proyecto. Y a Federico Padrón que nos permitió realizar pruebas en el muelle de náutica.

Gracias también a Iván Rodríguez por su colaboración en el proceso de diseño y fabricación de las PCB.

Muchas gracias a mis amigos y compañeros David y Nicolás, por todos los buenos ratos montando el dron de madrugada, de excursión para el sur a hacer pruebas o simplemente estando un rato hablando. Y también a nuestro amigo José, que nos ha aguantado todo este curso que no hemos parado de hablar del dron, y siempre ha estado dispuesto a ayudarnos a realizar pruebas en el momento que necesitáramos.

Quiero agradecerle a mi familia y en especial a mis padres y a mi hermana el apoyo incondicional que me han brindado en todo momento.

Índice general

Índice de figuras

Índice de tablas

Capítulo 1. Resumen/Abstract

1.1. Resumen

Este Trabajo de Fin de Grado se enmarca dentro de un proyecto mayor denominado "*U-Water Explorer Dron*" consistente en la construcción de un prototipo de dron submarino funcional para tareas de investigación o inspección ya sea en puertos o lugares afines.

Para la realización de este prototipo va a ser necesario, entre otros, un sistema de sensores, un sistema de visión, otro de sonido y una interfaz para mejorar la interacción con el usuario. Este trabajo versa sobre el diseño y la implementación de dichos sistemas.

El sistema de sensores será construido sobre dos PCB (*Printed Circuit Board*) que irán conectadas a una Raspberry Pi y a un Arduino Mega.

Durante el desarrollo se equipa al dron con un sistema de visión en el que se implementarán el registro de imágenes en tiempo real y métodos para grabar, además de realizar seguimiento y detección de objetos. Todo el desarrollo software vendrá acompañado del proceso de puesta en práctica del sistema donde se verán sus limitaciones. Este sistema trabajará conjuntamente con un sistema de iluminación que se controlará desde tierra.

Paralelamente, en este trabajo se llevará a cabo un desarrollo que permita enlazar un micrófono con el que captar en tiempo real los sonidos, pudiendo hacer grabaciones para luego reproducirlos en tierra.

Junto a todo esto, se acomete el diseño de una interfaz gráfica, que permita la cómoda visualización de toda la información relevante.

1.2. Abstract

This Final Degree Project is part of a larger project named "U-Water Explorer Dron" consisting of the construction of a functional submarine drone prototype for research or inspection tasks in ports or similar places. For the realization of this prototype it will be necessary, among others, a sensor system, a vision and a sound system and an interface to improve the interaction with the user. This Final Degree Project deals with the design and implementation of these systems.

The sensor system will be built on two PCB (Printed Circuit Board) that will be connected to a Raspberry Pi and an Arduino Mega.

The drone is equipped with a vision system in which real-time view and recording methods will be implemented, as well as tracking and object detection. All this software development will be concluded with the implementation on the drone, in this moment the limitations will be

seen. This system will be supported by a lighting system that will be controlled from the ground.

Parallel to this work a development that allows to link a microphone for sound streaming and recording will be done.

In addition, the design of a graphical interface that allows comfortable viewing of all relevant information is undertaken.

Capítulo 2. Introducción

2.1. Proyecto

Este Trabajo de Fin de Grado (TFG) es parte de un proyecto que consiste en el desarrollo de un dron submarino capaz de operar a una profundidad máxima de 20 metros. Este dron submarino o ROUV (*Remote Operated Underwater Vehicle*) es multipropósito, pensado para realizar tareas de investigación, tareas de inspección en muelles y puertos o simplemente usarlo para exploración.

Este proyecto tiene por objetivo dar respuesta a necesidades que se pueden encontrar en un entorno como el de las Islas Canarias u otras zonas costeras en las que se realizan a diario tareas de inspección en muelles a buques de carga y transporte.

La decisión de que la máxima profundidad alcanzable por el mismo sea de 20 metros viene directamente ligada a este objetivo ya que el calado de los muelles de Canarias está en torno a los 14 metros, tal y como se puede consultar en la web de la autoridad portuaria [1].

El proyecto global de desarrollo del dron submarino se ha llevado a cabo junto a otros dos compañeros del grado, distribuyendo el trabajo conjunto en tres partes, cada una de las cuales constituye el Trabajo de Fin de Grado realizado por cada uno de nosotros:

- David Henry González, *Diseño de dron submarino: diseño estructural, control de movimiento y sistema de alimentación*: centrado en el diseño mecánico y el sistema de alimentación, todo ello condicionado por la especificación de profundidad máxima de inmersión del dron.
- Nicolás Adrián Rodríguez Linares, *Diseño de dron submarino: Diseño e implementación del sistema de comunicaciones y diseño de sistema de control*: centrado en la implementación de todo un sistema de comunicaciones que permita el tráfico de datos desde el ordenador de mando situado en el exterior con el ordenador interno del dron, así como en el control de movimiento del mismo.
- Iván Jesús Torres Rodríguez, *Diseño de dron submarino: Desarrollo del sistema de sensores y visión gestionado mediante software basado en ROS*: centrado en los sistemas de percepción requeridos para que el operador pueda conocer las condiciones de entorno del dron una vez está sumergido.

El diseño del dron va a guardar mucha semejanza con el diseño de un quadrotor de aire. Esto nos dará mucha maniobrabilidad y sólo se hará uso de cuatro motores por lo que conseguimos una buena relación entre movimiento y coste.

Teniendo el diseño anterior como base, es fundamental implementar un sistema de comunicación con el dron que sea robusto y que a su vez proporcione una velocidad de datos

suficiente. Con esto se puede comenzar a tratar el funcionamiento en detalle del dron, empezando por el control y movimiento del dron haciendo uso de los cuatro motores.

Por otro lado, hay que llevar a cabo el desarrollo del sistema de sensores que permita monitorizar el dron y conocer datos claves del entorno. Además, al tratarse de un dron submarino es fundamental que posea un sistema de visión en tiempo real que permita el manejo del dron desde fuera, junto a sistemas de detección de objetos que puedan ayudar en las tareas de inspección. La estructura de desarrollo del proyecto global se puede ver de forma esquemática en la Figura 2.1.

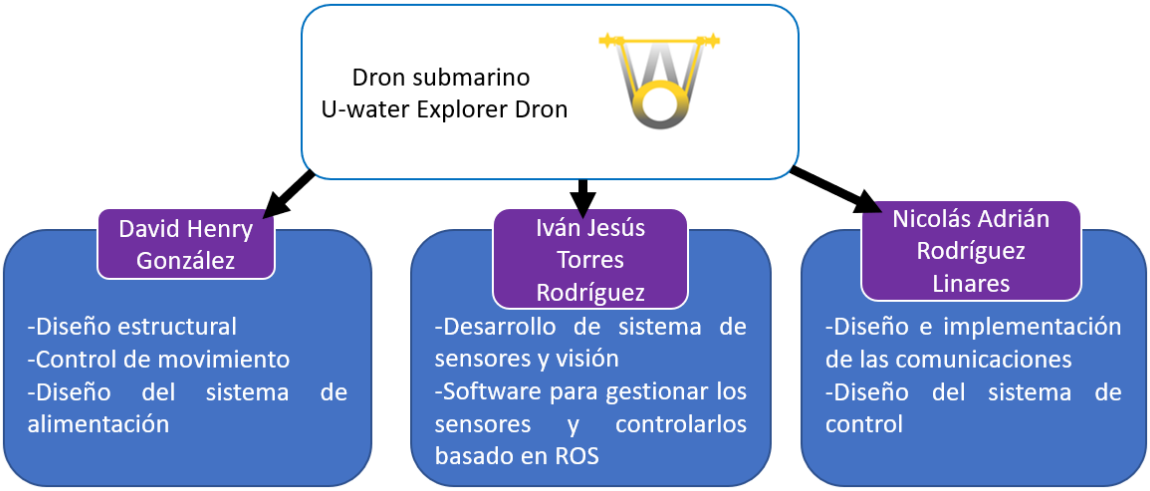


Figura 2.1: Esquema del proyecto

2.2. Objetivo del proyecto

El objetivo final del proyecto es tener un prototipo funcional de dron submarino capaz de ser controlado desde el exterior. Se ha realizado un prototipo perfectamente utilizable de forma que, una vez configurado, el usuario no tenga que realizar complejas tareas para ponerlo a funcionar, sino que tan solo sea necesario conectarlo mediante un cable Ethernet con interfaz RJ45 al PC (*Personal Computer*) y un mando USB (*Universal Serial Bus*) de Playstation 3 ®, para controlar los movimientos y algunas funcionalidades extras (Figura 2.2). Es por tanto sencillo de utilizar.

Figura 2.2: Conexionado del sistema

Por otro lado, se priorizó que el dron pueda ser utilizado como una herramienta de trabajo más que no requiere de complicados mantenimientos o tareas de preparación. En este sentido el tubo donde va alojada la electrónica es de fácil acceso ya que se puede abrir por tres partes, pero no es necesario abrirlo para realizar tareas como la recarga de la batería del submarino o el encendido y apagado del mismo. Esto es gracias a un sistema utilizado para tener acceso a conectores e interruptores sin más trabajo que abrir una pequeña caja estanca situada fuera del submarino.

Para que el usuario pueda tener en todo momento controlado lo que pasa con el dron debemos tener una interfaz de usuario completa, pero simple, el usuario no debe tener que realizar navegación a través de menús para realizar las distintas tareas, sino que se busca una única pantalla en la que se vea la información en tiempo real del submarino.

Junto a esto, dado que el dron puede estar destinado a inspección, es crucial poder disponer de un sistema de detección que podamos adaptar a diferentes situaciones. Además, es fundamental el guardado de la información, para poder repasar sesiones de inmersión.

El dron también va a contar con una opción para grabar y hacer *streaming* de audio, lo que puede ser de utilidad para tareas de investigación submarina. Para que sea estándar el dispositivo de grabación se ha de conectar vía USB a una Raspberry Pi [2].

2.3. Objetivos del sistema de sensores y visión

Este Trabajo de Fin de Grado va a centrarse en el desarrollo de todo el sistema de sensores que lleva el dron junto a la interfaz gráfica para mostrar la información al usuario. Se abordará por tanto el uso de la placa necesaria para leer los sensores, la preparación de la información para

ser enviada al PC externo, el tratamiento de esta información para ser mostrada y el uso de ROS (*Robotic Operating System*) [3] para integrarlo todo. Además, se ha contemplado la posibilidad de ampliación del sistema de sensores que permitan añadir funcionalidades al dron. Para realizar esto, junto con un microcontrolador, será necesario el uso de una PCB (*Printed Circuit Board*) que nos permita conectar los sensores de forma robusta y en un espacio reducido.

Por otro lado, tenemos el sistema de visión que consiste en una cámara USB que va conectada a la Raspberry Pi. Esta cámara va situada en la parte del submarino alojada en una carcasa de metacrilato que permite la visión. Esta carcasa es sumergible hasta 30 metros por lo que cumple de sobra nuestra especificación de 20 metros de profundidad.

Con esta cámara somos capaces de enviar imágenes en tiempo real que puedan ser vista por el usuario en la interfaz diseñada para ayudar al manejo del dron. También se pueden realizar grabaciones: el usuario tiene una pestaña para comenzar a grabar y posteriormente parar la grabación; el archivo resultante se almacena en el PC externo.

En lo que se refiere a la detección de objetos se tienen dos alternativas que se pueden usar indistintamente, una de ellas se puede usar sobre CPU. Esta opción se basa en el uso el detector de características ORB (*Oriented FAST and Rotated BRIEF*) [4]. Por otro lado, tenemos una detección basado en redes neuronales implementado usando *TensorFlow* [5]. En concreto se ha usado una estructura de red ya existente y se ha entrenado con un conjunto de imágenes etiquetadas o *dataset* totalmente personalizado.

2.4. Análisis del estado del arte

Cada vez se habla más de drones submarinos, una industria que en la actualidad está en auge. Esto se debe en gran parte a la curiosidad del ser humano ya que se estima que el 95% de los océanos del mundo están aún inexplorados [6]. Es por ello que cada vez se pueden encontrar más modelos de estos drones en el mercado, muchos de ellos desarrollado por pequeños grupos de emprendedores que han acabado formando pequeñas empresas como pueden ser NIDO Robotics [7] o NAVATICS [8] (Figura 2.3), esta última está desarrollando el dron conocido como MITO que se ha financiado mediante Kickstarter [9] donde en solo cinco horas consiguió alcanzar la meta de patrocinios.

Figura 2.3: ROUV a la izquierda el del NIDO Robotics y a la derecha el MITO de NAVATICS

Por norma general ninguno de estos drones submarinos posee una gran carga de sensores para medir el medio, sino que centran sus esfuerzos en un buen sistema de visión e iluminación. Por ejemplo, el dron MITO posee un sistema de visión con cámara 4K, mientras el dron SIBIU PRO de NIDO Robotics tiene una cámara 1080p y cuenta con cuatro luces de 1500 lúmenes.

Otro aspecto que se trabaja mucho es la modularidad de los drones. En este sentido, NIDO Robotics está desarrollando una sonda multiparamétrica (Figura 2.4) con entradas para sensores de PH, redox (ORP), oxígeno disuelto, conductividad, temperatura, presión ecosonda y GPS, con lo que se consigue realizar medidas de múltiples parámetros del medio marino.

Figura 2.4: Sonda multiparamétrica de NIDO Robotics

Un ejemplo claro de este concepto modular es el trabajo que realiza la empresa BlueRobotics [10], que es la empresa suministradora de los propulsores de nuestro dron. Esta empresa distribuye un dron como base, el BlueROV2 (Figura 2.5).

Figura 2.5: BlueROV2

A dicho dron se le pueden añadir distinto número de luces, tiene diversas configuraciones de motores y también existe la posibilidad de equiparlo con múltiples cámaras como puede ser la básica de Raspberry Pi, modelos para baja luminosidad o con gran angular. Aparte de esto, comercializan sistemas GPS para el dron, o productos como un sonar (Figura 2.6) que en este momento se encuentra en beta.

Figura 2.6: Sonar BlueRobotics

Si bien los sistemas de visión están muy avanzados no existe tanto desarrollo en cuanto a sistemas de detección de objetos bajo el agua en parte porque es difícil conseguir *dataset* de objetos que sean de interés para detectar en el agua. Se tratan de dos campos que avanzan más bien en paralelo, pues los drones no incorporan detección y no hay muchas facilidades para implementar dichos sistemas en los drones actuales.

Estos son algunos ejemplos que podemos encontrar de drones submarinos de producción. Sin embargo, en cuanto a construcción en un primer prototipo no se pretende alcanzar esos niveles de calidad. Es por esto que se realizó una búsqueda de otras estructuras a un nivel más simple para tener alguna referencia. En este sentido hay un dron diseñado por la Plataforma Oceánica de Canarias (PLOCAN) [11] que se usa para dar talleres en centros educativos, en los que construyen el dron (Figura 2.7) que es bastante simple y no requiere de materiales muy complejos de trabajar ni de conseguir.

Figura 2.7: Dron basado en el diseño de PLOCAN para el proyecto EDUROV

Capítulo 3. Material

En este apartado se van a tratar de forma esquemática los elementos hardware y software que se han empleado en el desarrollo del proyecto.

3.1. Hardware

3.1.1. Raspberry Pi

Estos son ordenadores en miniatura que hacen uso de la tecnología SoC (*System on Chip*). Son dispositivos mucho más potentes que un microprocesador como puede ser Arduino teniendo en cuenta su reducido tamaño. Además, son dispositivos de bajo coste lo que posibilita que haya mucho desarrollo en torno a ellos y una gran comunidad detrás. Por lo que podemos encontrar diversas opciones para montar como sistema operativo. Algunos ejemplos pueden ser Raspbian [12] o Ubuntu Mate [13].

Existen varios modelos, podemos encontrar la Raspberry Pi Zero W que tiene un tamaño reducido en comparación a otros modelos como la Raspberry Pi *model* 3B+ [2] que es el modelo más avanzado del fabricante. En primera instancia se utilizó una Raspberry Pi *model* 3B que tiene características peores al modelo +. Aunque todo el trabajo se pudo realizar de forma satisfactoria con el modelo más básico, la comunicación con el cable Ethernet largo no era viable con este modelo. Una de las mejoras más importantes en el modelo + se encuentra en la tarjeta de red que pasa de ser de hasta 100 Mbps a 300 Mbps. Esta diferencia nos permite poder usar el cable de Ethernet largo para conectar el PC y la Raspberry Pi. Además de incluir un mejor procesador. Por esto la Raspberry utilizada en el prototipo final tiene que ser la *model* 3B+ (Figura 3.1). En el TFG de Nicolás se encuentra información ampliada sobre el rendimiento de las comunicaciones.

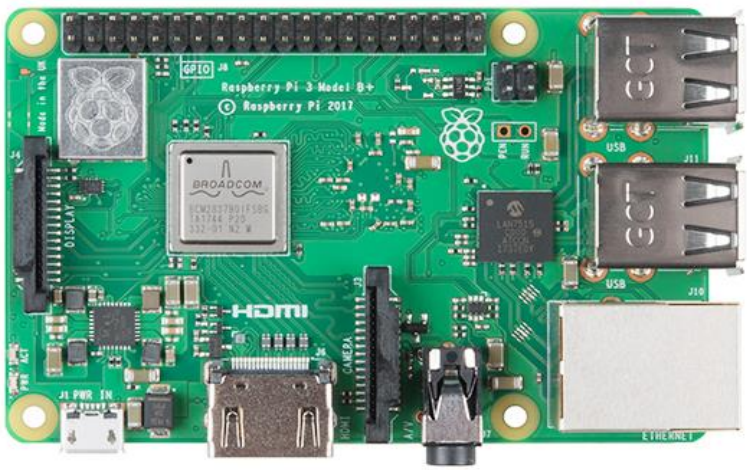


Figura 3.1: Raspberry Pi model 3B+

3.1.2. Arduino

Es uno de los microcontroladores más utilizados en el mundo, tiene una enorme comunidad y existen multitud de proyectos realizados con los distintos modelos. Además, hay una gran cantidad de sensores adaptados para el uso con estos microcontroladores de los que se puede consultar tanto código como conexionado.

Otra ventaja de estos dispositivos es que pueden trabajar de manera sencilla con ROS lo que nos facilitará mucho el trabajo a la hora de pasar información entre dispositivos.

Para nuestro proyecto vamos a hacer uso de dos Arduino Mega [14]. Uno de ellos es una versión adaptada por la comunidad, conocida como ArduPilot [15] que lo usaremos para el control de movimientos. El otro, que es el que más importancia cobra para el desarrollo de este TFG, es una placa de Arduino Mega original (Figura 3.2). En esta irán conectados todos los sensores a excepción de la IMU (*Inertial Measurement Unit*).

Figura 3.2: Arduino Mega original

3.1.3. Webcam USB de Creative

Vamos a utilizar una Webcam de Creative [16] con micrófono que funciona mediante USB (Figura 3.3) para realizar las tareas de visión y sonido. Esta cámara cuenta con un sensor CMOS y es capaz de grabar hasta 640x480 a 30 fotogramas por segundo. No cuenta con *zoom* óptico. El sensor es de 1 *megapixel*.

Figura 3.3: Webcam USB de Creative

3.1.4. Sensores

Se van a utilizar un conjunto de sensores necesarios para poder monitorizar el submarino y tomar ciertos datos básicos. El sistema de sensores se explica en profundidad en el capítulo 4.

3.2. Software

3.2.1. Ubuntu 16.04

Ubuntu [17] es una distribución de GNU/Linux basada en Debian que nos proporciona un sistema operativo de código abierto. Este software es creado por la empresa Canonical Ltd.. Cada dos años se lanza una versión que adquiere la denominación LTS (*Long Term Support*), que tiene actualizaciones durante cinco años. Aunque las versiones con esta denominación sean las que más soporte tienen, cada seis meses hay una nueva versión de software que tiene soporte de la comunidad durante nueve meses.

En el momento de inicio del proyecto se disponía de dos versiones LTS, la 16.04 (*Xerial Xerus*) y la 18.04 (*Bionic Beaver*), que fue lanzada en 2018. En nuestro caso, tenemos que elegir una versión con denominación LTS pues son las que llevan asociada una versión de ROS. Debido a la mayor cantidad de documentación y material desarrollado, en torno a la versión de ROS compatible con Ubuntu 16.04, fue esta la opción elegida.

3.2.2. Ubuntu Mate 16.04

Es el sistema operativo que va a llevar montado la Raspberry Pi. Ubuntu Mate [13] se trata de una distribución de Linux basada en Ubuntu pero más ligera que su homólogo de Ubuntu. Esto hace que se utilice mucho en dispositivos como la Raspberry Pi que tiene menos capacidad de procesamiento que un PC de sobremesa.

Las versiones de Ubuntu Mate están ligadas con las versiones de Ubuntu, en este sentido vamos a utilizar Ubuntu Mate en la misma versión que el Ubuntu del ordenador. Con esto podemos instalar la misma versión de ROS en la Raspberry Pi y en el PC lo que simplifica todo el proceso de trabajo, pues compatibilizar dos versiones de ROS requiere de un trabajo añadido, que ahorramos usando la misma versión.

3.2.3. ROS

3.2.3.1. Introducción

La base del proyecto a nivel de software es ROS [3], en concreto la versión ROS Kinetic [18] que va ligada a la distribución 16.04 de Ubuntu, cuyo soporte oficial para estas versiones se extiende hasta Abril de 2021.

ROS es un sistema meta-operativo para robots, se trata de software open-source lo que sin duda es de gran ayuda para poder realizar el desarrollo pues hay mucha información acerca del mismo. De hecho, ROS no busca ser el framework con más características del mercado, sino que trata de conseguir que el código que se desarrolla pueda ser reutilizado. Además, presenta una característica muy valiosa que es que se puede integrar código de forma sencilla. Por ejemplo, si tenemos un programa en Python de OpenCV [19] o para capturar audio, el integrar estos códigos en ROS no conlleva gran complejidad. Otro punto clave es que soporta diferentes lenguajes como C++ o el ya mencionado Python.

Sin embargo, es bastante complicado el hecho de adaptarse a trabajar con ROS, pues todo el sistema de archivos, de compilación y el funcionamiento general de ROS hace que, para poder trabajar con él de manera ágil, sean necesarias muchas horas de práctica.

3.2.3.2. Funcionamiento básico de ROS

La gran ventaja de ROS es que permite que tengamos varios ordenadores sincronizados, para llevar a cabo distintas tareas.

En ROS tenemos un maestro que es el que permite a los nodos de ROS localizarse entre ellos. El maestro registra los tópicos, nodos y servicios que se encuentran en ejecución. Para ejecutar este nodo hace falta ejecutar un *roscore*. En nuestro caso el maestro se encuentra en la Raspberry Pi model 3B+ ubicada en el submarino.

Los nodos (Figura 3.4) son programas que realizan funciones concretas, como es el nodo de capturar la imagen de la cámara. Son precisamente estos nodos los que aportan una gran flexibilidad a ROS. Cada nodo es compilado de forma individual, tras lo cual son ejecutados y gestionados por el nodo maestro.

Estos nodos se comunican entre sí haciendo uso de los tópicos (Figura 3.4). Los diferentes nodos publican en tópicos la información que se intercambia con los demás nodos. Desde los distintos nodos se puede publicar información en tópicos o suscribirse a tópicos que ya están publicados lo que hace muy sencillo el trasvase de información.

Estos tópicos se pueden ver como el asunto de los mensajes que contienen. En ROS existen muchos tipos de mensaje y se pueden crear mensajes personalizados. Estos mensajes se publican en tópicos, los mensajes pueden ser simples como *int* o *float* o mensajes compuestos como puede ser el de temperatura que está compuesto por un encabezado, un *float64* que es la temperatura y un *float64* que es la varianza.

Por otro lado, están los servicios que son para comunicar directamente dos nodos. Un servicio (Figura 3.4) está definido por dos mensajes uno, de petición y otro, de respuesta.

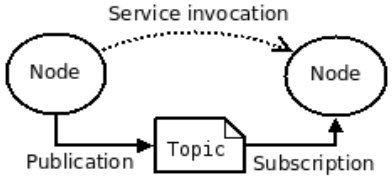


Figura 3.4: Esquema de funcionamiento de ROS

3.2.4. Arduino IDE

Este es el entorno de desarrollo integrado de Arduino. Se trata de una aplicación multiplataforma que está escrita en Java. Con ella podemos escribir programas y cargarlos a las placas compatibles con Arduino.

El lenguaje de programación de Arduino está basado en C++, por lo que, aunque Arduino tiene su propia página de referencia, para el lenguaje de programación, también se pueden usar comandos estándar de C++.

3.2.5. Git

Git [20] es un software de control de versiones que nos va a permitir llevar a cabo la gestión de los diversos cambios que se van realizando sobre distintos archivos de código o texto.

Este tipo de software es ideal cuando se tienen un proyecto en grupo ya que nos permite trabajar en local y cuando tenemos una versión operativa o que simplemente queremos compartir con el resto, la subimos al repositorio en la nube y pasa a ser visible para todos los miembros. Los cambios van acompañados de un comentario que nos ayuda a identificar que se ha realizado en cada actualización del repositorio.

Como servicio de hosting para el repositorio vamos a utilizar GitHub [21].

3.2.6. Geany

Geany [22] es un editor de texto ligero basado en Scintilla, que es un componente libre de edición de código fuente, con características básicas de entorno de desarrollo integrado (IDE). Este editor está disponible para múltiples plataformas. En cuanto a la licencia cuenta con Licencia Pública General de GNU. Con este programa se han trabajado los códigos de C++ y Python.

3.2.7. C++

Se trata de un lenguaje de programación orientado a objetos que toma como base el lenguaje C. El C++ [23] es un lenguaje ampliamente utilizado, puesto que es versátil, potente y general. Cualquier programa C es un programa válido C++. Con respecto a C, C++ añade aspectos interesantes como pueden ser los comentarios de fin de línea con //, palabras reservadas o nuevos tipos como *bool*.

3.2.8. Python

Python [24] es otro lenguaje de programación que se centra en que la sintaxis favorezca un código legible. Es un lenguaje de programación interpretado, esto significa que no necesitamos compilar el código ya que usamos *scripts* que son interpretados en tiempo real por un intérprete.

Además, es multiparadigma ya que soporta varios paradigmas de programación como pueden ser la orientación a objetos o la programación imperativa.

3.2.9. KiCAD

KiCAD [25] es un paquete de software libre que se ha utilizado para realizar las placas de circuito impreso que son necesarias para el dron. Este programa facilita pasar del diseño de un esquemático a una placa de circuito impreso. Además, posee funcionalidades muy interesantes como son el visor 3D que nos permite saber cómo va a quedar la placa antes de pasar a realizarla físicamente. Este también es multiplataforma y cuenta con Licencia Pública General de GNU.

3.2.10. Google Images Download

Con esta aplicación [26] podemos descargar un gran volumen de imágenes desde *Google Images* al ordenador, lo que nos va a ser muy útil para obtener todas las imágenes que necesitamos para etiquetar y posteriormente poder entrenar la red neuronal para detección de imagen.

3.2.11. LabelImg

Es una aplicación [27] (Figura 3.5) desarrollada en Python que nos permite realizar el etiquetado de imágenes para poder construir un *dataset* personalizado, que posteriormente usamos para entrenar la red neuronal de *TensorFlow*. Con ella trabajamos en un directorio con todas las imágenes y una vez decididos los objetos de interés comenzamos a etiquetar.

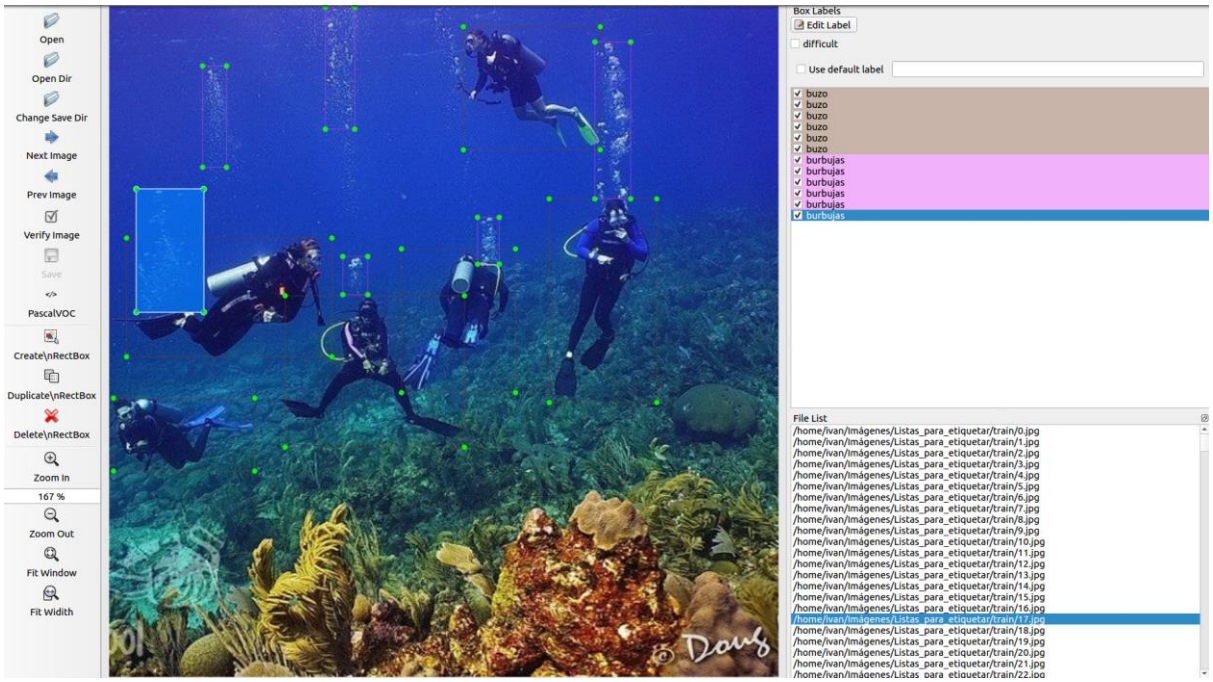


Figura 3.5: Etiquetado de imagen con LabelImg

3.2.12. OpenCV

OpenCV (*Open Source Computer Vision Library*) [19] es una librería de software *open source* ampliamente utilizada en el mundo de la visión por ordenador. Sirve tanto, para visión por computador, como para *Machine Learning*. El objetivo de OpenCV era dar una infraestructura común para las aplicaciones de visión por ordenador y acelerar el uso de la percepción con máquinas en los productos comerciales.

Esta librería cuenta con más de 2500 algoritmos optimizados, que incluyen desde algoritmos clásicos hasta los que utilizan *machine learning*. En este amplio conjunto encontramos soluciones para detectar y reconocer caras, identificar objetos, hacer *tracking* o generar nubes de puntos 3D usando cámaras estéreo.

Esta librería, aunque está construida en C++ es compatible con C++, Python, Java y Matlab en Windows, Linux, Android y Mac OS.

3.2.13. TensorFlow

Es una biblioteca de código abierto que se basa en un sistema de redes neuronales para resolver problemas de *Deep Learning*. Esta biblioteca ha sido desarrollada por Google y es utilizada para tareas como traducción o recomendación de contenido. *TensorFlow* [5] nos proporciona la capacidad de construir y entrenar redes neuronales.

El funcionamiento de *TensorFlow* se basa en imitar el funcionamiento de un cerebro humano, tratando de relacionar imágenes, textos o palabras en función de la información con la que se ha entrenado la red neuronal.

TensorFlow puede ser ejecutado usando la CPU o con GPU. Sin embargo, para poder realizar detección de imagen en tiempo real es recomendable ejecutarlo en GPU.

3.2.14. Cuda

CUDA (*Compute Unified Device Architecture*) [28] es una plataforma de computación en paralelo que incluye un compilador y un conjunto de herramientas de desarrollo creadas por nVidia que permite usar una variación del lenguaje de programación C para codificar algoritmos en GPU de nVidia.

Esto es muy importante para usar las redes neuronales, ya que permite que se puedan ejecutar en GPU, con los beneficios que esto conlleva. En cuanto a velocidad, es difícil dar una cifra, por la cantidad de dispositivos GPU y CPU que existen. Sin embargo, en algunos casos el usar la GPU para *TensorFlow* puede llevar a doblar la velocidad de ejecución.

3.2.15. Google Colab

Se trata de un entorno gratuito [29] de Jupyter Notebook que se ejecuta completamente en la nube. Este nos permite escribir y ejecutar código teniendo acceso a una gran cantidad de GPU,

lo que nos permitirá realizar el entrenamiento de las redes neuronales de forma más ágil que si lo hiciéramos en local.

Además, nos permite enlazar con Drive para almacenar los resultados que obtengamos de los entrenamientos, hacer pruebas con ellos y si funcionan de forma satisfactoria descargarlos para usarlos en local.

Capítulo 4. Sistema de sensores

4.1. Requerimientos

En lo que se refiere al sistema de sensores se puede distinguir claramente sensores con dos objetivos. Por un lado, están los que se utilizan para monitorizar el submarino, la temperatura interna, la humedad, la presión interna o la presencia de agua. Por otro lado, se encuentran los que tienen como objetivo conocer parámetros del entorno. En este sentido tenemos la sonda para medir la temperatura externa, el sensor de presión externa, el micrófono y la cámara.

En lo que se refiere a las medidas de parámetros internos del dron es importante conocer la temperatura dado que hay muchos dispositivos electrónicos que disipan calor. También es de bastante interés saber la humedad en el interior del tubo. Además, tenemos que tener información sobre la presión interna del tubo para poder detectar cambios bruscos de presión que puedan indicar que el submarino tiene alguna abertura. El hecho de que entre agua en el submarino puede acabar con toda la electrónica que en él está alojada, por lo que también debe haber un sistema que pueda detectar la entrada de agua por las zonas más sensibles del dron, es decir, la junta central y las tapas laterales, para en tal caso actuar buscando preservar la electrónica.

Otro punto clave del submarino es que, al ir alimentado por batería, debemos conocer el voltaje de la misma para saber cuánta autonomía nos queda.

Por otro lado, tenemos sensores externos, concretamente uno para conocer la presión externa, y poder estimar la profundidad a la que se encuentra el submarino, sabiendo que cada 10 m de profundidad la presión aumenta en torno a 1 bar. Y otro para conocer la temperatura del agua. También tenemos la cámara que incluye micrófono, pero este tema se tratará en un apartado distinto pues engloba cosas como la detección de objetos.

4.2. Hardware

Para cumplir con los requerimientos en lo que a sensórica se refiere primero se realiza un estudio de las distintas opciones que encontramos en el mercado. En este punto se busca encontrar sensores con una buena relación de prestaciones costo para no encarecer el producto final. A continuación, se verá el microcontrolador escogido y se hará un análisis de distintas opciones para medir los parámetros requeridos.

4.2.1. Microcontrolador

La placa microcontroladora que vamos a usar para leer los sensores a excepción de la IMU va a ser un Arduino Mega 2560. Esta está basada en el uso del ATMega 2560 y tiene 54 pines de entrada/salida digitales y 16 pines analógicos de entradas. Con estos será más que suficiente

para cubrir las necesidades del sistema. Por otro lado, cuenta con comunicación serial que usaremos para conectarlo con la Raspberry Pi y bus I^2C (*Inter-Integrated Circuit*) que es usado por alguno de los sensores.

4.2.2. Análisis de mercado

4.2.2.1. Medida de temperatura interior

En primera instancia se planteó el uso de un termistor, como una opción sencilla y económica. Los termistores son dispositivos que cambian su resistencia en función de la temperatura. Dentro de los termistores se encuentran los NTC (*Negative Temperature Coefficient*), como el que se ve en la Figura 4.1, cuya resistencia disminuye si aumenta la temperatura y los PTC (*Positive Temperature Coefficient*) cuya resistencia aumenta con la temperatura. Los más comunes son los NTC.

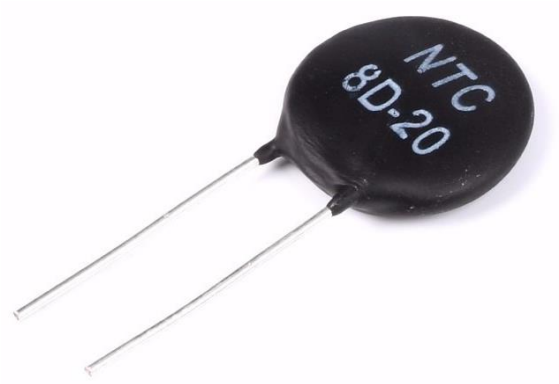


Figura 4.1: Termistor NTC

Estos dispositivos son muy compactos y baratos, se pueden comprar 10 termistores por un euro aproximadamente. Tienen un buen rango de medición, son rápidos, tienen buena precisión y son poco susceptibles al ruido. Sin embargo, los termistores tienen un comportamiento fuertemente no lineal, que no supone un gran problema porque es una tecnología bien conocida y por tanto en las hojas de características (*datasheet*) se puede consultar con precisión la respuesta de los mismos. El problema que esto implica es que los cálculos matemáticos pueden ralentizar el procesador.

Por otro lado, se contempló el uso de un sensor de temperatura digital como puede ser el LM35 (Figura 4.2), que es un integrado con un circuito de control propio que ofrece a la salida directamente un voltaje proporcional a la temperatura. Este voltaje podría leerse directamente con una de las entradas analógicas del Arduino. Otra característica interesante de estos sensores es que la salida varía de forma lineal con la temperatura. El coste de cada sensor de este tipo está en torno al euro la unidad.

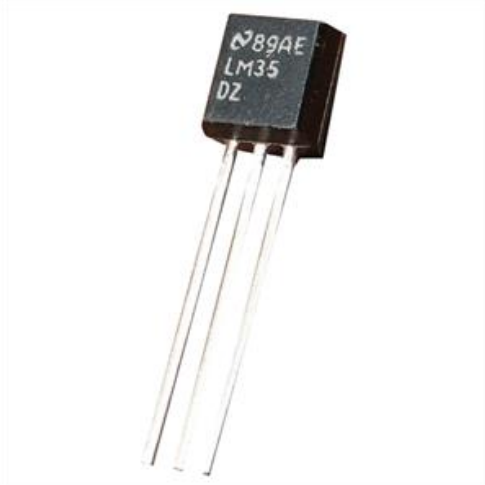


Figura 4.2: LM35

Por último, tenemos sensores como el DHT22 (Figura 4.3) que permiten realizar medidas de temperatura y humedad al mismo tiempo. Esto constituye una gran ventaja con respecto a los demás pues con un mismo encapsulado podemos obtener ambas medidas. Si bien es cierto que el coste es ligeramente superior situándose en los cinco euros, la ventaja de obtener las dos medidas suple esta desventaja. El funcionamiento de este sensor se verá en profundidad posteriormente en el apartado de sensores escogidos.

4.2.2.2. Medida de humedad interior

Para la realización de la medida de humedad vamos a utilizar el DHT22 (Figura 4.3) aprovechando que hace medida de humedad relativa y temperatura. Este sensor es digital, sólo utiliza un pin digital para mandar información, lo que hace que le afecte menos el ruido.

Figura 4.3: DHT22

4.2.2.3. Medida de presión interna

Para realizar esta medida decidimos utilizar un sensor que está bastante extendido en el mundo de Arduino, el BMP180 (Figura 4.4) es una versión mejorada del BMP085. Se trata de un

barómetro digital que nos permite medir la presión de aire y funciona mediante el protocolo I^2C el cual está disponible en el Arduino. Es capaz de medir en un rango de 300 hPa a 1110 hPa. Lo que buscamos con este sensor es corroborar que durante las inmersiones la presión se mantenga estable. En cuanto al coste, es económico en torno a los 2 euros.

Figura 4.4: BMP180

4.2.2.4. Detección de agua

Para esta tarea optamos por un sensor comercial sencillo que se suele usar para detectar lluvia, posee una placa conductora (Figura 4.5) con pistas separadas ligeramente entre ellas que cuando se deposita agua encima pasan a conducir; responde a las necesidades del proyecto en lo que a detección de agua se refiere y el coste no es muy elevado, aproximadamente dos euros.

Figura 4.5: Placa conductora del sensor de lluvia

4.2.2.5. Medida de temperatura externa

En este caso también optamos por usar un sensor bastante extendido en el mundo de Arduino. Se trata del DS18B20 (Figura 4.6) que se puede comprar en forma de integrado, pero también, se vende en forma de sonda impermeable, lo que la hace ideal para medir la temperatura del agua. Además, el material usado para la misma es acero inoxidable de alta calidad para prevenir la oxidación. Se trata de un sensor que funciona entre -55 ºC y 125 ºC. Usa el bus de comunicación *one-wire*, que se caracteriza por usar un solo conductor para la comunicación. Además, es bastante económico, sobre los tres euros.

Figura 4.6: DS18B20

4.2.2.6. Medida de presión externa

Se analizaron dos alternativas, por un lado, nos planteamos el uso de algún sensor como el MS5803-14BA (Figura 4.7) o similar, muy sencillo de usar con Arduino vía I^2C o SPI (*Serial Peripheral Interface*), con el obtenemos la presión del fluido que entra en contacto con el mismo. Es muy compacto pero dada la morfología del dron puede resultar un poco difícil de acoplar en el mismo.

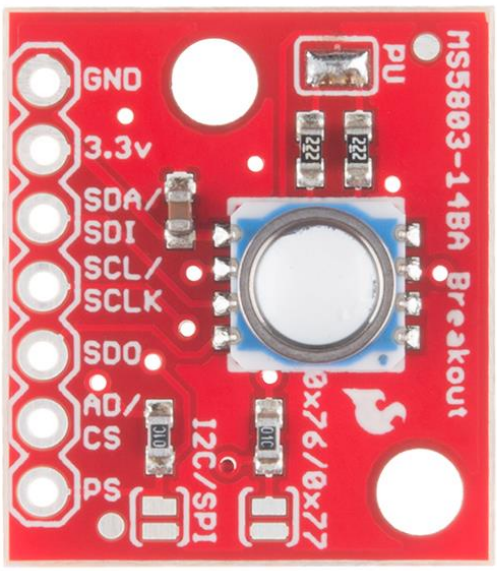


Figura 4.7: MS5803-14BA

Por otro lado, tenemos el SICK PBT-RB010SG1SSNAMA0Z (Figura 4.8) que es un sensor capaz de medir en un rango de 0 a 10 bares, es decir hasta 99,55 m de profundidad en agua salada, para obtener esta medida usamos el software de cálculo de BlueRobotics [30]. Al ser alimentado mediante dos conductores con una tensión entre 8 y 30 V de continua nos proporciona una salida de 4-20mA. Una gran ventaja de este sensor es que tiene un acople comercial roscado, que nos sería fácil adaptar para el dron, lo que hizo que lo escogiéramos por delante de la otra opción.

Figura 4.8: Sensor SICK para medida de presión

4.2.2.7. Medida de voltaje de la batería

Para conocer el voltaje de la batería vamos a utilizar un sensor (Figura 4.9) capaz de medir hasta 25 V en sistemas de 5 V como es el nuestro. Su uso está bastante extendido en la comunidad de Arduino, al ser muy económico, con un costo inferior a dos euros, y bastante compacto. Todo esto lo hace ideal para el proyecto.

Figura 4.9: Sensor de voltaje

4.2.2.8. Unidad de medición inercial (IMU)

Una unidad de medición inercial (IMU) es el nombre genérico para denominar a un dispositivo que es capaz de medir la velocidad angular, orientación y aceleración de un sistema. El objetivo de esta unidad es posicionar en el espacio un objeto, tanto en posición como en orientación.

Es frecuente hacer referencia a la cantidad de grados de libertad (DOF, *Degree Of Freedom*) que tiene una IMU. Los grados de libertad de un sensor representan la cantidad de magnitudes independientes que es capaz de medir.

Optamos por un sensor de 6 DOF, que es el MPU-6050, compuesto por un giroscopio de 3 ejes, que aporta la información del *roll* (alabeo), y un acelerómetro de 3 ejes, que da la información del *pitch* (cabeceo). Este sensor posee un DMP (*Digital Motion Processor*) que es un procesador que nos permite obtener las medidas más precisas de *roll* y *pitch*, además de proporcionarnos una medida de *yaw* (guiñada) usando la información de magnetómetro y giroscopio. Con este, podemos conocer la información necesaria para manejar el dron.

4.2.3. Sensores escogidos

Uso	Sensor
Temperatura y humedad interior	DHT22
Presión interna	BMP180
Detección de agua	YL-83
Temperatura externa	DS18B20
Presión externa	PBT-RB010SG1SSNAMA0Z
Medir voltaje	F85
IMU	MPU-6050

Tabla 4.1: Sensores elegidos

4.2.3.1. Sensor de temperatura y humedad interior

4.2.3.1.1. Características

El sensor escogido para esta tarea fue el DHT22, una versión mejorada del DHT11, que permite realizar medidas con menos error que este último, necesita una tensión de alimentación de entre 3,3 V y 5,5 V, normalmente se usan 5 V para alimentarlo. Las medidas de temperatura y

humedad las puede dar con una resolución de un decimal. Una característica importante es que su tiempo de muestreo es de dos segundos.

El rango de funcionamiento es entre -40 ºC y 80 ºC, y toma medidas con una precisión de 0,5 ºC en condiciones normales, mientras en condiciones extremas la precisión pasa a 1 ºC.

En lo que se refiere a humedad el sensor hace medidas entre 0% y 99,9% de humedad relativa. Siendo la precisión de ± 2% asumiendo una temperatura ambiente de 25 ºC.

Al final del documento se puede consultar el *datasheet* del sensor, anexo 14.

4.2.3.1.2. Funcionamiento

El funcionamiento de este sensor se puede dividir en dos partes, la que mide la humedad relativa y la que mide la temperatura.

Para la medida de temperatura posee un termistor, que varía la resistencia con la temperatura. En este caso tenemos un termistor de tipo NTC que decrementa la resistencia cuando la temperatura aumenta.

Por la parte del sensor de humedad tiene dos electrodos con un sustrato entre ellos. Cuando la humedad varía cambia la conductividad del sustrato, por lo tanto, la resistencia entre electrodos también cambia, además aprovecha este fenómeno para medir la humedad relativa.

Esto podría verse como un condensador que cambia la capacidad dependiendo de la humedad. El funcionamiento se basa en el cambio del dieléctrico, por ejemplo, el aire cambia su permitividad dependiendo de la humedad del ambiente.

4.2.3.1.3. Conexionado

El conexionado (Figura 4.10) es bastante simple, por un lado, conectamos el pin de alimentación a 5 V y el pin de tierra a GND. El pin de salida lo conectamos a una de las entradas digitales del Arduino. Pero necesitamos colocar una resistencia de 10 k**Ω** entre la alimentación y el pin de salida del sensor.

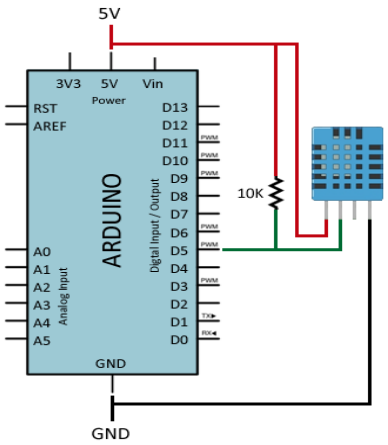


Figura 4.10: Esquema de conexión DHT22

4.2.3.1.4. Obtención de datos

Este sensor utiliza su propio sistema de comunicación bidireccional. Ambas medidas se pasan a través del mismo *driver*. Esto se consigue usando señales temporizadas. Las dos señales se mandan en paquetes de 40 bits cada 4 ms. Dentro de estos paquetes hay 2 bytes dedicados a mandar el mensaje de humedad y otros 2 bytes para el mensaje de temperatura, aparte de un byte para comprobar si hay errores. Este mensaje lo lee el Arduino y lo interpreta gracias a una librería específicamente diseñada para el DHT22.

4.2.3.2. Sensor presión interna

4.2.3.2.1. Características

Vamos a usar el BMP180, un sensor barométrico digital que incorpora un sensor de temperatura. La tensión de entrada puede ser entre 1,8 V y 3,6 V, pero el módulo que vamos a utilizar incorpora la electrónica necesaria para poder conectarlo de forma sencilla al Arduino, incorporando un regulador de voltaje que permite conectarlo directamente a 5 V.

Tiene un rango de medida de 300 hPa a 1110 hPa. La temperatura máxima a la que trabaja es 85 ºC aunque para tener una mejor precisión se recomienda no superar los 65 ºC, mientras que la temperatura mínima es - 40 ºC, no se recomienda bajar de 0 ºC.

4.2.3.2.2. Funcionamiento

La presión depende de varios factores, entre ellos la temperatura porque afecta a la densidad del aire. Es por esto que el BMP180 incorpora un sensor de temperatura para compensar el efecto de la misma en la presión.

4.2.3.2.3. Conexionado

El sensor debe tener conectada la alimentación (Figura 4.11), es decir, los pines Vcc a 5 V y GND a tierra y por otro lado tenemos que conectar los pines SDA y SCL a los pines correspondientes del Arduino que permiten la conexión I^2C.

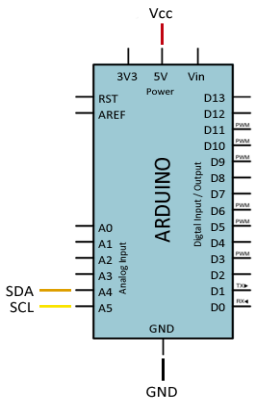


Figura 4.11: Esquema conexión BMP180

4.2.3.2.4. Obtención de datos

El sensor usa el protocolo de comunicación I^2C por lo que es sencillo obtener datos, este es un protocolo serial para establecer la comunicación entre varios dispositivos como pueden ser un microcontrolador y un sensor usando solo dos cables. Consiste en dos líneas SCL y SDA. La primera funciona como reloj y la segunda es un bus de datos. Dentro de este hay dos mensajes principales, la lectura de presión y la de temperatura.

4.2.3.3. Sensor presencia de agua

4.2.3.3.1. Características

Se seleccionó el sensor de lluvia YL-83, capaz de detectar presencia de agua en una placa que tiene pistas conductoras separadas entre sí. De estos usaremos tres uno para la parte central, otro para la parte delantera y otro para la parte de atrás del tubo.

Nos puede dar una lectura digital o analógica. La lectura analógica no es de interés puesto que el sensor no permite determinar con precisión la cantidad de agua que hay sobre la placa. Es por esto que vamos a usar el valor digital que está a baja (*low*) cuando no hay agua y a alta (*high*) cuando se detecta agua. La sensibilidad para que el sensor se dispare se puede ajustar con un potenciómetro.

4.2.3.3.2. Funcionamiento

El funcionamiento consiste en que cuando hay agua encima de la placa esta hace que las pistas, que en condiciones normales están separadas, entren en contacto eléctrico. Posee un comparador LM393 cuyo umbral se ajusta mediante el potenciómetro del sensor, con esto conseguimos que el sensor se active con una menor o mayor cantidad de agua.

4.2.3.3.3. Conexionado

La conexión (Figura 4.12) es muy simple ya que solo hay que conectar la alimentación, es decir, 5 V y GND, y el pin digital a un pin digital de Arduino para ver qué valor tiene el sensor.

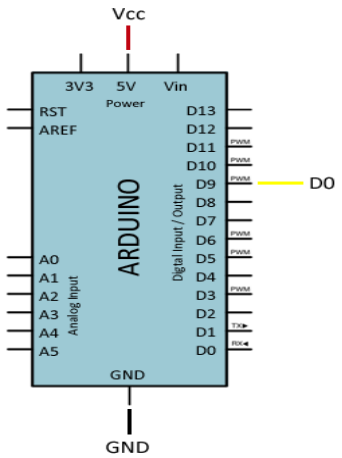


Figura 4.12: Esquema de conexión sensor de detección de agua

4.2.3.3.4. Obtención de datos

Simplemente tenemos que leer el pin digital para ver si está en *high* o *low*.

4.2.3.4. Sensor de temperatura externa

4.2.3.4.1. Características

Para esta tarea se seleccionó la sonda DS18B20, que permite realizar medidas de temperatura en líquidos, ya que es impermeable y está protegida contra la corrosión.

Este sensor tiene un amplio rango de medición desde -55 ºC a 125 ºC, teniendo una resolución de 0,5 ºC en el rango de -10 ºC a 85 ºC. Utiliza el bus de comunicación *One-Wire*, que solo necesita de un conductor para comunicarse. Se podría alimentar por la línea de datos, nosotros vamos a optar por usar el cable de alimentación. El sensor se alimenta entre 3 y 5,5 V, por lo que nosotros utilizaremos los 5 V que nos proporciona la placa de Arduino.

Existe la posibilidad de configurar alarmas, que se activan en función de las condiciones de temperatura, pero en nuestra aplicación, no es de gran interés.

4.2.3.4.2. Funcionamiento

El funcionamiento de este sensor es más complejo de lo que a priori pueda parecer, esto se debe principalmente al sistema de comunicación que tiene implementado.

El sensor está formado por un procesador (Figura 4.13) que tiene varios módulos que controlan la comunicación, miden la temperatura y controlan las alarmas.

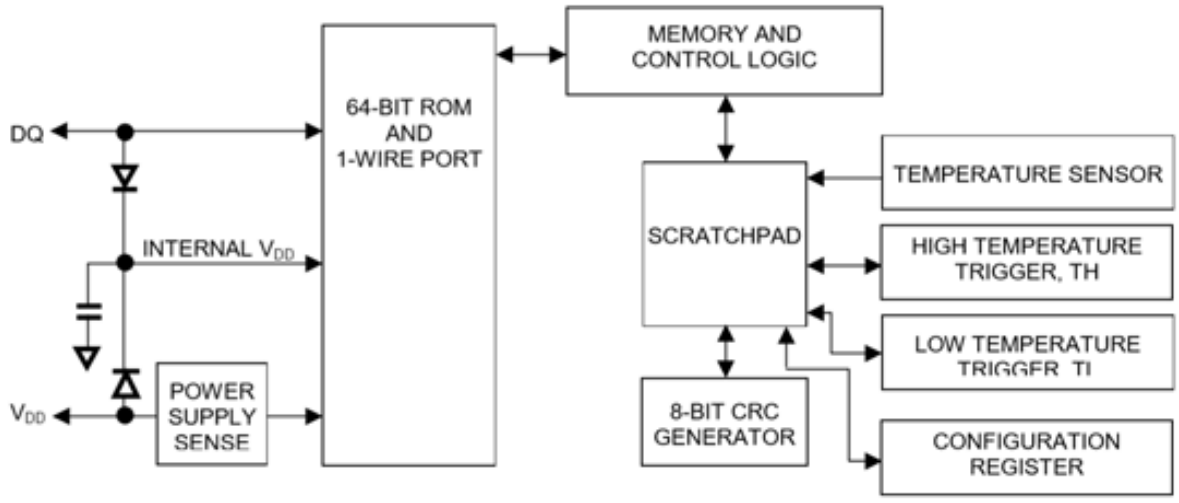


Figura 4.13: Diagrama de componentes del sensor DS18B20

Este sensor tiene la ventaja de que solo necesita un cable de datos, pero el sistema de comunicaciones carga bastante el procesador, ya que se trata de un sistema bastante complejo, cuyo funcionamiento detallado se puede consultar en el *datasheet* del sensor, anexo 17.

4.2.3.4.3. Conexionado

Conectamos el sensor utilizando los tres cables que tenemos (Figura 4.14): el rojo va conectado a 5 V, el negro a GND y el amarillo, que es el de datos, a un pin digital. Además, debemos poner una resistencia de 4,7 k**Ω** entre 5 V y el pin digital (Figura 4.15).

Figura 4.14: Pines del DS18B20

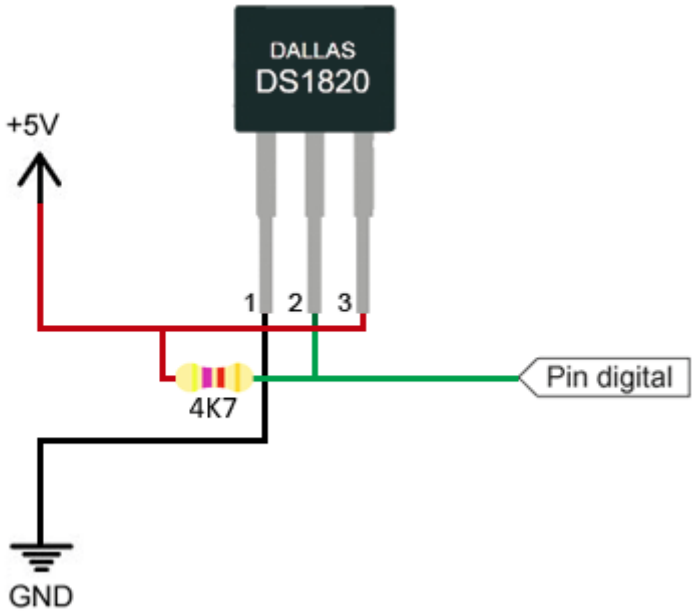


Figura 4.15: Conexionado del DS18B20

4.2.3.4.4. Obtención de datos

Para la lectura de los datos utilizamos las librerías *OneWire* y *DallasTemperature*. De estas librerías es de especial importancia la función *getTempCByIndex*, que nos permite conocer la temperatura del sensor correspondiente al índice especificado. En nuestro caso, al tener solo uno, usamos el índice cero.

4.2.3.5. Sensor de presión externa

4.2.3.5.1. Características

El sensor utilizado es del fabricante SICK, en concreto el PBT-RB010SG1SSNAMA0Z, que está destinado a hacer medidas de presión en líquidos y gases.

Este nos proporciona la capacidad de medir presiones entre 0 y 10 bar, lo que es suficiente, dada nuestra especificación de máximo 20 metros de profundidad. Las temperaturas de funcionamiento van desde 0 ºC hasta 80 ºC. Se puede alimentar entre 8 V y 30 V de continua. Vamos a usar la alimentación de 12 V que nos proporciona el regulador que tenemos instalado en la placa de alimentación.

La señal de salida es 4-20mA lo que significa que dado que con el Arduino podemos medir hasta 5 V la resistencia a utilizar sería de 5/0,02 = 250 **Ω**. Para usar una resistencia de la que teníamos disponibilidad en SMD usamos 220 **Ω** que es el valor más próximo admisible que estaba disponible.

El conector que se usa para este sensor es de rosca M12.

4.2.3.5.2. Funcionamiento

A nivel interno el fabricante no ofrece demasiada información de cómo funciona, pero sí nos proporciona la configuración de conexión para que el sensor trabaje dando una salida de 4-20 mA. Esto se explica más en profundidad en el siguiente apartado.

4.2.3.5.3. Conexionado

Para conectar el sensor utilizamos la configuración de 2 cables con el acople M12, tal y como se puede ver en la Figura 4.16.

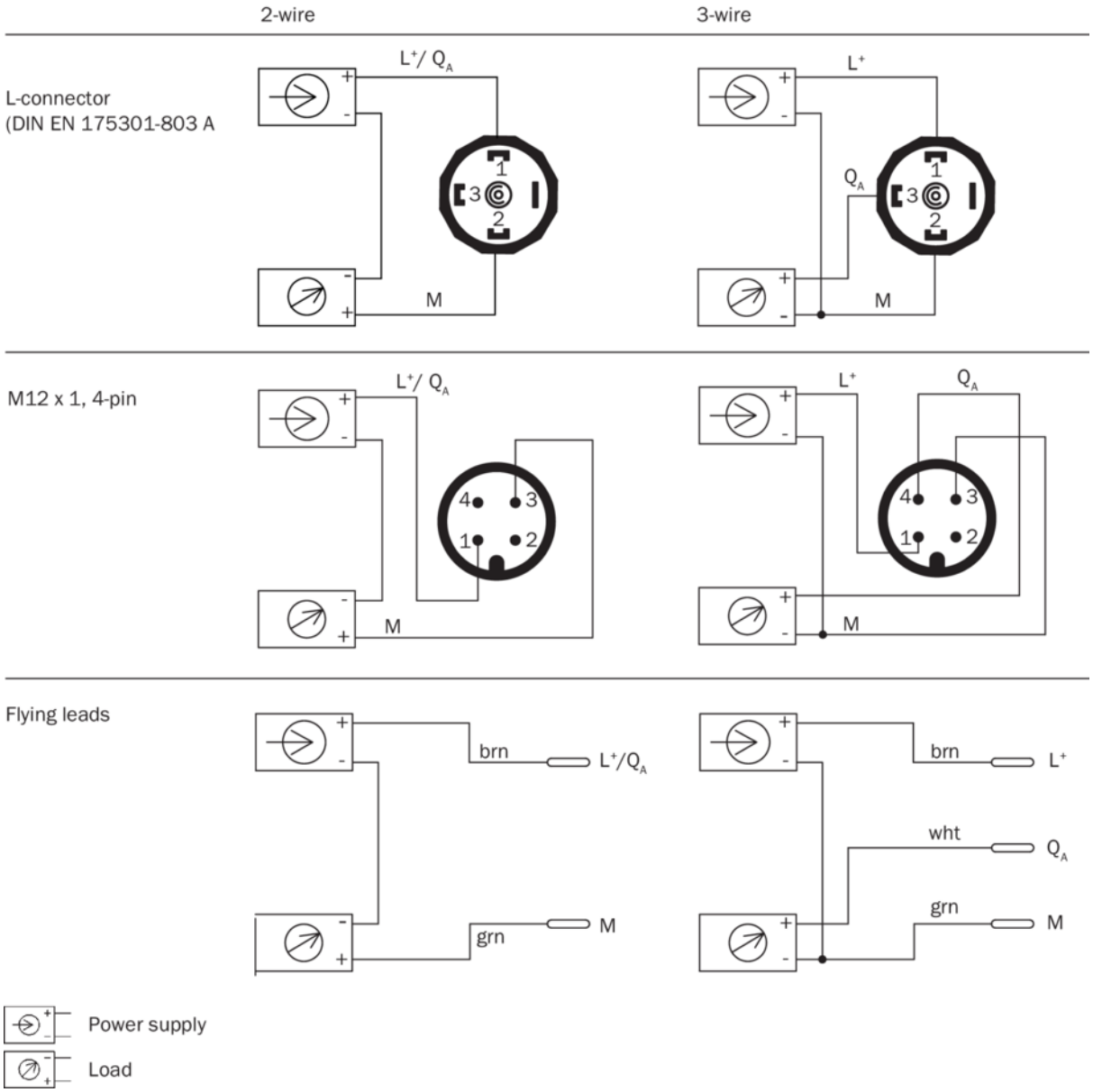


Figura 4.16: Esquemas posibles de conexión del sensor de presión de la marca SICK

Para llevar a cabo el conexionado de forma correcta debemos mirar el conector que tenemos en el cable para ver con qué colores de cable se corresponde cada pin. El cable con conector que vamos a utilizar es el DOL-1204-G02M.

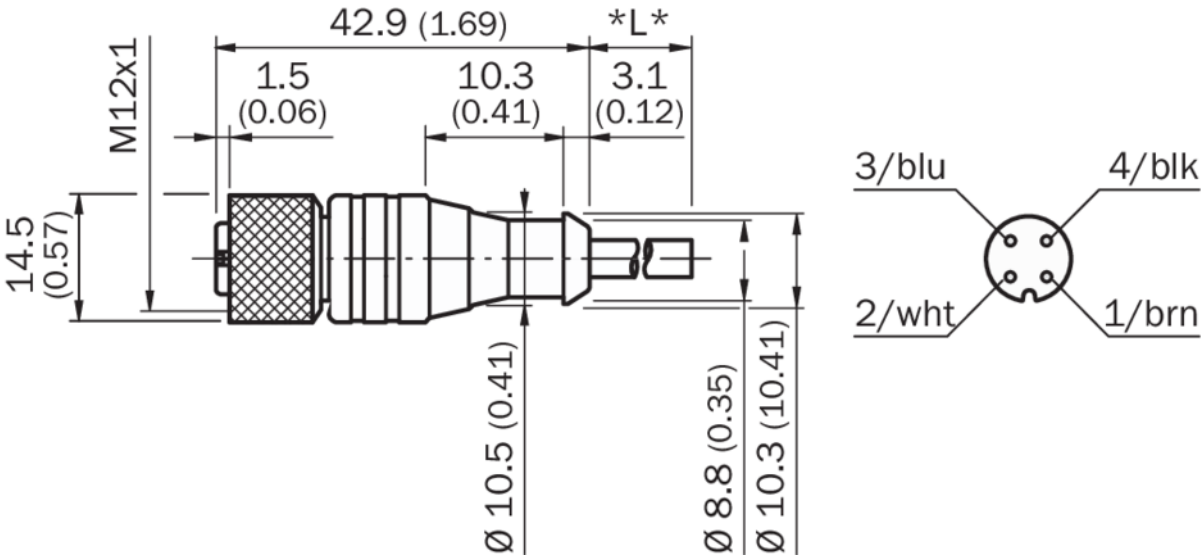


Figura 4.17: Pines del conector utilizado para el sensor de presión de SICK

Como vemos en la Figura 4.17 el número con el que nombran los pines es igual que en el sensor. Para nuestra configuración conectaremos el conector en la posición única que puede ponerse y tendremos cuatro cables que podemos conectar. De estos vamos a usar el marrón que va conectado a la alimentación y el azul que va conectado a la resistencia.

4.2.3.5.4. Obtención de datos

Para obtener el valor de presión vamos a realizar la medida de la caída de voltaje en la resistencia con una de las entradas analógicas del Arduino. Las entradas analógicas del Arduino dan un valor entre 0 y 1023. Nuestro sensor, como funciona con 4-20 mA, nos daría una lectura mínima de 0,004×220 = 0,88 V, que se correspondería con un valor de lectura del Arduino de 180. Por debajo de este valor el sensor estaría dando error. Y como valor máximo tendríamos 0,02×220 = 4,4 V, que sería un valor de 900 en el Arduino. El valor de 180 se corresponde con 0 bar y el de 900 con 10 bar.

4.2.3.6. Sensor de voltaje

4.2.3.6.1. Características

El sensor de voltaje F85 es de fácil utilización que nos permite medir el voltaje de la batería para saber la autonomía restante.

Tiene un rango de entrada de voltaje de entre 0 y 25 V, al juntarlo con el Arduino tenemos una resolución de unos 0,00488 V, ya que el Arduino tiene un ADC de 10 bits.

4.2.3.6.2. Funcionamiento

El funcionamiento se basa en un divisor de tensión (Figura 4.18) con resistencias de 30 k**Ω** y 7,5 k**Ω**. Atendiendo al funcionamiento de un divisor de tensión, para un procesador alimentado con 5 V se pueden medir hasta 25 V.

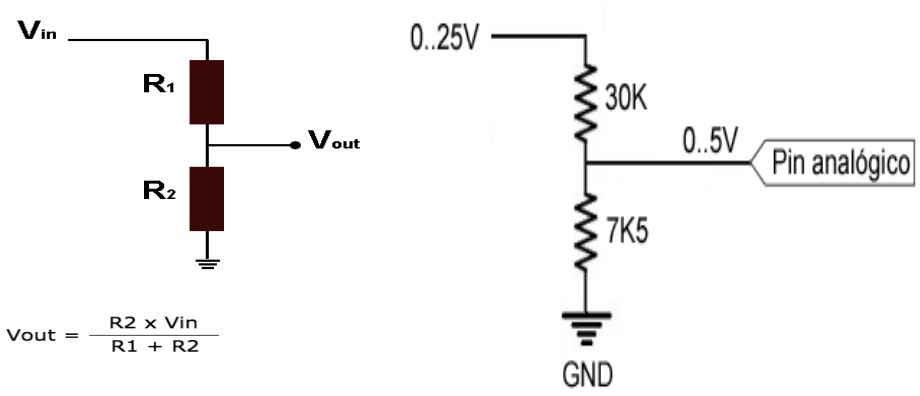


Figura 4.18: A la izquierda divisor de tensión genérico, a la derecha tenemos el divisor de tensión implementado en el sensor de voltaje

4.2.3.6.3. Conexionado

Este es el apartado donde este sensor nos proporciona una gran ventaja, ya que, al ser un simple divisor de tensión, no sería difícil de hacer directamente. Sin embargo, este sensor ya trae incorporado un conector donde poner los cables de alimentación de la batería (Figura 4.19) y 3 pines Dupont para conectar los cables que van al Arduino. Aunque de estos tres pines solo usaremos dos, el de señal que va a un pin analógico del Arduino y el de tierra que va a GND.

Figura 4.19: Esquema de conexión sensor de voltaje

4.2.3.6.4. Obtención de datos

Para obtener la medida de voltaje hemos de realizar un cambio de escala de los valores que se obtienen con el *analogRead* a los valores de 0 a 25 V. Esto se hace con un simple cambio de escala lineal.

4.2.3.7. IMU

4.2.3.7.1. Características

La IMU que vamos a utilizar es el modelo MPU-6050. Que consta de un acelerómetro y un giroscopio de tres ejes cada uno.

En lo que se refiere al giroscopio tiene salida digital. Este sensor se puede alimentar con voltajes desde 3 a 5 V. Posee un rango de escala completamente programable por el usuario de ±250, ±500, ±1000, y ±2000 °/sec.

Pasando al acelerómetro también tiene salida digital y al igual que el giroscopio posee un rango de escala completamente programable de ±2g, ±4g, ±8g y ±16g.

4.2.3.7.2. Funcionamiento

Para tratar el funcionamiento vamos a analizar por separado el giroscopio y el acelerómetro, puesto que realmente son dos sensores distintos, aunque vengan unidos en una misma placa.

4.2.3.7.2.1. Giroscopio

Para poder entender los sensores giroscópicos electrónicos primero vamos a analizar en qué se basa un sensor de este tipo. El giróscopo fue inventado por Foucault para aplicarlo en sus experimentos sobre la rotación terrestre. Este aparato consiste en un disco que gira alrededor de un eje a gran velocidad, cuando se produce una alteración en la inclinación se provoca un movimiento de precisión que contrarresta a este último. El efecto giroscópico asociado a este aparato es apreciable por ejemplo en las motos donde a partir de los 30 km/h la moto se mantiene recta gracias a este. De hecho, esto se está aprovechando para desarrollar motos que no se caen, al incorporar un gran giroscopio que cuando se desequilibra la moto produce un movimiento para contrarrestar el desequilibrio.

Pasando a cómo se aplica todo esto a los sensores para electrónica, lo primero que hemos de tener en cuenta es que son aparatos diseñados para conocer la velocidad de rotación angular y son por lo general sensores pequeños, no se puede usar un gran giroscopio. A este respecto la tecnología ha evolucionado enormemente. Tradicionalmente estos sensores tenían funcionamiento mecánico usando unas bolas. Hoy en día lo más extendido son los giroscopios de estructura vibrante que se basan en que un objeto vibrante tiende a continuar vibrando en el mismo plano que gira su apoyo. Con esto y teniendo en cuenta el efecto de la fuerza de Coriolis que hace que se deforme la estructura, podemos medir la variación de la capacitancia del sistema. Este principio ha permitido la construcción de diversos tipos de giroscopio entre los que destacan los sistemas micro-electromecánicos o MEMS (*Microelectromechanical systems*) y los giroscopios piezoeléctricos. Estos sensores por construcción, ya que usan la fuerza de Coriolis, miden la velocidad angular. Si quisiéramos conocer el ángulo girado, es decir, la posición, habría que realizar una integración. Esto sin embargo no es nada bueno pues introduce ruido y conlleva acumulación de errores. Esto se debe a que la integral se hace con una función continua pero el sensor lo que nos da son muestras discretas cada cierto intervalo de tiempo, por lo que para hacer esta operación habría que tomar muchas muestras en un intervalo corto de tiempo y después intentar obtener una función que se aproxime a lo que hemos muestreado, lo que inevitablemente va a introducir ruido al sistema. El principal problema de este ruido de medida es que al integrarlo dispara el valor final.

Los giroscopios funcionan muy bien para movimientos cortos o bruscos, a medio o largo plazo acumulan error, sobre todo aquellos que integran para obtener el ángulo, por lo que son menos fiables.

4.2.3.7.2.2. Acelerómetro

Los acelerómetros se utilizan para medir de aceleraciones, proporcionando una señal eléctrica según la variación física. Para comprender el funcionamiento de un sensor de este tipo es importante tener en cuenta la segunda ley de Newton, que postula que la fuerza neta aplicada sobre un cuerpo es proporcional a la aceleración que adquiere dicho cuerpo. Este efecto es el que se aprovecha para construir este tipo de sensores. En la Figura 4.20 se aprecia claramente el principio de funcionamiento.

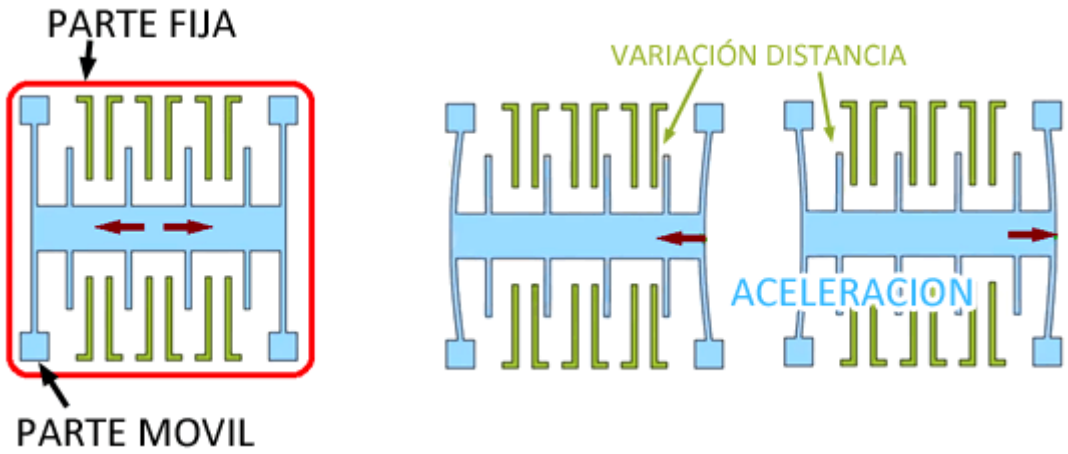


Figura 4.20: Principio de funcionamiento del acelerómetro

Cuando el sensor experimente una aceleración, la masa (parte azul) ejercerá una fuerza sobre los muelles, haciendo que la posición relativa varie. Si medimos la variación de distancia vamos a poder determinar la magnitud de aceleración. El desplazamiento será proporcional a la aceleración soportada, dado que la masa se mantiene constante.

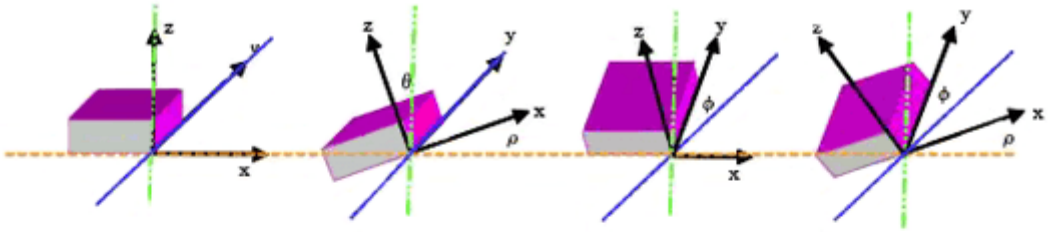
Dentro de los acelerómetros encontramos varios tipos, uno de ellos es el capacitivo, que además se ve claramente cómo podría funcionar, pues tendríamos dos placas con un dieléctrico en medio que variaría su espesor en función de la aceleración. Otros tipos son los mecánicos, piezoeléctricos o piezo resistivos. El mecánico usa galgas extensiométricas que miden en este caso la deformación, que se traduce en variación de corriente detectada por un puente de Wheatstone. La deformación es directamente proporcional a la aceleración.

Un tipo muy interesante de acelerómetro es el piezoeléctrico, sabiendo que los cristales piezoeléctricos adquieren una diferencia de potencial en su superficie al serles aplicada fuerza mecánica. Se aprovecha esto de forma que cuando una masa conocida debido a una aceleración ejerce fuerza sobre el material podemos calcular la aceleración.

Al igual que ocurre con el giroscopio, el acelerómetro es adecuado para medir aceleraciones, pero no para determinar velocidades angulares ni posiciones pues tal y como se explicó antes

los procesos integrativos respecto al tiempo generan errores. Además de esto los acelerómetros son muy sensibles a las vibraciones por lo que por norma general las mediciones presentaran ruido de alta frecuencia, por lo que se debe filtrar la señal antes de usarla. Estos dispositivos funcionan mejor a medio o largo plazo que a corto plazo donde afecta mucho el ruido.

Otro dato a destacar de estos sensores es que se ven afectados por la gravedad terrestre, por tanto, es un dato que registran. El sensor, si es de tres ejes como el que incluye la IMU que vamos a usar, registra la aceleración en estos, lo que es interesante de cara a que podemos determinar la posición del sensor usando relaciones trigonométricas. Esto se aprecia claramente en la Figura 4.21.



$$\theta_x = atan\frac{A_x}{\sqrt{{A_y}^2+{A_z}^2}}$$

$$\theta_y = atan\frac{A_y}{\sqrt{{A_x}^2+{A_z}^2}}$$

$$\theta_z = atan\frac{\sqrt{{A_x}^2+{A_y}^2}}{A_z}$$

Figura 4.21: Obtención de la posición en función de las aceleraciones de los tres ejes

4.2.3.7.3. Conexionado

Este sensor va a ir conectado directamente a la Raspberry Pi (Figura 4.22) para evitar tener que pasar un mensaje de tipo IMU que es bastante pesado con el Arduino. El sensor va a tomar la alimentación de 3,3 V de la Raspberry, y la vamos a conectar con I^2C a la misma.

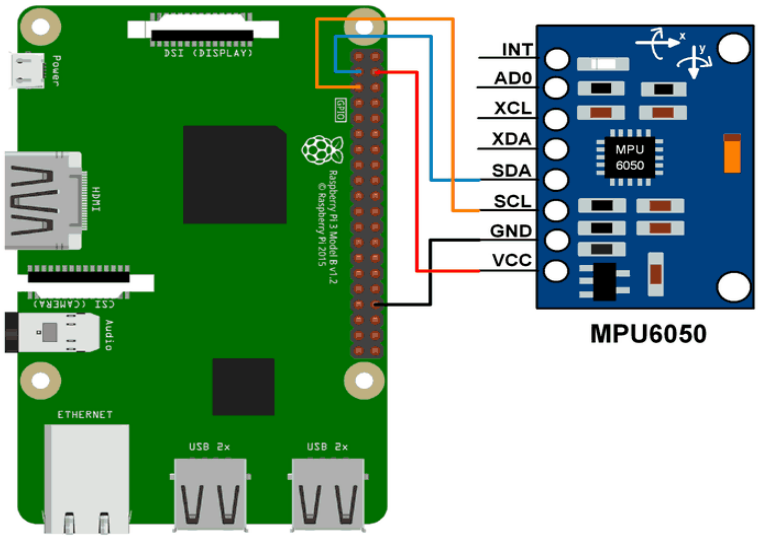


Figura 4.22: Conexionado de la IMU MPU-6050 con la Raspberry Pi

4.2.3.7.4. Obtención de datos

Para obtener los datos vamos a utilizar el bus de comunicación I^2C que permite el uso de solo dos cables para el funcionamiento: uno la señal de reloj (CLK) y otro para envío de datos (SDA). Una característica importante de este bus es que es síncrono. La información del sensor la vamos a obtener en forma de cuaterniones, para poder trabajar con ella en el entorno de ROS.

4.3. Pruebas

Para comprobar el funcionamiento de los sensores se realizaron pruebas de los mismos por separado, utilizando códigos simples, para posteriormente unirlos y montar todo el sistema de sensores.

Para poder realizar todas las pruebas fue necesario instalar una serie de librerías para Arduino que se pueden ver en los *include* del código.

Las pruebas de los sensores más simples se hicieron sin demasiados problemas. El sensor que más tiempo llevó probar fue el de presión externa pues requerimos de un compresor para poder comprobar las medidas que obtenemos. En este apartado se va a detallar el proceso de prueba de este sensor y la implementación del conjunto ya que las pruebas de los demás sensores de forma individual no tienen tanto interés.

4.3.1. Prueba del sensor de presión

Lo primero que tenemos que hacer es implementar una función que nos dé el valor de presión en función de la caída de tensión en la resistencia. Para ello utilizamos una hoja de Excel (Tabla 4.2) en el que representemos la función que relaciona valor analógico medido por el Arduino y presión en bares tal y como se observa en la Figura 4.23.

R (Ohmios)	Presión (Bares)	Valor sensor (Analog output)	Corriente (Amperios)	Voltaje (Voltios)
220	0	180	0,004	0,88
	10	900	0,02	4,4

Tabla 4.2: Datos para obtener respuesta sensor de presión

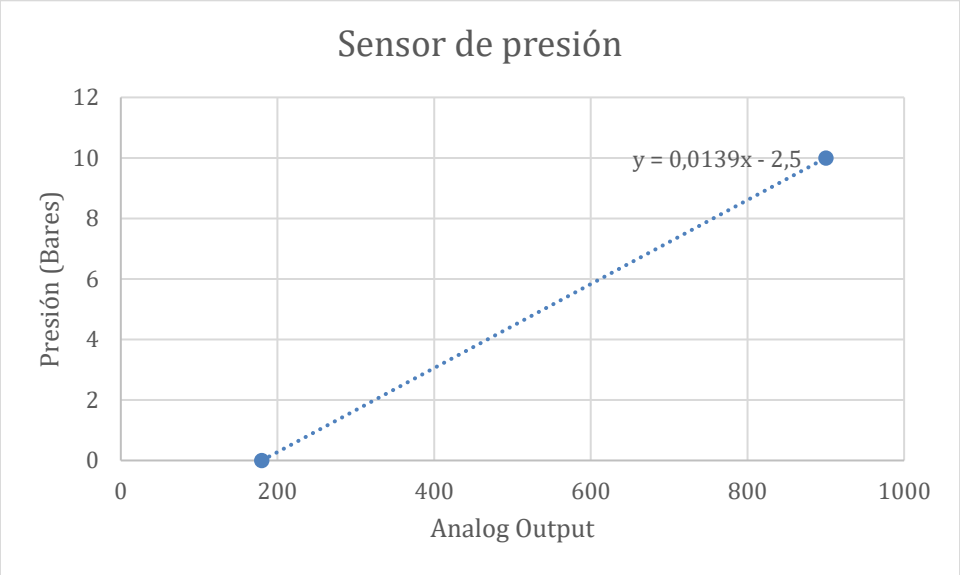


Figura 4.23: Recta de respuesta del sensor de presión

Teniendo en cuenta la resistencia de 220 **Ω** y la corriente que puede circular por la misma, en caso de que esté a 0 bares será 4 mA y si está a 10 bar será 20 mA. Con esto obtenemos la caída de tensión en la resistencia, que es la columna Voltaje, y pasamos eso a los valores que vamos a obtener del conversor analógico-digital del Arduino. Hacemos una interpolación lineal de los puntos y obtenemos la función que relaciona el valor obtenido del Arduino con la presión.

Para comprobar la adecuación de la función obtenida a la realidad en primer lugar realizamos el conexionado del sensor (Figura 4.24: Conexionado del sensor de presión SICK), lo conectamos a una fuente de tensión y comprobamos con un *tester* que obtenemos caída de tensión en la resistencia.

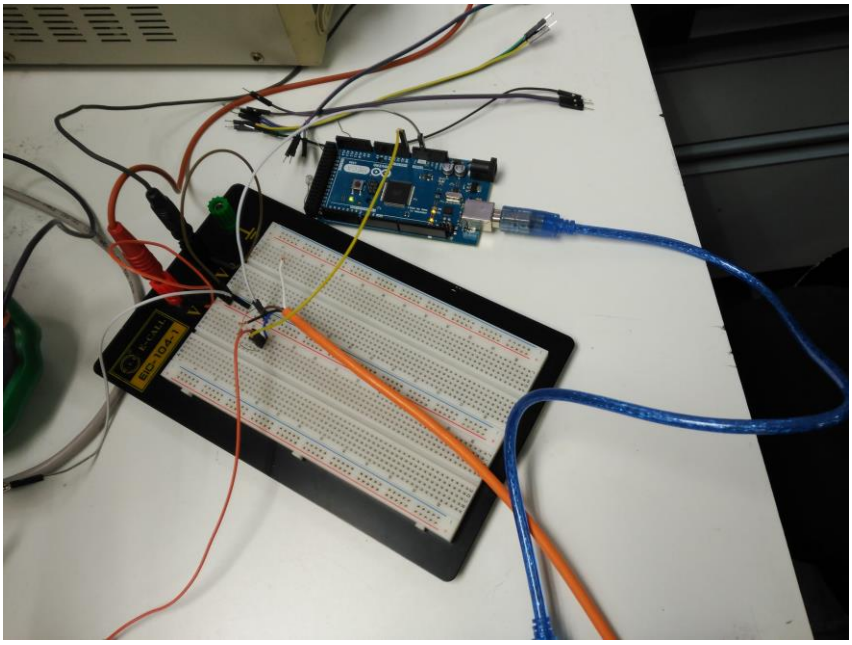


Figura 4.24: Conexionado del sensor de presión SICK

A continuación, conectamos el sensor a un compresor (Figura 4.25) y vamos probando con distintos valores de presión y observando la salida para ver si se corresponden.

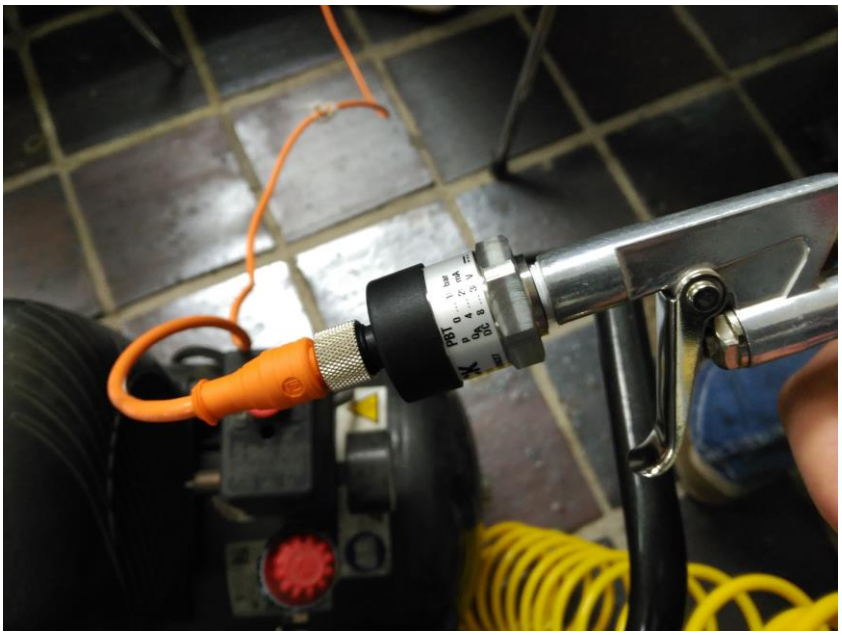


Figura 4.25: Sensor de presión SICK conectado al compresor

Presión compresor (Bares)	Valor sensor (Analog output)	Presión medida (Bares)	Error relativo
1	229	0.99	1.00%
1.5	269	1.53	2.00%
2	297	1.98	1.00%
2.5	333	2.52	0.80%
3	369	3.01	0.33%
3.5	400	3.5	0.00%
4	433	3.96	1.00%
4.5	466	4.49	0.22%
5	501	5.01	0.20%

Tabla 4.3: Valores medidos con el sensor SICK y error relativo

Los resultados obtenidos son satisfactorios, teniendo en cuenta que el valor de presión que damos con el compresor no se puede conocer de forma exacta porque tiene una aguja para indicar el valor de presión que no permite tener toda la precisión que nos gustaría. Tal y como se puede observar en la Tabla 4.3 el máximo error relativo que obtenemos es un 2%, es un valor admisible, que puede venir dado por la propia imprecisión de las pruebas.

4.3.2. Prueba del sistema de sensores

4.3.2.1. Arduino

Para realizar la prueba del sistema de sensores primero montamos todo el circuito en una *protoboard* (Figura 4.26).

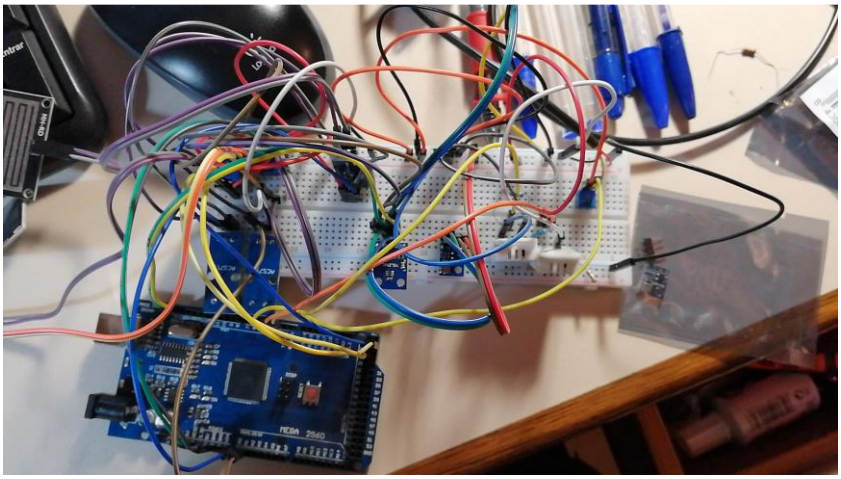
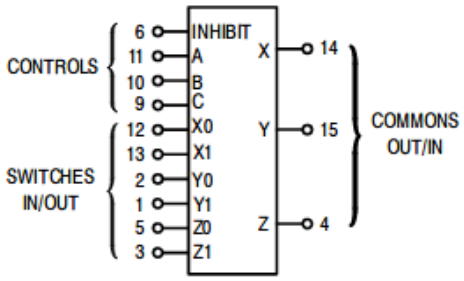


Figura 4.26: Sistema de sensores montado en protoboard

Las pruebas se realizaron con varios sensores duplicados para poder proporcionar capacidad de ampliación en caso de que fuera necesario, en este sentido se montaron dos DHT22 y dos BMP180.

A la hora de realizar las pruebas nos encontramos con que el BMP180 que usa I^2C tiene siempre la misma dirección I^2C por lo que no se podrían utilizar dos a la vez. Para solventar este problema se recurrió al uso del multiplexor/demultiplexor analógico MC14053B, de forma que activemos cada BMP180 de forma independiente. La *datasheet* se puede consultar en el anexo 13.

MC14053B
Triple 2-Channel Analog Multiplexer/Demultiplexer



V_{DD} = PIN 16
V_{SS} = PIN 8
V_{EE} = PIN 7

Control Inputs				ON Switches					
	Select								
Inhibit	C*	B	A	MC14051B	MC14052B		MC14053B		
0	0	0	0	X0	Y0	X0	Z0	Y0	X0
0	0	0	1	X1	Y1	X1	Z0	Y0	X1
0	0	1	0	X2	Y2	X2	Z0	Y1	X0
0	0	1	1	X3	Y3	X3	Z0	Y1	X1
0	1	0	0	X4			Z1	Y0	X0
0	1	0	1	X5			Z1	Y0	X1
0	1	1	0	X6			Z1	Y1	X0
0	1	1	1	X7			Z1	Y1	X1
1	x	x	x	None	None		None		

*Not applicable for MC14052
x = Don't Care

Figura 4.27: A la izquierda pines del MC14053B, a la derecha tabla de verdad de este

Para hacer esto conectamos los pines del Arduino de I^2C, SDA y SCL a los pines X e Y del demultiplexor. Por otro lado, conectamos A y B a dos pines digitales del Arduino para controlar los interruptores activos, y el pin de inhabilitación (*inhibit*) a tierra. Por último, conectamos los pines SDA y SCL de un sensor a X0 e Y0 y los del otro sensor a X1 e Y1 (Figura 4.27). Con

esto estamos conectando un sensor cuando los dos pines digitales de control están a ‘0’ y el otro cuando están los dos a ‘1’.

De esta forma ya tendríamos todo el sistema de sensores funcionando así que ya podríamos pasar al diseño de la PCB.

4.3.2.2. Raspberry Pi

En la Raspberry Pi vamos a probar el funcionamiento de la IMU. Para ello la conectamos utilizando el bus I^2C de la Raspberry, como se explicó anteriormente. Para esta prueba implementamos directamente la lectura de la IMU con ROS para poder ver la orientación de la misma utilizando la ventana del módulo *Pose_View* que se puede ver en la Figura 4.28. Para esto vamos a usar un código de la que se encuentra en el repositorio de *github* [31] con el que vamos a obtener las medidas de orientación en cuaterniones, así que para poder usar el código con *Pose View* vamos a publicar un mensaje de tipo *pose* en el que pongamos a 0 la posición y actualicemos la información de orientación con el valor de los cuaterniones. El código adaptado se puede consultar en el anexo 1.

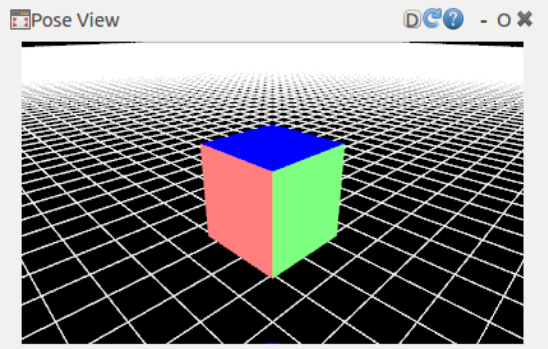


Figura 4.28: Ventana Pose view

Capítulo 5. Iluminación

5.1. Requerimientos

El objetivo de este apartado es dotar a la cámara del dron de un sistema de iluminación que nos permita conectar una luz que se alimente con corriente continua a máximo 12 V. Este sistema tiene que poder ser controlado por medio del Arduino para poder apagar y encender la luz en cualquier momento con el mando de control.

5.2. Hardware

Para realizar este sistema usamos una bombilla de coche de 12 V que nos permita ilustrar el funcionamiento del sistema. La luz que tenemos utiliza tecnología halógena, esto implica un mayor consumo que una de tecnología LED, además de que la experiencia de iluminación no es del todo positiva por el tono de la luz, muy amarillento. Sin embargo, el sistema permite un cambio de forma sencilla de la luz. Es por esto que usamos esta bombilla a modo de prueba, esperando disponer de una de tecnología LED.

Para poder controlar esta luz desde el Arduino utilizaremos un par Darlington NPN, concretamente el BD649, su *datasheet* se puede consultar en el anexo 19. También vamos a utilizar una resistencia entre la base del transistor y el pin PWM (*Pulse Width Modulation*) de control del Arduino.

5.3. Diseño

Para poder controlar la luz con el pin PWM del Arduino conectamos este a la base del transistor unido con una resistencia de 2,2 k**Ω** que nos sirva de limitador de corriente (Figura 5.1). Por otro lado, unimos el emisor del transistor a la tierra común de la batería y el Arduino. El polo positivo de la batería va conectado a la luz y la otra conexión de la bombilla va directamente al colector del transistor.

Con esta configuración podemos controlar no solo el encendido y apagado de la luz, sino que también controlamos la intensidad.

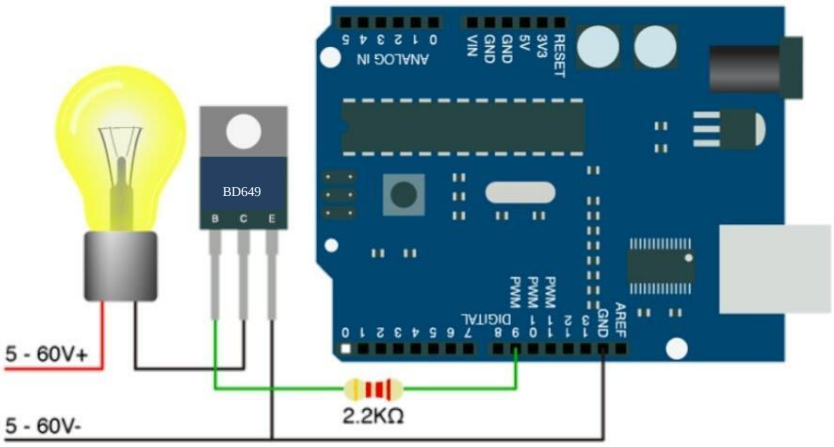


Figura 5.1: Esquema de conexión del sistema de iluminación

5.4. Pruebas

Una vez montado todo el circuito se realizó una sencilla prueba (Figura 5.2) en la que el valor de PWM iba variando, consiguiendo un efecto de encendido y apagado progresivo. Durante esta prueba se conectó un amperímetro con el fin de conocer la intensidad que va a consumir la luz. Además, se realizaron pruebas con distintas luces para ver la que nos ofrecía mejor relación de consumo y calidad de iluminación.

Figura 5.2: Prueba de sistema de iluminación

La bombilla que finalmente seleccionamos consumía en torno a los 0,8 A a pleno rendimiento.

5.5. Montaje

Esta bombilla va a ir dentro de la misma carcasa en la que se encuentra la cámara (Figura 5.3). La explicación detallada del montaje se encuentra en el TFG de David Henry. En la siguiente figura se puede ver cómo quedó el resultado final.

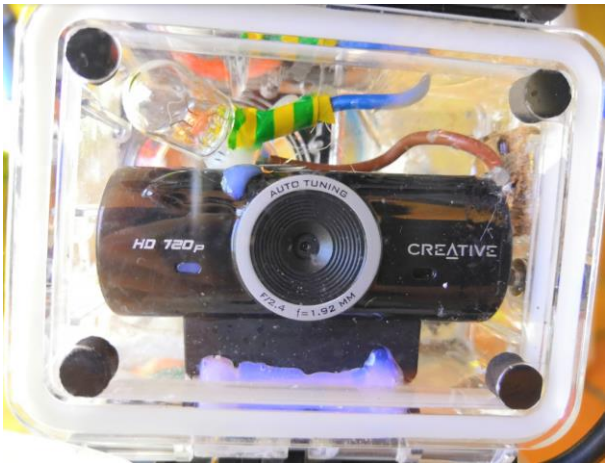


Figura 5.3: Montaje de la cámara y sistema de iluminación en carcasa de metacrilato

Capítulo 6. Diseño de PCB

Para el proyecto hay que diseñar dos placas, una va a ser utilizada para todos los sensores que van conectados con el Arduino, al igual que para la alimentación del sistema de iluminación de la cámara.

Por otro lado, vamos a realizar una PCB que irá acoplada encima de la Raspberry Pi, con el conversor necesario para la alimentación, seleccionado en el Trabajo de Fin de Grado de David Henry, y con la IMU.

6.1. Arduino

6.1.1. Esquema

El esquema electrónico del sistema de sensores se realiza en KiCAD para posteriormente realizar el diseño físico de la PCB. Este esquema se puede consultar en la Figura 6.1.

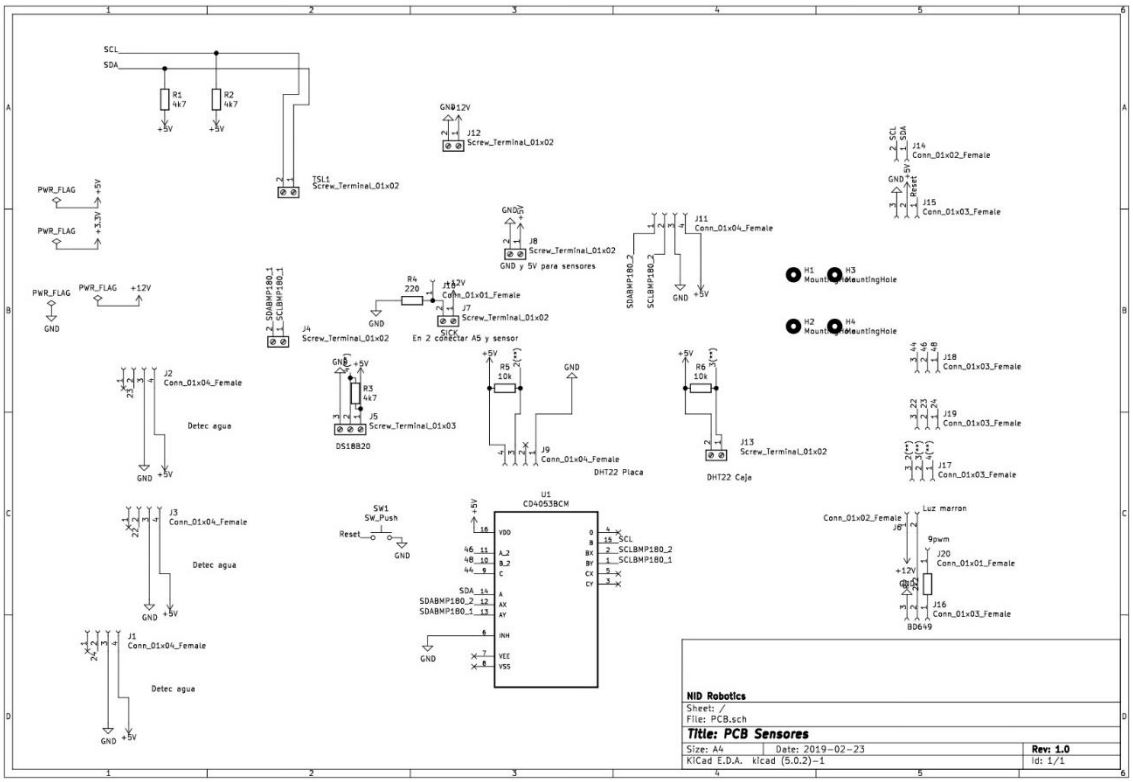


Figura 6.1: Esquema eléctrico de la PCB para el sistema de sensores

6.1.2. Diseño

Para el diseño de la PCB también se va a utilizar el KiCAD. Los sensores para realizar las medidas interiores van a ir directamente soldados a la placa mientras que los sensores que se usan para realizar medidas del exterior se van a conectar con terminales de tornillo (Figura 6.2).

Figura 6.2: Terminal de tornillo

Además, en la placa se va a disponer de varios terminales que en este primer prototipo no se van a usar, pero que nos proporcionan posibilidades de ampliar el sistema. La PCB que se va a realizar va a ser de una sola capa de cobre. Tanto las resistencias como el MC14053B van a ser de tecnología SMD lo que nos va a permitir ganar bastante espacio a la hora de hacer el diseño.

El diseño final de esta placa se puede ver en la Figura 6.3.

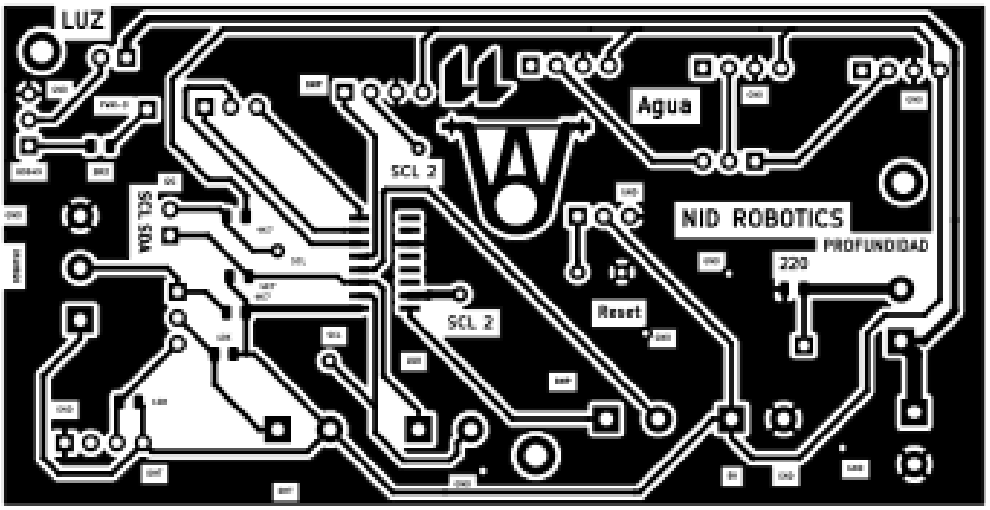


Figura 6.3: Fotolito de la PCB para los sensores

6.1.3. Fabricación

A continuación, pasamos a describir el proceso de fabricación de la placa de circuito impreso.

En primer lugar, recortamos el fotolito y limpiamos la placa de cobre.

Figura 6.4: Recorte del fotolito

Figura 6.5: Limpieza de la placa de cobre

Una vez hemos hecho esto pasamos a pegar el fotolito a la placa de cobre utilizando calor.

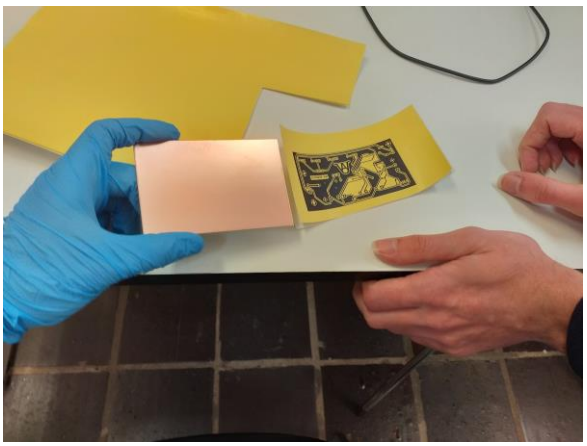


Figura 6.6: Placa de cobre y fotolito antes de pegarlos

Figura 6.7: Plastificadora utilizada para que el fotolito quede adherido a la placa de cobre

Figura 6.8: Fijación del fotolito a la placa de cobre

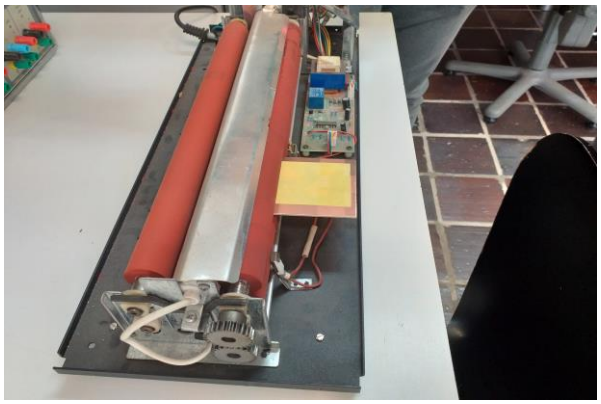


Figura 6.9: Proceso de pegado con calor del fotolito a la placa de cobre

Pasado unos 10 minutos sacamos la placa del calentador, y retiramos el papel amarillo, como se ve a continuación tenemos en la placa de cobre la impresión del fotolito. Como quedan algunos huecos debido a la mala adhesión repasamos los huecos con rotulador.

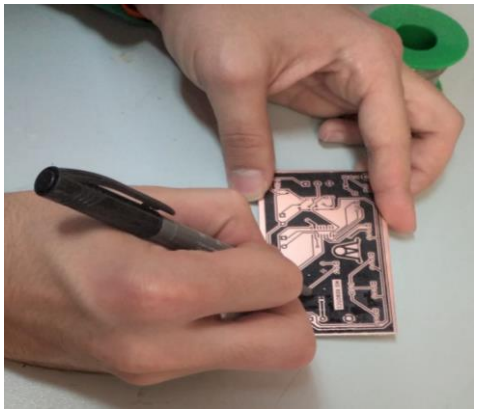


Figura 6.10: Eliminar los pequeños huecos donde no se pegó bien el fotolito

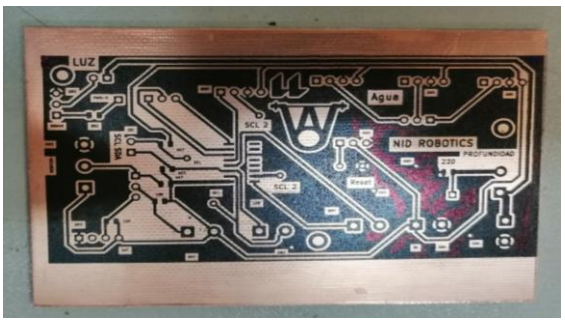


Figura 6.11: Placa de cobre con las imperfecciones corregidas

Ahora recortamos los bordes de la placa y posteriormente metemos la placa en el baño de ácido donde tenemos que esperar hasta que desaparezca todo el cobre visible.

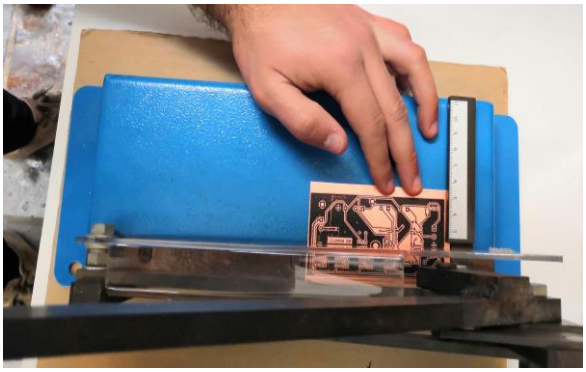


Figura 6.12: Recorte de la placa de cobre

Figura 6.13: Baño de ácido

Cuando acabamos este proceso limpiamos la placa y comenzamos a hacer los taladros.

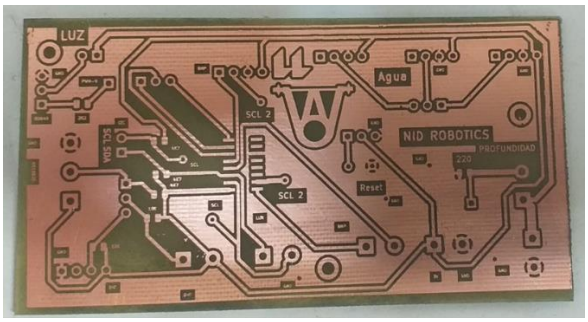


Figura 6.14: Placa limpia tras sacarla del ácido

Figura 6.15: Realización de taladros en la placa

Con todos los taladros hechos podemos pasar a situar la pasta de soldadura y el estaño para los componentes SMD y encima los componentes SMD.

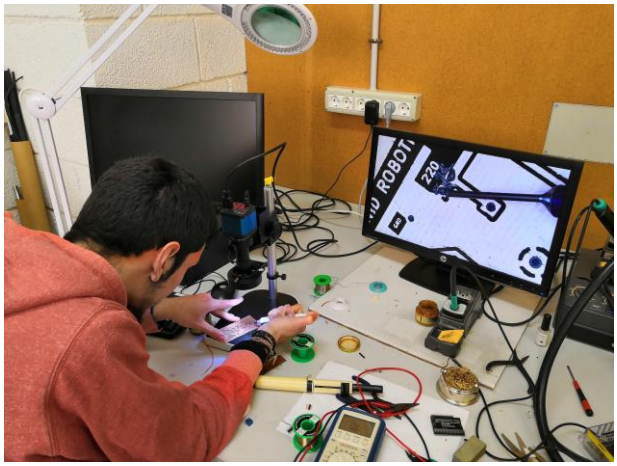


Figura 6.16: Ponemos el estaño para los componentes SMD

Figura 6.17: Puesta de estaño para componentes SMD

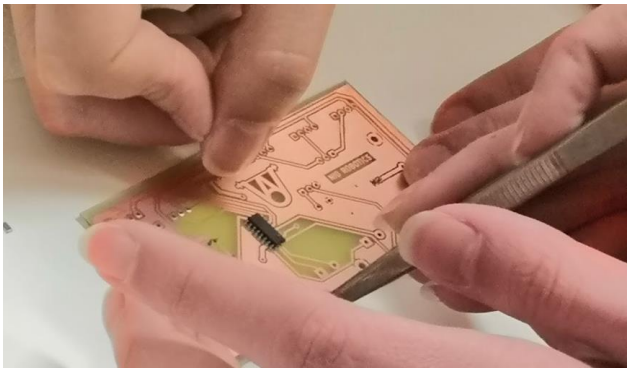


Figura 6.18: Situamos los componentes SMD en la placa

Una vez situados todos los componentes SMD vamos a poner la placa en el horno que tiene un perfil de temperatura establecido para que se suelden los componentes de forma correcta.

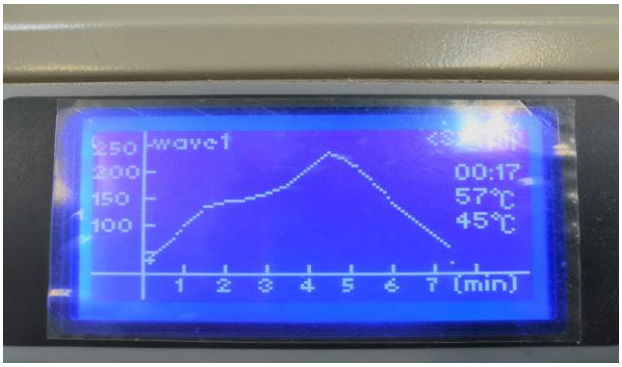


Figura 6.19: Perfil de temperatura del horno para la soldadura de los componentes SMD

Figura 6.20: PCB en el horno con los componentes SMD soldados

Cuando termina este proceso, pasamos a comprobar que las soldaduras hayan quedado bien, es decir que no hay cortocircuitos y que los componentes hagan buen contacto, lo que hacemos con la lupa electrónica y el multímetro.

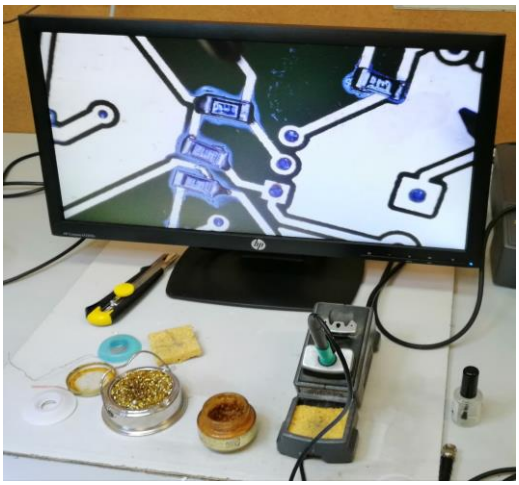


Figura 6.21: Comprobación de la soldadura de los componentes SMD

Las soldaduras que no hayan quedado bien las corregimos haciendo uso del soldador y de la pistola de calor.

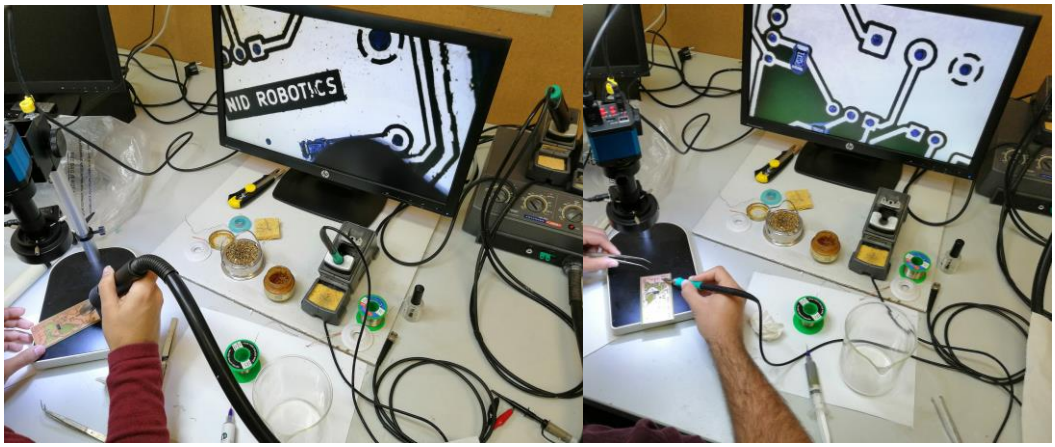


Figura 6.22: A la izquierda corrección de soldaduras SMD con pistola de calor a la derecha con soldador

Cuando terminamos este paso, realizamos las soldaduras de los demás componentes.

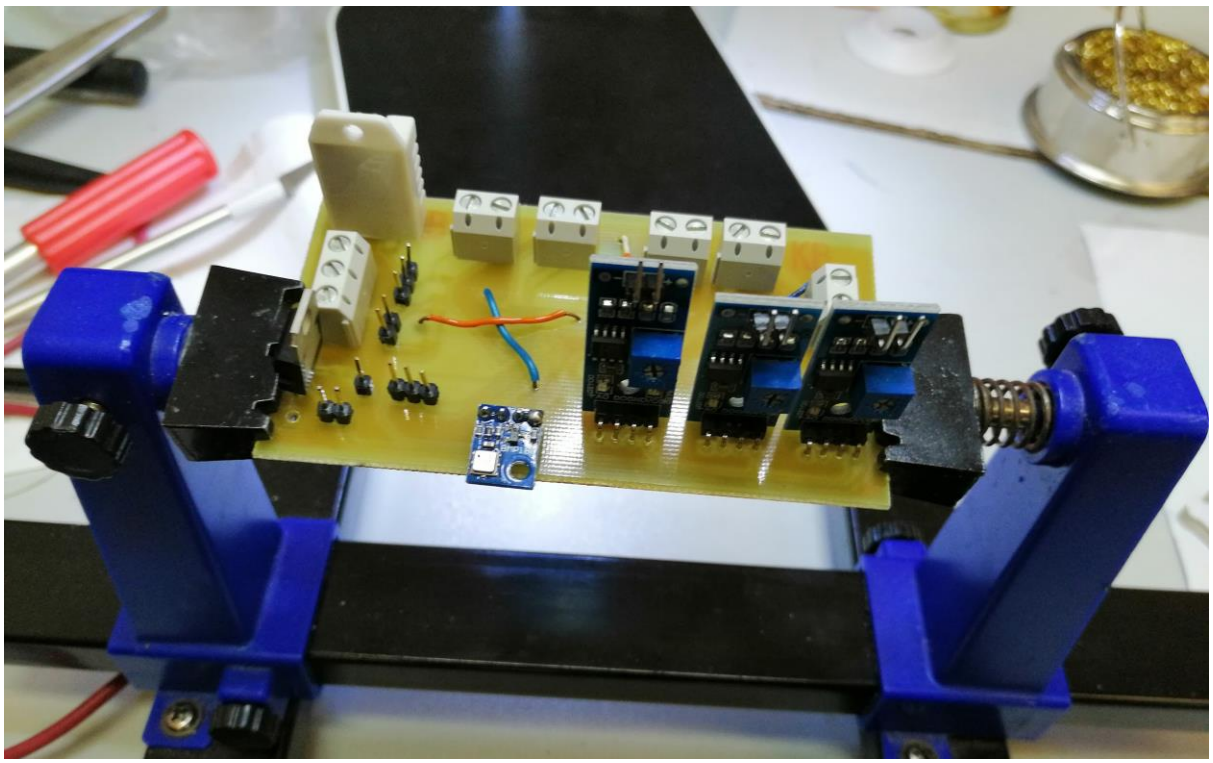


Figura 6.23: PCB de los sensores terminada

Ahora comprobamos que no hay ningún cortocircuito o mal contacto. Tras esto comprobamos el correcto funcionamiento de la placa y le ponemos varias capas de laca en la cara de cobre, esto es de especial importancia para evitar la oxidación.

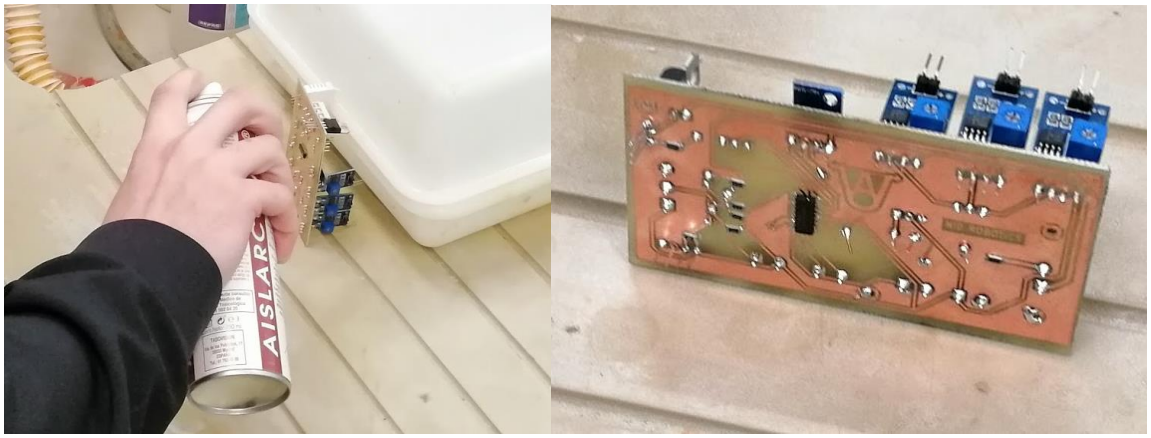


Figura 6.24: Puesta de laca en la PCB para evitar oxidación y desgaste en general

6.2. Raspberry

6.2.1. Esquema

El esquema de esta PCB (Figura 6.25) es bastante más sencillo que en el caso de los sensores ya que en este caso la PCB solo va a contar con un conector para la alimentación que llega del sistema de rectificación usado para estabilizar el voltaje de la batería, la IMU y el conversor DC-DC para pasar a los 5V de entrada de la Raspberry.

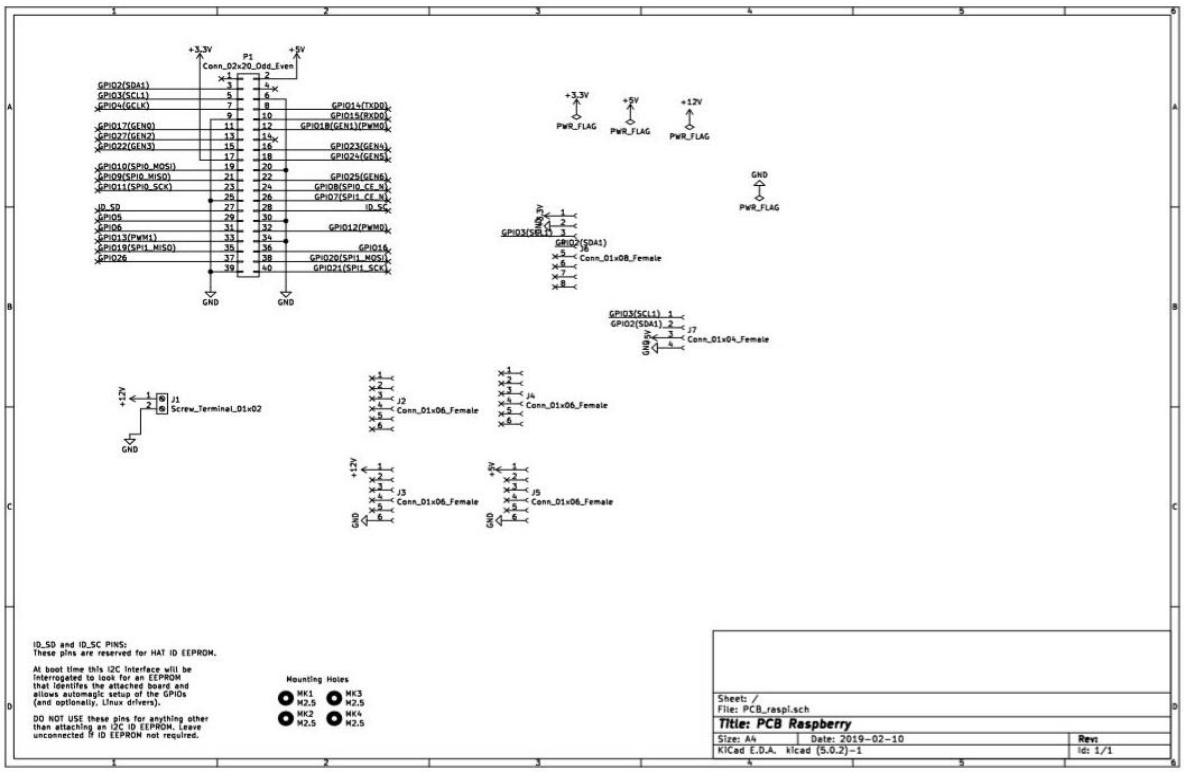


Figura 6.25: Esquema eléctrico de la PCB para la Raspberry Pi

6.2.2. Diseño

En este caso el diseño va a ser el de un módulo que va directamente acoplado encima de la Raspberry Pi como si se tratase de un *shield* de Arduino, pero para la Raspberry Pi, con lo que de esta forma ahorramos espacio y cableado.

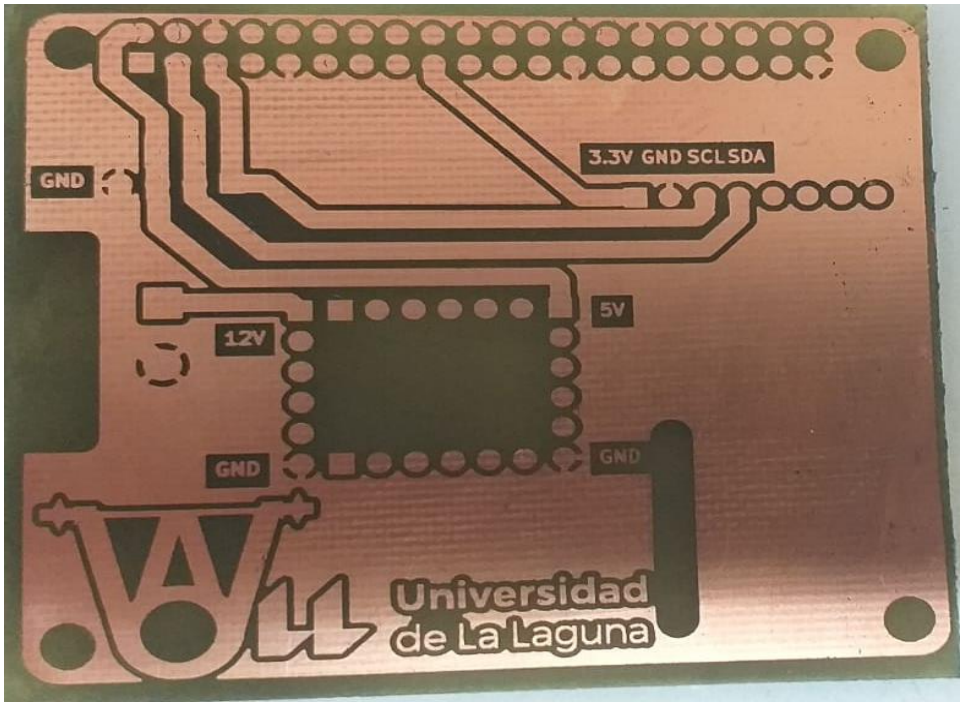


Figura 6.26: PCB de la Raspberry Pi tras el ataque ácido

6.2.3. Fabricación

El proceso de fabricación fue el mismo que el descrito para la PCB que va conectada con el Arduino. Para no repetir todo el proceso de fabricación sólo se muestra el resultado final en la Figura 6.27.

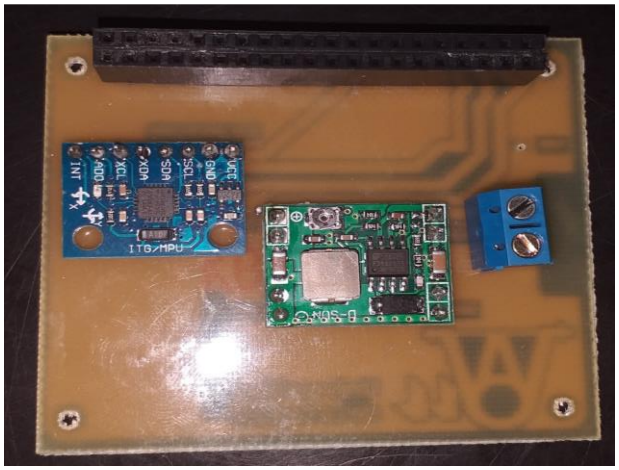


Figura 6.27: PCB para la Raspberry Pi terminada

Capítulo 7. ROS en el sistema de sensores

7.1. Introducción

En este apartado se va a explicar cómo se adapta el código de Arduino, para que usando la librería de ROS y el paquete *rosserial_arduino*, mandemos todos los mensajes necesarios con la información de los sensores y recibamos información del mando de control para activar la luz. El código completo se puede consultar en el anexo 2.

7.2. Enviar datos de los sensores

Lo que se va a tratar es la elección de los mensajes que utilizaremos para enviar la información de cada sensor. Como se comentó en la parte inicial de ROS, este tiene definido una serie de mensajes que podemos utilizar. Para este caso vamos a hacer uso de los paquetes de mensaje *common_msgs* [32] y *std_msgs* [33]. Por otro lado, para la detección de agua vamos a servirnos de las herramientas de *logging* de ROS [34].

7.2.1. Detección de agua

Para este apartado utilizamos los *logs* que nos proporciona ROS, que pueden ser después mostrados en la interfaz de usuario de forma sencilla. Existen diferentes niveles de *logs*:

- *Debug*
- *Information*
- *Warning*
- *Error*
- *Fatal*

Dada la implicación que tiene que en el submarino entre agua vamos a tomar el nivel *Fatal*, porque es un fallo crítico. El fallo irá acompañado de un *string* que nos dirá en qué lugar se ha detectado agua.

7.2.2. Temperatura

Dentro de los *common_msgs* de ROS encontramos un mensaje destinado para la temperatura. Tanto para la temperatura interior como exterior vamos a utilizar este tipo de mensaje que tiene la siguiente estructura:

- std_msgs/Header *header* [35]
- *float64 temperature*

- *float64 variance*

En el *header* se incluye un *time stamp* que usamos para conocer el momento en que se envió el mensaje. Después tenemos un *float64* para poner la temperatura y la varianza, que si se deja a 0 se considera como desconocida, que es lo que vamos a hacer en nuestro caso.

7.2.3. Presión

Para enviar los datos de la presión interna y externa vamos a usar el mensaje definido dentro de *common_msgs* llamado *FluidPressure,* cuya estructura es:

- *std_msgs/Header header* [35]
- *float64 fluid_pressure*
- *float64 variance*

El *header* es igual en todos los casos, y la varianza funciona igual en todos los mensajes que usamos, si está a 0 se toma como desconocida. En *fluid_pressure* ponemos el valor de la presión, en la documentación se habla de usar como unidad los pascales. Sin embargo, se van a utilizar los bares que es la unidad en la que obtenemos las dos medidas y lo mandamos a través de este mensaje especificando en el programa este cambio para no crear un nuevo mensaje ni hacer la conversión a pascales, ya que estamos más acostumbrados a trabajar con bares.

7.2.4. Humedad

En cuanto al dato de la humedad, existe un tipo de mensaje que es *RelativeHumidity* que está pensado para enviar la humedad relativa, su estructura es:

- *std_msgs/Header header* [35]
- *float64 relative_humidity*
- *float64 variance*

Como se puede observar, todos los mensajes tienen una estructura igual. Este mensaje recomienda enviar la humedad relativa en valores de 0 a 1 en lugar de 0 a 100.

7.2.5. Voltaje

Para enviar el voltaje de la batería vamos a utilizar un mensaje del paquete *std_msgs*, que será simplemente un *float32*. Aquí solo tenemos que incluir el número que queremos enviar.

7.2.6. IMU

Para la IMU se va a hacer uso del mensaje *Pose* que está definido dentro de los *geometry_msgs* que están en el paquete *common_msgs*. Esto se debe a que para poder ver la representación de la posición en la interfaz necesitamos este tipo de mensaje, su estructura es la siguiente:

- *geometry_msgs/Point position* [36]
- *geometry_msgs/Quaternion orientation* [37]

Por un lado, tenemos la posición que nosotros no vamos a conocer pues no tenemos ningún sistema de localización, ni vamos a realizar ninguna estimación, por tanto, la establecemos a 0. Por otro lado, tenemos la orientación que se define utilizando cuaterniones que son una herramienta matemática para representar orientaciones y rotaciones de objetos en tres dimensiones. El valor del cuaternión en los diferentes instantes de tiempo lo obtenemos a partir de la IMU.

7.3. Leer datos del mando

En el sistema de ROS existe un nodo que publica los botones que están pulsados en el mando de Playstation 3 ®. La información ampliada sobre este sistema se encuentra en el TFG de Nicolás Rodríguez. Lo que hay que hacer con esos botones publicados es leerlos en el Arduino para al pulsarse una combinación de botones, en este caso 'L1' + '△' se debe encender la luz y 'L1' + '□' apaga la luz. Para hacer esto hay que construir un suscriptor que se suscriba al tópico de los botones que es un array donde está el valor de los botones ('1' totalmente pulsado y '0' suelto) en el siguiente orden: 'X', 'O', '△', '□', 'L1', 'R1', 'L2' y 'R2'. Este tiene asociado un *callback* que se ejecuta cuando se pulsan los botones para activar o desactivar la luz.

7.4. Pruebas

Aparte de las pruebas de los sistemas individualmente se realizó una prueba del conjunto. Para ello conectamos vía Ethernet el PC con la Raspberry Pi y ésta a su vez vía serial con el Arduino. El Arduino estaba conectado a la PCB con todos los sensores y la Raspberry con la IMU. La explicación detallada de la comunicación se puede consultar en el TFG de Nicolás Rodríguez. Una vez tuvimos todo conectado y el programa de los sensores cargado al Arduino, lanzamos un *roscore* y empezamos a lanzar los nodos necesarios para hacer la prueba que serían:

- *Rosserial_python* (Raspberry)
- *Ros_mpu6050_node* (Raspberry)
- *Joy_command* (PC)

Una vez tenemos todo esto vemos que se publican los tópicos de los sensores y podemos acceder a la información. También comprobamos que se enciende la luz al pulsar los botones correspondientes lo que quiere decir que tanto el *publisher* como el *subscriber* de los botones están funcionando correctamente.

Capítulo 8. Visión

8.1. Introducción

En este apartado se va a explicar todo lo relacionado con el sistema de visión que hay en el submarino. Por un lado, tenemos dos funciones básicas que son las de *streaming* de vídeo, fundamental para que el operador pueda manejar el submarino, y la de grabación de vídeo, que nos permite capturar los vídeos de las inmersiones. Por otro lado, nos encontramos con los sistemas de *tracking* y detección de objetos. Dentro de estos se plantean varias alternativas de las que analizaremos las ventajas y desventajas.

8.2. Requerimientos

En lo que se refiere a los requerimientos del sistema de visión, tiene que ser capaz de proporcionar al usuario una imagen en tiempo real de la cámara del submarino para facilitar la tarea de manejo del dron. Además de esto, se le debe proporcionar al usuario la capacidad de realizar una grabación de las imágenes que está viendo en cualquier momento que lo crea conveniente. También debe haber una opción por si lo que necesita es simplemente hacer una captura de la imagen. Esto en cuanto a las funciones básicas del sistema de visión.

Pasando a los requerimientos del sistema de detección, lo que se busca es que haya herramientas que permitan realizar una detección de cualquier objeto que se quiera, aunque esto lleve un proceso de preparación previo. Aparte de esto, debe ser posible tener algún sistema de asistencia a la visión, aunque sea más básico, para cuando el ordenador de tierra no es tan potente. En este sentido se ha implementado un seguimiento basado en características menos prestacional pero más ligero y un sistema de detección complejo apoyado en el uso de redes neuronales, más pesado.

8.3. Hardware

Para la realización de este sistema se empleó una Webcam USB de Creative descrita en el apartado inicial de hardware. También empleamos una Raspberry Pi model 3B+ a la que conectamos la cámara, que está en el dron, y un PC vía Ethernet, que está situado en tierra.

Las pruebas de visión se han realizado con tres equipos de características muy distintas para hacernos una idea del hardware del PC que necesitamos para utilizar las distintas funcionalidades. Para realizar las funciones básicas de *streaming*, grabación y *tracking* basado en características podemos usar un ordenador sin tarjeta gráfica dedicada de nVidia, con un procesador AMD A9 o equivalente y 8 GB de RAM. Para poder realizar detección usando redes neuronales pudimos llegar a utilizar un ordenador sin tarjeta gráfica dedicada de nVidia, cuyas características eran: procesador i7 y 8 GB de RAM. Sin embargo, esto sobrecarga el

ordenador y la experiencia de detección no llega a ser tan positiva como la que obtenemos con el siguiente ordenador que utilizamos. Las características de este último ordenador son: procesador i7, 8 GB de RAM y tarjeta gráfica dedicada de nVidia GT920M con 2 GB de memoria. Como vemos para obtener buenos resultados en la detección de imagen con redes neuronales no es necesario tener una tarjeta gráfica de las más potentes del mercado.

8.4. Funciones básicas

8.4.1. Streaming de vídeo

8.4.1.1. Introducción

Para este apartado usaremos un paquete de ROS ampliamente empleado en sistemas de visión: *usb_cam* [38]. Con este nodo vamos a capturar la imagen de la webcam con la Raspberry Pi y, tras realizar una compresión de esta, se publica el tópico */usb_cam/image_raw/compressed*. Accediendo a este tópico usando el paquete *image_view* podemos ver la imagen en tiempo real que está captando la webcam del dron.

8.4.1.2. Configuración

Para poder utilizar este paquete previamente se deben configurar varios parámetros, en este apartado vamos a ver con detenimiento como se hace. Cuando vamos a utilizar la cámara y queremos incluir el paquete en un archivo *.launch* que nos permite lanzar varios paquetes con una sola línea de comando, se debe incluir lo que vemos en el Código 8.1.

```
<launch>
            <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
              <param name="video_device" value="/dev/video0" />
              <param name="image_width" value="640" />
              <param name="image_height" value="480" />
              <param name="pixel_format" value="yuyv" />
              <param name="camera_frame_id" value="usb_cam" />
              <param name="io_method" value="mmap"/>
```

Código 8.1: Launch file de la Webcam

En primer lugar, tenemos que configurar el dispositivo al que nos vamos a conectar, en caso de que no haya más dispositivos de vídeo conectados, como será el caso de la Raspberry Pi, lo normal sería que la webcam sea el dispositivo *video0*. De todas maneras, esto se puede comprobar usando el comando *ls /dev | grep video**.

Una vez configuramos esto pasamos a establecer el ancho y alto de la imagen en píxeles, en nuestro caso vamos a utilizar 640x480, ya que nos ofrece buenos resultados.

También vamos a elegir el formato de pixel *yuyv* que se basa en el uso del espacio de color YCbCr.

El parámetro *camera_frame_id* nos sirve para identificar la posición de la cámara, vamos a dejar el valor por defecto.

Como parámetro *io_method* vamos a usar *mmap* que es la opción que se utiliza por defecto y aporta buenos resultados.

Si queremos ver la imagen en tiempo real tenemos que usar el paquete *image_view* especificando que estamos utilizando una imagen comprimida, esto se hace con el siguiente comando *rosrun image_view image_view image:=/usb_cam/image_raw _image_transport:=compressed*. Sin embargo, esto solo nos sirve para realizar pruebas, ya que una vez implementemos la interfaz esto se hará de forma automática usando un desplegable. Así el usuario podrá ver los datos y la imagen en una pantalla sin necesidad de estar ejecutando comandos por separado con las distintas configuraciones.

Figura 8.1: Captura obtenida con el dron antes de una inmersión

8.4.2. Grabación de vídeo

Para realizar la tarea de grabación de vídeo vamos a utilizar la librería OpenCV [19]. El código utilizado en este nodo se puede consultar en el anexo 3.

En este nodo necesitamos hacer un suscriptor que se suscriba al tópico */usb_cam/image_raw/compressed* del que obtendremos las imágenes con las que iremos componiendo el archivo de vídeo.

El funcionamiento de este sistema es el siguiente: para comenzar a hacer una grabación deberemos abrir la ventana que indica "*No grabando video*" y pulsar la tecla *V*. En ese momento se comenzará la grabación y se abrirá una ventana con el título "*Grabando video*" al abrir esta ventana y pulsar *B* se parará la grabación.

Todos los vídeos que se graben se guardarán en un directorio que en caso de no existir se creará en *home* en una carpeta llamada *U_Records* y dentro de esta en la carpeta *video*. El nombre del archivo se construirá de la siguiente manera: *outputVideo_Año-Mes-Día Hora.avi*.

Figura 8.2: Fotograma obtenido de un archivo de vídeo de una inmersión realizada con el dron

8.5. Visión por ordenador

8.5.1. Introducción

En este apartado se va a abordar todo lo relacionado con la detección y seguimiento de objetos. En primer lugar, se van a plantear las características deseables que debería tener el sistema. Para a continuación empezar a tratar las diferentes opciones que se han contemplado para resolver el sistema de detección y seguimiento. Además, se va a tratar la adaptación de los sistemas de detección y seguimiento escogidos al entorno de ROS.

8.5.2. Requerimientos

El objetivo del sistema de detección es tener un método para poder detectar diferentes objetos. Como el dron no tiene un propósito específico, sino que es de uso general tiene que ser versátil en este sentido.

En tareas de inspección puede ser de interés detectar objetos de distintas clases. Por poner un ejemplo, si usamos el dron para detectar fugas sería de interés detectar burbujas que puedan indicar dónde existe una fuga. Mientras, en tareas de investigación puede ser de más interés detectar un animal en concreto.

Es por esto que en los sistemas de detección que se implementen debe existir la posibilidad de poder cambiar los objetos a detectar.

Además, dado que la detección usando redes neuronales es una tarea pesada en cuanto a cómputo, debe plantearse una alternativa que permita hacer seguimiento si el equipo del que se dispone no es tan potente. Este método de seguimiento no será tanto para objetos, sino que nos permitirá identificar códigos o marcas que puedan ser de utilidad en las tareas que realiza el dron.

8.5.3. Tracking

8.5.3.1. Introducción

El *tracking* consiste en ser capaz de localizar uno o varios objetos en *frames* sucesivos de un vídeo, que puede tratarse de una grabación o un *streaming* en tiempo real de imágenes. Las técnicas de *tracking* puro se basan en conocer la posición de un objeto en base a la posición anterior del objeto en la imagen y a alguna característica simple. Por esto muchas veces se hace trabajar en conjunto al *tracking* con la detección de objetos.

En nuestro caso vamos a realizar un estudio de distintos métodos de *tracking* implementados en la librería OpenCV [39] sin ayudar a estos algoritmos de ninguna función de reconocimiento extra para ver qué tal se comportan y en qué medida podrían ser de utilidad.

El punto positivo del *tracking* es que es más rápido que hacer detección de objetos. Esto se debe a que cuando usamos *tracking* partimos de un objeto seleccionado en un *frame* de vídeo en la situación concreta en la que vamos a intentar localizarlo, es por esto que vamos a tener mucha información concreta del objeto. Podemos conocer la posición del objeto en un *frame*, y a partir de varios *frames* podremos tener su dirección y velocidad. Con toda esta información es más rápido localizar un objeto que si tenemos que hacer detección de objetos en la cual no tenemos información de partida, solo tendremos información del objeto buscado, pero en otras circunstancias. Aun así, los métodos de *tracking* como comentamos al inicio no solo hacen uso de la información de movimiento, sino que tratan de, basándose en los *frames* en que se tiene localizado el objeto seleccionado, tener una idea de cómo es el objeto.

El problema de esto es que un objeto puede cambiar mucho su apariencia en un corto periodo de tiempo con simples cambios de orientación. Por esta razón se usan los clasificadores "*online*" cuyo objetivo es ser capaces de separar un objeto del fondo, y se van entrenando para hacer esta distinción en tiempo real. A diferencia de los clasificadores "*offline*" que se entrenan previamente a la ejecución usando muchas imágenes.

8.5.3.2. Métodos

Para realizar esta función de *tracking* existen muchos algoritmos y cada uno tiene sus ventajas y desventajas. Lo que vamos a hacer es implementar un programa adaptado para que funcione con ROS, y que mediante un parámetro nos deje seleccionar el método de *tracking*. En concreto vamos a implementar un *multiTracker* que nos deje seleccionar varios objetos para hacerles el seguimiento.

8.5.3.2.1. BOOSTING

Este método de *tracking* [40] usa un recuadro de selección o *bounding box* inicial dado por el usuario como muestra del objeto y toma partes de la imagen fuera de este rectángulo como fondo. Cuando se tiene un nuevo *frame* el clasificador se ejecuta en los pixeles cercanos a donde anteriormente estaba el objeto. La nueva posición del objeto será aquella en que la puntuación que se obtiene del clasificador sea máxima.

8.5.3.2.2. MIL

Este método [41] funciona de manera similar al método BOOSTING. La diferencia reside en que este método toma en los alrededores del objeto muestras que son potencialmente positivas, no solo toma la actual posición del objeto.

La clave de esto está en que en el método MIL (*Multiple Instance Learning*) no se definen muestras positivas ni negativas, sino que se habla de "*bags*" positivas y negativas. El conjunto de imágenes en el *bag* positivo incluye tanto la información de las cercanías del objeto como la información de la posición actual del objeto.

Al incluir esta información de los alrededores existen más posibilidades de que el conjunto contenga al menos una imagen en la que el objeto esté centrado.

8.5.3.2.3. KCF

El método *Kernelized Correlation Filters* (KCF) [42] se apoya en las ideas de los dos métodos anteriores. Concretamente aprovecha que el uso de múltiples muestras positivas lleva a que haya grandes regiones superpuestas. Esto conlleva una serie de propiedades matemáticas que sirven a este método para realizar un *tracking* más rápido y preciso.

8.5.3.2.4. TLD

Tal y como el nombre del método indica, *Tracking, Learning and Detection* (TLD) [43], lo que se hace es descomponer la tarea de seguimiento en tres partes. Por un lado, tenemos el *tracking* que sigue el objeto *frame* a *frame*. Por otro lado, tenemos una parte de detección que se centra en reconocer el objeto basándose en los *frames* que va leyendo. Y también una parte de aprendizaje en la que se mide el error del detector y se trata de corregir.

8.5.3.2.5. MEDIANFLOW

Este método [44] se basa en hacer un seguimiento del objeto en dirección hacia delante y hacia atrás en el tiempo y se miden las discrepancias entre las dos trayectorias. El minimizar este error es lo que permite a este método detectar fallos de *tracking* de manera fiable y seleccionar trayectorias relevantes en vídeos.

8.5.3.2.6. MOSSE

El *Minimum Output Sum of Squared Error* (MOSSE) [45] usa correlación adaptativa para hacer el seguimiento de objetos. Este de los que hemos visto, es el que en teoría mejor debería funcionar, además debería ser bastante rápido.

8.5.3.3. ROS

Para integrar el sistema de *tracking* con ROS vamos a tener que hacer dos tareas. Por un lado, nos tenemos que suscribir al tópico de la imagen como ya hicimos para la grabación. Por otro lado, tenemos que poner un *publisher* que nos permita una vez ejecutado el algoritmo de *tracking* publicar la imagen resultante, ya sea especificando que hay un error o remarcando un *bounding box* alrededor del objeto de interés. El código se puede consultar en el anexo 4.

Esto en lo que se refiere a la estructura de mensajes que vamos a usar para ROS. Pero aparte de esto tenemos que proporcionar alguna manera al usuario de interactuar con el *tracker*, para remarcar el objeto de interés, para poder parar el seguimiento o para marcar nuevos objetos. Para llevar a cabo esta tarea vamos a hacer uso de la función *waitKey* de OpenCV que nos permite, cuando abrimos una imagen, interactuar con el teclado.

La implementación es la siguiente: en primer lugar, se está mostrando una imagen con el logo de OpenCV en una ventana pequeña que tiene como título "Pulsa t para empezar a *trackear*" (Figura 8.3). Cuando, con esta ventana abierta, pulsamos 't' se muestra una imagen capturada por la cámara (Figura 8.4) que nos permite seleccionar varios objetos de interés usando las teclas que se especifican en el título. Una vez hecho esto comenzará la tarea de seguimiento y tendremos el resultado del mismo en la interfaz cuando en el *image view* seleccionemos el tópico de la lista desplegable: */tracker/image_raw/compressed*. Cuando se esté realizando un seguimiento hay otra pequeña ventana que cuando pulsamos *Esc* hace que se detenga el seguimiento de los objetos seleccionado. Podríamos volver a activar el seguimiento de la misma manera que cuando comenzamos.

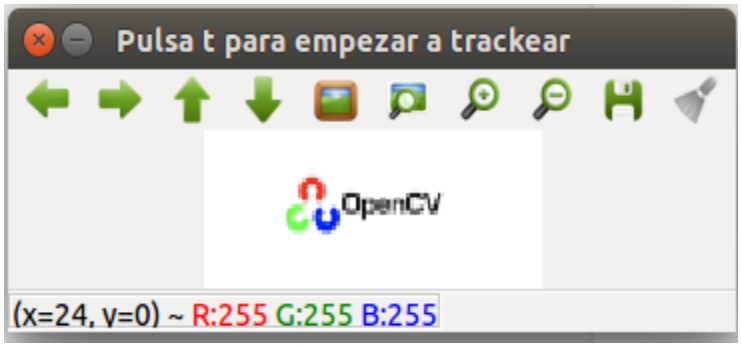


Figura 8.3: Pestaña para comenzar el proceso de seguimiento

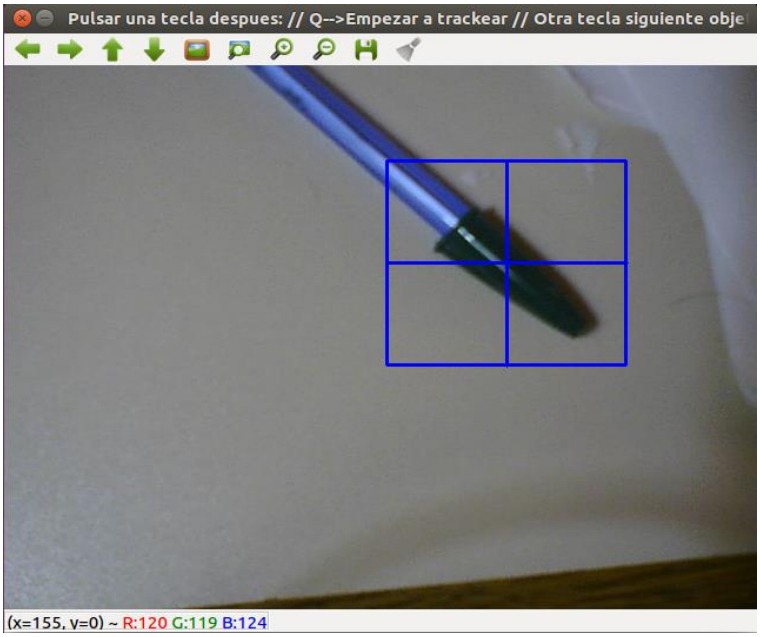


Figura 8.4: Captura tomada para seleccionar el objeto a seguir

8.5.3.4. Pruebas

Una vez tenemos el código funcionando dentro de ROS, podemos seleccionar el *tracker* que vamos a utilizar haciendo uso de un parámetro a la hora de lanzar el programa. Así que en esta fase se va a realizar una prueba de seguimiento con condiciones muy similares utilizando cada uno de los métodos de *tracking* que se plantean. En todas las pruebas se va a seleccionar como objeto a seguir un *pen drive* negro y rojo.

8.5.3.4.1. BOOSTING

En primer lugar, hicimos la prueba con este algoritmo que es el más sencillo (Figura 8.5). Inicialmente se observó que, aunque los movimientos del objeto fueran lentos se producían errores, además de que una vez el objeto se perdía no detectaba de forma correcta que el seguimiento había fallado. Por otro lado, cuando el objeto se tapaba completamente el seguimiento se perdía moviéndose de forma aleatoria, como se ve en la Figura 8.6. Tras tapar un momento el objeto, el *bounding box* se ha desplazado completamente.

Figura 8.5: Tracking con el algoritmo BOOSTING

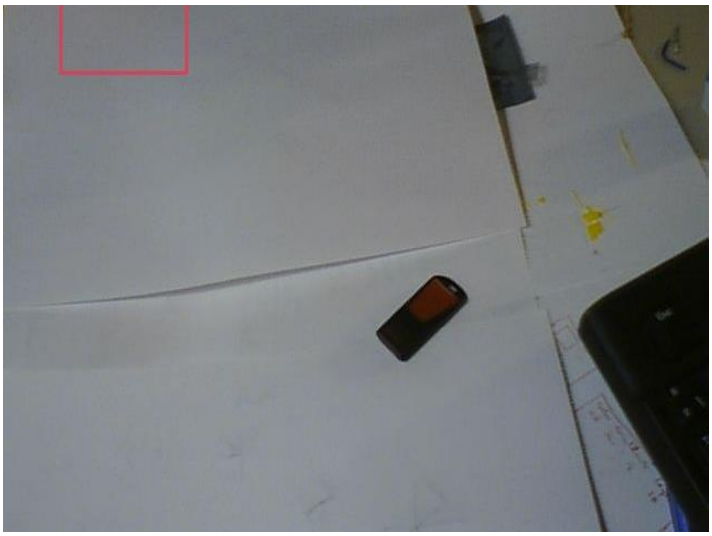


Figura 8.6: Tracking con el algoritmo BOOSTING tras ocultar el objeto de interés

8.5.3.4.2. MIL

En esta segunda prueba se pueden observar importantes mejoras, sobre todo en el seguimiento se experimenta que, aunque haya movimientos a velocidad normal el *tracker* se comporta bastante bien. Sin embargo, no es perfecto y se ve una carencia importante cuando pierde el objeto pues no es capaz de detectar correctamente este hecho. Como vemos en la Figura 8.7 tras ocultar completamente el objeto pierde su localización y no es capaz de detectar que ya no sabe la posición del mismo.

Figura 8.7: Resultado del algoritmo MIL tras tapar el objeto de interés

8.5.3.4.3. KCF

Con el uso de este método podemos apreciar una gran mejoría con respecto a los anteriores. Cuando el objeto se está moviendo lo sigue sin demasiadas dificultades, pero el gran salto con respecto a los demás viene a la hora de detectar que el objeto se ha perdido. Como se puede ver en las imágenes de abajo al tapar el objeto detecta que ya este no está en escena (Figura 8.8), y al destaparlo vuelve a detectarlo y continúa con la tarea de seguimiento (Figura 8.9).

Figura 8.8: Respuesta del algoritmo KCF tras tapar el objeto de interés

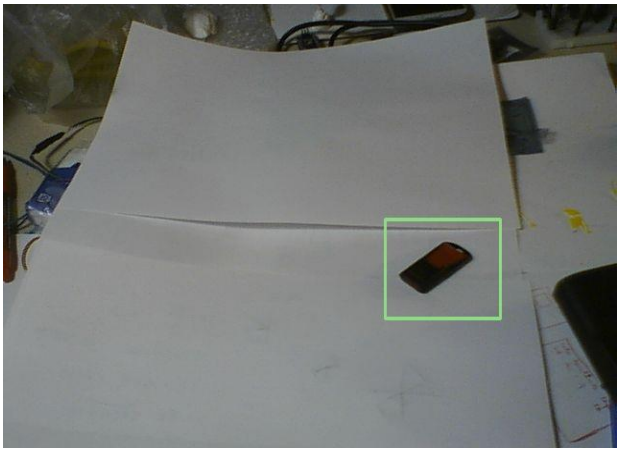


Figura 8.9: Respuesta del algoritmo KCF cuando el objeto de interés regresa a escena

8.5.3.4.4. TLD

En el uso de este método se aprecia un fenómeno llamativo, pues se comporta bien cuando acercamos o alejamos la cámara tal y como se ve en la Figura 8.10. Sin embargo, por el funcionamiento del método cuando tapamos completamente el objeto busca en la imagen algo similar al objeto seleccionado y pasa a seguirlo como se ve en la Figura 8.11. En la Figura 8.12 ponemos dos objetos similares y pasa a seguir el que no debería, cuando desaparece el objeto que era parecido vuelve a seguir el pen seleccionado inicialmente tal y como se ve en la Figura 8.13.

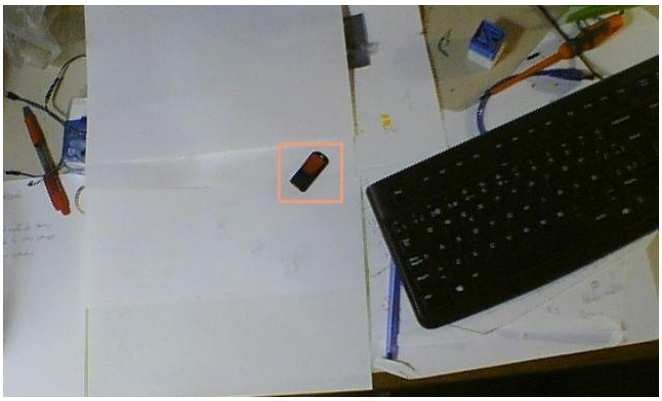


Figura 8.10: Seguimiento con el algoritmo TLD al alejar la cámara

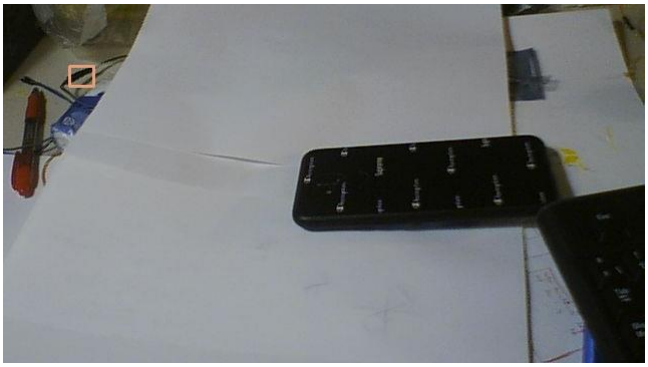


Figura 8.11: Algoritmo TLD al tapar el objeto de interés

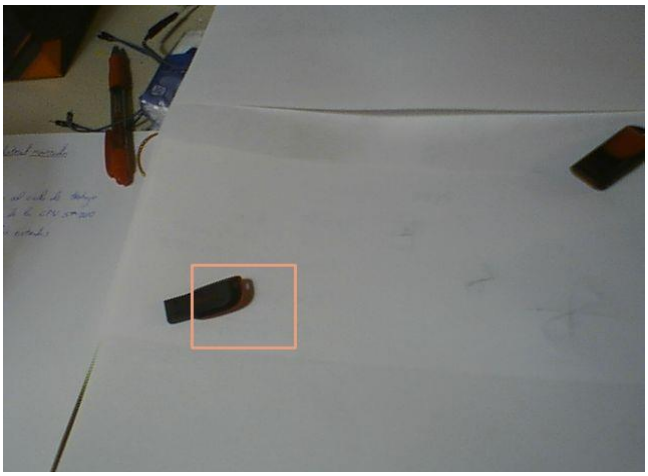


Figura 8.12: Algoritmo TLD cuando tenemos dos objetos similares en escena

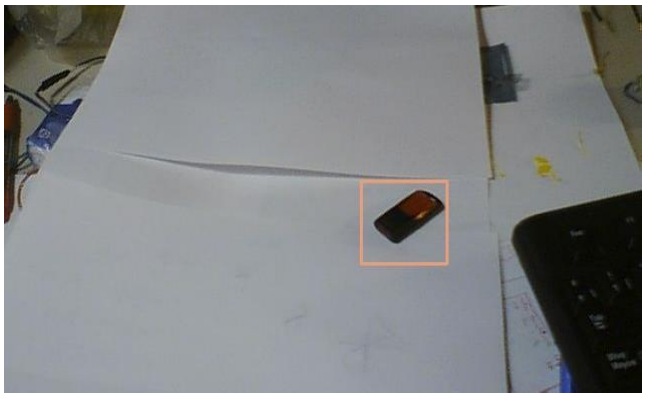


Figura 8.13: Algoritmo TLD cuando retiramos uno de los dos objetos similares

8.5.3.4.5. MEDIANFLOW

Con este método también sucede un fenómeno llamativo. El seguimiento del objeto es correcto (Figura 8.14), pero cuando lo tapamos por completo no es capaz de detectar que el objeto ya no está en escena y lo que hace es comenzar a hacer el *bounding box* mayor tal y como se ve en la Figura 8.15.

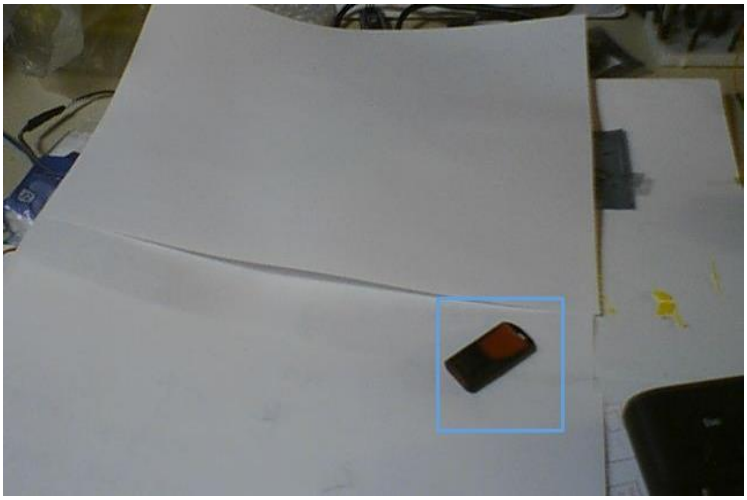


Figura 8.14: Algoritmo MEDIANFLOW realizando seguimiento

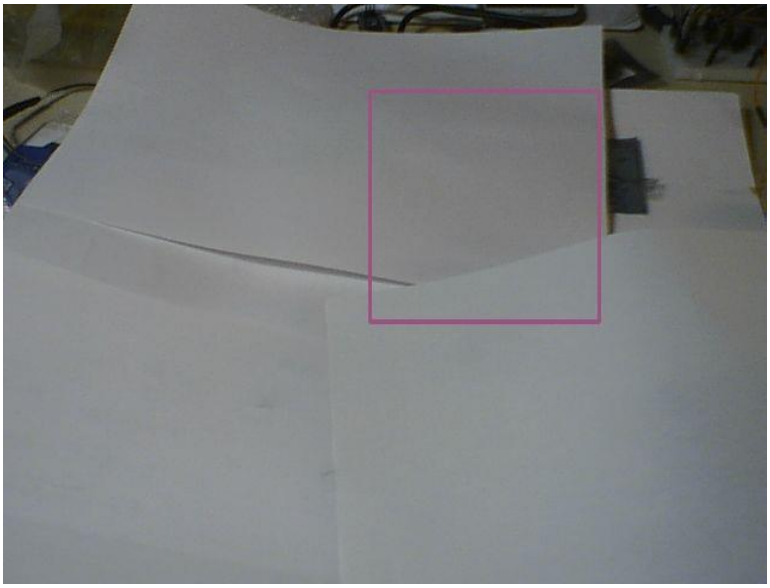


Figura 8.15: Algoritmo MEDIANFLOW al tapar el objeto de interés

8.5.3.4.6. MOSSE

Este método funciona de forma correcta en las pruebas de seguimiento normales. Además, maneja muy bien el tema de la oclusión (Figura 8.16) siendo capaz de volver a situar el objeto una vez este desaparece completamente de escena, tal y como se puede observar en la Figura 8.17.

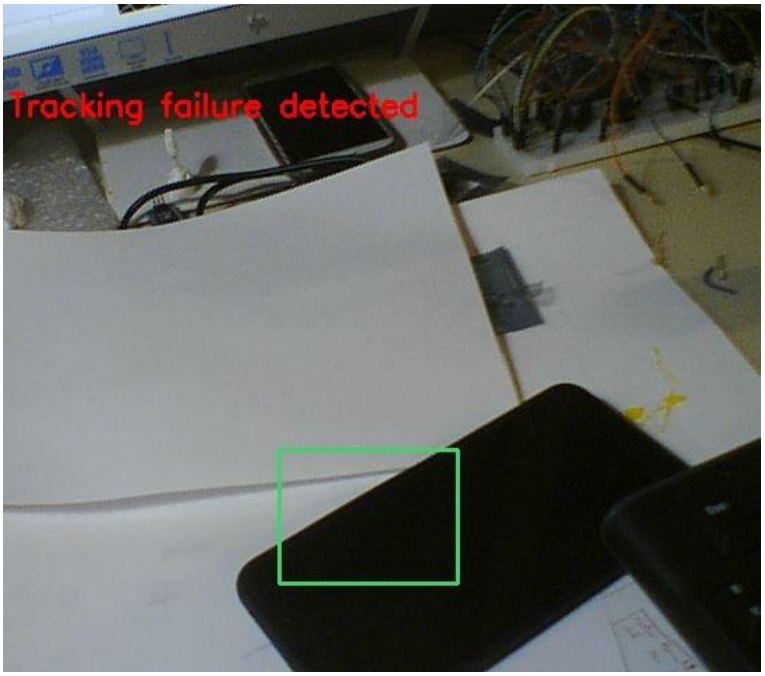


Figura 8.16: Algoritmo MOSSE al desaparecer el objeto de interés de la escena

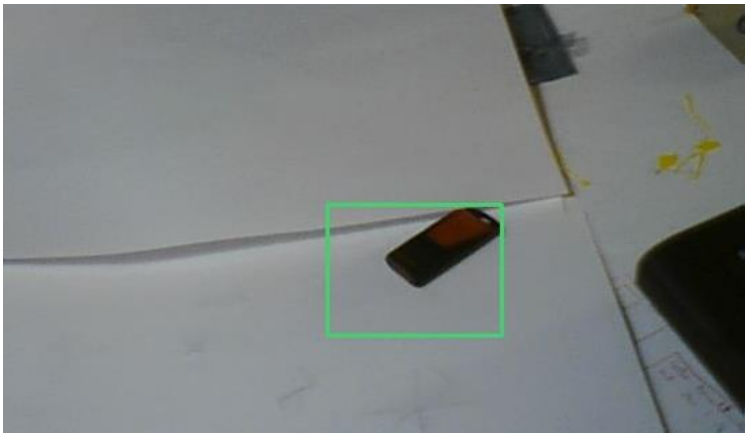


Figura 8.17: Algoritmo MOSSE tras destapar el objeto de interés

8.5.3.5. Resultados tracking

De todas las opciones comparadas las dos que mejor se han comportado son el método KCF y el MOSSE. El punto que hace la diferencia con respecto a los demás es el manejo de la oclusión. Mientras que los demás pierden por completo el objeto o no detectan que el objeto ya no está en escena, estos dos métodos manejan muy bien estas situaciones.

Dentro de estas dos opciones el método MOSSE se plantea como mejor solución en este apartado ya que, en las pruebas realizadas, el seguimiento del objeto cuando este se mueve rápido es notoriamente mejor. Además, maneja mejor los cambios de sentido del objeto, esto lo hace la opción más robusta para tareas de seguimiento de las opciones analizadas.

8.5.4. Tracking basado en características

8.5.4.1. Introducción

En este apartado se va a tratar la realización de un sistema de *tracking* más elaborado que los anteriores que nos va a permitir detectar objetos de interés o códigos. En tareas de mantenimiento puede ser de mucha utilidad para identificar una parte del casco de un barco, ya que con que este tuviera una marca, algo similar a un código de barras, lo podríamos detectar.

Lo interesante de este sistema es que podemos seleccionar el objeto de interés en tiempo de ejecución cosa que la detección más potente con redes neuronales que vamos a tratar después no nos va a permitir.

Para utilizar este tipo de sistemas nos vamos a servir de un descriptor local del que hablaremos a continuación y de un "*matcher*" [46] que nos permita relacionar los puntos de interés de la imagen inicial con la actual.

Un punto fundamental de este sistema es que se pueda ejecutar en un ordenador sin especificaciones especialmente pensadas para la detección de imágenes, ya que nos va a permitir poder tener ayuda de seguimiento de objetos si lo necesitamos cuando no tengamos disponibilidad del sistema de detección con *TensorFlow* o cuando simplemente se vayan a hacer tareas más simples donde, por ejemplo, tengamos que detectar un boya a la que nos hemos de dirigir y este sistema nos ayude a seguirla para no perder la trayectoria. Para la realización de esta tarea usaremos la librería OpenCV.

8.5.4.2. Descriptor local

Para el descriptor local existen diversas alternativas. Entre ellas las más conocidas son los algoritmos SURF, SIFT y ORB. En este caso vamos a utilizar el algoritmo ORB [47] que según la documentación del mismo es mucho más rápido que el SURF y SIFT.

8.5.4.2.1. ORB

El *Oriented Fast and Rotated Brief* [47] es una mezcla entre el detector de puntos de interés FAST (*Features from Accelerated Segment Test*) [48] y el descriptor BRIEF (*Binary Robust Independent Elementary Features*) [49], que es el que nos proporciona características elementales de la imagen. Pero tiene implementadas varias modificaciones buscando mejorar el rendimiento.

El funcionamiento se basa en detectar puntos claves con FAST y aplicar una serie de operaciones. El problema es que el algoritmo FAST no tiene en cuenta la orientación. Así que ORB lo que hace es, usando el centroide y una esquina situada en el centro, obtener un vector con la dirección.

Por otro lado, el descriptor BRIEF que normalmente no se comporta bien con los cambios de orientación también se le incorporan mejoras para mitigar esta carencia, en este caso se aprovecha la orientación de los puntos de interés.

8.5.4.2.1.1. Pruebas

Para poder observar la salida del algoritmo ORB (Figura 8.18) se va a utilizar un código que se encuentra en la *wiki* de ROS [50]. Con este código nos vamos a suscribir a la imagen de la cámara, vamos a ejecutar el algoritmo y podemos observar la salida de este en la imagen publicada */feature/image_raw/compressed*.

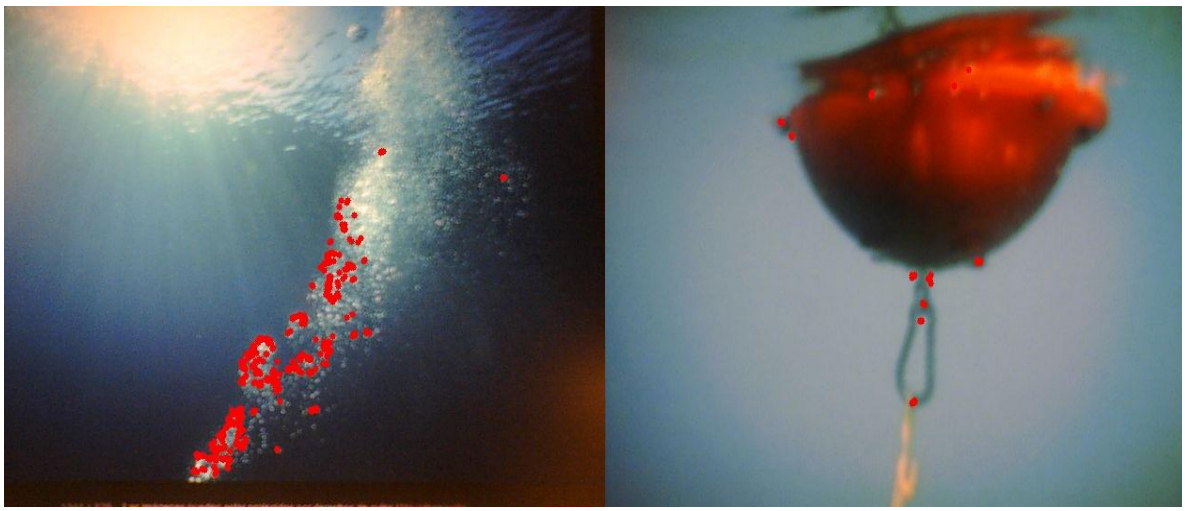


Figura 8.18: Salida del algoritmo ORB

8.5.4.3. Feature matcher

8.5.4.3.1. FLANN

El objetivo del *matcher* FLANN [51] (*Fast Library for Approximate Nearest Neighbors*) es el de encontrar la relación entre una imagen y otra teniendo un vector de características (Figura 8.19) que se obtiene usando el método ORB.

Figura 8.19: Funcionamiento del matcher FLANN

8.5.4.4. Tracker

El *tracker* funciona haciendo uso del descriptor local ORB y del *feature matcher* FLANN, con estas dos herramientas vamos a ser capaces de marcar un objeto y realizar un seguimiento del mismo. Para realizar esta tarea vamos a hacer uso como punto de partida de un código [52] de ejemplo de OpenCV.

A partir de este código que implementa el seguimiento haciendo uso de estas dos herramientas vamos a realizar la adaptación a ROS del mismo para que quede integrado en el sistema del submarino.

8.5.4.4.1. ROS

El código adaptado para su uso en ROS se puede encontrar en el anexo 5. Para poder utilizar este código, la principal diferencia al implementar ROS tal y como tenemos el sistema es que como el procesado de imagen se hace en el PC, pero la cámara está conectada a la Raspberry, tenemos que suscribirnos a la imagen de la cámara y convertirla a formato OpenCV para posteriormente trabajar con ella. Una vez hacemos el procesado correspondiente, tenemos que publicar el resultado para que pueda verse desde la interfaz en el PC.

El hecho de que en el programa no nos conectemos directamente a un dispositivo es lo que más altera el programa, las modificaciones se ven en la clase *App* donde tenemos que trabajar con los *frames* que van llegando mientras esté funcionando un *roscore*.

8.5.4.4.2. Pruebas

Para las pruebas vamos a utilizar el programa y en tiempo de ejecución vamos a seleccionar un objeto que sea de interés. Una vez hecho esto, vamos a tapar y rotar la imagen para ver cómo responde el sistema. Los resultados de estas pruebas se pueden ver en la Figura 8.20 y la Figura 8.21 respectivamente.

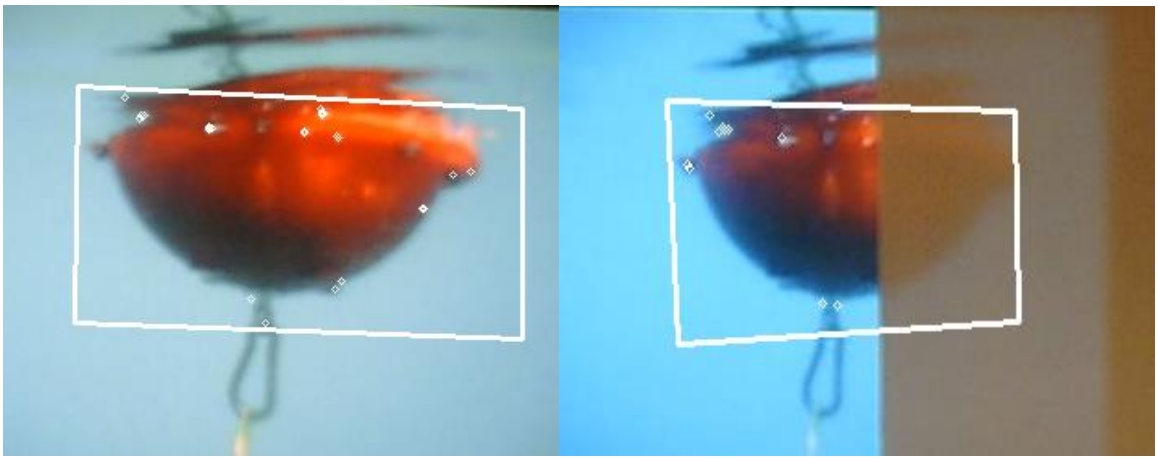


Figura 8.20: Selección de objeto de interés a la izquierda y a la derecha objeto tapado parcialmente

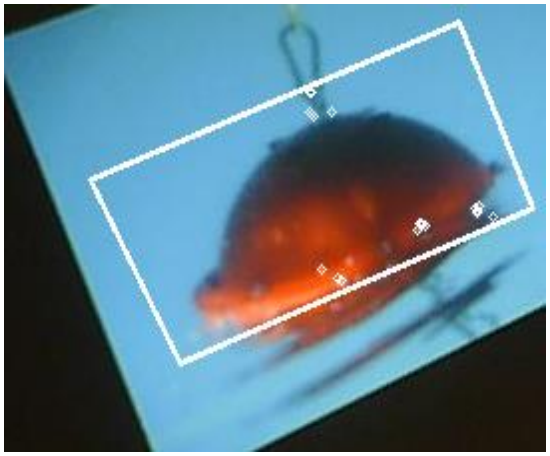


Figura 8.21: Objeto cambiado de orientación

8.5.5. Detección usando inteligencia artificial

8.5.5.1. Introducción

En este apartado vamos a tratar la implementación de un sistema de detección de imágenes avanzado haciendo uso de redes neuronales para la detección de los objetos de interés. Empezaremos con una explicación de las redes neuronales aplicadas a visión por ordenador y después haremos una comparativa entre dos opciones que son ampliamente usadas para resolver el problema de detección de objetos, YOLO [53] y *TensorFlow* [5].

Lo importante de este sistema tal y como se ha ido comentando a lo largo del documento es que tiene que poder ser adaptado para detectar objetos diferentes a los que encontramos en *datasets* estándar. Existen muchos *datasets* para entrenar redes neuronales que nos permiten detectar coches, animales, móviles, balones y muchos más objetos. Sin embargo, dada la aplicación del dron hay que tener una manera de entrenar la red utilizando un *dataset* personalizado, ya que no hay demasiados *datasets* orientados a la detección de objetos bajo el agua, principalmente podemos implementar detección de diferentes tipos de peces.

Para nuestro caso que el dron puede tener aplicación para inspeccionar puertos en busca de fugas en tuberías, vamos a preparar la red para reconocer burbujas que son un indicador de donde se está produciendo una fuga, también identificará las propias tuberías y los buzos que pudieran estar realizando trabajos en las tuberías, de esta forma el controlador del dron va a tener un apoyo para que le sea más fácil realizar la tarea de inspección pues cualquiera de estos objetos se marcará con un *bounding box* advirtiendo su presencia en el campo de visión.

8.5.5.2. Explicación

Existen distintos tipos de redes neuronales, la más sencilla es la red neuronal monocapa o perceptrón simple, cuya estructura se puede observar en la Figura 8.22. En este tipo de redes tenemos una única capa de neuronas que reciben las entradas y las pasan a una capa de neuronas de salida donde se realizan los cálculos. Esto pasa después por una función de activación como puede ser una tangente hiperbólica, sigmoide o una ReLU [54]. Cada función de activación

tiene unas propiedades concretas que la hacen más apropiadas para ciertos tipos de redes. Esta función de activación es la que proporciona la salida.

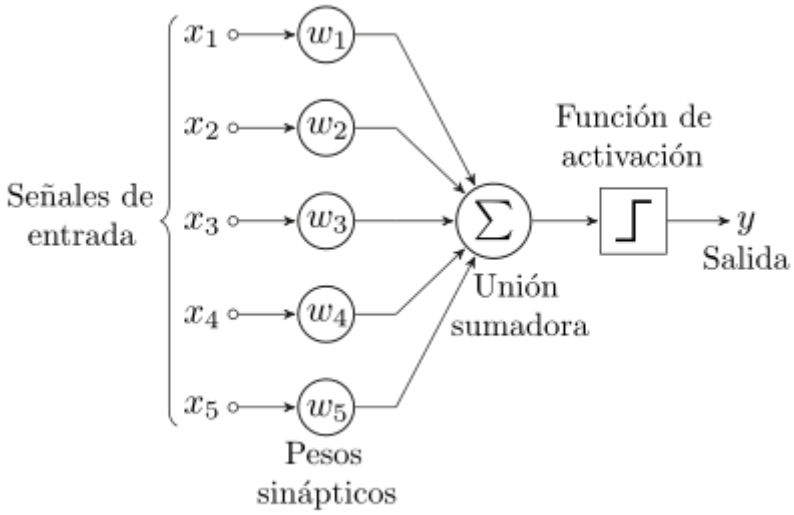


Figura 8.22: Estructura de perceptrón simple

Como una opción más avanzada que el perceptrón simple tenemos el perceptrón multicapa (Figura 8.23). La diferencia con la primera reside en que no sólo tenemos una capa de entrada y salida, sino que hay capas intermedias conocidas como capas ocultas.

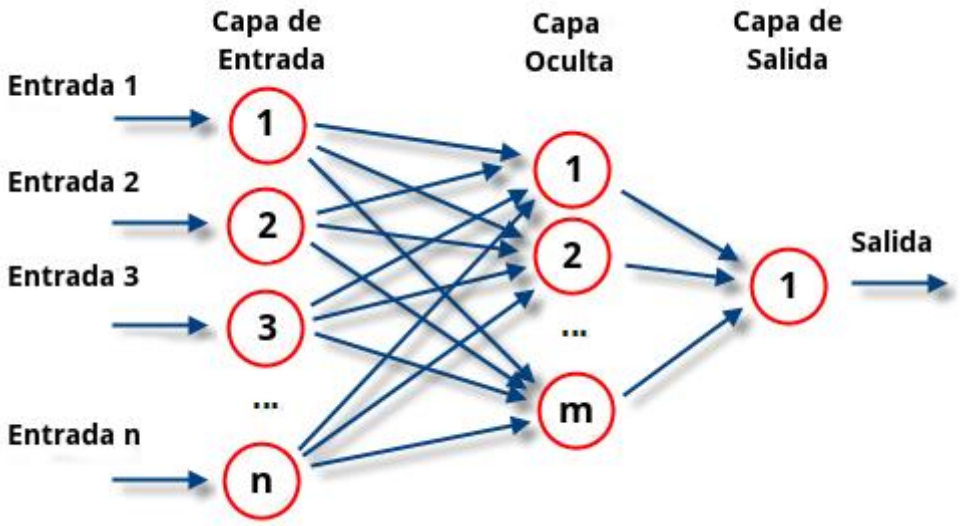


Figura 8.23: Perceptrón multicapa

También existen otros tipos de redes con estructuras más complejas como la red neuronal recurrente que no tiene estructura de capas o la red de base radial donde la salida se calcula en función de la distancia a un punto denominado centro.

Y por otro lado tenemos las redes que son más interesantes para la aplicación de visión que son las redes neuronales convolucionales (Figura 8.24). La diferencia con las redes más básicas como el perceptrón multicapa es que todas las neuronas no tienen por qué unirse con todas las

capas siguientes, con lo que se consigue que los subgrupos de neuronas se especialicen, de esta forma reduciéndose el número de neuronas y la carga de cómputo.

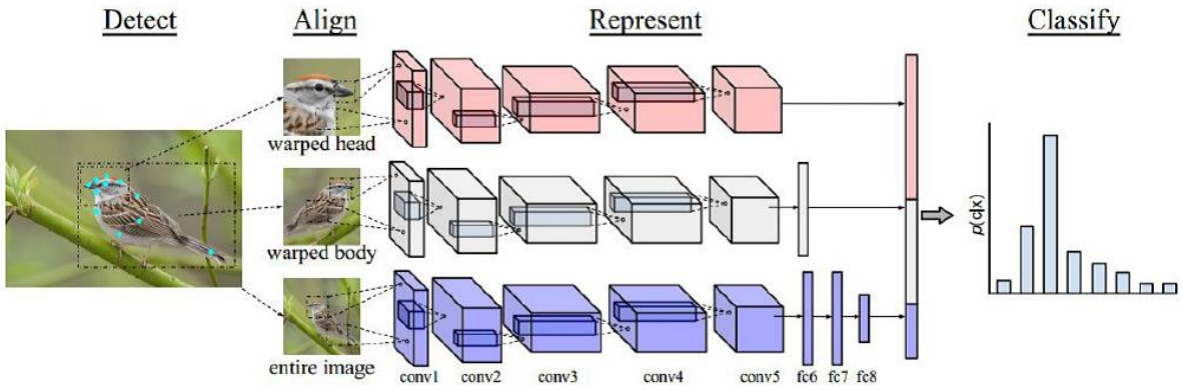


Figura 8.24: Estructura de red neuronal convolucional

8.5.5.3. Redes neuronales convolucionales

8.5.5.3.1. Introducción

Como ya comentamos anteriormente, en este tipo de redes se entrena cada parte para realizar una tarea concreta. Además, este tipo de redes supone que la entrada de la misma ha de ser una matriz (una imagen en nuestro caso). Estas permiten resolver problemas de las redes neuronales más simples como que las imágenes no se escalan bien.

La clave del uso de este tipo de redes en análisis de imágenes es que cada parte es capaz de detectar características simples de la imagen. Con esto nos referimos por ejemplo a detección de bordes, realizando la detección de bastantes de estas características simples la red puede componer características más complejas hasta que es capaz de detectar el objeto de interés.

8.5.5.3.2. Arquitectura

Estas redes en realidad son redes multicapas, pero más complejas en el sentido de que cuentan con capas convolucionales y al final tienen una capa que hace de clasificador con capas totalmente conectadas como lo que encontramos en un perceptrón multicapa.

De forma más detallada nos encontramos con una primera capa convolucional que es la que da nombre a la red, después tenemos una capa de reducción o *pooling* que reduce la cantidad de parámetros buscando quedarse con las características comunes, como pueden ser el máximo de activación de una zona (*max pooling*) o el promedio de activación (*mean pooling*), por ejemplo, y por último nos encontramos con una capa clasificadora totalmente conectada que es la que nos proporciona el resultado final.

- **Capa convolucional**: Este nombre proviene de que estas capas hacen uso de la operación convolución en algunas de sus capas en vez de realizar la multiplicación de matrices de pesos por entradas que se usa habitualmente. La convolución recibe una

imagen como entrada y aplica un filtro que nos va a proporcionar una imagen donde se pueden apreciar unas características concretas de la imagen.

- **Capa de reducción**: Esta se suele situar después de la capa convolucional, y se utiliza para reducir las dimensiones espaciales (ancho × alto). Con esto conseguimos aligerar las operaciones que tiene que realizar la red y además ayuda a evitar el *overfitting*, es decir, que la red trabaje muy bien con las imágenes de entrenamiento pero que no sea capaz de generalizar.
- **Capa clasificadora**: Tras aplicar las capas convolucionales y de reducción nos encontramos con esta capa en la que cada *pixel* se toma como una neurona. Esta capa tendrá tantas neuronas como número de clases se pretenda reconocer.

8.5.5.4. Alternativas

Para implementar la detección usando redes neuronales, tal y como comentamos al inicio del apartado, vamos a probar YOLO y *TensorFlow*. Estas dos alternativas nos permiten implementar una detección en tiempo real.

8.5.5.4.1. YOLO

A diferencia de otros sistemas de detección, YOLO propone aplicar una única red neuronal a toda la imagen. Esta red se encarga de dividir la imagen en regiones y predecir *bounding boxes* y probabilidades para cada región. Estas *bounding boxes* son pesadas por las probabilidades predichas.

Este sistema de detección presenta algunas ventajas frente a otros sistemas como R-CNN. Con YOLO se mira toda la imagen en el momento de ejecución por lo que las predicciones que realizan poseen información del contexto de la imagen. Además, al usar una red es mucho más rápida que alternativas como R-CNN [55], en concreto es como 1000 veces más rápida que esta última.

8.5.5.4.2. TensorFlow

Aunque YOLO tiene diferentes redes como puede ser la versión *tiny* o la v3, es un sistema centrado en resolver problemas de visión. *TensorFlow* sin embargo, es una biblioteca con la que se pueden construir redes para resolver problemas no solo de visión. De hecho, dentro de *TensorFlow* existen muchos modelos para resolver el problema de visión que se pueden descargar. Esto es una gran ventaja respecto a YOLO pues tenemos más capacidad de elección.

Los modelos disponibles para la descarga se pueden encontrar en el repositorio de *GitHub* de *TensorFlow* [56] y por otro lado están los archivos de configuración que los encontramos en el mismo repositorio pero en el directorio *configs* [57]. Nosotros vamos a hacer uso del modelo *ssd_mobilenet_v1_coco* que permite detección en tiempo real.

8.5.5.4.3. Pruebas

Para las pruebas simplemente vamos a utilizar la red con la que viene entrenada cada opción, la vamos a ejecutar y observar cómo se comporta cada una, teniendo en cuenta que lo prioritario es la fluidez de la imagen. Vamos a comenzar con YOLO. La Figura 8.25 muestra el sistema de detección de YOLO funcionando. Como vemos es capaz de detectar que el buzo es una persona. La detección es bastante precisa, pero da sensación de que la imagen no va lo suficientemente fluida. Esto se nota sobre todo haciendo la comparativa directa con *TensorFlow*, donde haciendo uso de un *Single Shot Detector* (SSD) [58] que usa *mobilenet* para extraer características y entrenado con el *dataset* de *COCO* [59] se puede apreciar una diferencia sustancial en lo que a fluidez se refiere.

Figura 8.25: Detección de objetos con YOLO

Ahora vamos a realizar la prueba con el modelo *ssd_mobilenet_v1_coco* en *TensorFlow*. En la Figura 8.26 se puede apreciar cómo la detección funciona correctamente, pero tal y como comentábamos antes la experiencia es más fluida que con YOLO. Por esto, porque tiene más modelos que se pueden utilizar, y por opciones como el uso de *google Collab* que está perfectamente adaptado para poder hacer entrenamiento de redes de *TensorFlow*, este será el sistema elegido a utilizar en la detección de imágenes.

Figura 8.26: Detección de objetos usando TensorFlow con mobilenet

8.5.5.5. Personalizar detección

8.5.5.5.1. Introducción

En este apartado vamos a tratar de preparar al sistema de detección para detectar los objetos que nosotros queramos, sin utilizar un *dataset* que ya esté construido. Tal y como se comentó antes, nuestro objetivo va a ser realizar una detección capaz de identificar buzos, tuberías y burbujas. Para esto vamos a partir del repositorio de *GitHub* [60] donde tenemos las herramientas para realizar el entrenamiento de la red.

8.5.5.5.2. Obtención de imágenes

Una vez hemos seleccionado las clases que va a tener nuestro detector tenemos que obtener muchas imágenes donde aparezcan los objetos de interés que nos sirvan para realizar el entrenamiento. El formato de las imágenes tiene que ser *.jpg* o *.jpeg*, todas deben ser en formato de color RGB, y no pueden superar ni en ancho ni en alto los 1000 px. Estos datos se han obtenido de forma experimental realizando entrenamientos de redes con el modelo *ssd_mobilenet_v1_coco*.

Teniendo esta información descargamos usando la herramienta *google_images_download* las imágenes que consideremos de interés. Esta herramienta nos deja seleccionar algunos parámetros como el número de imágenes a descargar, el formato, el espacio de color, el tamaño o si se realiza la búsqueda de imágenes con el *safe_search* activado.

Para nuestro caso usamos las siguientes configuraciones:

- *googleimagesdownload --keywords "Lo que vamos a descargar" --limit 550 --format jpg --color_type full-color --output_directory ~/Descargas/Tensorflow_explained/imagenes_des --safe_search --size icon --chromedriver /usr/bin/chromedriver*
- *googleimagesdownload --keywords "Lo que vamos a descargar, Lo otro que vamos a descargar" --limit 550 --format jpg --color_type full-color --output_directory ~/Descargas/Tensorflow_explained/imagenes_des --safe_search --size medium --chromedriver /usr/bin/chromedriver*

Descargando imágenes de burbujas submarinas, *underwater pipe* (tubería), *diver* (buzo) y otras opciones similares. De cada opción descargamos 550 imágenes en formato pequeño y en formato mediano. Descargamos tantas imágenes porque muchas de estas no son adecuadas para el entrenamiento, una vez descargamos todo tenemos como 10000 imágenes. Lo siguiente que hacemos es eliminar poco a poco las imágenes que no son adecuadas para el etiquetado. Una vez hecho esto nos quedamos con unas 1200 imágenes.

Lo siguiente que hacemos es separar todas las imágenes para el conjunto de entrenamiento y test, usamos un 80% para entrenamiento y un 20% para test. Esta separación para hacerla de forma aleatoria y rápida usamos un *script* en Python llamado *split.py*, que se puede consultar

en el anexo 6. A este script le pasamos el directorio donde se encuentran las imágenes. La estructura de este directorio debe ser la siguiente:

- burbujas
    - *.jpg
    - *.jpg
- buzo
    - *.jpg
    - *.jpg

Al terminar este proceso tenemos una carpeta llamada *train*, con las imágenes para entrenamiento, y otra *val*, con las imágenes para test. A continuación, metemos dentro de la carpeta *train* el script *renombrar_train.py*, y lo ejecutamos. Con esto vamos a hacer que las imágenes no tengan el nombre aleatorio con el que se descargan, sino que se llamen con números crecientes de uno en uno. Ahora apuntamos el número de la última imagen de la carpeta *train* y vamos a la carpeta val, donde tenemos que poner el archivo *renombrar_val.py* en el que tenemos que poner como número inicial el de la carpeta *train* más uno. Estos códigos para renombrar se pueden consultar en los anexos 7 y 8, respectivamente.

Ya podemos pasar al proceso de etiquetado.

8.5.5.5.3. Etiquetado

Para realizar el etiquetado de las imágenes vamos a utilizar la aplicación llamada *labelImg* [27]. Con esta podemos abrir el directorio donde están las imágenes y empezar a etiquetar una por una marcando los objetos de las clases de interés (Figura 8.27).

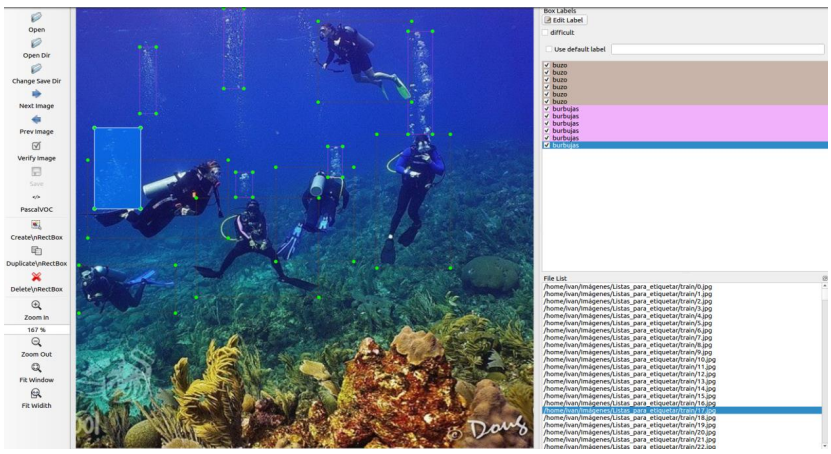


Figura 8.27: Captura de la aplicación LabelImg durante el proceso de etiquetado

Se tienen que etiquetar tanto las imágenes del conjunto *train* como del conjunto *val*. Una vez hemos etiquetado todas las imágenes pasamos al proceso de entrenamiento.

8.5.5.5.4. Entrenamiento

Para realizar el entrenamiento de la red neuronal tenemos que pasar las etiquetas de las imágenes a formato *TFRecords* que es el que usa *TensorFlow*. Para ello tenemos que dirigirnos al directorio *~/Descargas/Tensorflow/deteccion_objetos-master/object_detection* y hacer *export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim* después vamos a *~/Descargas/Tensorflow/deteccion_objetos-master/* y creamos dos carpetas una llamada CSV y otra llamada TFRecords. Cuando tenemos esto ejecutamos los siguientes comandos:

- *python xml_a_csv.py --inputs=img_test --output=test*
- *python xml_a_csv.py --inputs=img_entrenamiento --output=entrenamiento*
- *python csv_a_tf.py --csv_input=CSV/test.csv --output_path=TFRecords/test.record --images=images*
- *python csv_a_tf.py --csv_input=CSV/entrenamiento.csv --output_path=TFRecords/entrenamiento.record --images=images*

Con esto deberíamos tener listo los archivos TFRecords que vamos a utilizar para el entrenamiento en google Colab.

Antes de comenzar con el entrenamiento tenemos que preparar los archivos de etiquetas, por un lado en *Tensorflow/deteccion_objetos-master/configuracion* creamos el archivo *label_map.pbtxt*, que debe contener lo siguiente:

```
item {
  id: 1
  name: 'tuberia'
}
item {
  id: 2
  name: ‘burbujas’
}

item {
  id: 3
  name: 'buzo'
}
```

Y en el mismo directorio creamos otro archivo que será labels.txt, cuyo contenido será:

```
null
tuberia
burbujas
buzo
```

Es muy importante que los nombres que se ponen aquí sean exactamente los mismos que hemos usado en el proceso de etiquetado. Antes de entrenar solo nos falta preparar los archivos de configuración para el entrenamiento.

8.5.5.5.5. Configuración

Para realizar esta tarea de configuración necesitamos descargar el archivo de configuración de la red correspondiente este lo obtenemos del repositorio de *github* de *TensorFlow* en concreto del apartado *configs* [57].

En este archivo hemos de cambiar distintos parámetros, vamos a ver de forma detallada la configuración utilizada para realizar el entrenamiento.

En primer lugar, cambiamos el número de clases a tres. También modificamos el *batch_size* a 32 que nos permite entrenar la red sin problemas de que se llene la memoria en el *google Colab*.

Por otro lado, tenemos que indicar la ruta donde se pueden localizar los *checkpoints* para en caso de acabar el entrenamiento si obtenemos más imágenes para entrenar la red podamos continuar desde donde lo dejamos y no tengamos que empezar desde el principio. Por último, tenemos que indicar los directorios donde están los archivos *TFRecords* tanto de entrenamiento como de test e indicar la ruta del archivo donde se encuentran las etiquetas *label_map.pbtxt*.

8.5.5.5.6. Entrenamiento

Llegamos al momento donde vamos a comenzar a entrenar la red neuronal para ello vamos a ir a *google Colab* que nos va a permitir hacer el entrenamiento de forma más fluida y lo vamos a vincular con nuestra carpeta de *google drive* donde tenemos la carpeta para entrenar la red. Para el proceso de entrenamiento se ha hecho uso de 872 imágenes etiquetadas y para la parte de pruebas se han usado 221 imágenes. Para comenzar el entrenamiento tenemos que realizar una serie de acciones antes, estas se pueden ver en el cuaderno que se incluye en el anexo 9.

Una vez hemos ejecutado todas las celdas anteriores a la del entrenamiento vamos a dar comienzo a este usando el siguiente comando: *!python3 train.py --logtostderr --train_dir=train_5 --pipeline_config_path=modelo/ssd_mobilenet_v1_coco.config*

En el directorio train_5 se irán guardando los resultados del entrenamiento. Cuando han transcurridos 13579 pasos paramos el entrenamiento. A continuación tenemos que congelar el modelo esto lo hacemos con el siguiente comando: *!python object_detection/export_inference_graph.py --input_type image_tensor --pipeline_config_path modelo/ssd_mobilenet_v1_coco.config --trained_checkpoint_prefix train_5/model.ckpt-13579 --output_directory modelo_congelado_5*.

Con esto ya hemos completado el entrenamiento ahora solo hay que introducir este modelo en la carpeta de TensorFlow de ROS.

8.5.5.5.7. Implementación del modelo

Ahora descargamos la carpeta con el modelo congelado 5 y copiamos los archivos *frozen_inference_graph.pb, model.ckpt.data-00000-of-00001, model.ckpt.index* y

model.ckpt.meta a la carpeta *data/models/ssd_mobilenet_v1_coco* sustituyendo el contenido anterior si lo hubiera. Por otro lado tenemos que poner en la carpeta *data/labels* los archivos *label_map.pbtxt* y *labels.txt*.

Por último, en el archivo dentro del *scripts detect_ros.py* hay que cambiar el número de clases a tres y si cambiáramos el nombre del modelo también cambiaríamos el *model_name*.

Con todo esto ya tendríamos todo listo para poder detectar tuberías, buzos y burbujas.

8.5.5.5.8. Pruebas

Una vez tenemos todo configurado e integrado con ROS vamos a realizar pruebas de detección en tiempo real a ver qué tal se comporta la red. En las figuras siguientes se puede apreciar cómo el sistema es capaz de realizar la detección de manera satisfactoria. Aunque en algunas ocasiones da falsos positivos debido a que tampoco disponemos de una gran cantidad de imágenes, el sistema es perfectamente usable, ya que cuando se realizó una prueba con unas 500 imágenes el sistema daba demasiados falsos positivos como para poder ser utilizado.

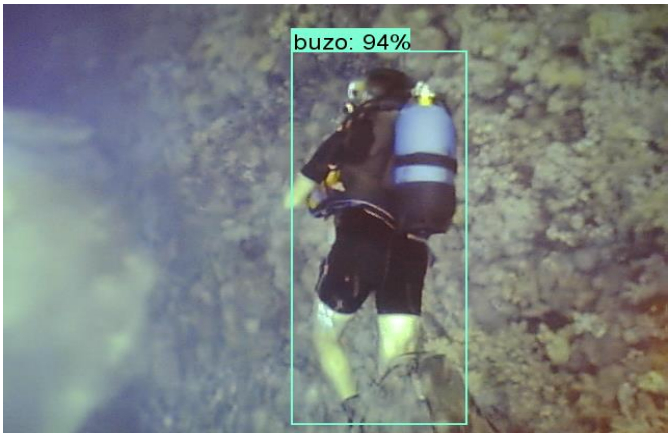


Figura 8.28: Detección de buzo usando la red con reconocimiento personalizado

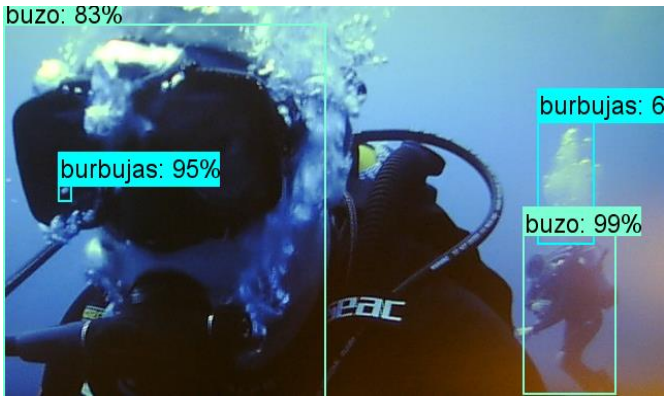


Figura 8.29: Detección de buzos y burbujas usando la red con reconocimiento personalizado

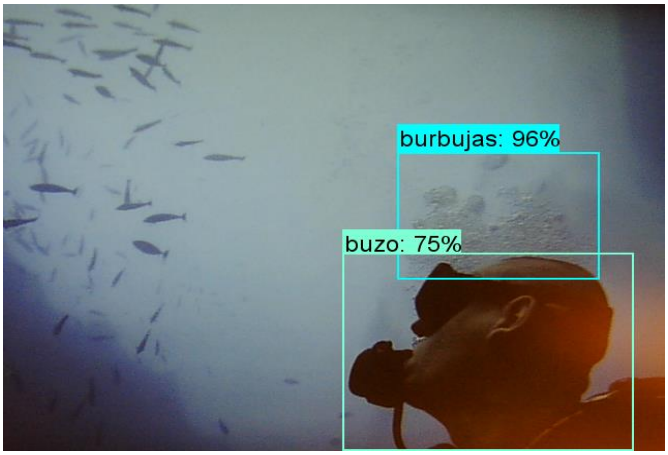


Figura 8.30: Detección de buzo y burbujas usando la red con reconocimiento personalizado

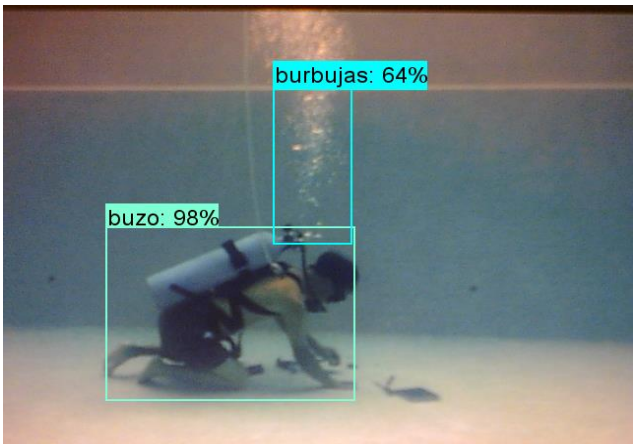


Figura 8.31: Detección de buzo y burbujas usando la red con reconocimiento personalizado

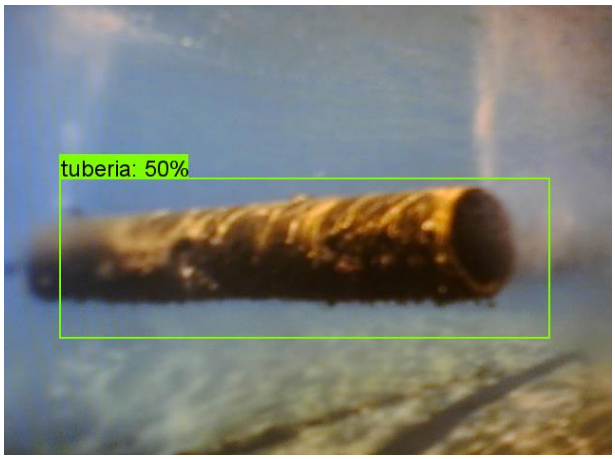


Figura 8.32: Detección de tubería usando la red con reconocimiento personalizado

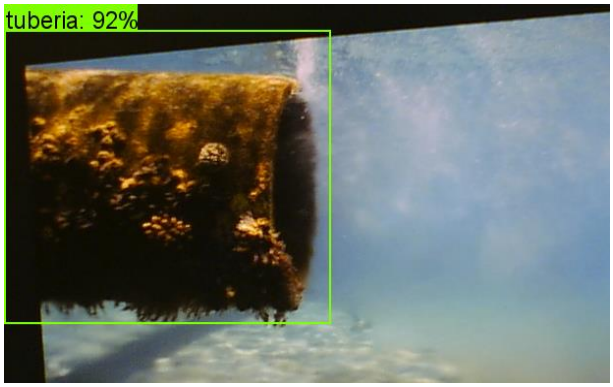


Figura 8.33: Detección de tubería usando la red con reconocimiento personalizado

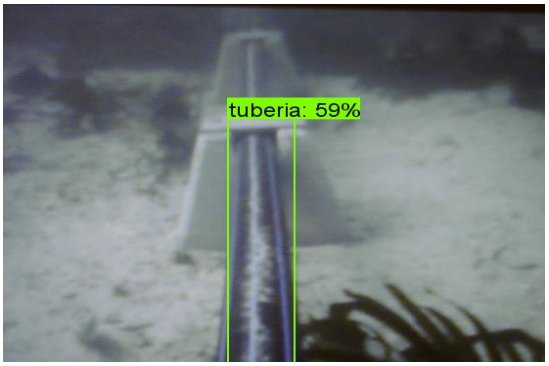


Figura 8.34: Detección de tubería usando la red con reconocimiento personalizado

8.6. Ejecución de paquetes de visión

Para poder ejecutar todos los paquetes con una sola línea ROS permite crear archivos *.launch* que nos permiten hacer precisamente esto. Para lanzar todo lo referente a visión en un ordenador local y poder realizar pruebas el archivo a utilizar contendría el Código 8.2.

```
<launch>
            <node name="usb_cam" pkg="usb_cam" type="usb_cam_node" output="screen" >
              <param name="video_device" value="/dev/video0" />
              <param name="image_width" value="640" />
              <param name="image_height" value="480" />
              <param name="pixel_format" value="yuyv" />
              <param name="camera_frame_id" value="usb_cam" />
              <param name="io_method" value="mmap"/>

            </node>
            <node pkg= "tensorflow_object_detector" name="detect_ros"
          type="detect_ros.py"  output="screen">
              <remap from="image" to="/usb_cam/image_raw"/>
            </node>
            <node pkg="record_video" type="record_video.py" name="record_video">
```

```
    <param name="identity" value="video"/>
  </node>
  <node pkg="tracker_planne" type="planne_tracker.py" name="tracker_planne">
  </node>
  <node pkg="feature_detector" type="feature.py" name="feature_detector">
  </node>
  <node pkg="tracker" type="multiTracker.py" name="tracker">
    <param name="tracker" value="MOSSE" />
  </node>

  <node name="rqt_gui" pkg="rqt_gui" type="rqt_gui"/>
</launch>
```

Código 8.2: Launch file de visión

Capítulo 9. Audio

9.1. Introducción

En este apartado describiremos la implementación de un sistema que nos permita captar sonido a través de un dispositivo USB conectado a la Raspberry. El objetivo es proveer al usuario de una manera de poder conectar un dispositivo de audio como pueda ser un hidrófono mediante interfaz USB, orientado principalmente a tareas de investigación. En este caso se va a utilizar el micrófono de la webcam como dispositivo captador de audio.

9.2. Requerimientos

Este sistema de sonido tiene que estar integrado con ROS y debe permitir la grabación de sonido a través del dispositivo conectado en cuanto el usuario desee. Además de esto el audio que está captando el dispositivo puede ser retransmitido en *streaming* a través del ordenador.

9.3. ROS

9.3.1. Obtener la señal de audio

El código utilizado para resolver esta parte se puede consultar en el anexo 10. Para usar la información captada por el micrófono conectado a la Raspberry Pi usamos la librería *pyAudio* [61] con la que nos conectamos a los dispositivos y tomamos los datos. Esta información la publicamos usando el mensaje de ROS *AudioData* [62].

9.3.2. Streaming

El código utilizado en este apartado se puede consultar en el anexo 11. Con este código que se ejecuta en el ordenador nos suscribimos al mensaje de audio y lo que se recibe se emite a través de los altavoces.

9.3.3. Grabación

Este código también se ejecuta en el ordenador, se puede consultar en el anexo 12. Con este el usuario puede realizar grabaciones de audio que se guardan en la carpeta */U_Records/audio/*. El sistema para empezar y finalizar las grabaciones es exactamente igual que para la grabación de vídeo. Tenemos una ventana abierta que tiene por título "*No grabando audio*" cuando la abrimos y pulsamos la *A* se comienza la grabación y en este momento se cierra la ventana y se abre otra que pone "*Grabando audio*" cuando abrimos esta ventana y pulsamos *S* se detiene la grabación de audio.

Capítulo 10. Interfaz de usuario

10.1. Introducción

El objetivo de este apartado es diseñar una interfaz perfectamente integrada con ROS que nos permita ver toda la información de *logs*, sensores y cámara de forma cómoda. Esta interfaz tiene que ser modular en el sentido de que debemos poder hacer diferentes disposiciones de la interfaz atendiendo al uso que se le va a dar al submarino, si vamos a centrarnos en imagen nos puede interesar tener una zona más grande de visión y tener menos información de los sensores, o en una aplicación más genérica poder acceder a todos los sensores, aunque perdamos espacio para ver la información de la cámara.

10.2. Requerimientos

La prioridad es que la interfaz sea sencilla en cuanto a uso y que los datos se puedan consultar de forma clara utilizando gráficos o en el caso de los mensajes teniendo una jerarquía que permita identificar la importancia de los mismos. Para la modularidad de la misma el usuario no debe estar construyendo cada interfaz de forma personalizada sino que tendrá una serie de archivos con configuraciones prediseñadas que podrá cargar de forma sencilla.

10.3. Diseño

Para el diseño de la interfaz vamos a usar el paquete de ROS *rqt* con este podemos crear disposiciones que se llaman en la aplicación "*perspectives*". Para poder crear las disposiciones que necesitamos con gráficas tenemos que usar la librería *PyQtGraph* [63], con la que podemos juntar varias gráficas en una misma columna. Una vez tenemos esto seleccionado, creamos la interfaz arrastrando los diferentes tipos de bloques que necesitamos, en nuestro caso necesitamos una consola donde se vean los *logs*, gráficos, una ventana de *image view*, la cual nos deja seleccionar qué imagen queremos ver, sólo la imagen en tiempo real, la imagen con detección, o cualquiera de las que se están publicando, y otra de *pose view*, para saber la orientación del submarino. En algunos casos podemos no utilizar la ventana donde vemos la pose pues con esto ganamos espacio para la imagen y dependiendo de la pantalla que usemos esto será de gran ayuda. Un ejemplo de interfaz genérica lo podemos observar en la Figura 10.1. Como se ve en esta figura la información de logs, donde aparecen errores importantes, se encuentra arriba a la izquierda. Esto se hace para cumplir con lo propuesto por el Diagrama de Gutenberg [64], que dicta que el movimiento del ojo es de arriba a abajo y de izquierda a derecha, por lo que la información más importante debe ir arriba a la izquierda.

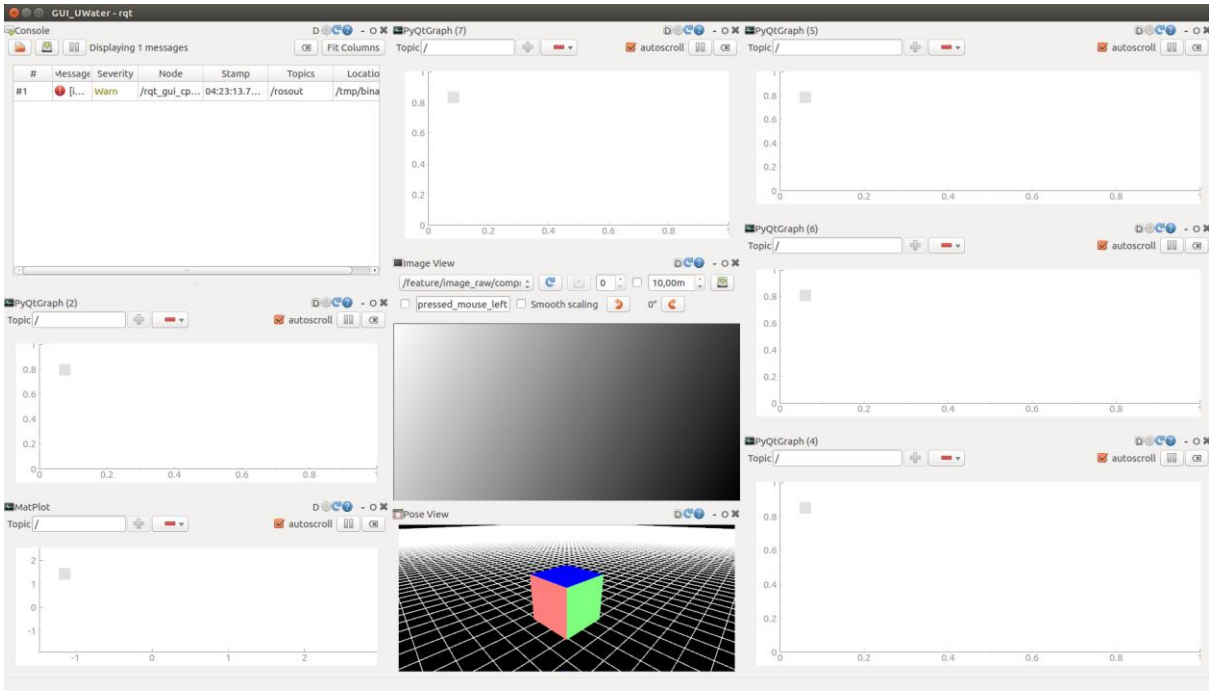


Figura 10.1: Diseño de interfaz para el dron

Una vez tenemos creada la interfaz la podemos guardar para posteriormente importarla, podemos guardar tantas interfaces como queramos para posteriormente usar la que más se ajuste a las necesidades.

10.4. Pruebas

En la realización de pruebas con el dron submarino hemos hecho uso de una interfaz general que nos permita tener información de los sensores y la imagen de la cámara. Además de la ventana de *logs* donde se puede ver si al dron le entra agua en alguno de los lados.

En la Figura 10.2 se puede apreciar cómo funciona la ventana de *logs*.

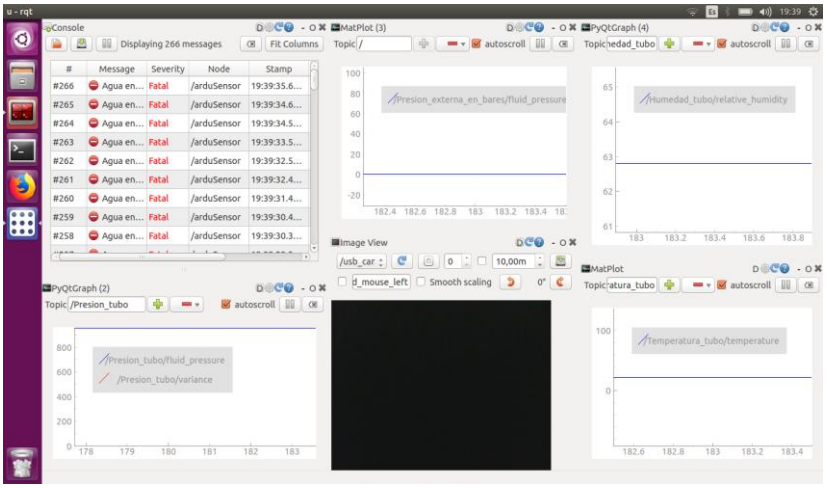


Figura 10.2: Interfaz funcionando con logs de aviso activados

En la Figura 10.3 podemos ver la interfaz antes de realizar una inmersión, tal y como se puede apreciar en la imagen de la cámara. Por otro lado, en la Figura 10.4, observamos que ya el dron está sumergido.

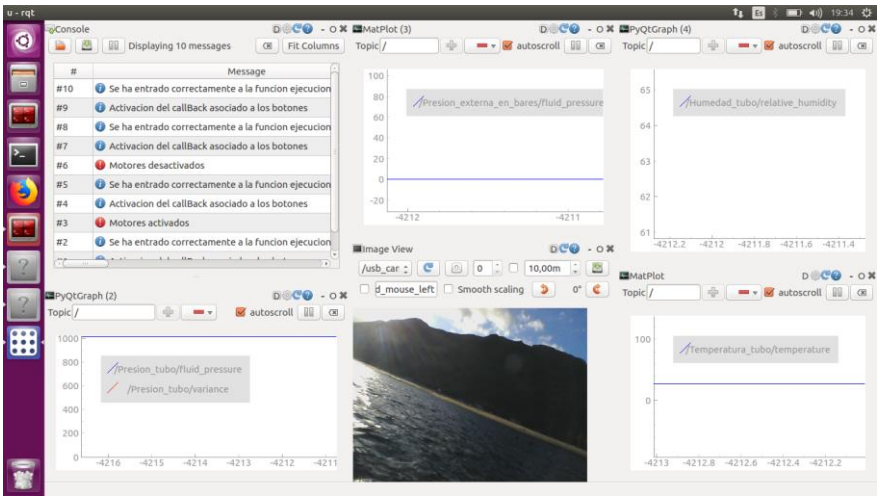


Figura 10.3: Interfaz funcionando antes de una inmersión

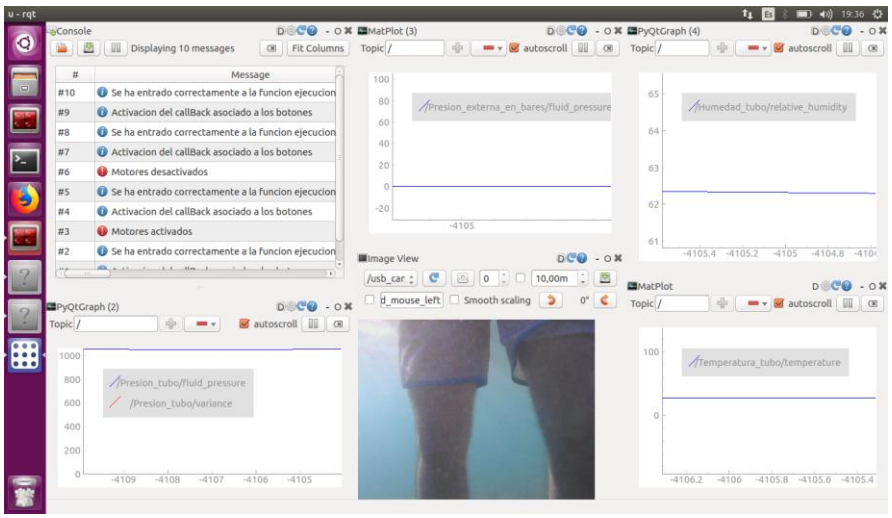


Figura 10.4: Interfaz funcionando durante una inmersión

Capítulo 11. Estructura final de ROS

En cuanto a todo el desarrollo de ROS el esquema resultante del proyecto se puede consultar en la Figura 11.1 y Figura 11.2. En estos no están incluidas todas las aplicaciones de visión por limitaciones en el portátil utilizado.

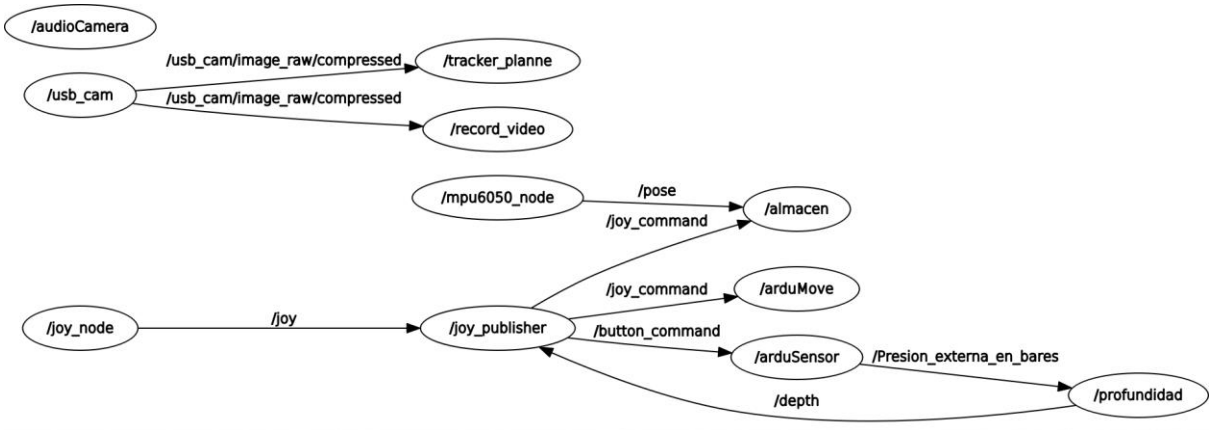


Figura 11.1: Esquema de nodos de ROS del proyecto

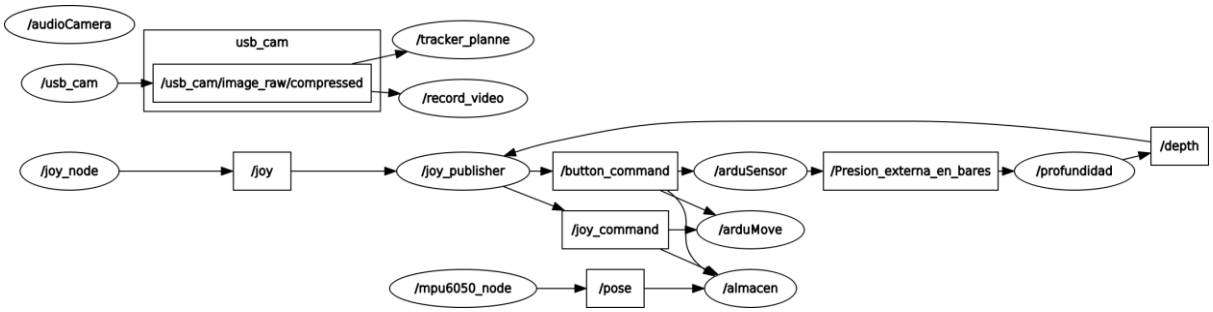


Figura 11.2: Esquema de nodos y tópicos de ROS del proyecto

El esquema de todo el sistema de visión en ROS es el que encontramos en la Figura 11.3 donde solo se ven los nodos o en la Figura 11.4 en la que se ven los nodos y tópicos, solo aparece el tópico *feature* porque es al que estábamos suscrito en el *image view*.

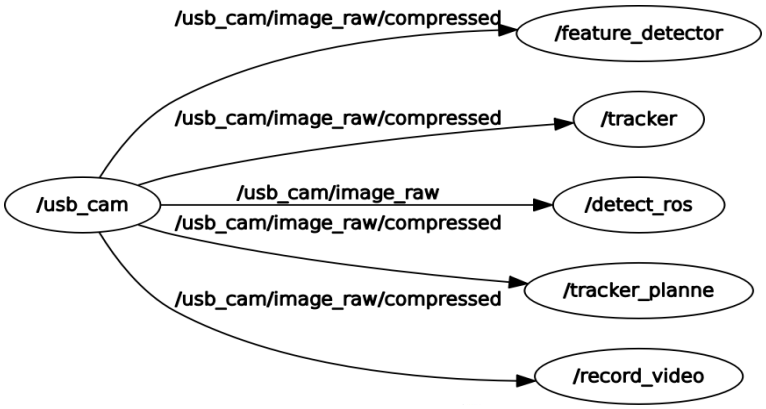


Figura 11.3: Esquema de nodos del sistema de visión

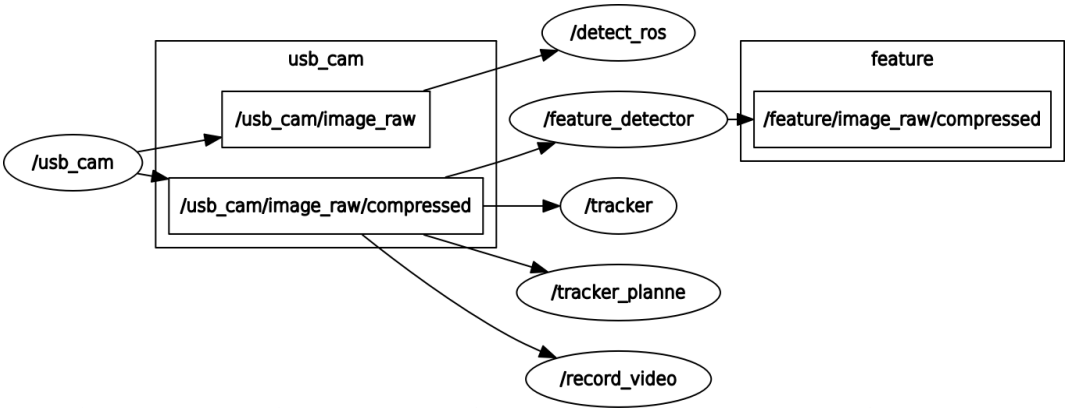


Figura 11.4: Esquema de nodos y tópicos del sistema de visión

En el repositorio del proyecto de GitHub [65] se puede acceder a todos los paquetes que son necesarios para el correcto funcionamiento del sistema. En concreto en la carpeta *Code/pc* se encuentran los paquetes que se usan en el ordenador y en la carpeta *Code/raspberry* los que se usan en la Raspberry Pi.

Capítulo 12. Resultados

En lo referente a todo el sistema de sensores el funcionamiento ha sido positivo, pudiendo realizar con ellos las tareas a las que están destinados. Se han hecho varias inmersiones con el dron en el mar, y no se ha apreciado ningún desgaste fuera del habitual.

En lo que se refiere al sistema de visión, en las primeras inmersiones se utilizó un cable de alrededor de 85 metros de longitud, con esta longitud la conexión era de 10 Mbps *half-duplex*. Con esta velocidad de transmisión se experimentó que la imagen se detenía durante un corto periodo de tiempo, haciendo difícil todas las tareas asociadas al sistema de visión, desde la grabación hasta las tareas de detección o *tracking*.

Tras estas pruebas se realizó una reducción de la longitud del cable hasta unos 65 metros, con lo que se obtuvo una velocidad de transmisión de 100 Mbps *full duplex*, mejorando notablemente la experiencia del sistema de visión en todos los aspectos.

Capítulo 13. Conclusiones/Conclusions

13.1. Conclusiones

En el desarrollo de este Trabajo de Fin de Grado se han conseguido implementar los sistemas de sensores, visión y audio que se plantearon al inicio del proyecto. Todo esto, apoyado en la interfaz de usuario para controlar el dron de forma más sencilla.

En lo que se refiere a la complejidad de la implementación de todos los sistemas, uno de los retos más importantes en el desarrollo de este trabajo ha sido aprender ROS, puesto que no conocía nada sobre el mismo. Una vez familiarizado con ROS, el aprendizaje ha sido continuo, ya que es muy amplio y no es lo mismo trabajar con los sensores y el Arduino que con la cámara o la interfaz gráfica, cada parte tenía su problemática.

En lo que a la implementación de los distintos sistemas se refiere, la parte más compleja ha sido la de detección de objetos, pues era un mundo que no conocía, en el que se plantearon retos, como el de adaptar la detección realizando el proceso de etiquetado, que llevaba mucho tiempo y en el que no había garantías de que fuera a rendir positivamente. Teniendo, en esta parte, que hacer muchas pruebas para poder obtener unos buenos resultados. Por otro lado, todo el apartado de sensórica, aunque su puesta en marcha inicial no supusiera muchos problemas, el diseño de las PCB y testeo de las mismas dentro del sistema, supuso mucho tiempo, pues para poder hacer las pruebas en condiciones de uso real, teníamos que desplazarnos, si fallaba algo, analizar las causas, arreglarlo y volver a probarlo, hasta verificar que todo estuviera bien.

Además, se han podido realizar pruebas del prototipo en el mar, concretamente en la playa de Las Teresitas situada en Santa Cruz de Tenerife. En la Figura 13.1, podemos ver el dron sumergido durante una de las varias pruebas que se han hecho a lo largo del desarrollo del proyecto.

Este proyecto me ha permitido adentrarme en mundos que no conocía como el de ROS, detección de objetos o diseño de interfaces. Aparte de todo el aprendizaje teórico, el hecho de lograr construir un prototipo real me permitió afrontar muchos problemas que en un laboratorio no se pueden apreciar, aprendiendo a prever lo que puede fallar y plantear soluciones para ello. Además, el construir el prototipo nos ha dado la posibilidad de resolver desafíos constructivos, como el de tener una cámara en la parte externa del dron. En lo que se refiere a los sensores, también hemos tenido que enfrentarnos a problemas con la condensación de agua dentro del tubo que podía provocar fallos en los sensores de detección de agua. Todo este proceso de ver cómo llevar a la realidad los diferentes sistemas, ha supuesto una experiencia muy enriquecedora.

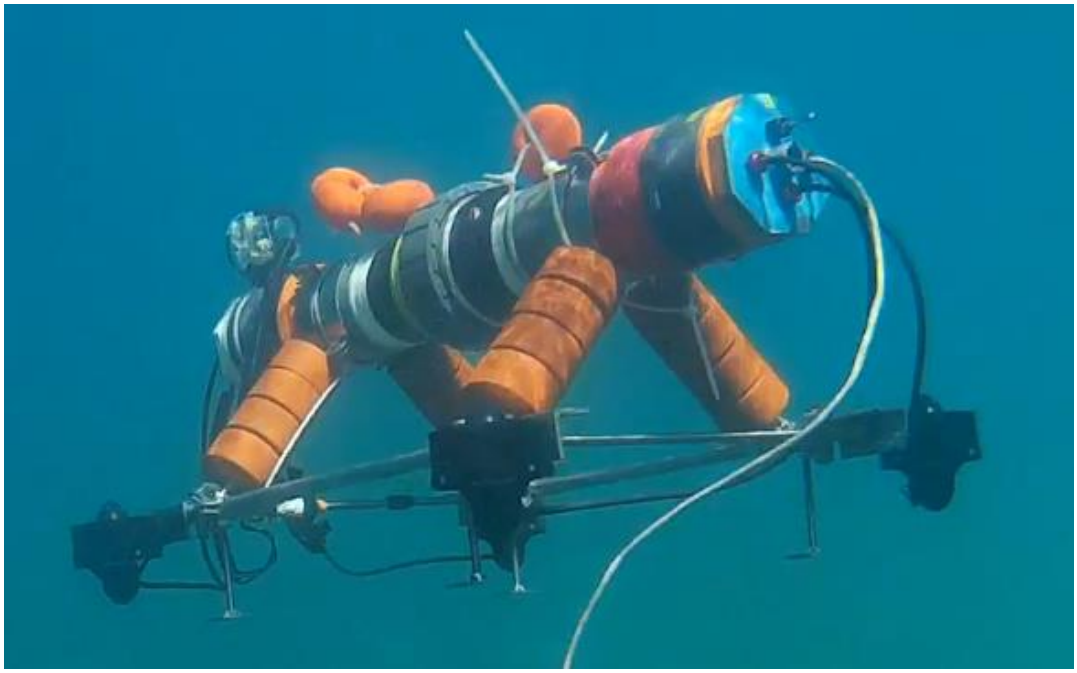


Figura 13.1: Dron durante una inmersión

13.2. Conclusions

In the development of this Final Degree Project sensor, vision and audio systems have been satisfactorily implemented. Everything supported by the user interface to control the drone easily.

Regarding the complexity of the implementation of all the systems, one of the most important challenges in the development of this work has been learning ROS, since I did not know anything about it. Once familiar with ROS, the learning process has been continuous, as it is very large, so it is not the same working with sensors and Arduino, with the camera or the graphic interface, each part had its problems.

As far as the implementation of the different systems is concerned, the most complex part has been the object detection, since it was a new world for me, in which I found challenges such as adapting detection by carrying out the process of labelling, which took a long time and in which there was no guarantee that it would work fine. That is why in this part, I had to do many tests in order to obtain good results. On the other hand, the whole section of sensorics was really difficult, because although its initial start-up did not pose many problems, the design of the PCB and testing them within the system, took a long time, because to be able to do the tests in real conditions, we had to move to the coast, if something failed, analyse the causes, fix it and try again, until we verified that everything was fine.

In addition, tests of the prototype have been carried out at sea, specifically on Las Teresitas beach located in Santa Cruz de Tenerife. In Figure 13.1, we can see the submerged drone during one of the various tests that have been done throughout the development of the project.

This project has allowed me to get to know new worlds I did not know, such as ROS, object detection or interface design. Apart from all the theoretical learning, the fact of building a real prototype allowed me to face many problems that cannot be seen in a laboratory, learning to foresee what can fail and propose solutions for it. In addition, building the prototype has given us the opportunity to solve constructive challenges, such as having a camera on the outside of the drone. As far as the sensors are concerned, we also had to deal with problems with the condensation of water inside the tube that could cause failures in the leak sensors. This whole process of seeing how to bring the different systems to reality has been a very enriching experience.

Capítulo 14. Bibliografía

[1] «Puertos de tenerife,» [En línea]. Available: https://www.puertosdetenerife.org/index.php/tf-muelles-y-darsenas/tf-santa-cruz-de-tenerife.

[2] «Raspberry Pi,» [En línea]. Available: https://www.raspberrypi.org/.

[3] ROS. [En línea]. Available: https://www.ros.org/.

[4] «Documentación ORB,» [En línea]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.

[5] «TensorFlow,» [En línea]. Available: https://www.tensorflow.org/.

[6] C. L. ZUBIMENDI, «El País,» 30 Julio 2017. [En línea]. Available: https://elpais.com/elpais/2017/07/28/ciencia/1501252929_307074.html.

[7] N. Robotics. [En línea]. Available: https://www.nidorobotics.com/.

[8] NAVATICS. [En línea]. Available: https://www.navatics.com/.

[9] «Kickstarter,» [En línea]. Available: https://www.kickstarter.com/?lang=es.

[10] «BlueRobotics,» [En línea]. Available: https://www.bluerobotics.com/.

[11] «PLOCAN EDUROV,» [En línea]. Available: https://www.plocan.eu/index.php/es/206-proyectos/proyectos-finalizados/1168-edurovs.

[12] «Raspbian,» [En línea]. Available: https://www.raspbian.org/.

[13] «Ubuntu Mate,» [En línea]. Available: https://ubuntu-mate.org/.

[14] «Arduino,» [En línea]. Available: https://www.arduino.cc/.

[15] «ArduPilot,» [En línea]. Available: http://ardupilot.org/.

[16] «Creative,» [En línea]. Available: https://es.creative.com/p/peripherals/live-cam-sync-hd.

[17] «Ubuntu,» [En línea]. Available: https://www.ubuntu.com/.

[18] «ROS Kinetic,» [En línea]. Available: http://wiki.ros.org/kinetic.

[19] «OpenCV,» [En línea]. Available: https://opencv.org/.

[20] «Git,» [En línea]. Available: https://git-scm.com/.

[21] «GitHub,» [En línea]. Available: https://github.com/.

[22] «Geany,» [En línea]. Available: https://www.geany.org/.

[23] «C++,» [En línea]. Available: http://www.cplusplus.com/.

[24] «Python,» [En línea]. Available: https://www.python.org/.

[25] «KiCAD,» [En línea]. Available: http://www.kicad-pcb.org/.

[26] «Google Images Download,» [En línea]. Available: https://github.com/hardikvasa/google-images-download.

[27] «LabelImg,» [En línea]. Available: https://github.com/tzutalin/labelImg.

[28] «CUDA,» [En línea]. Available: https://developer.nvidia.com/cuda-zone.

[29] «Google Colab,» [En línea]. Available: https://colab.research.google.com/.

[30] «BlueRobotics Cálculo Profundidad/Presión,» [En línea]. Available: http://docs.bluerobotics.com/calc/pressure-depth/.

[31] «GitHub ROS MPU-6050,» [En línea]. Available: https://github.com/chrisspen/ros_mpu6050_node.

[32] «ROS Common msgs,» [En línea]. Available: http://wiki.ros.org/common_msgs.

[33] «ROS Std msgs,» [En línea]. Available: http://wiki.ros.org/std_msgs.

[34] «ROS Logging,» [En línea]. Available: http://wiki.ros.org/rosserial_arduino/Tutorials/Logging)..

[35] «ROS Header message,» [En línea]. Available: http://docs.ros.org/api/std_msgs/html/msg/Header.html.

[36] «ROS Point Message,» [En línea]. Available: http://docs.ros.org/api/geometry_msgs/html/msg/Point.html.

[37] «ROS Quaternion message,» [En línea]. Available: http://docs.ros.org/api/geometry_msgs/html/msg/Point.html.

[38] «ROS USB Cam,» [En línea]. Available: http://wiki.ros.org/usb_cam.

[39] «OpenCV Tracking,» [En línea]. Available: https://docs.opencv.org/3.0-beta/modules/tracking/doc/tracker_algorithms.html#trackermil.

[40] «OpenCV Boosting,» [En línea]. Available: https://docs.opencv.org/2.4/modules/ml/doc/boosting.html.

[41] «OpenCV MIL,» [En línea]. Available: https://docs.opencv.org/3.4/d0/d26/classcv_1_1TrackerMIL.html.

[42] «OpenCV KCF,» [En línea]. Available: https://docs.opencv.org/3.4.2/d2/dff/classcv_1_1TrackerKCF.html.

[43] «OpenCV TLD,» [En línea]. Available: https://docs.opencv.org/3.4/dc/d1c/classcv_1_1TrackerTLD.html.

[44] «OpenCV MedianFlow,» [En línea]. Available: https://docs.opencv.org/3.4/d7/d86/classcv_1_1TrackerMedianFlow.html.

[45] «OpenCV MOSSE,» [En línea]. Available: https://docs.opencv.org/3.4/d0/d02/classcv_1_1TrackerMOSSE.html.

[46] «OpenCV Feature Matching,» [En línea]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_matcher/py_matcher.html.

[47] «OpenCV ORB,» [En línea]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_orb/py_orb.html.

[48] «OpenCV FAST,» [En línea]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_fast/py_fast.html.

[49] «OpenCV BRIEF,» [En línea]. Available: https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_feature2d/py_brief/py_brief.html.

[50] «Wiki ROS,» [En línea]. Available: http://wiki.ros.org/rospy_tutorials/Tutorials/WritingImagePublisherSubscriber.

[51] «OpenCV FLANN,» [En línea]. Available: https://docs.opencv.org/3.4/d5/d6f/tutorial_feature_flann_matcher.html.

[52] «GitHub Plane Tracker,» [En línea]. Available: https://github.com/opencv/opencv/blob/master/samples/python/plane_tracker.py.

[53] J. a. F. A. Redmon, «YOLO,» [En línea]. Available: https://pjreddie.com/darknet/yolo/.

[54] D. Calvo. [En línea]. Available: http://www.diegocalvo.es/funcion-de-activacion-redes-neuronales/.

[55] «Towards Data Science,» [En línea]. Available: https://towardsdatascience.com/r-cnn-fast-r-cnn-faster-r-cnn-yolo-object-detection-algorithms-36d53571365e.

[56] «GitHub TensorFlow Model Zoo,» [En línea]. Available: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md.

[57] «GitHub TensorFlow Configs,» [En línea]. Available: https://github.com/tensorflow/models/tree/master/research/object_detection/samples/configs.

[58] «Towards Data Science SSD,» [En línea]. Available: https://towardsdatascience.com/review-ssd-single-shot-detector-object-detection-851a94607d11.

[59] «COCO Dataset,» [En línea]. Available: http://cocodataset.org/#home.

[60] «GitHub Deteccion Objetos,» [En línea]. Available: https://github.com/puigalex/deteccion_objetos.

[61] «PyAudio,» [En línea]. Available: https://pypi.org/project/PyAudio/.

[62] «GitHub Audio Common,» [En línea]. Available: https://github.com/ros-drivers/audio_common.

[63] «PyQtGraph,» [En línea]. Available: http://www.pyqtgraph.org/.

[64] F. L. Montosa, «Revista Digital,» [En línea]. Available: https://revistadigital.inesem.es/diseno-y-artes-graficas/diagrama-de-gutenberg/.

[65] «GitHub Sub Master,» [En línea]. Available: https://github.com/nicolasarl/sub_master.

Capítulo 15. Presupuesto

En este apartado se procede al desglose de los costes materiales y de mano de obra que ha supuesto la realización de esta parte del proyecto del prototipo de dron submarino. Los costes abajo desglosados incluyen impuestos directos e indirectos, además de la seguridad social si procede.

15.1. Gastos materiales

En esta sección se pueden ver todos los materiales de los que se ha hecho uso para poder realizar con éxito todo lo descrito en este Trabajo de Fin de Grado.

Artículo	Cantidad	Coste unitario (€/unidad)	Coste total (€)
Webcam	1	15,99	15,99
Action Cam Case	1	25,00	25,00
Sensor de presión 10 bares PBT SICK	1	212,00	212,00
Cable 1.5 mm 10 m	2	2,85	5,70
Cable 6 mm 10 m	2	6,95	13,90
Raspberry Pi Model 3 B+	1	41,90	41,90
40 cables DuPont macho-hembra	1	2,50	2,50
40 cables DuPont macho-macho	2	2,50	5,00
40 cables DuPont hembra-hembra	2	2,40	4,80
Cable de telefonía 4 hilos 50 m	1	13,80	13,80
Arduino Mega 2560 original	1	51,12	51,12
Sonda DS18B20	1	2,56	2,56
Sensor de tensión	1	1,60	1,60
Tarjeta SD 16 GB	1	12,00	12,00
DHT22	1	5,33	5,33
BMP180	1	2,66	2,66
YL-83	3	2,13	6,39
Bombilla 12V	1	3,12	3,12
BD649	1	0,48	0,48
Total	**25**		**425,85**

Tabla 15.1: Materiales empleados

15.2. Mano de obra

A continuación, se detallan las horas de dedicación junto al coste que supone la realización de este trabajo.

Concepto	Horas	Coste (€/hora)	Coste total (€)
Diseño de sistemas	120	25	3.000
Programación de los sistemas	250	25	6.250
Generación de *dataset*	80	18	1.440
Realización de PCB	40	18	720
Realización de pruebas	90	20	1.800
Redacción de proyecto	40	18	720
Total	**620**		**13.930**

Tabla 15.2: Desglose de horas trabajadas y coste

15.3. Total

El presupuesto total sumando los materiales empleados y el trabajo realizado sería de 14.335,85 euros.

Capítulo 16. Anexos

16.1. Anexo 1: Código IMU Raspberry

```
1.  // https://github.com/richardghirst/PiBits/blob/master/MPU6050-
     Pi-Demo/demo_dmp.cpp
2.  #include <stdio.h>
3.  #include <stdlib.h>
4.  #include <unistd.h>
5.  #include <stdint.h>
6.  #include <string.h>
7.  #include <math.h>
8.  #include <signal.h>
9.
10. #include <ros/ros.h>
11. #include <sensor_msgs/Imu.h>
12. #include <geometry_msgs/PoseStamped.h>
13. #include <geometry_msgs/Pose.h>
14. #include <std_msgs/Bool.h>
15. #include <std_srvs/Empty.h>
16. #include <geometry_msgs/Vector3Stamped.h>
17. #include <tf/transform_datatypes.h>
18.
19. #include "I2Cdev.h"
20. #include "MPU6050_6Axis_MotionApps20.h"
21.
22. //Typically, motion processing algorithms should be run at a high
     rate, often around 200Hz,
23. //in order to provide accurate results with low latency. This is
     required even if the application
24. //updates at a much lower rate; for example, a low power user
     interface may update as slowly
25. //as 5Hz, but the motion processing should still run at 200Hz.
26. //Page 25 of MPU6050 datasheet.
27. #define DEFAULT_SAMPLE_RATE_HZ  10
28.
29. #define MPU_FRAMEID "base_imu"
30.
31. //#include "AccelGyroSensorOffsets.h"
32.
33. ros::Publisher imu_calib_pub;
34.
35. ros::ServiceClient *clientptr;
36.
37. // class default I2C address is 0x68
38. // specific I2C addresses may be passed as a parameter here
39. // AD0 low = 0x68 (default for SparkFun breakout and InvenSense
     evaluation board)
40. // AD0 high = 0x69
41. MPU6050 mpu;
```

```
42.
43.    // uncomment "OUTPUT_READABLE_QUATERNION" if you want to see the
        actual
44.    // quaternion components in a [w, x, y, z] format (not best for
        parsing
45.    // on a remote host such as Processing or something though)
46.    //#define OUTPUT_READABLE_QUATERNION
47.
48.    // uncomment "OUTPUT_READABLE_EULER" if you want to see Euler
        angles
49.    // (in degrees) calculated from the quaternions coming from the
        FIFO.
50.    // Note that Euler angles suffer from gimbal lock (for more info,
        see
51.    // http://en.wikipedia.org/wiki/Gimbal_lock)
52.    //#define OUTPUT_READABLE_EULER
53.
54.    // uncomment "OUTPUT_READABLE_YAWPITCHROLL" if you want to see
        the yaw/
55.    // pitch/roll angles (in degrees) calculated from the quaternions
        coming
56.    // from the FIFO. Note this also requires gravity vector
        calculations.
57.    // Also note that yaw/pitch/roll angles suffer from gimbal lock
        (for
58.    // more info, see: http://en.wikipedia.org/wiki/Gimbal_lock)
59.    #define OUTPUT_READABLE_YAWPITCHROLL
60.
61.    // uncomment "OUTPUT_READABLE_REALACCEL" if you want to see
        acceleration
62.    // components with gravity removed. This acceleration reference
        frame is
63.    // not compensated for orientation, so +X is always +X according
        to the
64.    // sensor, just without the effects of gravity. If you want
        acceleration
65.    // compensated for orientation, us OUTPUT_READABLE_WORLDACCEL
        instead.
66.    #define OUTPUT_READABLE_REALACCEL
67.
68.    // uncomment "OUTPUT_READABLE_WORLDACCEL" if you want to see
        acceleration
69.    // components with gravity removed and adjusted for the world
        frame of
70.    // reference (yaw is relative to initial orientation, since no
        magnetometer
71.    // is present in this case). Could be quite handy in some cases.
72.    //#define OUTPUT_READABLE_WORLDACCEL
73.
74.    // uncomment "OUTPUT_TEAPOT" if you want output that matches the
75.    // format used for the InvenSense teapot demo
76.    //#define OUTPUT_TEAPOT
77.
78.    // MPU control/status vars
79.    bool dmpReady = false;  // set true if DMP init was successful
```

```
80.   uint8_t mpuIntStatus;   // holds actual interrupt status byte
       from MPU
81.   uint8_t devStatus;      // return status after each device
       operation (0 = success, !0 = error)
82.   uint16_t packetSize;    // expected DMP packet size (default is
       42 bytes)
83.   uint16_t fifoCount;     // count of all bytes currently in FIFO
84.   uint8_t fifoBuffer[64]; // FIFO storage buffer
85.
86.   // orientation/motion vars
87.   Quaternion q;           // [w, x, y, z]         quaternion
       container
88.   VectorInt16 aa;         // [x, y, z]            accel sensor
       measurements
89.   VectorInt16 aaReal;     // [x, y, z]            gravity-free
       accel sensor measurements
90.   VectorInt16 aaWorld;    // [x, y, z]            world-frame accel
       sensor measurements
91.   VectorFloat gravity;    // [x, y, z]            gravity vector
92.   float euler[3];         // [psi, theta, phi]    Euler angle
       container
93.   float ypr[3];           // [yaw, pitch, roll]   yaw/pitch/roll
       container and gravity vector
94.
95.   int sample_rate;
96.   std::string frame_id;
97.   //ros::NodeHandle pn;
98.   //ros::NodeHandle n;
99.
100.  // mpu offsets
101.  int ax, ay, az, gx, gy, gz;
102.
103.  bool debug = false;
104.
105.  // AD0 low = 0x68 (default for SparkFun breakout and InvenSense
       evaluation board)
106.  // AD0 high = 0x69
107.  bool ado = false;
108.
109.  double angular_velocity_covariance, pitch_roll_covariance,
       yaw_covariance, linear_acceleration_covariance;
110.  double linear_acceleration_stdev_, angular_velocity_stdev_,
       yaw_stdev_, pitch_roll_stdev_;
111.
112.  ros::Publisher imu_pub;
113.  ros::Publisher pose_ss_pub;
114.  ros::Publisher imu_euler_pub;
115.  ros::Publisher mag_pub;
116.  ros::Publisher pose_pub;
117.
118.  void mySigintHandler(int sig){
119.    ROS_INFO("Shutting down mpu6050_node...");
120.
121.    mpu.reset();
```

```
122.
123.     // All the default sigint handler does is call shutdown()
124.     ros::shutdown();
125.   }
126.
127.   //
       =============================================================
128.   // ===                    MAIN PROGRAM
       LOOP                     ===
129.   //
       =============================================================
130.
131.   void loop(ros::NodeHandle pn, ros::NodeHandle n) {
132.
133.       // if programming failed, don't try to do anything
134.       if (!dmpReady) return;
135.
136.       ros::Time now = ros::Time::now();
137.
138.     //http://docs.ros.org/kinetic/api/sensor_msgs/html/msg/Imu.html
139.       sensor_msgs::Imu imu_msg;
140.       imu_msg.header.stamp = now;
141.       imu_msg.header.frame_id = frame_id;
142.
143.       geometry_msgs::Pose pose_ss_msg;
144.
145.       geometry_msgs::PoseStamped pose_msg;
146.       pose_msg.header.stamp = now;
147.       pose_msg.header.frame_id = frame_id;
148.
149.       geometry_msgs::Vector3Stamped imu_euler_msg;
150.       imu_euler_msg.header.stamp = now;
151.       imu_euler_msg.header.frame_id = frame_id;
152.
153.       geometry_msgs::Vector3Stamped mag_msg;
154.       mag_msg.header.stamp = now;
155.       mag_msg.header.frame_id = frame_id;
156.
157.       // http://www.i2cdevlib.com/forums/topic/4-understanding-raw-
      values-of-accelerometer-and-gyrometer/
158.       //The output scale for any setting is [-32768, +32767] for
      each of the six axes.
159.   //The default setting in the I2Cdevlib class is +/- 2g for the
      accel and +/- 250 deg/sec
160.       //for the gyro. If the device is perfectly level and not
      moving, then:
161.     //
162.     //    X/Y accel axes should read 0
163.     //    Z accel axis should read 1g, which is +16384 at a
      sensitivity of 2g
164.     //    X/Y/Z gyro axes should read 0
165.     //
166.     //In reality, the accel axes won't read exactly 0 since it is
      difficult to be perfectly level
```

```
167.        //and there is some noise/error, and the gyros will also not
    read exactly 0 for the same
168.        //reason (noise/error).
169.
170.        // get current FIFO count
171.        fifoCount = mpu.getFIFOCount();
172.
173.        if (fifoCount == 1024) {
174.
175.            // reset so we can continue cleanly
176.            mpu.resetFIFO();
177.            if(debug) printf("FIFO overflow!\n");
178.
179.        // otherwise, check for DMP data ready interrupt (this should
    happen frequently)
180.        } else if (fifoCount >= 42) {
181.
182.        // read a packet from FIFO
183.            mpu.getFIFOBytes(fifoBuffer, packetSize);
184.
185.        // display quaternion values in easy matrix form: w x y z
186.        mpu.dmpGetQuaternion(&q, fifoBuffer);
187.        if(debug) printf("quat %7.2f %7.2f %7.2f %7.2f    ",
    q.w,q.x,q.y,q.z);
188.
189.        imu_msg.orientation.x = q.x;
190.        imu_msg.orientation.y = q.y;
191.        imu_msg.orientation.z = q.z;
192.        imu_msg.orientation.w = q.w;
193.
194.        pose_msg.pose.orientation.x = q.x;
195.        pose_msg.pose.orientation.y = q.y;
196.        pose_msg.pose.orientation.z = q.z;
197.        pose_msg.pose.orientation.w = q.w;
198.
199.        pose_msg.pose.position.x = 0;
200.        pose_msg.pose.position.y = 0;
201.        pose_msg.pose.position.z = 0;
202.
203.        pose_ss_msg.orientation.x = q.x;
204.        pose_ss_msg.orientation.y = q.y;
205.        pose_ss_msg.orientation.z = q.z;
206.        pose_ss_msg.orientation.w = q.w;
207.
208.        pose_ss_msg.position.x = 0;
209.        pose_ss_msg.position.y = 0;
210.        pose_ss_msg.position.z = 0;
211.
212.
213.        imu_msg.linear_acceleration_covariance[0] = linear_accelerati
    on_covariance;
214.        imu_msg.linear_acceleration_covariance[4] = linear_accelerati
    on_covariance;
```

```
215.        imu_msg.linear_acceleration_covariance[8] = linear_accelerati
    on_covariance;
216.
217.        imu_msg.angular_velocity_covariance[0] = angular_velocity_cov
    ariance;
218.        imu_msg.angular_velocity_covariance[4] = angular_velocity_cov
    ariance;
219.        imu_msg.angular_velocity_covariance[8] = angular_velocity_cov
    ariance;
220.
221.        imu_msg.orientation_covariance[0] = pitch_roll_covariance;
222.        imu_msg.orientation_covariance[4] = pitch_roll_covariance;
223.        imu_msg.orientation_covariance[8] = yaw_covariance;
224.
225.    //        #ifdef OUTPUT_READABLE_EULER
226.    //            // display Euler angles in degrees
227.    //            mpu.dmpGetQuaternion(&q, fifoBuffer);
228.    //            mpu.dmpGetEuler(euler, &q);
229.      //    imu_euler_msg.vector.y=-
    mpu.fusedEuler[VEC3_Y]*RAD_TO_DEGREE;
230.      //    imu_euler_msg.vector.x=mpu.fusedEuler[VEC3_X]*RAD_TO_DEGR
    EE;
231.      //    imu_euler_msg.vector.z=-
    mpu.fusedEuler[VEC3_Z]*RAD_TO_DEGREE;
232.      //    imu_euler_pub.publish(imu_euler_msg);
233.    //            if(debug) printf("euler %7.2f %7.2f %7.2f    ",
    euler[0] * 180/M_PI, euler[1] * 180/M_PI, euler[2] * 180/M_PI);
234.    //        #endif
235.
236.        // http://docs.ros.org/api/sensor_msgs/html/msg/Imu.html
237.        // Accelerations should be in m/s^2 (not in g's), and
    rotational velocity should be in rad/sec
238.
239.            #ifdef OUTPUT_READABLE_YAWPITCHROLL
240.                // display Euler angles in degrees
241.                mpu.dmpGetQuaternion(&q, fifoBuffer);
242.                mpu.dmpGetGravity(&gravity, &q);
243.                mpu.dmpGetYawPitchRoll(ypr, &q, &gravity);
244.
245.                // Should be in rad/sec.
246.                imu_msg.angular_velocity.x = ypr[2];
247.                imu_msg.angular_velocity.y = ypr[1];
248.                imu_msg.angular_velocity.z = ypr[0];
249.
250.                if(debug) printf("ypr (degrees)  %7.2f %7.2f
    %7.2f    ", ypr[0] *180/M_PI, ypr[1] * 180/M_PI,
    ypr[2] * 180/M_PI);
251.            #endif
252.
253.            #ifdef OUTPUT_READABLE_REALACCEL
254.                // display real acceleration, adjusted to remove
    gravity
255.                //
    https://github.com/jrowberg/i2cdevlib/blob/master/Arduino/MPU605
    0/MPU6050_6Axis_MotionApps20.h
```

```
256.              mpu.dmpGetQuaternion(&q, fifoBuffer);
257.              mpu.dmpGetAccel(&aa, fifoBuffer);
258.              mpu.dmpGetGravity(&gravity, &q);
259.              mpu.dmpGetLinearAccel(&aaReal, &aa, &gravity);
260.
261.              // By default, accel is in arbitrary units with a
     scale of 16384/1g.
262.              // Per http://www.ros.org/reps/rep-0103.html
263.              // and
     http://docs.ros.org/api/sensor_msgs/html/msg/Imu.html
264.              // should be in m/s^2.
265.              // 1g = 9.80665 m/s^2, so we go arbitrary -> g ->
     m/s^s
266.              imu_msg.linear_acceleration.x = aaReal.x * 1/16384. *
      9.80665;
267.              imu_msg.linear_acceleration.y = aaReal.y * 1/16384. *
      9.80665;
268.              imu_msg.linear_acceleration.z = aaReal.z * 1/16384. *
      9.80665;
269.
270.              if(debug) printf("areal (raw) %6d %6d %6d    ",
     aaReal.x, aaReal.y, aaReal.z);
271.              if(debug) printf("areal (m/s^2) %6d %6d %6d    ",
     imu_msg.linear_acceleration.x, imu_msg.linear_acceleration.y,
     imu_msg.linear_acceleration.z);
272.            #endif
273.
274.         imu_pub.publish(imu_msg);
275.         pose_pub.publish(pose_msg);
276.         pose_ss_pub.publish(pose_ss_msg);
277.    //            mag_msg.vector.x=mpu.calibratedMag[VEC3_X];
278.    //            mag_msg.vector.y=mpu.calibratedMag[VEC3_Y];
279.    //            mag_msg.vector.z=mpu.calibratedMag[VEC3_Z];
280.    //            mag_pub.publish(mag_msg);
281.
282.    //        #ifdef OUTPUT_READABLE_WORLDACCEL
283.    //            // display initial world-frame acceleration,
      adjusted to remove gravity
284.    //            // and rotated based on known orientation from
      quaternion
285.    //            mpu.dmpGetQuaternion(&q, fifoBuffer);
286.    //            mpu.dmpGetAccel(&aa, fifoBuffer);
287.    //            mpu.dmpGetGravity(&gravity, &q);
288.    //            mpu.dmpGetLinearAccelInWorld(&aaWorld, &aaReal,
      &q);
289.    //            if(debug) printf("aworld %6d %6d %6d    ",
      aaWorld.x, aaWorld.y, aaWorld.z);
290.    //        #endif
291.
292.         if(debug) printf("\n");
293.         }
294.    }
295.
296.    int main(int argc, char **argv){
```

```
297.
298.       ros::init(argc, argv, "mpu6050");
299.
300.       // Allows parameters passed in via <param>
301.       ros::NodeHandle pn("~");
302.
303.       // Does not allow parameters being passed in.
304.       ros::NodeHandle n;
305.
306.       signal(SIGINT, mySigintHandler);
307.
308.       ROS_INFO("Starting mpu6050_node...");
309.
310.       pn.param<int>("frequency", sample_rate,
      DEFAULT_SAMPLE_RATE_HZ);
311.       std::cout << "Using sample rate:
      " << sample_rate << std::endl;
312.
313.       pn.param<std::string>("frame_id", frame_id, MPU_FRAMEID);
314.       std::cout << "Using frame_id: " << frame_id << std::endl;
315.
316.       pn.param<int>("ax", ax, 0);
317.       pn.param<int>("ay", ay, 0);
318.       pn.param<int>("az", az, 0);
319.       pn.param<int>("gx", gx, 0);
320.       pn.param<int>("gy", gy, 0);
321.       pn.param<int>("gz", gz, 0);
322.
323.       pn.param<bool>("ado", ado, false);
324.       std::cout << "ADO: " << ado << std::endl << std::flush;
325.
326.       pn.param<bool>("debug", debug, false);
327.       std::cout << "Debug: " << debug << std::endl << std::flush;
328.
329.       // NOISE PERFORMANCE: Power Spectral Density @10Hz, AFS_SEL=0
      & ODR=1kHz 400 ug/&#8730;Hz (probably wrong)
330.       pn.param("linear_acceleration_stdev",
      linear_acceleration_stdev_, (400 /1000000.0) * 9.807 );
331.
332.       // Total RMS Noise: DLPFCFG=2 (100Hz) 0.05 °/s-rms (probably
      lower (?) @ 42Hz)
333.       pn.param("angular_velocity_stdev",
      angular_velocity_stdev_, 0.05 * (M_PI /180.0));
334.
335.       // 1 degree for pitch and roll
336.       pn.param("pitch_roll_stdev",
      pitch_roll_stdev_, 1.0 * (M_PI / 180.0));
337.
338.       // 5 degrees for yaw
339.       pn.param("yaw_stdev", yaw_stdev_, 5.0 * (M_PI / 180.0));
340.
341.       angular_velocity_covariance = angular_velocity_stdev_ *angula
      r_velocity_stdev_;
342.       linear_acceleration_covariance = linear_acceleration_stdev_ *
      linear_acceleration_stdev_;
```

```
343.        pitch_roll_covariance = pitch_roll_stdev_ * pitch_roll_stdev_
    ;
344.        yaw_covariance = yaw_stdev_ * yaw_stdev_;
345.
346.        //
    ================================================================
347.    // === INITIAL SETUP ===
348.        //
    ================================================================
349.
350.        printf("Initializing I2C...\n");
351.        I2Cdev::initialize();
352.
353.        // verify connection
354.        printf("Testing device connections...\n");
355.        mpu = MPU6050(ado ? 0x69 : 0x68);
356.        if(mpu.testConnection()){
357.            std::cout << "MPU6050 connection
    successful" << std::endl << std::flush;
358.        }else{
359.            std::cout << "MPU6050 connection
    failed" << std::endl << std::flush;
360.            return 1;
361.        }
362.
363.        // initialize device
364.        printf("Initializing I2C devices...\n");
365.        mpu.initialize();
366.
367.        // load and configure the DMP
368.        printf("Initializing DMP...\n");
369.        devStatus = mpu.dmpInitialize();
370.
371.        // Set accel offsets.
372.        std::cout << "Setting X accel offset: " << ax << std::endl;
373.        mpu.setXAccelOffset(ax);
374.        std::cout << "Setting Y accel offset: " << ay << std::endl;
375.        mpu.setYAccelOffset(ay);
376.        std::cout << "Setting Z accel offset: " << az << std::endl;
377.        mpu.setZAccelOffset(az);
378.
379.        // Set gyro offsets.
380.        std::cout << "Setting X gyro offset: " << gx << std::endl;
381.        mpu.setXGyroOffset(gx);
382.        std::cout << "Setting Y gyro offset: " << gy << std::endl;
383.        mpu.setYGyroOffset(gy);
384.        std::cout << "Setting Z gyro offset: " << gz << std::endl;
385.        mpu.setZGyroOffset(gz);
386.
387.        // make sure it worked (returns 0 if so)
388.        if (devStatus == 0) {
389.            // turn on the DMP, now that it's ready
390.            printf("Enabling DMP...\n");
391.            mpu.setDMPEnabled(true);
```

```
392.
393.            // enable Arduino interrupt detection
394.            //Serial.println(F("Enabling interrupt detection (Arduino
    external interrupt 0)..."));
395.            //attachInterrupt(0, dmpDataReady, RISING);
396.            mpuIntStatus = mpu.getIntStatus();
397.
398.            // set our DMP Ready flag so the main loop() function
    knows it's okay to use it
399.            printf("DMP ready!\n");
400.            dmpReady = true;
401.
402.            // get expected DMP packet size for later comparison
403.            packetSize = mpu.dmpGetFIFOPacketSize();
404.        } else {
405.            // ERROR!
406.            // 1 = initial memory load failed
407.            // 2 = DMP configuration updates failed
408.            // (if it's going to break, usually the code will be 1)
409.            printf("DMP Initialization failed (code %d)\n",
    devStatus);
410.        }
411.
412.        usleep(100000);
413.
414.        imu_pub = n.advertise<sensor_msgs::Imu>("imu/data", 10);
415.        pose_pub = n.advertise<geometry_msgs::PoseStamped>("pose", 10
    );
416.        imu_euler_pub = n.advertise<geometry_msgs::Vector3Stamped>("i
    mu/euler", 10);
417.        mag_pub = n.advertise<geometry_msgs::Vector3Stamped>("imu/mag
    ", 10);
418.        pose_ss_pub = n.advertise<geometry_msgs::Pose>("pose_ss", 10)
    ;
419.        ros::Rate r(sample_rate);
420.        while(ros::ok()){
421.            loop(pn, n);
422.            ros::spinOnce();
423.            r.sleep();
424.        }
425.
426.        std::cout << "Shutdown." << std::endl << std::flush;
427.
428.        return 0;
429.
430.    }
```

16.2. Anexo 2: Código Arduino para sensores

```
#include <ros.h>
#include <std_msgs/String.h>
#include <std_msgs/Float32.h>
#include <std_msgs/Int32MultiArray.h>
#include <sensor_msgs/Temperature.h>
#include <sensor_msgs/RelativeHumidity.h>
#include <sensor_msgs/Illuminance.h>
#include <sensor_msgs/FluidPressure.h>
#include "DHT.h"
#define DHTTYPE DHT22
#include <Wire.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_TSL2561_U.h>
#include <OneWire.h>
#include <DallasTemperature.h>
#include <SFE_BMP180.h>
#include <Adafruit_LSM303_U.h>
#include <Adafruit_Simple_AHRS.h>

const int NUM_BOTONES = 8; // Numero de botones habilitados
int botones_temp[] ={0,0,0,0,0,0,0,0};

//::::::::::::::::::ROS::::::::::::::::::
  ros::NodeHandle nh;

  //sensor_msgs::Temperature temp_msgC;
  sensor_msgs::Temperaturetemp_msgT;
  sensor_msgs::Temperaturetemp_msgA;
  //ros::Publisher pub_temp_caja("Temperatura_caja", &temp_msgC);
  ros::Publisher pub_temp_tubo("Temperatura_tubo", &temp_msgT);
  ros::Publisher pub_temp_agua("Temperatura_agua", &temp_msgA);

  //sensor_msgs::RelativeHumidity hum_msgC;
  sensor_msgs::RelativeHumidity hum_msgT;
  //ros::Publisher pub_hum_caja("Humedad_caja", &hum_msgC);
  ros::Publisher pub_hum_tubo("Humedad_tubo", &hum_msgT);

  std_msgs::String str_msg;
  ros::Publisher logs("Logs", &str_msg);

  sensor_msgs::Illuminance luxes_msg;
  ros::Publisher pub_luxes("Luxes", &luxes_msg);

  sensor_msgs::FluidPressure bares_msg;
  ros::Publisher pub_bares("Presion_externa_en_bares", &bares_msg
 );

  std_msgs::Float32 volts_msg;
  ros::Publisher pub_volts("Voltaje_bateria", &volts_msg);

```

```
50.     //sensor_msgs::FluidPressure pres_msgC;
51.     sensor_msgs::FluidPressure pres_msgT;
52.     //ros::Publisher pub_pres_caja("Presion_caja", &pres_msgC);
53.     ros::Publisherpub_pres_tubo("Presion_tubo", &pres_msgT);
54.
55.
56.   //::::::::::::::::::::::::::::::::::::
57.
58.   //PINES
59.
60.   const int detectorWaterTuboIzquierda = 22;
61.   const int detectorWaterTuboDerecha = 23;
62.   const int detectorWaterCaja = 24;
63.   const int DHTPinTubo = 2;
64.   //const int DHTPinCaja = 3;
65.   DHT dhtTubo(DHTPinTubo, DHTTYPE);
66.   //DHT dhtCaja(DHTPinCaja, DHTTYPE);
67.   const int pinBMPTubo = 48;
68.   const int pinBMPCaja = 46;
69.   const int pinBMPC = 44;
70.   const int pinDatosDQ = 4;
71.   const int voltagePin = A4;   // seleccionar la entrada para el
       sensor voltaje
72.   const int analogInPin = A5; //Para presion externa
73.   const int cameraLight = 9; // Luz de la camara
74.
75.   //END_PINES
76.
77.   //¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡BMP180¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡
78.
79.   SFE_BMP180 bmp180;
80.
81.   //¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡¡
82.
83.   //--------------DS18B20---------------------------
84.
85.   OneWire oneWireObjeto(pinDatosDQ);
86.   DallasTemperature sensorDS18B20(&oneWireObjeto);
87.
88.   //-------------------------------------------------
89.
90.   //Variables
91.
92.   long tiempoLecturaDHT = 0;
93.   long tiempoLecturaAgua = 0;
94.   long tiempoLecturaLuz = 0;
95.   long tiempoLecturaProfundidad = 0;
96.   float th[2]; //Vector con temperatura y humedad
97.   float thTubo[2]; //Vector con temperatura y humedad del tubo
98.   float thCaja[2]; //Vector con temperatura y humedad de la caja
99.   int sensorValueVolt = 0; // variable que almacena el valor raw
       para determinar el voltaje de la bateria (0 a 1023)
100.  float voltsValue = 0;
101.  float sensorValue = 0;        // valor raw del sensor sick
```

```
102.   float barValue = 0;         // value output to the PWM (analog
        out)
103.   char status; //Para bmp180
104.   double T = 0; //Para bmp180
105.   double P = 0; //Para bmp180
106.
107.   //End_Variables
108.
109.    //$$$ Encendido/Apagado de la luz de la camara $$$//
110.
111.   // Ejecucion de las acciones asociadas a los botones
112.   void ejecucion_botones(int botones[]){
113.
114.     nh.loginfo("Se ha entrado correctamente a la funcion ejecucion
        de los botones");
115.
116.     if((botones[4] == 1) && (botones[2] == 1)){//L1 + Triangulo
117.       analogWrite(cameraLight,200);
118.       nh.loginfo("Luz encendida");
119.     }
120.     if((botones[4] == 1) && (botones[3] == 1)){//L1 + Cuadrado
121.       analogWrite(cameraLight,0);
122.       nh.loginfo("Luz apagada");
123.     }
124.
125.   }
126.   // Callback de los botones
127.
128.   void botones(const std_msgs::Int32MultiArray& buttons_msg){
129.       nh.loginfo("Activacion del callBack asociado a los botones");
130.       for(int i = 0; i < NUM_BOTONES; i++){
131.         botones_temp[i] = buttons_msg.data[i];
132.         /* X,O,Triangulo,Cuadrado
133.          * L1,R1,L2,R2*/
134.       }
135.       ejecucion_botones(botones_temp);
136.   }
137.
138.   ros::Subscriber<std_msgs::Int32MultiArray>sub_buttons("button_com
        mand",&botones);
139.
140.   // $$$ Fin de encendido/ apagado de la luz de la camara $$$ //
141.
142.   bool hayAgua(int detectorWater){ //Funcion para determinar si hay
        agua con los sensores de presencia de agua
143.     bool notWaterPresence = false;
144.     notWaterPresence = digitalRead(detectorWater);
145.     if(notWaterPresence == LOW) {
146.       return true; //Ha entrado agua
147.     }
148.     else{
149.       return false; //No ha entrado agua
150.     }
151.   }
```

```
152.
153.   bool tempHum(float th[], DHT dht){
154.     th[0] = dht.readTemperature();
155.     th[1] = dht.readHumidity();
156.
157.     if (isnan(th[1]) || isnan(th[0])) {
158.       return false; //Devuelve false si hay error en el DHT22
159.     }
160.     else{
161.       return true;
162.     }
163.   }
164.
165.   float tempAgua(){
166.       sensorDS18B20.requestTemperatures();
167.       delay(100);
168.       // Leemos y devolvemos los datos del sensor DS18B20
169.       returnsensorDS18B20.getTempCByIndex(0);
170.   }
171.
172.   float fmap(float x, float in_min, float in_max, float out_min, fl
       oatout_max){ //Cambio de escala para floats (se usa en sensor de
       voltaje)
173.       return (x - in_min) * (out_max -
       out_min) / (in_max - in_min) +out_min;
174.   }
175.
176.   void setup() {
177.     Serial.begin(57600);   //Iniciar puerto serie
178.
179.     //:::::::::::::::ROS:::::::::::::::
180.     nh.initNode();
181.
182.     //logs
183.     nh.advertise(logs);
184.     //temperaturas
185.     nh.advertise(pub_temp_tubo);
186.     //nh.advertise(pub_temp_caja);
187.     nh.advertise(pub_temp_agua);
188.     //humedades
189.     nh.advertise(pub_hum_tubo);
190.     //nh.advertise(pub_hum_caja);
191.     //luxes
192.     nh.advertise(pub_luxes);
193.     //presion externa
194.     nh.advertise(pub_bares);
195.     //voltios
196.     nh.advertise(pub_volts);
197.     //presion interna
198.     nh.advertise(pub_pres_tubo);
199.     //nh.advertise(pub_pres_caja);
200.     nh.subscribe(sub_buttons);
201.     //::::::::::::::::::::::::::::::::
202.
```

```
203.     //Se establecen los pines para controlar el BMP180 que se está
     utilizando como salida
204.     pinMode(pinBMPTubo, OUTPUT);
205.     pinMode(pinBMPCaja, OUTPUT);
206.
207.     //definir los pines del sensor de agua como entrada
208.     pinMode(detectorWaterTuboIzquierda, INPUT);
209.     pinMode(detectorWaterTuboDerecha, INPUT);
210.     pinMode(detectorWaterCaja, INPUT);
211.
212.     pinMode(cameraLight, OUTPUT);
213.     //Inicializamos los DHT
214.     dhtTubo.begin();
215.     //dhtCaja.begin();
216.
217.     //Iniciamos el sensor de temperatura del agua
218.     sensorDS18B20.begin();
219.
220.     //Activamos el BMP180 del tubo
221.     digitalWrite(pinBMPC, LOW);
222.     digitalWrite(pinBMPTubo, LOW);
223.     digitalWrite(pinBMPCaja, LOW);
224.
225.     delay(50);
226.     if (! bmp180.begin()){
227.       Serial.println("Error BMP180 2");
228.       nh.logerror("Error BMP180 2");
229.     }
230.
231.   }
232.
233.   void loop() {
234.
235.     //+++++++++++++++++Comprobacion de si hay agua en algun lugar
     del submarino++++++++++++++++++++++++++++++++
236.
237.     if(hayAgua(detectorWaterTuboIzquierda) == true){
238.       str_msg.data = "Hay agua detras";
239.       nh.logfatal("Agua detras");
240.       logs.publish( &str_msg );
241.     }
242.     else{
243.       str_msg.data = "No hay agua detras";
244.       logs.publish( &str_msg );
245.     }
246.     if(hayAgua(detectorWaterTuboDerecha) == true){
247.       str_msg.data = "Hay agua delante";
248.       nh.logfatal("Agua delante");
249.       logs.publish( &str_msg );
250.     }
251.     else{
252.       str_msg.data = "No hay agua delante";
253.       logs.publish( &str_msg );
254.     }
```

```
255.     if(hayAgua(detectorWaterCaja) == true){
256.       str_msg.data = "Agua en centro";
257.       nh.logfatal("Agua en centro");
258.       logs.publish( &str_msg );
259.     }
260.     else{
261.       str_msg.data = "No hay agua en centro";
262.       logs.publish( &str_msg );
263.     }
264.
265.     //++++++++++++++++++++Fin comprobación de presencia de
    agua++++++++++++++++++++++++++
266.
267.
268.     //----------------------Comienzo lectura sensores DHT22-------
    ----------------------------
269.
270.     long tiempo = millis();
271.
272.     if((millis() - tiempoLecturaDHT) >= 2000){
273.
274.       tiempoLecturaDHT = millis();
275.       if(tempHum(th, dhtTubo) == true){
276.         thTubo[0] = th[0];
277.         temp_msgT.temperature = thTubo[0];
278.         temp_msgT.header.stamp = nh.now();
279.         pub_temp_tubo.publish(&temp_msgT);
280.         thTubo[1] = th[1];
281.         hum_msgT.relative_humidity = thTubo[1];
282.         hum_msgT.header.stamp = nh.now();
283.         pub_hum_tubo.publish(&hum_msgT);
284.       }
285.       else{
286.         str_msg.data = "Error DHT22 Tubo en loop:";
287.         nh.logerror("Error DHT22 Tubo en loop");
288.         logs.publish( &str_msg );
289.       }
290.     }
291.
292.     //---------------------Fin lectura DHTs 22--------------------
    ------
293.
294.     //^^^^^^^^^^^^^^^^^^^^^^Comienzo lectura Temperatura
    Agua^^^^^^^^^^^^^^^^^^^^^^^^
295.
296.     if((millis() - tiempoLecturaAgua) >= 1000){
297.       tiempoLecturaAgua = millis();
298.       temp_msgA.temperature = tempAgua();
299.       temp_msgA.header.stamp = nh.now();
300.       pub_temp_agua.publish(&temp_msgA);
301.     }
302.
303.     //^^^^^^^^^^^^^^^^^^^^^^Fin lectura Temperatura
    Agua^^^^^^^^^^^^^^^^^^^^^^^^
304.
```

```
305.
306.     //%%%%%%%%%%%%%%%%%%%%%%Comienzo lectura presion externa (usar R
    = 220ohmios)%%%%%%%%%%%%%%%%%%%%%%%%
307.
308.     if((millis() - tiempoLecturaProfundidad) >= 200){
309.       sensorValue = analogRead(analogInPin);
310.       tiempoLecturaProfundidad = millis();
311.       if(sensorValue <= 100){//0.004*220=0.88//0.88*(1023/5)=180.04
    8
312.         str_msg.data = "Error sensor presion externa en loop";
313.         nh.logerror("Error sensor presion externa en loop");
314.         logs.publish( &str_msg );
315.       }
316.       else{
317.         barValue = 2.8409 * (sensorValue * (5.0/1023.0)) - 2.5;
318.         bares_msg.fluid_pressure = barValue;
319.         bares_msg.header.stamp = nh.now();
320.         pub_bares.publish(&bares_msg);
321.       }
322.     }
323.
324.     //%%%%%%%%%%%%%%%%%%%%%%Fin lectura presion
    externa%%%%%%%%%%%%%%%%%%%%%%%%
325.
326.     //_____________________________Comienzo lectura
    voltage_____________________________
327.
328.     sensorValueVolt = analogRead(voltagePin);          // realizar
    la lectura
329.     voltsValue = fmap(sensorValueVolt, 0, 1023, 0.0, 25.0);   //
    cambiar escala a 0.0 - 25.0
330.     volts_msg.data = voltsValue;
331.     pub_volts.publish(&volts_msg);
332.
333.     //_____________________________Fin lectura
    voltage_____________________________
334.
335.
336.     //#########################Comienzo lectura presion
    interna################################
337.     digitalWrite(pinBMPTubo, LOW);
338.     digitalWrite(pinBMPCaja, LOW);
339.     delay(50);
340.     //BMP180 Tubo
341.     status = bmp180.startTemperature(); //Inicio de lectura de
    temperatura
342.     if (status != 0){
343.       delay(status); //Pausa para que finalice la lectura
344.       status = bmp180.getTemperature(T); //Obtener la temperatura
345.       if (status != 0){
346.         status = bmp180.startPressure(3); //Inicio lectura de
    presión
347.         if (status != 0){
```

```
348.            delay(status); //Pausa para que finalice la
    lectura
349.            status = bmp180.getPressure(P,T); //Obtenemos la presión
350.            if (status != 0){
351.              pres_msgT.fluid_pressure = P; //presion en mb
352.              pres_msgT.header.stamp = nh.now();
353.              pub_pres_tubo.publish(&pres_msgT);

354.            }
355.          }
356.        }
357.      }
358.      //#########################Fin lectura presion
    interna################################
359.
360.      nh.spinOnce();
361.    }
```

16.3. Anexo 3: Código para grabación de vídeo

```
#!/usr/bin/env python
#coding: utf-8


import numpy as np
import cv2
import datetime
import rospy
import os
from os.path import expanduser
from std_msgs.msg import String
import sys
import keyboard
from sensor_msgs.msg import CompressedImage


out = cv2.VideoWriter(None, 0, 30.0, (640,480))

class image_record:

    def __init__(self):
        # Hacemos la subscripcion al topico de la camara
        self.subscriber =
 rospy.Subscriber("/usb_cam/image_raw/compressed",
 CompressedImage, self.callback,  queue_size = 240)

    def callback(self, ros_data):
        global out
        #Aqui se hace la conversion de imagen en formato ROS a
 formato OpenCV
        np_arr = np.fromstring(ros_data.data, np.uint8)
        #image_np = cv2.imdecode(np_arr, cv2.CV_LOAD_IMAGE_COLOR)
        image_np = cv2.imdecode(np_arr, cv2.IMREAD_COLOR) #
 OpenCV >= 3.0:

        frame = image_np
        # Guardamos el frame convertido
  out.write(frame)


if __name__ == '__main__':

# Iniciamos el nodo
  rospy.init_node("record_video")
        ic = image_record()
  home = expanduser("~") #Definimos la ruta a home
  pathFolder = home + "/U_Records/" + "video" #Establecemos el
 directorio para guardar el video
  # Creamos el directorio si no existe
  try:
```

```
46.       os.stat(pathFolder)
47.     except:
48.       os.makedirs(pathFolder)
49.
50.           # Especificamos el video codec
51.     fourcc = cv2.VideoWriter_fourcc(*'XVID')
52.
53.     while not rospy.is_shutdown():
54.             img_stop = cv2.imread(home
    + '/sub_master/Code/pc/src/record_video/nodes/stop.jpg') #Leemos
    la imagen que va a indicar que no estamos grabando
55.             #La escalamos
56.             r = 100.0 / img_stop.shape[1]
57.             dim = (100, int(img_stop.shape[0] * r))
58.             resized_stop = cv2.resize(img_stop, dim, interpolation
    = cv2.INTER_AREA)
59.             #Lamostramos
60.             cv2.imshow('No grabando video', resized_stop)
61.             #Si se pulsa v comenzamos la grabacion
62.             if cv2.waitKey(1) & 0xFF == ord('v'):
63.               cv2.destroyAllWindows()
64.         # Creamos el archivo de video con la informacion de fecha y
    hora
65.         date = datetime.datetime.now().strftime("%Y-%m-%d
    %H:%M:%S")
66.         path = pathFolder + "/outputVideo_"+ date + ".avi"
67.               out = cv2.VideoWriter(path,fourcc, 30.0, (640,480))
68.               img = cv2.imread(home
    + '/sub_master/Code/pc/src/record_video/nodes/rec blanco.jpg')
69.               resized = cv2.resize(img, dim, interpolation =
    cv2.INTER_AREA)
70.               #Mostramos una imagen que indique que estamos
    grabando
71.               cv2.imshow('Grabando video', resized)
72.         while True:
73.                 if cv2.waitKey(1) & 0xFF == ord('b'): #Si pulsamos
    b se para la grabacion
74.                       out.release()
75.                       cv2.destroyAllWindows()
76.                       break
77.
78.
79.
80.     rospy.spin()
```

16.4. Anexo 4: Código para multiTracker

```
#!/usr/bin/python
#
# Copyright 2018 BIG VISION LLC ALL RIGHTS RESERVED
#
from __future__ import print_function
import sys
import cv2
import numpy as np
from random import randint
import rospy
from sensor_msgs.msg import CompressedImage
from os.path import expanduser

home = expanduser("~")
OPENCV_OPENCL_CACHE_CLEANUP=0
frame = cv2.imdecode(np.array([0,0]), cv2.IMREAD_COLOR)
rectangle = cv2.imdecode(np.array([0,0]), cv2.IMREAD_COLOR)

class image_tracker:
  def __init__(self):
        '''Initialize ros publisher, ros subscriber'''
        # Publicamos la imagen del tracker
        self.image_pub =
 rospy.Publisher("/tracker/image_raw/compressed",
 CompressedImage, queue_size=10)
        # Nos suscribimos a la imagen de la camara
        self.subscriber =
 rospy.Subscriber("/usb_cam/image_raw/compressed",
 CompressedImage, self.callback,  queue_size = 1)

  def callback(self, ros_data):
        global frame
        global rectangle

        # Convertimos la imagen que llega de la camara en una que
 podamos usar con openCV
        np_arr = np.fromstring(ros_data.data, np.uint8)
        #image_np = cv2.imdecode(np_arr, cv2.CV_LOAD_IMAGE_COLOR)
        image_np = cv2.imdecode(np_arr, cv2.IMREAD_COLOR) #
 OpenCV >= 3.0:

        frame = image_np # Guardamos la imagen en frame que es
 global y la usamos en el main
        if np.shape(rectangle) != (): # Solo publicamos imagenes
 con rectangulos para evitar el parpadeo
          msg = CompressedImage()
          msg.header.stamp = rospy.Time.now()
          msg.format = "jpeg"
          msg.data = np.array(cv2.imencode('.jpg',
 rectangle)[1]).tostring()
```

```
42.             # Publicamos la imagen con el formato de ROS
43.             self.image_pub.publish(msg)
44.
45.
46.
47.   trackerTypes
       = ['BOOSTING', 'MIL', 'KCF','TLD', 'MEDIANFLOW', 'MOSSE'] #
       Definimos los tipos de tracker disponibles
48.
49.   def createTrackerByName(trackerType):
50.     # Create a tracker based on tracker name
51.     if trackerType == trackerTypes[0]:
52.       tracker = cv2.TrackerBoosting_create()
53.     elif trackerType == trackerTypes[1]:
54.       tracker = cv2.TrackerMIL_create()
55.     elif trackerType == trackerTypes[2]:
56.       tracker = cv2.TrackerKCF_create()
57.     elif trackerType == trackerTypes[3]:
58.       tracker = cv2.TrackerTLD_create()
59.     elif trackerType == trackerTypes[4]:
60.       tracker = cv2.TrackerMedianFlow_create()
61.     elif trackerType == trackerTypes[5]:
62.       tracker = cv2.TrackerMOSSE_create()
63.     else:
64.       tracker = None
65.       print('Nombre de tracker incorrecto')
66.
67.     return tracker
68.
69.   if __name__ == '__main__':
70.     rospy.init_node("tracker")
71.     ic = image_tracker()
72.     while not rospy.is_shutdown():
73.       trackerType = rospy.get_param('~tracker') #Especificar tipo
       de tracker
74.
75.       # Declaramos las cajas
76.       bboxes = []
77.       colors = []
78.
79.       img = cv2.imread(home
       + '/catkin_ws/src/tracker/src/opencv.png')
80.       r = 100.0 / img.shape[1]
81.       dim = (100, int(img.shape[0] * r))
82.       resized = cv2.resize(img, dim, interpolation =
       cv2.INTER_AREA)
83.       cv2.imshow('Pulsa t para empezar a trackear', resized)
84.       if cv2.waitKey(1) & 0xFF == ord('t'):
85.         # La funcion selectROI de OpenCV no funciona para multiples
       objetos en python asi que hacemos un bucle hasta acabar de
       seleccionar
86.         while True:
87.
```

```
88.           # Por defecto selectROI dibuja boxes desde el centro asi
      que ponemos fromCenter a false asi dibujamos boxes desde la
      esquina superior izquierda
89.           if np.shape(frame) != ():
90.             bbox = cv2.selectROI ('Pulsar una tecla despues: // Q--
      >Empezar a trackear // Otra tecla siguiente objeto', frame)
91.             bboxes.append(bbox)
92.             colors.append((randint(64, 255), randint(64, 255),
      randint(64, 255)))
93.             print("Q para empezar a trackear")
94.             print("Cualquier otra tecla para siguiente objeto")
95.             k = cv2.waitKey(0) & 0xFF
96.             if (k == 113):  # Q presionada
97.               break
98.
99.         print('Selected bounding boxes {}'.format(bboxes))
100.        cv2.destroyAllWindows()
101.
102.        # Crear el objeto MultiTracker
103.        multiTracker = cv2.MultiTracker_create()
104.
105.        # Inicializar el MultiTracker
106.        for bbox in bboxes:
107.          multiTracker.add(createTrackerByName(trackerType), frame,
      bbox)
108.
109.        cv2.imshow('ESC para salir', resized)
110.        # Procesar el video y trackear
111.        while True:
112.        # Obtener posicion actualizada del objeto
113.          success, boxes = multiTracker.update(frame)
114.          if success:
115.            # Dibujar los rectangulos de los objetos trackeados
116.            for i, newbox in enumerate(boxes):
117.              p1 = (int(newbox[0]), int(newbox[1]))
118.              p2 = (int(newbox[0] + newbox[2]), int(newbox[1] +
      newbox[3]))
119.              rectangle = cv2.rectangle(frame, p1, p2,
      colors[i], 2, 1)
120.          else :
121.            for i, newbox in enumerate(boxes):
122.              p1 = (int(newbox[0]), int(newbox[1]))
123.              p2 = (int(newbox[0] + newbox[2]), int(newbox[1] +
      newbox[3]))
124.              rectangle = cv2.rectangle(frame, p1, p2,
      colors[i], 2, 1)
125.            # Fallo en el tracking
126.            cv2.putText(frame, "Tracking failure
      detected", (100,80), cv2.FONT_HERSHEY_SIMPLEX, 0.75,(0,0,255),2)
127.
128.          # Salir con ESC
129.          if cv2.waitKey(1) & 0xFF == 27:  # Esc presionado
130.            cv2.destroyAllWindows()
131.            break
```

16.5. Anexo 5: Código para seguimiento basado en características

```
#!/usr/bin/env python


# Compatibilidad Python 2/3
from __future__ import print_function
import sys
PY3 = sys.version_info[0] == 3

if PY3:
    xrange = range

import numpy as np
import cv2 as cv

# built-in modules
from collections import namedtuple

# Modulos locales
import video
import common
from video import presets

import roslib
import rospy

# Mensajes de ROS
from sensor_msgs.msg importCompressedImage


FLANN_INDEX_KDTREE = 1
FLANN_INDEX_LSH    = 6
flann_params= dict(algorithm = FLANN_INDEX_LSH,
                   table_number = 6, # 12
key_size = 12,     # 20
                   multi_probe_level = 1) #2

MIN_MATCH_COUNT = 10

'''
 image     - image to track
 rect      - tracked rectangle (x1, y1, x2, y2)
 keypoints - keypoints detected inside rect
 descrs    - their descriptors
 data      - some user-provided data
'''
PlanarTarget = namedtuple('PlaneTarget', 'image, rect, keypoints,
 descrs, data')
```

```
48.
49.   '''
50.    target - reference to PlanarTarget
51.    p0     - matched points coords in target image
52.    p1     - matched points coords in input frame
53.    H      - homography matrix from p0 to p1
54.    quad   - target boundary quad in input frame
55.   '''
56.   TrackedTarget = namedtuple('TrackedTarget', 'target, p0, p1, H,
       quad')
57.   image_np = cv.imdecode(np.array([0,0]), cv.IMREAD_COLOR)
58.   vis = cv.imdecode(np.array([0,0]), cv.IMREAD_COLOR)
59.
60.   class image_feature:
61.
62.       def __init__(self):
63.           '''Initialize ros publisher, ros subscriber'''
64.           # Publicamos el resultado de la deteccion en el topico
       /plannar_tracker/image_raw/compressed
65.           self.image_pub =
       rospy.Publisher("/plannar_tracker/image_raw/compressed",
66.               CompressedImage, queue_size = 10)
67.           # self.bridge = CvBridge()
68.           # Nos suscribimos a la imagen captada por la camara
69.           self.subscriber =
       rospy.Subscriber("/usb_cam/image_raw/compressed",
70.               CompressedImage, self.callback,  queue_size = 1)
71.
72.
73.
74.       def callback(self, ros_data):
75.           global image_np
76.           global vis
77.
78.           #Convertimos la imagen de formato ROS a OpenCV
79.           np_arr = np.fromstring(ros_data.data, np.uint8)
80.           #image_np = cv2.imdecode(np_arr, cv2.CV_LOAD_IMAGE_COLOR)
81.           image_np = cv.imdecode(np_arr, cv.IMREAD_COLOR) # OpenCV
       >= 3.0:
82.
83.
84.           if np.shape(vis) != ():
85.             #Creamos la imagen comprimida
86.             msg = CompressedImage()
87.             msg.header.stamp = rospy.Time.now()
88.             msg.format = "jpeg"
89.             msg.data = np.array(cv.imencode('.jpg',
       vis)[1]).tostring()
90.             #Publicamos la imagen
91.             self.image_pub.publish(msg)
92.
93.
94.   class PlaneTracker:
95.       def __init__(self):
96.           self.detector = cv.ORB_create(nfeatures = 1000 )
```

```
97.             self.matcher = cv.FlannBasedMatcher(flann_params, {})  # bug : need to pass empty dict (#1329)
98.             self.targets = []
99.             self.frame_points = []
100.
101.        def add_target(self, image, rect, data=None):
102.            '''Add a new tracking target.'''
103.            x0, y0, x1, y1 = rect
104.            raw_points, raw_descrs = self.detect_features(image)
105.            points, descs = [], []
106.            for kp, desc in zip(raw_points, raw_descrs):
107.                x, y = kp.pt
108.                if x0 <= x <= x1 and y0 <= y <= y1:
109.                    points.append(kp)
110.                    descs.append(desc)
111.            descs = np.uint8(descs)
112.            self.matcher.add([descs])
113.            target = PlanarTarget(image = image, rect=rect, keypoints = points, descrs=descs, data=data)
114.            self.targets.append(target)
115.
116.        def clear(self):
117.            '''Remove all targets'''
118.            self.targets = []
119.            self.matcher.clear()
120.
121.        def track(self, frame):
122.            '''Returns a list of detected TrackedTarget objects'''
123.            self.frame_points, frame_descrs = self.detect_features(frame)
124.            if len(self.frame_points) < MIN_MATCH_COUNT:
125.                return []
126.            matches = self.matcher.knnMatch(frame_descrs, k = 2)
127.            matches = [m[0] for m in matches if len(m) == 2 and m[0].distance <m[1].distance * 0.75]
128.            if len(matches) < MIN_MATCH_COUNT:
129.                return []
130.            matches_by_id = [[] for _ in xrange(len(self.targets))]
131.            for m in matches:
132.                matches_by_id[m.imgIdx].append(m)
133.            tracked = []
134.            for imgIdx, matches in enumerate(matches_by_id):
135.                if len(matches) < MIN_MATCH_COUNT:
136.                    continue
137.                target = self.targets[imgIdx]
138.                p0 = [target.keypoints[m.trainIdx].pt for m in matches]
139.                p1 = [self.frame_points[m.queryIdx].pt for m in matches]
140.                p0, p1 = np.float32((p0, p1))
141.                H, status = cv.findHomography(p0, p1, cv.RANSAC, 3.0)
142.                status = status.ravel() != 0
143.                if status.sum() < MIN_MATCH_COUNT:
```

```
144.                     continue
145.                 p0, p1 = p0[status], p1[status]
146.
147.                 x0, y0, x1, y1 = target.rect
148.                 quad = np.float32([[x0, y0], [x1, y0], [x1, y1], [x0,
     y1]])
149.                 quad = cv.perspectiveTransform(quad.reshape(1, -
     1, 2), H).reshape(-1, 2)
150.
151.                 track = TrackedTarget(target=target, p0=p0, p1=p1,
     H=H, quad=quad)
152.                 tracked.append(track)
153.             tracked.sort(key = lambda t: len(t.p0), reverse=True)
154.             return tracked
155.
156.         def detect_features(self, frame):
157.             '''detect_features(self, frame) -> keypoints, descrs'''
158.             keypoints, descrs
     = self.detector.detectAndCompute(frame, None)
159.             if descrs is None:  # detectAndCompute returns descs=None
     if not keypoints found
160.                 descrs = []
161.             return keypoints, descrs
162.
163.
164.     class App:
165.         def __init__(self, src):
166.             #self.cap = video.create_capture(src, presets['book'])
167.             self.frame = image_np
168.             self.paused = False
169.             self.tracker = PlaneTracker()
170.
171.             cv.namedWindow('plane')
172.             self.rect_sel =
     common.RectSelector('plane', self.on_rect)
173.
174.         def on_rect(self, rect):
175.             self.tracker.add_target(self.frame, rect)
176.
177.         def run(self):
178.             global vis
179.             while not rospy.is_shutdown():
180.                 playing
     = not self.paused and not self.rect_sel.dragging
181.                 if playing or self.frame is None:
182.                     #ret, frame = self.cap.read()
183.                     #if not ret:
184.                      #   break
185.                     if np.shape(image_np) != ():
186.                       self.frame = image_np.copy()
187.                 if np.shape(self.frame) != ():
188.                   vis = self.frame.copy()
189.                   if playing:
190.                       tracked = self.tracker.track(self.frame)
191.                       for tr in tracked:
```

```
192.                         cv.polylines(vis, [np.int32(tr.quad)], True
    , (255, 255, 255), 2)
193.                         for (x, y) in np.int32(tr.p1):
194.                             cv.circle(vis, (x,
    y), 2, (255, 255, 255))
195.
196.                 self.rect_sel.draw(vis)
197.                 cv.imshow('plane', vis)
198.                 ch = cv.waitKey(1)
199.                 if ch == ord(' '):
200.                     self.paused = not self.paused
201.                 if ch == ord('c'):
202.                     self.tracker.clear()
203.                 if ch == 27:
204.                     break
205.
206. if __name__ == '__main__':
207.     print(__doc__)
208.     rospy.init_node('tracker_planne', anonymous=True)
209.     ic = image_feature()
210.     App(None).run()
211.     rospy.spin()
212.     import sys
213.     try:
214.         video_src = sys.argv[1]
215.     except:
216.         video_src = 0
```

16.6. Anexo 6: Código para separar las imágenes

```
import argparse
import errno
import os
import random
import shutil


def mkdirP(path):
    try:
        os.makedirs(path)
    except OSError as exc:  # Python >2.5
        if exc.errno == errno.EEXIST and os.path.isdir(path):
            pass
        else:
            raise


def getImgs(imageDir):
    exts = ["jpg", "png"]

    # All images with one image from each class put into the validation set.
    allImgsM = []
    classes = set()
    valImgs = []
    for subdir, dirs, files inos.walk(imageDir):
        for fName in files:
            (imageClass, imageName) = (os.path.basename(subdir), fName)
            if any(imageName.lower().endswith("." + ext) for ext in exts):
                if imageClass not in classes:
                    classes.add(imageClass)
                    valImgs.append((imageClass, imageName))
                else:
                    allImgsM.append((imageClass, imageName))
    print("+ Number of Classes: '{}'.".format(len(classes)))
    return (allImgsM, valImgs)


def createTrainValSplit(imageDir, valRatio):
    print("+ Val ratio: '{}'.".format(valRatio))

    (allImgsM, valImgs) = getImgs(imageDir)

    trainValIdx = int((len(allImgsM) + len(valImgs)) * valRatio) - len(valImgs)
    assert(trainValIdx > 0)  # Otherwise, valRatio is too small.

    random.shuffle(allImgsM)
```

```
47.         valImgs += allImgsM[0:trainValIdx]
48.     trainImgs = allImgsM[trainValIdx:]
49.
50.         print("+ Training set size: '{}'.".format(len(trainImgs)))
51.         print("+ Validation set size: '{}'.".format(len(valImgs)))
52.
53.     for person, img in trainImgs:
54.             origPath = os.path.join(imageDir, person, img)
55.             newDir = os.path.join(imageDir, 'train', person)
56.             newPath = os.path.join(imageDir, 'train', person, img)
57.             mkdirP(newDir)
58.             shutil.move(origPath, newPath)
59.
60.         for person, img in valImgs:
61.             origPath = os.path.join(imageDir, person, img)
62.             newDir = os.path.join(imageDir, 'val', person)
63.             newPath = os.path.join(imageDir, 'val', person, img)
64.             mkdirP(newDir)
65.             shutil.move(origPath, newPath)
66.
67.         for person, img in valImgs:
68.             d = os.path.join(imageDir, person)
69.             if os.path.isdir(d):
70.                 os.rmdir(d)
71.
72.     if __name__ == '__main__':
73.         parser = argparse.ArgumentParser()
74.         parser.add_argument('imageDir', type=str, help="Directory of
     images to partition   in-place to 'train' and 'val'
     directories.")
75.         parser.add_argument('--valRatio', type=float, default=0.10,
76.                             help="Validation to training ratio.")
77.         args = parser.parse_args()
78.
79.         createTrainValSplit(args.imageDir, args.valRatio)
```

16.7. Anexo 7: Código para renombrar el conjunto de entrenamiento

```
1.  import os
2.
3.  # Create a list of files from the current directory who's last 4
    characters
4.  # as lowercase are either '.jpg'
5.
6.  files = [f for f in os.listdir('.') if f[-4:].lower() == '.jpg']
7.
8.  DRYRUN = True
9.  archivos_train = 0
10.
11. for (index, filename) in enumerate(files):
12.     extension = os.path.splitext(filename)[1]
13.
14.     # Define new file name pattern
15.     newname = "%0d%s" % (index + archivos_train, extension)
16.
17.     if os.path.exists(newname):
18.         print "Cannot rename %s to %s, already
    exists" % (filename, newname)
19.         continue
20.
21.     if DRYRUN:
22.         print "Renaming %s to %s" % (filename, newname)
23.         os.rename(filename, newname)
```

16.8. Anexo 8: Código para renombrar el conjunto de *test*

```
1.  import os
2.
3.  # Create a list of files from the current directory who's last 4
    characters
4.  # as lowercase are either '.jpg'
5.
6.  files = [f for f in os.listdir('.') if f[-4:].lower() == '.jpg']
7.
8.  DRYRUN = True
9.  archivos_train = 872
10.
11. for (index, filename) in enumerate(files):
12.     extension = os.path.splitext(filename)[1]
13.
14.     # Define new file name pattern
15.     newname = "%0d%s" % (index + archivos_train, extension)
16.
17.     if os.path.exists(newname):
18.         print "Cannot rename %s to %s, already
    exists" % (filename, newname)
19.         continue
20.
21.     if DRYRUN:
22.         print "Renaming %s to %s" % (filename, newname)
23.         os.rename(filename, newname)
```

16.9. Anexo 9: Cuaderno para entrenar la red neuronal

```
1.  from google.colab import drive
2.  drive.mount('/content/gdrive')
3.
4.  !apt update -qq;
5.  !wget
     https://developer.nvidia.com/compute/cuda/8.0/Prod2/local_instal
     lers/cuda-repo-ubuntu1604-8-0-local-ga2_8.0.61-1_amd64-deb;
6.  !dpkg -i cuda-repo-ubuntu1604-8-0-local-ga2_8.0.61-1_amd64-deb;
7.  !apt-key add /var/cuda-repo-8-0-local-ga2/7fa2af80.pub;
8.  !apt-get update -qq;
9.  !apt-get install cuda gcc-5 g++-5 -y -qq;
10. !ln -s /usr/bin/gcc-5 /usr/local/cuda/bin/gcc;
11. !ln -s /usr/bin/g++-5 /usr/local/cuda/bin/g++;
12. !apt install cuda-8.0;
13.
14. %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-master
15. !python3 /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
     master/setup.py build
16. !python3 /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
     master/setup.py install
17. %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
     master/object_detection
18. !export PYTHONPATH=$PYTHONPATH:/content/gdrive/My\
     Drive/Tensorflow/deteccion_objetos-
     master/object_detection/models:/content/gdrive/My\
     Drive/Tensorflow/deteccion_objetos-
     master/object_detection/models/slim
19. !apt-get install -y -qq protobuf-compiler python-pil python-lxml
20. %set_env PYTHONPATH=/content/gdrive/My\
     Drive/Tensorflow/deteccion_objetos-
     master/object_detection/models/research:/content/gdrive/My\
     Drive/Tensorflow/deteccion_objetos-
     master/object_detection/models/research/slim
21. import sys
22. sys.path.append('/content/gdrive/My\ Drive/Tensorflow/deteccion_o
     bjetos-master/object_detection/models/slim')
23. %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-master
24. %run /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
     master/object_detection/builders/model_builder_test.py
25. !python3 /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
     master/object_detection/train.py --logtostderr --
     train_dir=train_7 --pipeline_config_path=/content/gdrive/My\
     Drive/Tensorflow/deteccion_objetos-
     master/object_detection/modelo/ssd_mobilenet_v1_coco.config
26.
27. %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
     master/slim
28. !python3 setup.py build
29. !python3 setup.py install
```

```
30.  %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
      master/object_detection
31.  !export PYTHONPATH=$PYTHONPATH:/content/gdrive/My\
      Drive/Tensorflow/deteccion_objetos-master/:/content/gdrive/My\
      Drive/Tensorflow/deteccion_objetos-master/slim
32.  !apt-get install -y -qq protobuf-compiler python-pil python-lxml
33.  %set_env PYTHONPATH=/content/gdrive/My\
      Drive/Tensorflow/deteccion_objetos-
      master/object_detection/models/research:/content/gdrive/My\
      Drive/Tensorflow/deteccion_objetos-
      master/object_detection/models/research/slim
34.  import sys
35.  sys.path.append('/content/gdrive/My\ Drive/Tensorflow/deteccion_o
      bjetos-master/slim')
36.  %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-
      master/object_detection/
37.  !export PYTHONPATH=$PYTHONPATH:`pwd`:`pwd`/slim
38.
39.
40.  %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-master
41.  !python3 train.py --logtostderr --train_dir=train_7 --
      pipeline_config_path=modelo/ssd_mobilenet_v1_coco.config
42.
43.  !ln -sf /opt/bin/nvidia-smi /usr/bin/nvidia-smi
44.  !pip install gputil
45.  !pip install psutil
46.  !pip install humanize
47.  import psutil
48.  import humanize
49.  import os
50.  import GPUtil as GPU
51.  GPUs = GPU.getGPUs()
52.  # XXX: only one GPU on Colab and isn't guaranteed
53.  gpu = GPUs[0]
54.  def printm():
55.   process = psutil.Process(os.getpid())
56.   print("Gen RAM Free: " +
      humanize.naturalsize(psutil.virtual_memory().available ), " |
      Proc size: " + humanize.naturalsize(process.memory_info().rss))
57.   print("GPU RAM Free: {0:.0f}MB | Used: {1:.0f}MB | Util
      {2:3.0f}% | Total {3:.0f}MB".format(gpu.memoryFree,
      gpu.memoryUsed, gpu.memoryUtil*100, gpu.memoryTotal))
58.  printm()
59.
60.  !pip install object_detection
61.
62.  %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-master
63.  !python object_detection/export_inference_graph.py --input_type
      image_tensor --pipeline_config_path
      modelo/ssd_mobilenet_v1_coco.config  --trained_checkpoint_prefix
      train_6/model.ckpt-13579 --output_directory modelo_congelado_6
64.
65.  %cd /content/gdrive/My\ Drive/Tensorflow/deteccion_objetos-master
66.  !python object_detection/object_detection_runner.py
```

16.10. Anexo 10: Código para publicar el audio

```
#!/usr/bin/env python


import pyaudio
import wave
import numpy

import roslib
import rospy

from audio_common_msgs.msg import AudioData

form_1 = pyaudio.paInt16 # Resolucion
chans = 1 # Canales
samp_rate = 16000 # Sampling rate
chunk = 256 # Muestras por buffer
dev_index = 2 # Indice del dispositivo dado por
 p.get_device_info_by_index(ii)


audio = pyaudio.PyAudio()

# crear pyAudio stream
stream = audio.open(format = form_1,rate = samp_rate,channels =
 chans, \
                    input_device_index = dev_index,input = True,
 \
                    frames_per_buffer=chunk)

def talker():
    global audio
    global stream
    pub = rospy.Publisher('audio', AudioData, queue_size=10)
    rospy.init_node('audio', anonymous=True)
    while not rospy.is_shutdown():
        msg = AudioData()
        msg.data = stream.read(chunk, exception_on_overflow
 = False)

        pub.publish(msg)
    rospy.spin()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

16.11. Anexo 11: Código para el streaming de audio

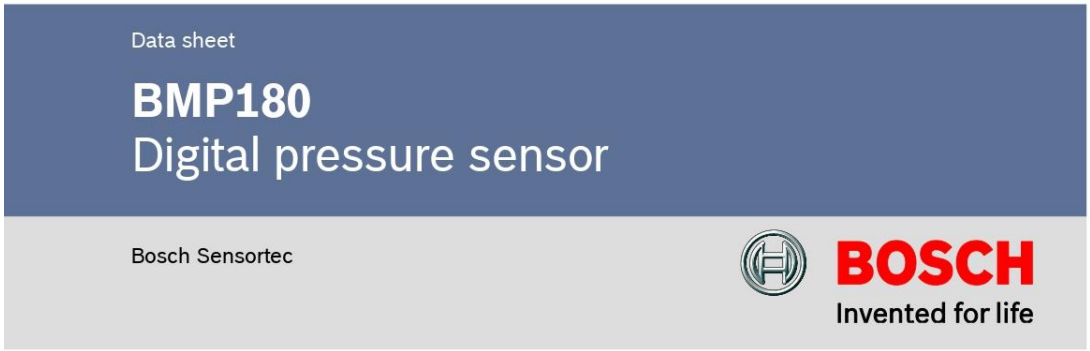
```
#!/usr/bin/env python
import pyaudio
import wave
import numpy
import cv2
import datetime
import os
from os.path import expanduser

import roslib
import rospy

from audio_common_msgs.msg import AudioData

form_1 = pyaudio.paInt16 # Resolucion
chans = 1 # Canales
samp_rate = 16000 # Sampling rate
chunk = 256 # Muestras por buffer

frames = []

audio = pyaudio.PyAudio()

stream = audio.open(format=form_1,
                    channels=chans,
                    rate=samp_rate,
                    output=True)

def callback(data):
    stream.write(data.data)

def listener():
    rospy.Subscriber("audio", AudioData, callback)

    rospy.spin()

if __name__ == '__main__':

   listener()
```

16.12. Anexo 12: Código para la grabación de audio

```
#!/usr/bin/env python
import pyaudio
import wave
import numpy
import cv2
import datetime
import os
from os.path import expanduser

import roslib
import rospy

from audio_common_msgs.msg import AudioData

form_1 = pyaudio.paInt16 # Resolucion
chans = 1 # Canales
samp_rate = 16000 # Sampling rate
chunk = 256 # Muestras por buffer
frames = []

home = expanduser("~")
pathFolder = home + "/U_Records/" + "audio"
# Creamos el directorio si no existe
try:
  os.stat(pathFolder)
except:
  os.makedirs(pathFolder)



def callback(data):

    frames.append(data.data)

def listener():

    rospy.Subscriber("audio", AudioData, callback)


if __name__ == '__main__':
    rospy.init_node('audio_listener', anonymous=True)
    listener()
    while not rospy.is_shutdown():
      img_stop = cv2.imread(home
 + '/catkin_ws/src/record_video/nodes/stop.jpg')
      r = 100.0 / img_stop.shape[1]
      dim = (100, int(img_stop.shape[0] * r))
      resized_stop = cv2.resize(img_stop, dim, interpolation =
 cv2.INTER_AREA)
```

```
48.          cv2.imshow('No grabando audio', resized_stop) # Icono para
     indicar que no se esta grabando audio
49.          if cv2.waitKey(1) & 0xFF == ord('a'):
50.            frames = []
51.            audio = pyaudio.PyAudio()
52.            img = cv2.imread(home
     + '/catkin_ws/src/record_video/nodes/rec blanco.jpg')
53.            resized = cv2.resize(img, dim, interpolation =
     cv2.INTER_AREA)
54.            cv2.destroyWindow('No grabando audio')
55.            while True:
56.              cv2.imshow('Grabando audio', resized) # Icono para
     indicar que se esta grabando audi
57.              if cv2.waitKey(1) & 0xFF == ord('s'):
58.                date = datetime.datetime.now().strftime("%Y-%m-%d
     %H:%M:%S")
59.                path = pathFolder + "/outputAudio_" + date + ".wav"
60.                wavefile = wave.open(path,'wb')
61.                wavefile.setnchannels(chans)
62.                wavefile.setsampwidth(audio.get_sample_size(form_1))
63.                wavefile.setframerate(samp_rate)
64.                wavefile.writeframes(b''.join(frames))
65.                wavefile.close()
66.                audio.terminate()
67.                cv2.destroyAllWindows()
68.                break
69.
70.       rospy.spin()
```

16.13. Anexo 13: Datasheet BMP180



BMP180 Data sheet	
Document revision	2.5
Document release date	5 April 2013
Document number	BST-BMP180-DS000-09
Technical reference code(s)	0 273 300 244
Notes	Data in this document are subject to change without notice. Product photos and pictures are for illustration purposes only and may differ from the real product's appearance.

BMP180

DIGITAL PRESSURE SENSOR

Key features

Pressure range:	300 ... 1100hPa (+9000m ... -500m relating to sea level)
Supply voltage:	1.8 ... 3.6V (V_{DD}) 1.62V ... 3.6V (V_{DDIO})
Package:	LGA package with metal lid Small footprint: 3.6mm x 3.8mm Super-flat: 0.93mm height
Low power:	5µA at 1 sample / sec. in standard mode
Low noise:	0.06hPa (0.5m) in ultra low power mode 0.02hPa (0.17m) advanced resolution mode

- Temperature measurement included
- I^2C interface
- Fully calibrated
- Pb-free, halogen-free and RoHS compliant,
- MSL 1

Typical applications

- Enhancement of GPS navigation (dead-reckoning, slope detection, etc.)
- In- and out-door navigation
- Leisure and sports
- Weather forecast
- Vertical velocity indication (rise/sink speed)

BMP180 general description

The BMP180 is the function compatible successor of the BMP085, a new generation of high precision digital pressure sensors for consumer applications.

The ultra-low power, low voltage electronics of the BMP180 is optimized for use in mobile phones, PDAs, GPS navigation devices and outdoor equipment. With a low altitude noise of merely 0.25m at fast conversion time, the BMP180 offers superior performance. The I^2C interface allows for easy system integration with a microcontroller.

The BMP180 is based on piezo-resistive technology for EMC robustness, high accuracy and linearity as well as long term stability.

Robert Bosch is the world market leader for pressure sensors in automotive applications. Based on the experience of over 400 million pressure sensors in the field, the BMP180 continues a new generation of micro-machined pressure sensors.

Index of Contents

1. Electrical characteristics

If not stated otherwise, the given values are ±3-Sigma values over temperature/voltage range in the given operation mode. All values represent the new parts specification; additional solder drift is shown separately.

Table 1: Operating conditions, output signal and mechanical characteristics

Parameter	Symbol	Condition	Min	Typ	Max	Units
Operating temperature	T_A	operational	-40		+85	°C
		full accuracy	0		+65	
Supply voltage	V_{DD}	ripple max. 50mVpp	1.8	2.5	3.6	V
			1.62	2.5	3.6	
Supply current @ 1 sample / sec. 25°C	I_{DDLOW}	ultra low power mode		3		µA
	I_{DDSTD}	standard mode		5		µA
	I_{DDHR}	high resolution mode		7		µA
	I_{DDUHR}	Ultra high res. mode		12		µA
	I_{DDAR}	Advanced res. mode		32		µA
Peak current	I_{peak}	during conversion		650	1000	µA
Standby current	I_{DDSBM}	@ 25°C		0.1	4[1]	µA
Relative accuracy pressure V_{DD} = 3.3V		950 . . . 1050 hPa @ 25 °C		±0.12		hPa
				±1.0		m
		700 ... 900hPa 25 . . . 40 °C		±0.12		hPa
				±1.0		m
Absolute accuracy pressure V_{DD} = 3.3V		300 . . . 1100 hPa 0 . . . +65 °C	-4.0	-1.0*	+2.0	hPa
		300 . . . 1100 hPa -20 . . . 0 °C	-6.0	-1.0*	+4.5	hPa
Resolution of output data		pressure		0.01		hPa
		temperature		0.1		°C
Noise in pressure		see table on page 12-13				
Absolute accuracy temperature V_{DD} = 3.3V		@ 25 °C	-1.5	±0.5	+1.5	°C
		0 . . . +65 °C	-2.0	±1.0	+2.0	°C

[1] at 85°C

Conversion time pressure	$t_{c\_p\_low}$	ultra low power mode		3	4.5	ms
	$t_{c\_p\_std}$	standard mode		5	7.5	ms
	$t_{c\_p\_hr}$	high resolution mode		9	13.5	ms
	$t_{c\_p\_luhr}$	ultra high res. mode		17	25.5	ms
	$t_{c\_p\_ar}$	Advanced res. mode		51	76.5	ms
Conversion time temperature	t_{C_temp}	standard mode		3	4.5	ms
Serial data clock	f_{SCL}				3.4	MHz
Solder drifts		Minimum solder height 50µm	-0.5		+2	hPa
Long term stability**		12 months		±1.0		hPa

* The typical value is: -1±1
** Long term stability is specified in the full accuracy operating pressure range 0 ... 65°C

2. Absolute maximum ratings

Table 2: Absolute maximum ratings

Parameter	Condition	Min	Max	Units
Storage temperature		-40	+85	°C
Supply voltage	all pins	-0.3	+4.25	V
ESD rating	HBM, R = 1.5kΩ, C = 100pF		±2	kV
Overpressure			10,000	hPa

The BMP180 has to be handled as 

Electrostatic Sensitive Device (ESD).

Figure 1: ESD

3. Operation

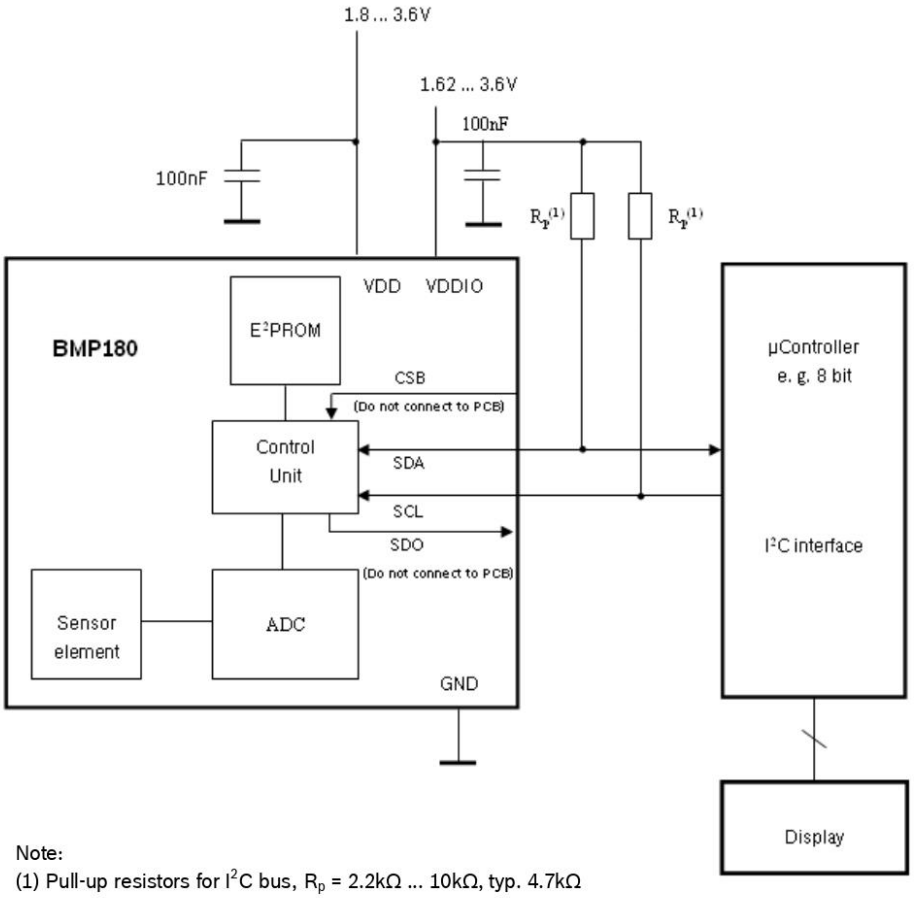
3.1 General description

The BMP180 is designed to be connected directly to a microcontroller of a mobile device via the I^2C bus. The pressure and temperature data has to be compensated by the calibration data of the E^2PROM of the BMP180.

3.2 General function and application schematics

The BMP180 consists of a piezo-resistive sensor, an analog to digital converter and a control unit with E^2PROM and a serial I^2C interface. The BMP180 delivers the uncompensated value of pressure and temperature. The E^2PROM has stored 176 bit of individual calibration data. This is used to compensate offset, temperature dependence and other parameters of the sensor.

- UP = pressure data (16 to 19 bit)
- UT = temperature data (16 bit)



Note:
(1) Pull-up resistors for I^2C bus, R_p = 2.2kΩ ... 10kΩ, typ. 4.7kΩ

Figure 2: Typical application circuit

3.3 Measurement of pressure and temperature

For all calculations presented here an ANSI C code is available from Bosch Sensortec ("BMP180 _API").

The microcontroller sends a start sequence to start a pressure or temperature measurement. After converting time, the result value (UP or UT, respectively) can be read via the I^2C interface. For calculating temperature in °C and pressure in hPa, the calibration data has to be used. These constants can be read out from the BMP180 E^2PROM via the I^2C interface at software initialization.

The sampling rate can be increased up to 128 samples per second (standard mode) for dynamic measurement. In this case, it is sufficient to measure the temperature only once per second and to use this value for all pressure measurements during the same period.

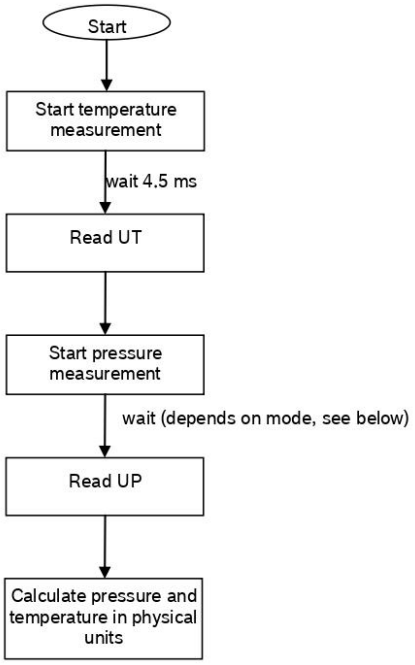


Figure 3: Measurement flow BMP180

3.3.1 Hardware pressure sampling accuracy modes

By using different modes the optimum compromise between power consumption, speed and resolution can be selected, see below table.

Table 3: Overview of BMP180 hardware accuracy modes, selected by driver software via the variable *oversampling_setting*

Mode	Parameter *oversampling_setting*	Internal number of samples	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [µA]	RMS noise typ. [hPa]	RMS noise typ. [m]
ultra low power	0	1	4.5	3	0.06	0.5
standard	1	2	7.5	5	0.05	0.4
high resolution	2	4	13.5	7	0.04	0.3
ultra high resolution	3	8	25.5	12	0.03	0.25

For further information on noise characteristics see the relevant application note “Noise in pressure sensor applications”.

All modes can be performed at higher speeds, e.g. up to 128 times per second for standard mode, with the current consumption increasing proportionally to the sample rate.

3.3.2 Software pressure sampling accuracy modes

For applications where a low noise level is critical, averaging is recommended if the lower bandwidth is acceptable. Oversampling can be enabled using the software API driver (with OSR = 3).

Table 4: Overview of BMP180 software accuracy mode, selected by driver software via the variable *software_oversampling_setting*

Mode	Parameter *oversampling_setting*	software_ oversampling_setting	Conversion time pressure max. [ms]	Avg. current @ 1 sample/s typ. [µA]	RMS noise typ. [hPa]	RMS noise typ. [m]
Advanced resolution	3	1	76.5	32	0.02	0.17

3.4 Calibration coefficients

The 176 bit E^2PROM is partitioned in 11 words of 16 bit each. These contain 11 calibration coefficients. Every sensor module has individual coefficients. Before the first calculation of temperature and pressure, the master reads out the E^2PROM data.
The data communication can be checked by checking that none of the words has the value 0 or 0xFFFF.

Table 5: Calibration coefficients

	BMP180 reg adr	
Parameter	MSB	LSB
AC1	0xAA	0xAB
AC2	0xAC	0xAD
AC3	0xAE	0xAF
AC4	0xB0	0xB1
AC5	0xB2	0xB3
AC6	0xB4	0xB5
B1	0xB6	0xB7
B2	0xB8	0xB9
MB	0xBA	0xBB
MC	0xBC	0xBD
MD	0xBE	0xBF

3.5 Calculating pressure and temperature

The mode (ultra low power, standard, high, ultra high resolution) can be selected by the variable *oversampling_setting* (0, 1, 2, 3) in the C code.

Calculation of true temperature and pressure in steps of 1Pa (= 0.01hPa = 0.01mbar) and temperature in steps of 0.1°C.

The following figure shows the detailed algorithm for pressure and temperature measurement.

This algorithm is available to customers as reference C source code ("BMP180_ API") from Bosch Sensortec and via its sales and distribution partners. **Please contact your Bosch Sensortec representative for details.**

Calculation of pressure and temperature for BMP180

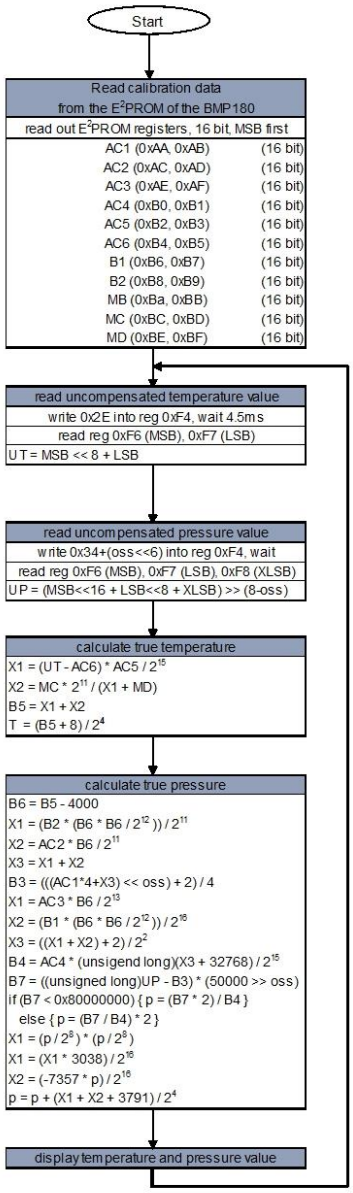


Figure 4: Algorithm for pressure and temperature measurement

3.6 Calculating absolute altitude

With the measured pressure p and the pressure at sea level p_0 e.g. 1013.25hPa, the altitude in meters can be calculated with the international barometric formula:

$$\text{altitude} = 44330 * \left(1 - \left(\frac{p}{p_0}\right)^{\frac{1}{5.255}}\right)$$

Thus, a pressure change of Δp = 1hPa corresponds to 8.43m at sea level.

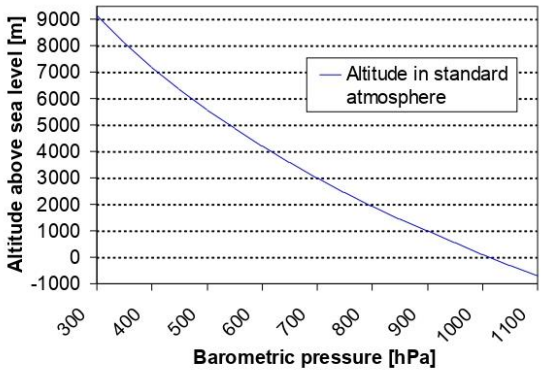


Figure 5: Transfer function: Altitude over sea level – Barometric pressure

3.7 Calculating pressure at sea level

With the measured pressure *p* and the absolute altitude the pressure at sea level can be calculated:

$$p_0 = \frac{p}{\left(1 - \frac{\text{altitude}}{44330}\right)^{5.255}}$$

Thus, a difference in altitude of Δaltitude = 10m corresponds to 1.2hPa pressure change at sea level.

4. Global Memory Map

The memory map below shows all externally accessible data registers which are needed to operate BMP180. The left columns show the memory addresses. The columns in the middle depict the content of each register bit. The colors of the bits indicate whether they are read-only, write-only or read- and writable. The memory is volatile so that the writable content has to be re-written after each power-on.
Not all register addresses are shown. These registers are reserved for further Bosch factory testing and trimming.

Register Name	Register Adress	bit7	bit6	bit5	bit4	bit3	bit2	bit1	bit0	Reset state
out_xlsb	F8h	adc_out_xlsb<7:3>					0	0	0	00h
out_lsb	F7h	adc_out_lsb<7:0>								00h
out_msb	F6h	adc_out_msb<7:0>								80h
ctrl_meas	F4h	oss<1:0>		sco	measurement control					00h
soft reset	E0h	reset								00h
id	D0h	id<7:0>								55h
calib21 downto calib0	BFh *down* to AAh	calib21<7:0> *down* to calib0<7:0>								n/a

Registers:	Control registers	Calibration registers	Data registers	Fixed
Type:	read / write	read only	read only	read only

Figure 6: Memory map

Measurement control (register F4h <4:0>): Controls measurements. Refer to Figure 6 for usage details.

Sco (register F4h <5>): Start of conversion. The value of this bit stays “1” during conversion and is reset to “0” after conversion is complete (data registers are filled).

Oss (register F4h <7:6>): controls the oversampling ratio of the pressure measurement (00b: single, 01b: 2 times, 10b: 4 times, 11b: 8 times).

Soft reset (register E0h): Write only register. If set to 0xB6, will perform the same sequence as power on reset.

Chip-id (register D0h): This value is fixed to 0x55 and can be used to check whether communication is functioning.

After conversion, data registers can be read out in any sequence (i.e. MSB first or LSB first). Using a burst read is not mandatory.

5. I²C Interface

- I²C is a digital two wire interface
- Clock frequencies up to 3.4Mbit/sec. (I²C standard, fast and high-speed mode supported)
- SCL and SDA needs a pull-up resistor, typ. 4.7kOhm to V_{DDIO}
  (one resistor each for all the I²C bus)

The I²C bus is used to control the sensor, to read calibration data from the E²PROM and to read the measurement data when A/D conversion is finished. SDA (serial data) and SCL (serial clock) have open-drain outputs.

For detailed I²C-bus specification please refer to:
http://www.nxp.com/acrobat_download/literature/9398/39340011.pdf

5.1 I²C specification

Table 6: Electrical parameters for the I²C interface

Parameter	Symbol	Min.	Typ	Max.	Units
Clock input frequency	f_{SCL}			3.4	MHz
Input-low level	V_{IL}	0		0.2 * V_{DDIO}	V
Input-high level	V_{IH}	0.8 * V_{DDIO}		V_{DDIO}	V
Voltage output low level @ V_{DDIO} = 1.62V, I_{OL} = 3mA	V_{OL}			0.3	V
SDA and SCL pull-up resistor	$R_{pull-up}$	2.2		10	kOhm
SDA sink current @ V_{DDIO} = 1.62V, V_{OL} = 0.3V	I_{SDA_sink}		9		mA
Start-up time after power-up, before first communication	t_{Start}	10			Ms

5.2 Device and register address

The BMP180 module address is shown below. The LSB of the device address distinguishes between read (1) and write (0) operation, corresponding to address 0xEF (read) and 0xEE (write).

Table 7: BMP180 addresses

A7	A6	A5	A4	A3	A2	A1	W/R
1	1	1	0	1	1	1	0/1

5.3 I²C protocol

The I²C interface protocol has special bus signal conditions. Start (S), stop (P) and binary data conditions are shown below. At start condition, SCL is high and SDA has a falling edge. Then the slave address is sent. After the 7 address bits, the direction control bit R/W selects the read or write operation. When a slave device recognizes that it is being addressed, it should acknowledge by pulling SDA low in the ninth SCL (ACK) cycle.
At stop condition, SCL is also high, but SDA has a rising edge. Data must be held stable at SDA when SCL is high. Data can change value at SDA only when SCL is low.
Even though V_{DDIO} can be powered on before V_{DD}, there is a chance of excessive power consumption (a few mA) if this sequence is used, and the state of the output pins is undefined so that the bus can be locked. Therefore, V_{DD} *must* be powered before V_{DDIO} unless the limitations above are understood and not critical.

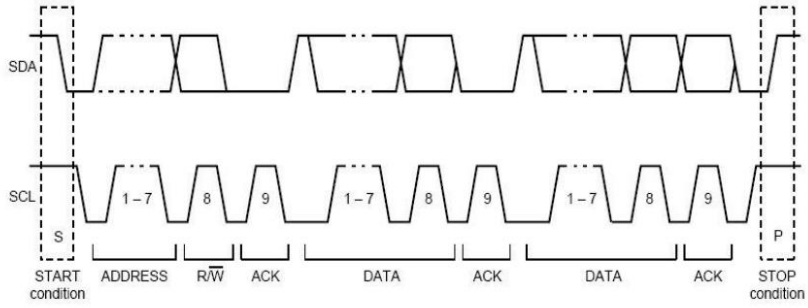


Figure 7: I²C protocol

5.4 Start temperature and pressure measurement

The timing diagrams to start the measurement of the temperature value UT and pressure value UP are shown below. After start condition the master sends the device address write, the register address and the control register data. The BMP180 sends an acknowledgement (ACKS) every 8 data bits when data is received. The master sends a stop condition after the last ACKS.

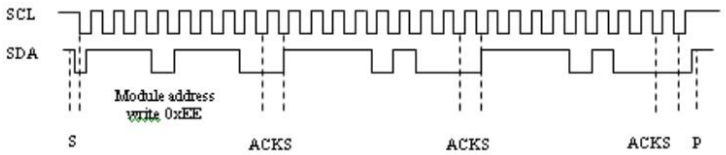


Figure 8: Timing diagram for starting pressure measurement

Abbreviations:

S	Start
P	Stop
ACKS	Acknowledge by Slave
ACKM	Acknowledge by Master
NACKM	Not Acknowledge by Master

Table 8: Control registers values for different internal oversampling_setting (oss)

Measurement	Control register value (register address 0xF4)	Max. conversion time [ms]
Temperature	0x2E	4.5
Pressure (oss = 0)	0x34	4.5
Pressure (oss = 1)	0x74	7.5
Pressure (oss = 2)	0xB4	13.5
Pressure (oss = 3)	0xF4	25.5

5.5 Read A/D conversion result or E^2PROM data

To read out the temperature data word UT (16 bit), the pressure data word UP (16 to 19 bit) and the E^2PROM data proceed as follows:

After the start condition the master sends the module address write command and register address. The register address selects the read register:

E^2PROM data registers 0xAA to 0xBF
Temperature or pressure value UT or UP 0xF6 (MSB), 0xF7 (LSB), optionally 0xF8 (XLSB)

Then the master sends a restart condition followed by the module address read that will be acknowledged by the BMP180 (ACKS). The BMP180 sends first the 8 MSB, acknowledged by the master (ACKM), then the 8 LSB. The master sends a "not acknowledge" (NACKM) and finally a stop condition.

Optionally for ultra high resolution, the XLSB register with address 0xF8 can be read to extend the 16 bit word to up to 19 bits; refer to the application programming interface (API) software rev. 1.1 ("BMP180_ API", available from Bosch Sensortec).

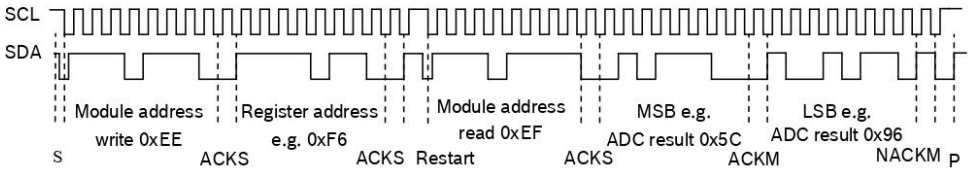


Figure 9: Timing diagram read 16 bit A/D conversion result

6. Package

6.1 Pin configuration

Picture shows the device in top view. Device pins are shown here transparently only for orientation purposes.

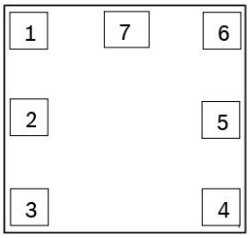


Figure 10: Layout pin configuration BMP180

Table 9: Pin configuration BMP180

in No	Name	Function
1	CSB*	Chip select
2	VDD	Power supply
3	VDDIO	Digital power supply
4	SDO*	SPI output
5	SCL	I2C serial bus clock input
6	SDA	I2C serial bus data (or SPI input)
7	GND	Ground

* A pin compatible product variant with SPI interface is possible upon customer's request. For I^2C (standard case) CSB and SDO are not used, they have to be left open.
All pins have to be soldered to the PCB for symmetrical stress input even though they are not connected internally.

6.2 Outline dimensions

The sensor housing is a 7Pin LGA package with metal lid. Its dimensions are 3.60mm (±0.1 mm) x 3.80mm (±0.1 mm) x 0.93mm (±0.07 mm).

Note: All dimensions are in mm.

6.2.1 Bottom view

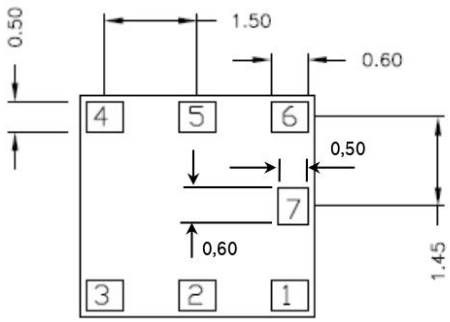


Figure 11: Bottom view BMP180

6.2.2 Top view

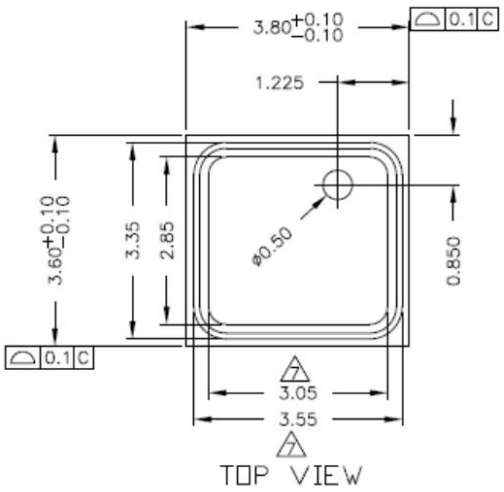


Figure 12: Top view BMP180

6.2.3 Side view

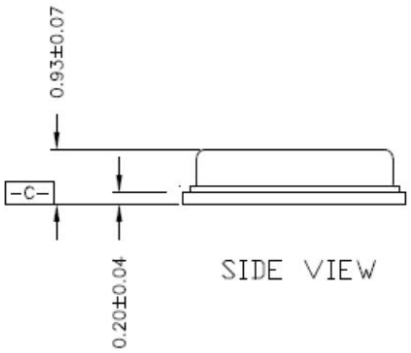


Figure 13: Side view BMP180

6.3 Moisture sensitivity level and soldering

The BMP180 is classified MSL 1 (moisture sensitivity level) according to IPC/JEDEC standards J-STD-020D and J-STD-033A.

The device can be soldered Pb-free with a peak temperature of 260°C for 20 to 40 sec. The minimum height of the solder after reflow shall be at least 50µm. This is required for good mechanical decoupling between the sensor device and the printed circuit board (PCB).

To ensure good solder-ability, the devices shall be stored at room temperature (20°C).

The soldering process can lead to an offset shift.

6.4 RoHS compliancy

The BMP180 sensor meets the requirements of the EC directive "Restriction of hazardous substances (RoHS)", please refer also to:

"Directive 2002/95/EC of the European Parliament and of the Council of 27 January 2003 on the restriction of the use of certain hazardous substances in electrical and electronic equipment".

The BMP180 sensor is also halogen-free.

6.5 Mounting and assembly recommendations

In order to achieve the specified performance for you design, the following recommendations and the “Handling, soldering & mounting instructions BMP180” should be taken into consideration when mounting a pressure sensor on a printed-circuit board (PCB):

- The clearance above the metal lid shall be 0.1mm at minimum.
- For the device housing appropriate venting needs to be provided in case the ambient pressure shall be measured.
- Liquids shall not come into direct contact with the device.
- During operation the sensor is sensitive to light, which can influence the accuracy of the measurement (photo-current of silicon).
- The BMP180 shall not the placed close the fast heating parts. In case of gradients > 3°C/sec. it is recommended to follow Bosch Sensortec application note ANP015, "Correction of errors induced by fast temperature changes". Please contact your Bosch Sensortec representative for details.

7. Legal disclaimer

7.1 Engineering samples

Engineering Samples are marked with an asterisk (*) or (e). Samples may vary from the valid technical specifications of the product series contained in this data sheet. They are therefore not intended or fit for resale to third parties or for use in end products. Their sole purpose is internal client testing. The testing of an engineering sample may in no way replace the testing of a product series. Bosch Sensortec assumes no liability for the use of engineering samples. The Purchaser shall indemnify Bosch Sensortec from all claims arising from the use of engineering samples.

7.2 Product use

Bosch Sensortec products are developed for the consumer goods industry. They may only be used within the parameters of this product data sheet. They are not fit for use in life-sustaining or security sensitive systems. Security sensitive systems are those for which a malfunction is expected to lead to bodily harm or significant property damage. In addition, they are not fit for use in products which interact with motor vehicle systems.

The resale and/or use of products are at the purchaser's own risk and his own responsibility. The examination of fitness for the intended use is the sole responsibility of the Purchaser.

The purchaser shall indemnify Bosch Sensortec from all third party claims arising from any product use not covered by the parameters of this product data sheet or not approved by Bosch Sensortec and reimburse Bosch Sensortec for all costs in connection with such claims.

The purchaser must monitor the market for the purchased products, particularly with regard to product safety, and inform Bosch Sensortec without delay of all security relevant incidents.

7.3 Application examples and hints

With respect to any examples or hints given herein, any typical values stated herein and/or any information regarding the application of the device, Bosch Sensortec hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights or copyrights of any third party. The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics. They are provided for illustrative purposes only and no evaluation regarding infringement of intellectual property rights or copyrights or regarding functionality, performance or error has been made.

8. Document history and modification

Rev. No	Chapter	Description of modifications/changes	Date
1.0		First edition for description of serial production material – Preliminary version	
1.1	5.1	New nomenclature of pin configuration	27 July 2010
1.2	5	Design change in package – hole in Lid and without slit	13 September 2010
1.3	3.2 5.1	- Standardizing pin naming over Bosch Sensortec products – typical application circuit - Optimizing pin description, SPI description	15 December 2010
2.0	1	- Non-preliminary version - Verifying parameter through characterization	28 January 2011
2.1	3.2 4 5.3 6.1 6.2.1	- Declaration of SDO and CSB pins in the typical application circuit - Adding global memory map and bits description - Power-up sequence - Description of used interfaces - Dimension pin7	1 April 2011
2.2	6.1	Correction of the pin configuration (editorial change)	14 April 2011
2.3	3.3	Optimizing noise performance	25 May 2011
2.4	6.3	Removed shelf-life constraints	26 January 2012
	page 2	Comparison removed	
	5.1	Voltage output low level added	
	5.3	Power on sequence of V_{DD} and V_{DDIO} defined	
2.5	1	Added max values for supply current for restricted version	15 Feb 2013
	1	Added max value for standby current for restricted version	5 Apr 2013
	Figure 4	Update of calculation of algorithm for pressure and temperature measurement	
	Page 2	Changed wording from “ultra high resolution mode” to “advanced resolution mode”	

Bosch Sensortec GmbH
Gerhard-Kindler-Strasse 8
72770 Reutlingen / Germany

contact@bosch-sensortec.com
www.bosch-sensortec.com

Modifications reserved | Printed in Germany
Specifications are subject to change without notice
Revision_2.5_042013
Document number: BST-BMP180-DS000-09

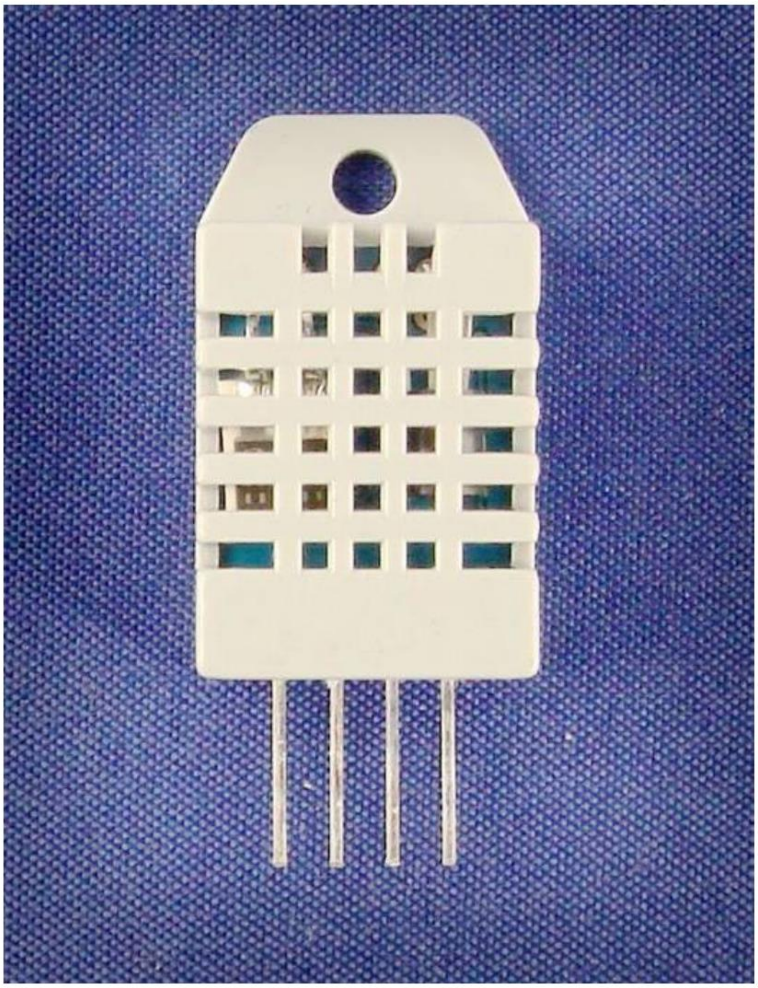
16.14. Anexo 14: Datasheet DHT22

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

Digital-output relative humidity & temperature sensor/module

DHT22 (DHT22 also named as AM2302)



Capacitive-type humidity and temperature module/sensor

1

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

1. Feature & Application:

* Full range temperature compensated * Relative humidity and temperature measurement
* Calibrated digital signal *Outstanding long-term stability *Extra components not needed
* Long transmission distance * Low power consumption *4 pins packaged and fully interchangeable

2. Description:

DHT22 output calibrated digital signal. It utilizes exclusive digital-signal-collecting-technique and humidity sensing technology, assuring its reliability and stability.Its sensing elements is connected with 8-bit single-chip computer.

Every sensor of this model is temperature compensated and calibrated in accurate calibration chamber and the calibration-coefficient is saved in type of programme in OTP memory, when the sensor is detecting, it will cite coefficient from memory.

Small size & low consumption & long transmission distance(20m) enable DHT22 to be suited in all kinds of harsh application occasions.

Single-row packaged with four pins, making the connection very convenient.

3. Technical Specification:

Model	DHT22
Power supply	3.3-6V DC
Output signal	digital signal via single-bus
Sensing element	Polymer capacitor
Operating range	humidity 0-100%RH; temperature -40~80Celsius
Accuracy	humidity +-2%RH(Max +-5%RH); temperature <+-0.5Celsius
Resolution or sensitivity	humidity 0.1%RH; temperature 0.1Celsius
Repeatability	humidity +-1%RH; temperature +-0.2Celsius
Humidity hysteresis	+-0.3%RH
Long-term Stability	+-0.5%RH/year
Sensing period	Average: 2s
Interchangeability	fully interchangeable
Dimensions	small size 14*18*5.5mm; big size 22*28*5mm

4. Dimensions: (unit----mm)

1) Small size dimensions: (unit----mm)

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

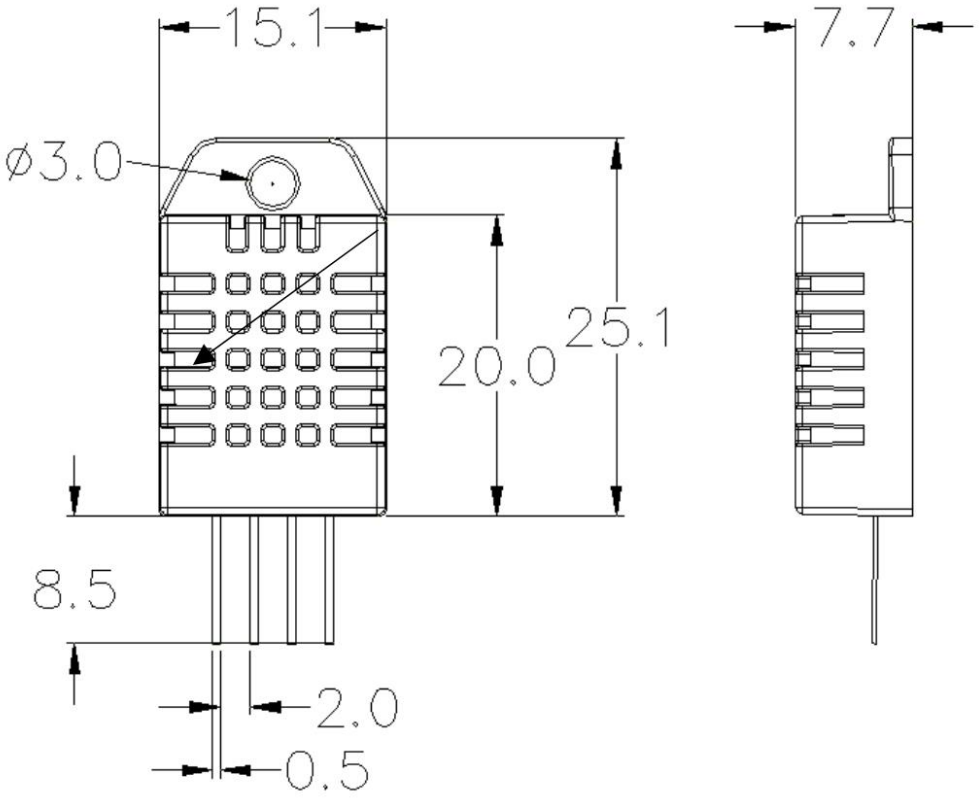
Your specialist in innovating humidity & temperature sensors

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors



Pin sequence number: 1 2 3 4 (from left to right direction).

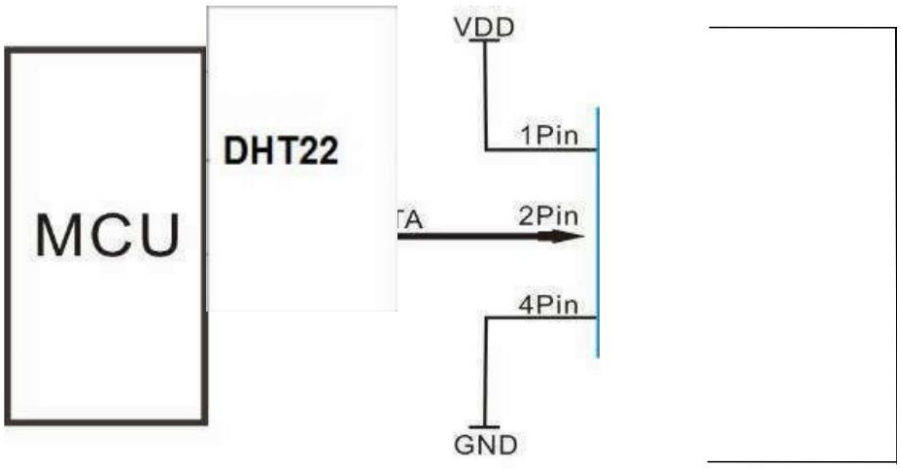
Pin	Function
1	VDD----power supply
2	DATA--signal
3	NULL
4	GND

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

5. Electrical connection diagram:



3Pin---NC, AM2302 is another name for DHT22

6. Operating specifications:

(1) Power and Pins

Power's voltage should be 3.3-6V DC. When power is supplied to sensor, don't send any instruction to the sensor within one second to pass unstable status. One capacitor valued 100nF can be added between VDD and GND for wave filtering.

(2) Communication and signal

Single-bus data is used for communication between MCU and DHT22, it costs 5mS for single time communication.

Data is comprised of integral and decimal part, the following is the formula for data.

DHT22 send out higher data bit firstly!

DATA=8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data+8 bit check-sum

If the data transmission is right, check-sum should be the last 8 bit of "8 bit integral RH data+8 bit decimal RH data+8 bit integral T data+8 bit decimal T data".

When MCU send start signal, DHT22 change from low-power-consumption-mode to running-mode. When MCU finishs sending the start signal, DHT22 will send response signal of 40-bit data that reflect the relative humidity

Thomas Liu (Business Manager)

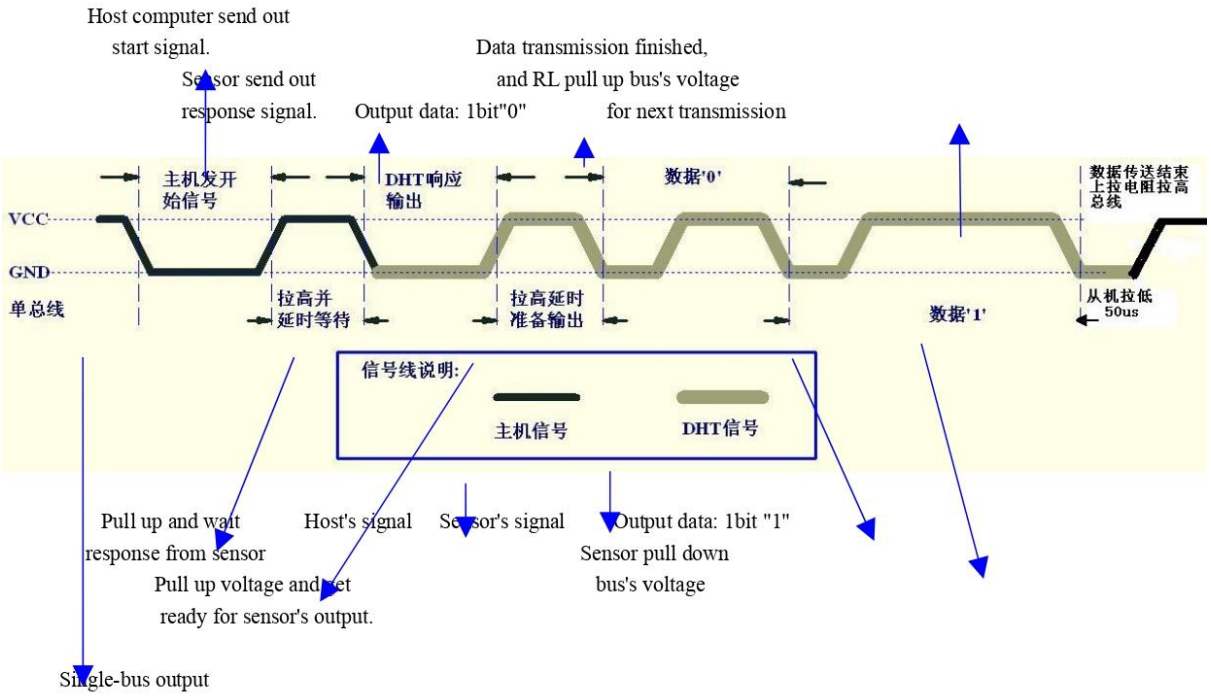
Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

and temperature information to MCU. Without start signal from MCU, DHT22 will not give response signal to MCU. One start signal for one time's response data that reflect the relative humidity and temperature information from DHT22. DHT22 will change to low-power-consumption-mode when data collecting finish if it don't receive start signal from MCU again.

1) Check bellow picture for overall communication process:



---

2) Step 1: MCU send out start signal to DHT22

Data-bus's free status is high voltage level. When communication between MCU and DHT22 begin, program of MCU will transform data-bus's voltage level from high to low level and this process must beyond at least 1ms to ensure DHT22 could detect MCU's signal, then MCU will wait 20-40us for DHT22's response.

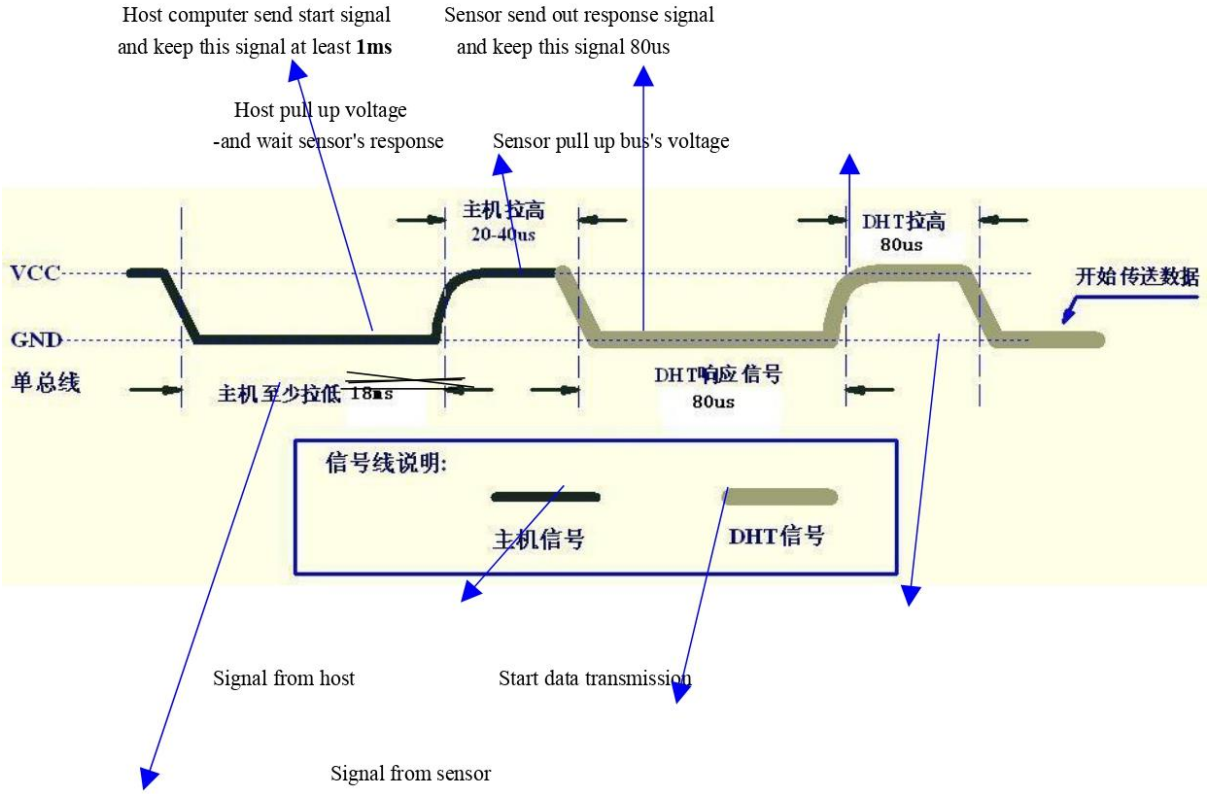
Check bellow picture for step 1:

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors



Single-bus signal

Step 2: DHT22 send response signal to MCU

When DHT22 detect the start signal, DHT22 will send out low-voltage-level signal and this signal last 80us as response signal, then program of DHT22 transform data-bus's voltage level from low to high level and last 80us for DHT22's preparation to send data.

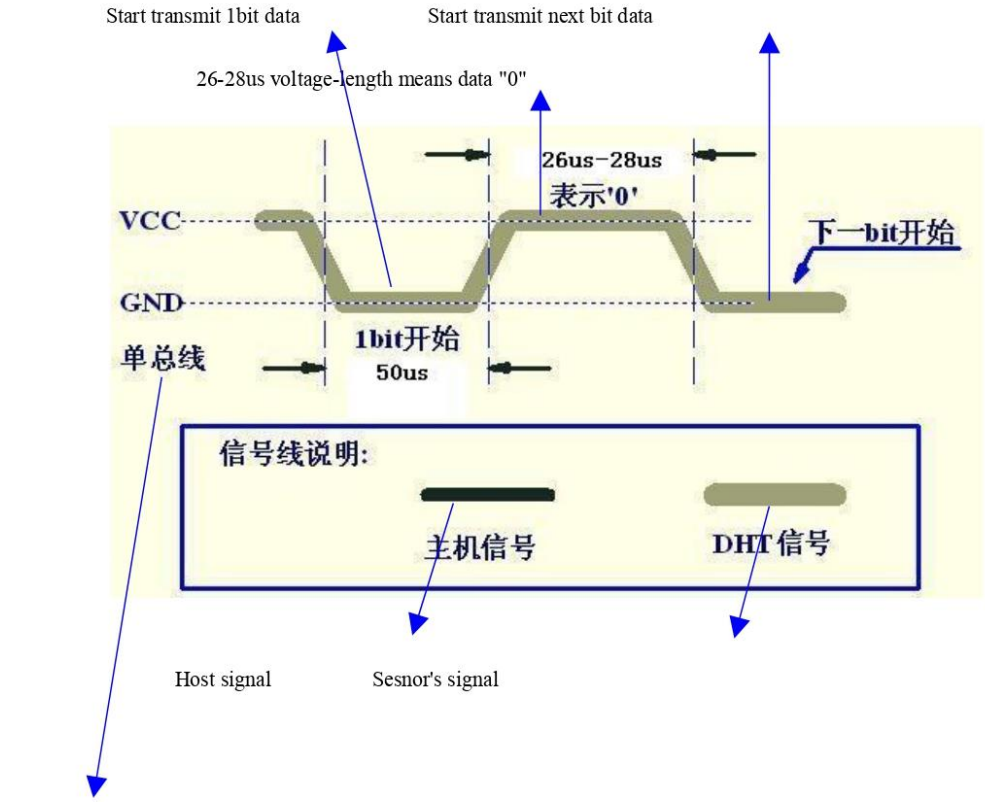
Check bellow picture for step 2:

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors



Step 3: DHT22 send data to MCU

When DHT22 is sending data to MCU, every bit's transmission begin with low-voltage-level that last 50us, the following high-voltage-level signal's length decide the bit is "1" or "0".

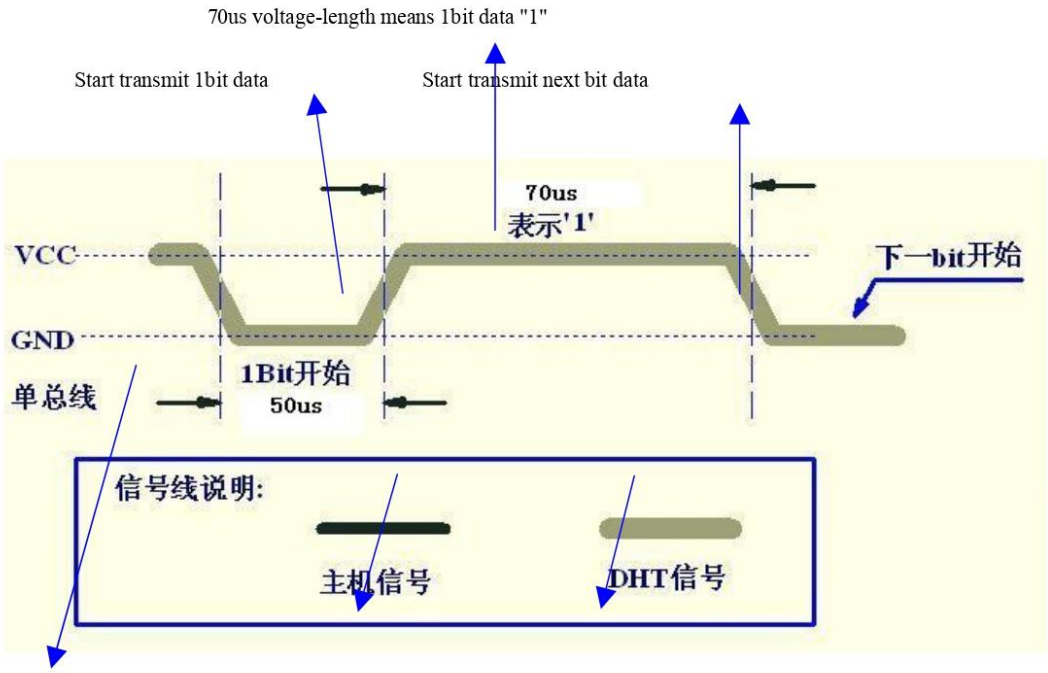
Check bellow picture for step 3:

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors



Host signal Sesnor's signal

Single-bus signal

If signal from DHT22 is always high-voltage-level, it means DHT22 is not working properly, please check the electrical connection status.

7. Electrical Characteristics:

Item	Condition	Min	Typical	Max	Unit
Power supply	DC	3.3	5	6	V
Current supply	Measuring	1		1.5	mA
	Stand-by	40	Null	50	uA
Collecting period	Second		2		Second

*Collecting period should be : >2 second.

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

Aosong Electronics Co.,Ltd

Your specialist in innovating humidity & temperature sensors

8. Attentions of application:

(1) Operating and storage conditions

We don't recommend the applying RH-range beyond the range stated in this specification. The DHT22 sensor can recover after working in non-normal operating condition to calibrated status, but will accelerate sensors' aging.

(2) Attentions to chemical materials

Vapor from chemical materials may interfere DHT22's sensitive-elements and debase DHT22's sensitivity.

(3) Disposal when (1) & (2) happens

Step one: Keep the DHT22 sensor at condition of Temperature 50~60Celsius, humidity <10%RH for 2 hours;

Step two: After step one, keep the DHT22 sensor at condition of Temperature 20~30Celsius, humidity >70%RH for 5 hours.

(4) Attention to temperature's affection

Relative humidity strongly depend on temperature, that is why we use temperature compensation technology to ensure accurate measurement of RH. But it's still be much better to keep the sensor at same temperature when sensing.

DHT22 should be mounted at the place as far as possible from parts that may cause change to temperature.

(5) Attentions to light

Long time exposure to strong light and ultraviolet may debase DHT22's performance.

(6) Attentions to connection wires

The connection wires' quality will effect communication's quality and distance, high quality shielding-wire is recommended.

(7) Other attentions

* Welding temperature should be bellow 260Celsius.
* Avoid using the sensor under dew condition.
* Don't use this product in safety or emergency stop devices or any other occasion that failure of DHT22 may cause personal injury.

Thomas Liu (Business Manager)

Email: thomasliu198518@yahoo.com.cn

16.15. Anexo 15: Datasheet sensor SICK

PBT-RB010SG1SSNAMA0Z

PBT

SENSORES DE PRESIÓN

SICK
Sensor Intelligence.

PBT-RB010SG1SSNAMA0Z | PBT

SENSORES DE PRESIÓN

Imagen aproximada

Información sobre pedidos

Tipo	N.º de artículo
PBT-RB010SG1SSNAMA0Z	6038615

Otros modelos del dispositivo y accesorios ➜ www.sick.com/PBT

Datos técnicos detallados

Características

Medio	Líquido, gaseoso
Tipo de presión	Presión relativa
Margen de medida	0 bar ... 10 bar
Temperatura de proceso	0 °C ... +80 °C
Salida de señal	4 mA ... 20 mA, 2 hilos
Particularidades	Sin

Mecánica/Electrónica

Conexión de proceso	G ¼ A según DIN 3852-E
Partes en contacto con el medio	Conexión de presión: acero inoxidable 316L Sensor de presión: acero inoxidable 316L (de 0 bar a 10 bar rel. acero inoxidable 13-8 PH)
Líquido interno de transmisión	Aceite sintético (solo para rangos de medición < de 0 bar a 10 bar y ≤ de 0 bar abs. a 25 bar abs.)
Orificio del canal	, Standard
Material de la carcasa	Acero inoxidable 316L
Tipo de conexión	1 conector cilíndrico M12 de 4 polos, IP67
Tensión de alimentación	8 V CC ... 30 V CC con señal de salida 4 mA ... 20 mA y 0 V ... 5 V 14 V CC ... 30 V CC con señal de salida 0 V ... 10 V 8 V DC ... 30 V DC [1)]
Consumo de corriente	Corriente de señal (máx. 25 mA) para salida de corriente Máx. 8 mA para señal de salida de tensión
Seguridad eléctrica	Protección contra sobretensión: 32 V CC, 36 V CC con una corriente de 4 mA ... 20 mA Resistencia a cortocircuitos: Q_A contra M Protección contra polarización inversa: L^+ contra M Clase de protección III
Tensión de aislamiento	500 V DC
Conformidad CE	Directiva de equipos bajo tensión: 2014/68/EU, Directiva CEM: 2014/30/CE, EN 61326-2-3
Peso del sensor	Aprox. 80 g
Junta	NBR

[1)] La alimentación del convertidor de medición de presión debe realizarse mediante un circuito con energía limitada según la sección 9.3 de la norma UL/EN/IEC 601010-1 o LPS según UL/EN/IEC 60950-1 o clase 2 según UL 1310/UL1585 (NEC o CEC). La fuente de alimentación debe ser apta para el servicio por encima de los 2.000 m, en caso de que el convertidor de medición de presión se utilice a partir de esta altura.

Grado de protección	IP67
Clase de protección III	✓
Condiciones de referencia	Condiciones de referencia: según IEC 61298-1
MTTF	815 años

[1] La alimentación del convertidor de medición de presión debe realizarse mediante un circuito con energía limitada según la sección 9.3 de la norma UL/EN/IEC 601010-1 o LPS según UL/EN/IEC 60950-1 o clase 2 según UL 1310/UL1585 (NEC o CEC). La fuente de alimentación debe ser apta para el servicio por encima de los 2.000 m, en caso de que el convertidor de medición de presión se utilice a partir de esta altura.

Rendimiento

No linealidad	≤ ± 0,5 %, (Best Fit Straight Line, BFSL) according to IEC 61298-2
Exactitud	≤ ± 1 % del margen
Precisión de compensación de la señal cero	≤ 0,5% del margen típ., ≤ 0,8% del margen máx. (con no linealidad 0,5%)
Histéresis	≤ 0,16 % del margen
No repetibilidad	≤ 0,1 % del margen
Tiempo de ataque	< 4 ms
Ruidos de señales	≤ 0,3 % del margen
Deriva a largo plazo/estabilidad por año	≤ 0,1 % del margen según IEC 61298-2
Rango de temperaturas de medición	0 °C ... +80 °C
Vida útil	Mín. 100 millones de alternancias de carga

Datos de ambiente

Temperatura ambiente	0 °C ... +80 °C
Temperatura de almacenamiento	–40 °C ... +70 °C
Humedad relativa del aire	45 % ... 75 %
Efecto de choque	500 g según IEC 60068-2-27 (impacto mecánico)
Carga de vibraciones	10 g según IEC 60068-2-6 (vibración con resonancia) 20 g opcional

Clasificaciones

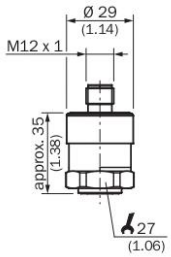
ECl@ss 5.0	27200614
ECl@ss 5.1.4	27200614
ECl@ss 6.0	27200614
ECl@ss 6.2	27200614
ECl@ss 7.0	27200614
ECl@ss 8.0	27200614
ECl@ss 8.1	27200614
ECl@ss 9.0	27200614
ETIM 5.0	EC002476
ETIM 6.0	EC002476
UNSPSC 16.0901	41112410

PBT-RB010SG1SSNAMA0Z | PBT

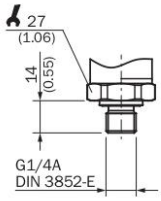
SENSORES DE PRESIÓN

Esquema de dimensiones (Medidas en mm)

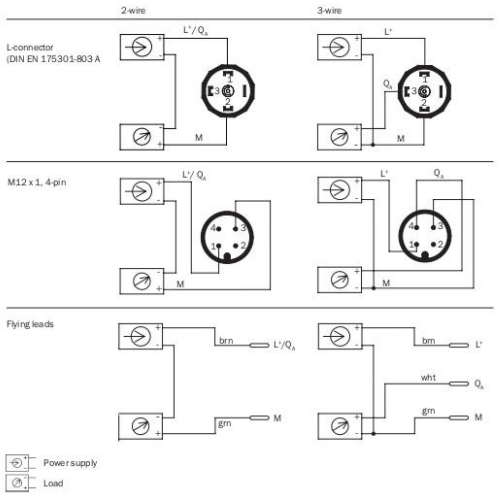
Carcasa con conector cilíndrico M12 x 1, IP67



G ¼ A DIN 3852-E



Tipo de conexión



Accesorios recomendados

Otros modelos del dispositivo y accesorios → www.sick.com/PBT

	Descripción breve	Tipo	N.º de artículo
Escuadra y placas de fijación			
	Escuadra de fijación para montaje sencillo y estable en la pared, para sensores de presión con hexágono de 27 mm, Aluminio	BEF-FL-ALUPBS-HLDR	5322501
Conectores y cables			
	Cabezal A: Conector hembra, M12, 4 polos, acodado Cabezal B: Extremo de cable suelto Cable: PVC, sin apantallar, 5 m	DOL-1204-W05MD	6020399
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 2 m	YF2A14-020UB3XLEAX	2095607
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 2 m	YF2A14-020VB3XLEAX	2096234
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 5 m	YF2A14-050VB3XLEAX	2096235
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 10 m	YF2A14-100UB3XLEAX	2095609
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 10 m	YF2A14-100VB3XLEAX	2096236
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 15 m	YF2A14-150UB3XLEAX	2095610
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 15 m	YF2A14-150VB3XLEAX	2096237
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 20 m	YF2A14-200UB3XLEAX	2095611
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 20 m	YF2A14-200VB3XLEAX	2096238
	Cabezal A: Conector hembra, M12, 4 polos, recto, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 25 m	YF2A14-250UB3XLEAX	2095615
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 2 m	YG2A14-020UB3XLEAX	2095766
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 2 m	YG2A14-020VB3XLEAX	2095895
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 5 m	YG2A14-050UB3XLEAX	2095767

PBT-RB010SG1SSNAMA0Z | PBT

SENSORES DE PRESIÓN

	Descripción breve	Tipo	N.º de artículo
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 5 m	YG2A14-050VB3XLEAX	2095897
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 10 m	YG2A14-100UB3XLEAX	2095768
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 10 m	YG2A14-100VB3XLEAX	2095898
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 15 m	YG2A14-150UB3XLEAX	2095769
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 15 m	YG2A14-150VB3XLEAX	2096213
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 20 m	YG2A14-200UB3XLEAX	2095770
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PVC, sin apantallar, 20 m	YG2A14-200VB3XLEAX	2096214
	Cabezal A: Conector hembra, M12, 4 polos, acodado, Con codificación A Cabezal B: Extremo de cable suelto Cable: Cable sensor/actuador, PUR sin halógenos, sin apantallar, 25 m	YG2A14-250UB3XLEAX	2095771

LO MÁS DESTACADO DE SICK

SICK es uno de los fabricantes líderes de sensores y soluciones de sensores inteligentes para aplicaciones industriales. Nuestro exclusivo catálogo de productos y servicios constituye la base perfecta para el control seguro y eficaz de procesos, para la protección de personas y para la prevención de accidentes y de daños medioambientales.

Nuestra amplia experiencia multidisciplinar nos permite conocer sus necesidades y procesos para ofrecer a nuestros clientes exactamente la clase de sensores inteligentes que necesitan. Contamos con centros de aplicación en Europa, Asia y Norteamérica, donde probamos y optimizamos las soluciones de sistemas específicas del cliente. Todo ello nos convierte en el proveedor y socio en el desarrollo de confianza que somos.

SICK LifeTime Services, nuestra completa oferta de servicios, garantiza la asistencia durante toda la vida útil de su maquinaria para que obtenga la máxima seguridad y productividad.

Para nosotros, esto es "Sensor Intelligence".

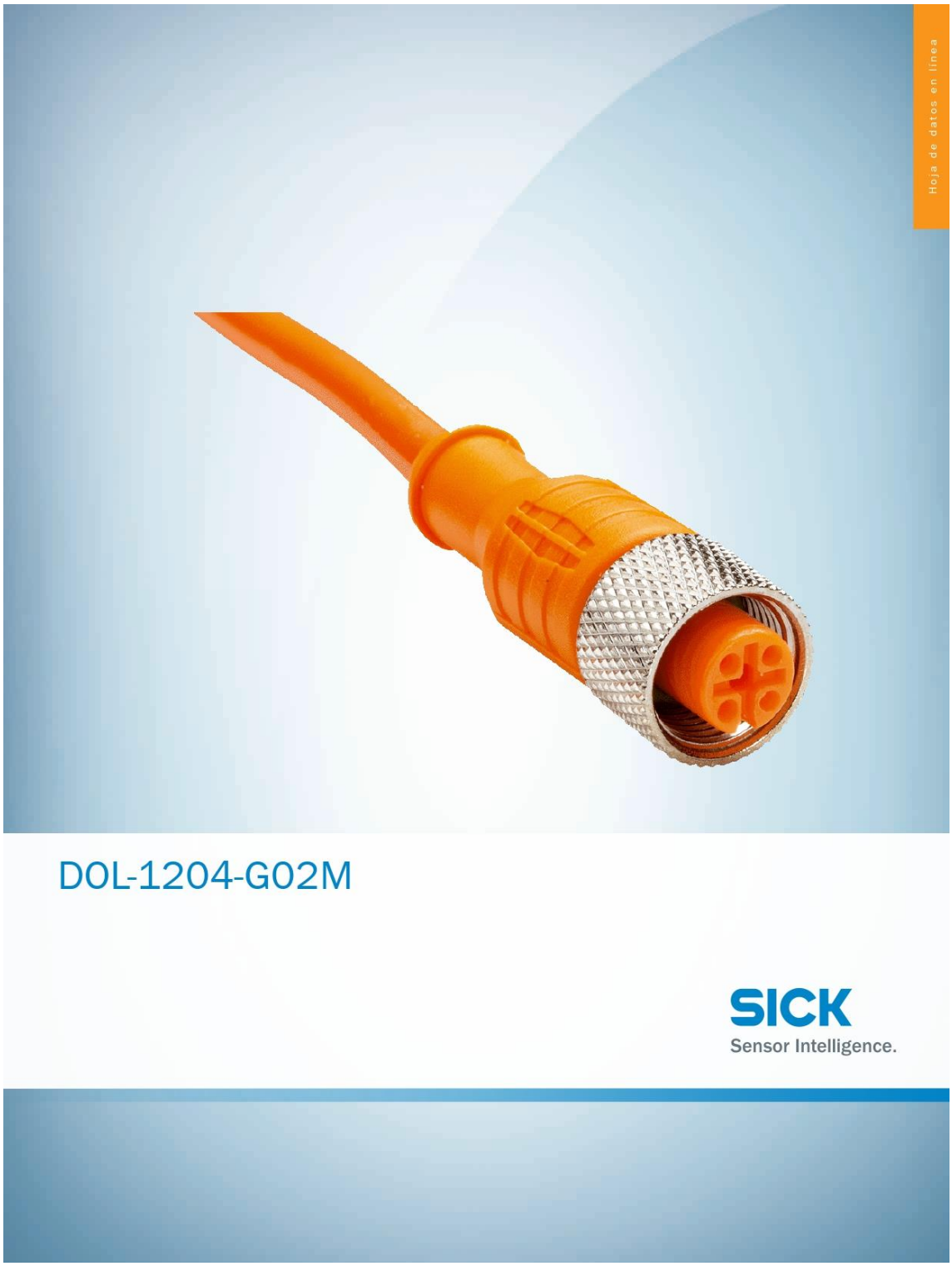
CERCA DE USTED EN CUALQUIER LUGAR DEL MUNDO:

Encontrará información detallada sobre todas las sedes y personas de contacto en nuestra página web: → www.sick.com

SICK AG | Waldkirch | Germany | www.sick.com

SICK
Sensor Intelligence.

16.16. Anexo 16: Datasheet cable para sensor SICK



DOL-1204-G02M

Imagen aproximada

Información sobre pedidos

Tipo	N.º de artículo
DOL-1204-G02M	6009382

Otros modelos del dispositivo y accesorios → www.sick.com/

Datos técnicos detallados

Datos técnicos

Grupos de accesorios	Conectores y cables
Gama de accesorios	Cables de conexión
Tipo de conexión cabezal A	Conector hembra, M12, 4 polos, recto
Tipo de conexión cabezal B	Extremo de cable suelto
Tipo de conexión	Extremo de cable suelto
Material del conector	TPU
Color, conector de enchufe	Naranja
Material de la tuerca de bloqueo	CuZn, niquelado
Par de apriete	0,5 Nm
Cable	2 m, de 4 hilos, PVC
Material de la envoltura	PVC
Color, envoltura	Naranja
Diámetro del cable	5 mm
Sección del conductor	0,25 mm²
Apantallamiento	Sin apantallar
Radio de curvatura	
En movimiento	> 10 veces el diámetro del cable
Con tendido fijo	> 5 veces el diámetro del cable en tendido fijo
Tensión asignada	≤ 250 V
Carga de corriente	4 A
Aplicación	Industria química
Grado de protección	IP67
Temperatura ambiente durante el servicio	
En movimiento	+5 °C ... +80 °C
Con tendido fijo	–40 °C ... +80 °C
Cabezal	–25 °C ... +80 °C

Clasificaciones

ECl@ss 5.0	19030312
ECl@ss 5.1.4	19030312
ECl@ss 6.0	27060304

ECl@ss 6.2	27060304
ECl@ss 7.0	27060304
ECl@ss 8.0	27060304
ECl@ss 8.1	27060304
ECl@ss 9.0	27060304
ETIM 5.0	EC000830
ETIM 6.0	EC000830
UNSPSC 16.0901	26121604

Hoja de datos en línea

LO MÁS DESTACADO DE SICK

SICK es uno de los fabricantes líderes de sensores y soluciones de sensores inteligentes para aplicaciones industriales. Nuestro exclusivo catálogo de productos y servicios constituye la base perfecta para el control seguro y eficaz de procesos, para la protección de personas y para la prevención de accidentes y de daños medioambientales.

Nuestra amplia experiencia multidisciplinar nos permite conocer sus necesidades y procesos para ofrecer a nuestros clientes exactamente la clase de sensores inteligentes que necesitan. Contamos con centros de aplicación en Europa, Asia y Norteamérica, donde probamos y optimizamos las soluciones de sistemas específicas del cliente. Todo ello nos convierte en el proveedor y socio en el desarrollo de confianza que somos.

SICK LifeTime Services, nuestra completa oferta de servicios, garantiza la asistencia durante toda la vida útil de su maquinaria para que obtenga la máxima seguridad y productividad.

Para nosotros, esto es "Sensor Intelligence".

CERCA DE USTED EN CUALQUIER LUGAR DEL MUNDO:

Encontrará información detallada sobre todas las sedes y personas de contacto en nuestra página web: → www.sick.com

SICK AG | Waldkirch | Germany | www.sick.com

SICK
Sensor Intelligence.

16.17. Anexo 17: Datasheet DS18B20

PRELIMINARY

DALLAS SEMICONDUCTOR

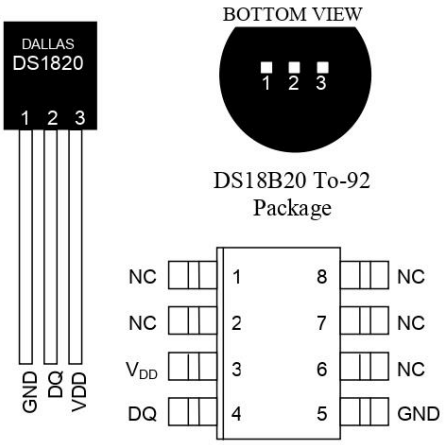
DS18B20
Programmable Resolution 1-Wire® Digital Thermometer

www.dalsemi.com

FEATURES

- Unique 1-Wire interface requires only one port pin for communication
- Multidrop capability simplifies distributed temperature sensing applications
- Requires no external components
- Can be powered from data line. Power supply range is 3.0V to 5.5V
- Zero standby power required
- Measures temperatures from -55°C to +125°C. Fahrenheit equivalent is -67°F to +257°F
- ±0.5°C accuracy from -10°C to +85°C
- Thermometer resolution is programmable from 9 to 12 bits
- Converts 12-bit temperature to digital word in 750 ms (max.)
- User-definable, nonvolatile temperature alarm settings
- Alarm search command identifies and addresses devices whose temperature is outside of programmed limits (temperature alarm condition)
- Applications include thermostatic controls, industrial systems, consumer products, thermometers, or any thermally sensitive system

PIN ASSIGNMENT



DS18B20 To-92 Package

DS18B20Z
8-Pin SOIC (150 mil)

PIN DESCRIPTION

GND - Ground
DQ - Data In/Out
V_{DD} - Power Supply Voltage
NC - No Connect

DESCRIPTION

The DS18B20 Digital Thermometer provides 9 to 12-bit (configurable) temperature readings which indicate the temperature of the device.

Information is sent to/from the DS18B20 over a 1-Wire interface, so that only one wire (and ground) needs to be connected from a central microprocessor to a DS18B20. Power for reading, writing, and performing temperature conversions can be derived from the data line itself with no need for an external power source.

Because each DS18B20 contains a unique silicon serial number, multiple DS18B20s can exist on the same 1-Wire bus. This allows for placing temperature sensors in many different places. Applications where this feature is useful include HVAC environmental controls, sensing temperatures inside buildings, equipment or machinery, and process monitoring and control.

DETAILED PIN DESCRIPTION Table 1

PIN 8PIN SOIC	PIN TO92	SYMBOL	DESCRIPTION
5	1	GND	Ground.
4	2	DQ	**Data Input/Output pin.** For 1-Wire operation: Open drain. (See "Parasite Power" section.)
3	3	V_{DD}	**Optional V_{DD} pin.** See "Parasite Power" section for details of connection. V_{DD} must be grounded for operation in parasite power mode.

DS18B20Z (8-pin SOIC): All pins not specified in this table are not to be connected.

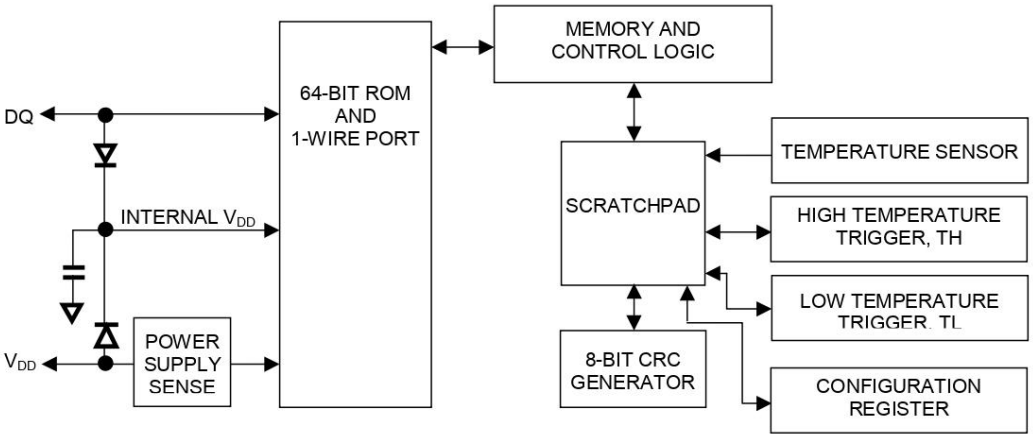
OVERVIEW

The block diagram of Figure 1 shows the major components of the DS18B20. The DS18B20 has four main data components: 1) 64-bit lasered ROM, 2) temperature sensor, 3) nonvolatile temperature alarm triggers TH and TL, and 4) a configuration register. The device derives its power from the 1-Wire communication line by storing energy on an internal capacitor during periods of time when the signal line is high and continues to operate off this power source during the low times of the 1-Wire line until it returns high to replenish the parasite (capacitor) supply. As an alternative, the DS18B20 may also be powered from an external 3 volt - 5.5 volt supply.

Communication to the DS18B20 is via a 1-Wire port. With the 1-Wire port, the memory and control functions will not be available before the ROM function protocol has been established. The master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. These commands operate on the 64-bit lasered ROM portion of each device and can single out a specific device if many are present on the 1-Wire line as well as indicate to the bus master how many and what types of devices are present. After a ROM function sequence has been successfully executed, the memory and control functions are accessible and the master may then provide any one of the six memory and control function commands.

One control function command instructs the DS18B20 to perform a temperature measurement. The result of this measurement will be placed in the DS18B20's scratch-pad memory, and may be read by issuing a memory function command which reads the contents of the scratchpad memory. The temperature alarm triggers TH and TL consist of 1 byte EEPROM each. If the alarm search command is not applied to the DS18B20, these registers may be used as general purpose user memory. The scratchpad also contains a configuration byte to set the desired resolution of the temperature to digital conversion. Writing TH, TL, and the configuration byte is done using a memory function command. Read access to these registers is through the scratchpad. All data is read and written least significant bit first.

DS18B20 BLOCK DIAGRAM Figure 1



PARASITE POWER

The block diagram (Figure 1) shows the parasite-powered circuitry. This circuitry "steals" power whenever the DQ or V_{DD} pins are high. DQ will provide sufficient power as long as the specified timing and voltage requirements are met (see the section titled "1-Wire Bus System"). The advantages of parasite power are twofold: 1) by parasiting off this pin, no local power source is needed for remote sensing of temperature, and 2) the ROM may be read in absence of normal power.

In order for the DS18B20 to be able to perform accurate temperature conversions, sufficient power must be provided over the DQ line when a temperature conversion is taking place. Since the operating current of the DS18B20 is up to 1.5 mA, the DQ line will not have sufficient drive due to the 5k pullup resistor. This problem is particularly acute if several DS18B20s are on the same DQ and attempting to convert simultaneously.

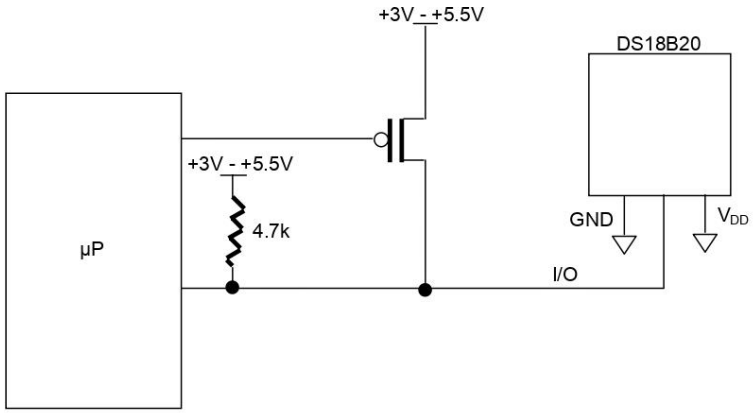
There are two ways to assure that the DS18B20 has sufficient supply current during its active conversion cycle. The first is to provide a strong pullup on the DQ line whenever temperature conversions or copies to the E^2 memory are taking place. This may be accomplished by using a MOSFET to pull the DQ line directly to the power supply as shown in Figure 2. The DQ line must be switched over to the strong pull-up within 10 µs maximum after issuing any protocol that involves copying to the E^2 memory or initiates temperature conversions. When using the parasite power mode, the V_{DD} pin must be tied to ground.

Another method of supplying current to the DS18B20 is through the use of an external power supply tied to the V_{DD} pin, as shown in Figure 3. The advantage to this is that the strong pullup is not required on the DQ line, and the bus master need not be tied up holding that line high during temperature conversions. This allows other data traffic on the 1-Wire bus during the conversion time. In addition, any number of DS18B20s may be placed on the 1-Wire bus, and if they all use external power, they may all simultaneously perform temperature conversions by issuing the Skip ROM command and then issuing the Convert T command. Note that as long as the external power supply is active, the GND pin may not be floating.

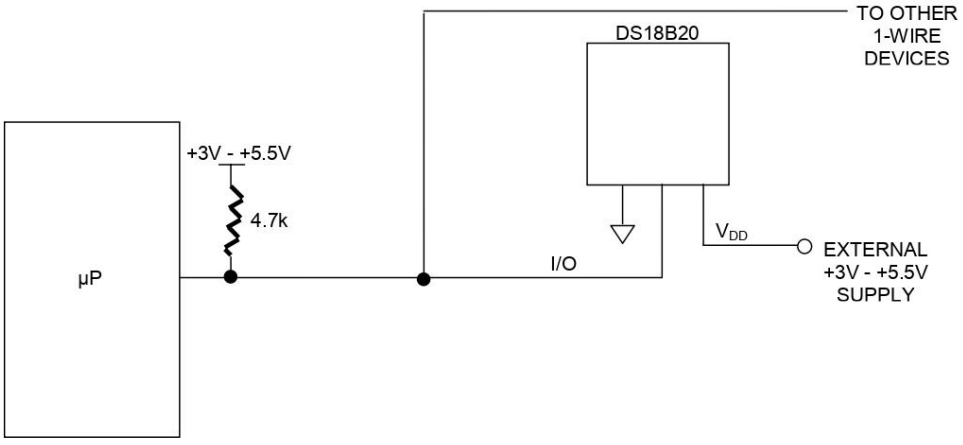
The use of parasite power is not recommended above 100°C, since it may not be able to sustain communications given the higher leakage currents the DS18B20 exhibits at these temperatures. For applications in which such temperatures are likely, it is strongly recommended that V_{DD} be applied to the DS18B20.

For situations where the bus master does not know whether the DS18B20s on the bus are parasite powered or supplied with external V_{DD}, a provision is made in the DS18B20 to signal the power supply scheme used. The bus master can determine if any DS18B20s are on the bus which require the strong pullup by sending a Skip ROM protocol, then issuing the read power supply command. After this command is issued, the master then issues read time slots. The DS18B20 will send back "0" on the 1-Wire bus if it is parasite powered; it will send back a "1" if it is powered from the V_{DD} pin. If the master receives a "0," it knows that it must supply the strong pullup on the DQ line during temperature conversions. See "Memory Command Functions" section for more detail on this command protocol.

STRONG PULLUP FOR SUPPLYING DS18B20 DURING TEMPERATURE CONVERSION Figure 2



USING V_{DD} TO SUPPLY TEMPERATURE CONVERSION CURRENT Figure 3



OPERATION - MEASURING TEMPERATURE

The core functionality of the DS18B20 is its direct-to-digital temperature sensor. The resolution of the DS18B20 is configurable (9, 10, 11, or 12 bits), with 12-bit readings the factory default state. This equates to a temperature resolution of 0.5°C, 0.25°C, 0.125°C, or 0.0625°C. Following the issuance of the Convert T [44h] command, a temperature conversion is performed and the thermal data is stored in the scratchpad memory in a 16-bit, sign-extended two's complement format. The temperature information can be retrieved over the 1-Wire interface by issuing a Read Scratchpad [BEh] command once the conversion has been performed. The data is transferred over the 1-Wire bus, LSB first. The MSB of the temperature register contains the "sign" (S) bit, denoting whether the temperature is positive or negative.

Table 2 describes the exact relationship of output data to measured temperature. The table assumes 12-bit resolution. If the DS18B20 is configured for a lower resolution, insignificant bits will contain zeros. For Fahrenheit usage, a lookup table or conversion routine must be used.

Temperature/Data Relationships Table 2

2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}	LSB

MSb (unit = °C) LSb

S	S	S	S	S	2^6	2^5	2^4	MSB

TEMPERATURE	DIGITAL OUTPUT (Binary)	DIGITAL OUTPUT (Hex)
+125°C	0000 0111 1101 0000	07D0h
+85°C	0000 0101 0101 0000	0550h*
+25.0625°C	0000 0001 1001 0001	0191h
+10.125°C	0000 0000 1010 0010	00A2h
+0.5°C	0000 0000 0000 1000	0008h
0°C	0000 0000 0000 0000	0000h
-0.5°C	1111 1111 1111 1000	FFF8h
-10.125°C	1111 1111 0101 1110	FF5Eh
-25.0625°C	1111 1110 0110 1111	FF6Fh
-55°C	1111 1100 1001 0000	FC90h

*The power on reset register value is +85°C.

OPERATION - ALARM SIGNALING

After the DS18B20 has performed a temperature conversion, the temperature value is compared to the trigger values stored in TH and TL. Since these registers are 8-bit only, bits 9-12 are ignored for comparison. The most significant bit of TH or TL directly corresponds to the sign bit of the 16-bit temperature register. If the result of a temperature measurement is higher than TH or lower than TL, an alarm flag inside the device is set. This flag is updated with every temperature measurement. As long as the alarm flag is set, the DS18B20 will respond to the alarm search command. This allows many DS18B20s to be connected in parallel doing simultaneous temperature measurements. If somewhere the temperature exceeds the limits, the alarming device(s) can be identified and read immediately without having to read non-alarming devices.

64-BIT LASERED ROM

Each DS18B20 contains a unique ROM code that is 64-bits long. The first 8 bits are a 1-Wire family code (DS18B20 code is 28h). The next 48 bits are a unique serial number. The last 8 bits are a CRC of the first 56 bits. (See Figure 4.) The 64-bit ROM and ROM Function Control section allow the DS18B20 to operate as a 1-Wire device and follow the 1-Wire protocol detailed in the section "1-Wire Bus System." The functions required to control sections of the DS18B20 are not accessible until the ROM function protocol has been satisfied. This protocol is described in the ROM function protocol flowchart (Figure 5). The 1-Wire bus master must first provide one of five ROM function commands: 1) Read ROM, 2) Match ROM, 3) Search ROM, 4) Skip ROM, or 5) Alarm Search. After a ROM function sequence has been successfully executed, the functions specific to the DS18B20 are accessible and the bus master may then provide one of the six memory and control function commands.

CRC GENERATION

The DS18B20 has an 8-bit CRC stored in the most significant byte of the 64-bit ROM. The bus master can compute a CRC value from the first 56-bits of the 64-bit ROM and compare it to the value stored within the DS18B20 to determine if the ROM data has been received error-free by the bus master. The equivalent polynomial function of this CRC is:

$$CRC = X^8 + X^5 + X^4 + 1$$

The DS18B20 also generates an 8-bit CRC value using the same polynomial function shown above and provides this value to the bus master to validate the transfer of data bytes. In each case where a CRC is used for data transfer validation, the bus master must calculate a CRC value using the polynomial function given above and compare the calculated value to either the 8-bit CRC value stored in the 64-bit ROM portion of the DS18B20 (for ROM reads) or the 8-bit CRC value computed within the DS18B20 (which is read as a ninth byte when the scratchpad is read). The comparison of CRC values and decision to continue with an operation are determined entirely by the bus master. There is no circuitry inside the DS18B20 that prevents a command sequence from proceeding if the CRC stored in or calculated by the DS18B20 does not match the value generated by the bus master.

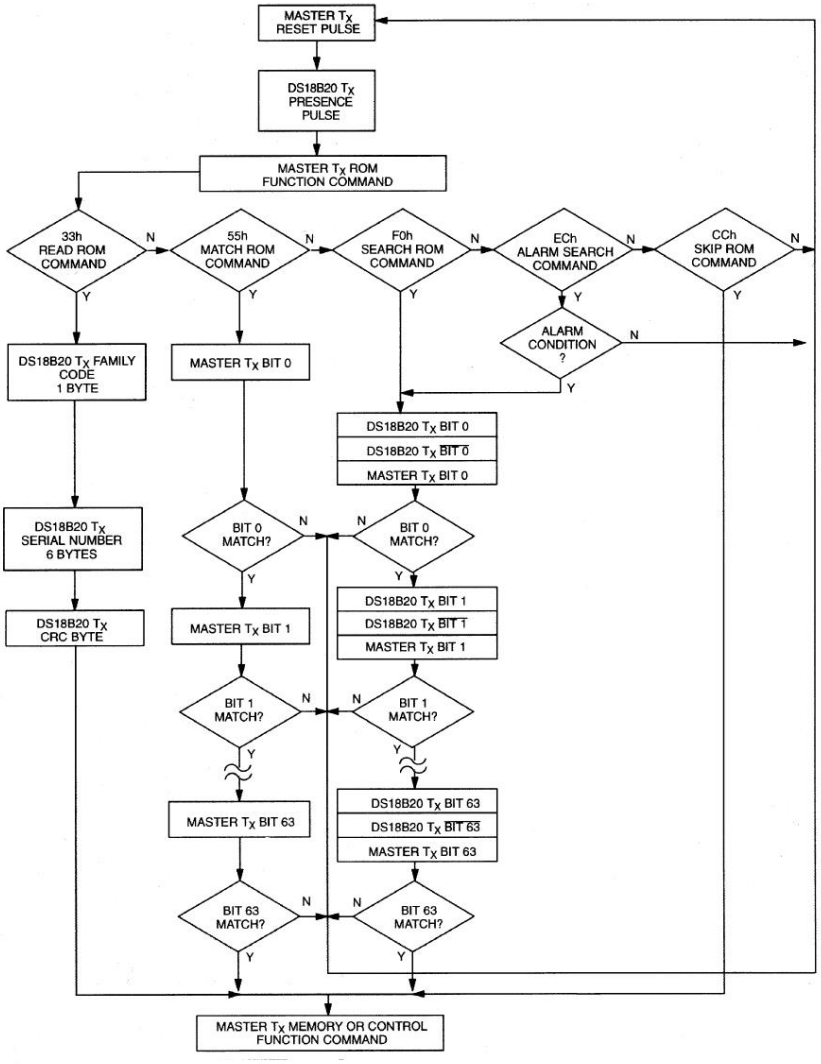
The 1-Wire CRC can be generated using a polynomial generator consisting of a shift register and XOR gates as shown in Figure 6. Additional information about the Dallas 1-Wire Cyclic Redundancy Check is available in Application Note 27 entitled "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Products."

The shift register bits are initialized to 0. Then starting with the least significant bit of the family code, 1 bit at a time is shifted in. After the 8th bit of the family code has been entered, then the serial number is entered. After the 48th bit of the serial number has been entered, the shift register contains the CRC value. Shifting in the 8 bits of CRC should return the shift register to all 0s.

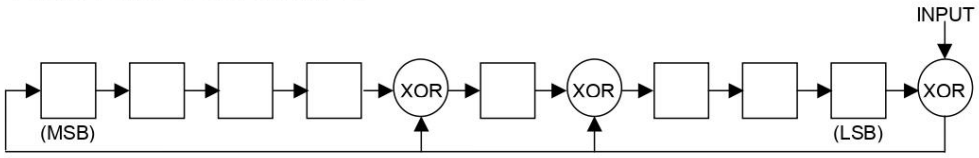
64-BIT LASERED ROM Figure 4

8-BIT CRC CODE		48-BIT SERIAL NUMBER		8-BIT FAMILY CODE (28h)	
MSB	LSB	MSB	LSB	MSB	LSB

ROM FUNCTIONS FLOW CHART Figure 5



1-WIRE CRC CODE Figure 6



MEMORY

The DS18B20's memory is organized as shown in Figure 8. The memory consists of a scratchpad RAM and a nonvolatile, electrically erasable (E^2) RAM, which stores the high and low temperature triggers TH and TL, and the configuration register. The scratchpad helps insure data integrity when communicating over the 1-Wire bus. Data is first written to the scratchpad using the Write Scratchpad [4Eh] command. It can then be verified by using the Read Scratchpad [BEh] command. After the data has been verified, a Copy Scratchpad [48h] command will transfer the data to the nonvolatile (E^2) RAM. This process insures data integrity when modifying memory. The DS18B20 EEPROM is rated for a minimum of 50,000 writes and 10 years data retention at T = +55°C.

The scratchpad is organized as eight bytes of memory. The first 2 bytes contain the LSB and the MSB of the measured temperature information, respectively. The third and fourth bytes are volatile copies of TH and TL and are refreshed with every power-on reset. The fifth byte is a volatile copy of the configuration register and is refreshed with every power-on reset. The configuration register will be explained in more detail later in this section of the datasheet. The sixth, seventh, and eighth bytes are used for internal computations, and thus will not read out any predictable pattern.

It is imperative that one writes TH, TL, and config in succession; i.e. a write is not valid if one writes only to TH and TL, for example, and then issues a reset. If any of these bytes must be written, all three must be written before a reset is issued.

There is a ninth byte which may be read with a Read Scratchpad [BEh] command. This byte contains a cyclic redundancy check (CRC) byte which is the CRC over all of the eight previous bytes. This CRC is implemented in the fashion described in the section titled "CRC Generation".

Configuration Register

The fifth byte of the scratchpad memory is the configuration register.

It contains information which will be used by the device to determine the resolution of the temperature to digital conversion. The bits are organized as shown in Figure 7.

DS18B20 CONFIGURATION REGISTER Figure 7

0	R1	R0	1	1	1	1	1

MSb LSb

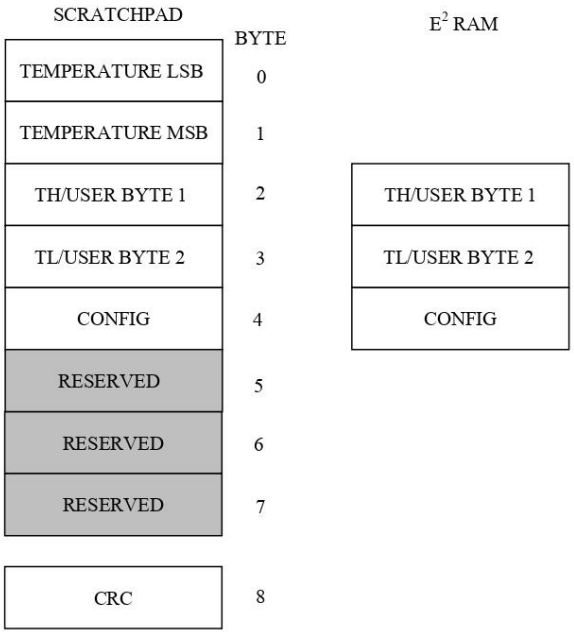
Bits 0-4 are don't cares on a write but will always read out "1".
Bit 7 is a don't care on a write but will always read out "0".

R0, R1: Thermometer resolution bits. Table 3 below defines the resolution of the digital thermometer, based on the settings of these 2 bits. There is a direct tradeoff between resolution and conversion time, as depicted in the AC Electrical Characteristics. The factory default of these EEPROM bits is R0=1 and R1=1 (12-bit conversions).

Thermometer Resolution Configuration Table 3

R1	R0	Thermometer Resolution	Max Conversion Time
0	0	9 bit	93.75 ms ($t_{conv}/8$)
0	1	10 bit	187.5 ms ($t_{conv}/4$)
1	0	11 bit	375 ms ($t_{conv}/2$)
1	1	12 bit	750 ms (t_{conv})

DS18B20 MEMORY MAP Figure 8



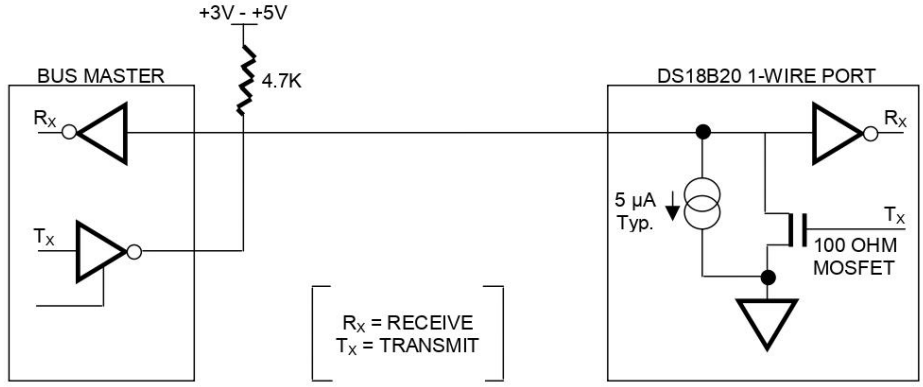
1-WIRE BUS SYSTEM

The 1-Wire bus is a system which has a single bus master and one or more slaves. The DS18B20 behaves as a slave. The discussion of this bus system is broken down into three topics: hardware configuration, transaction sequence, and 1-Wire signaling (signal types and timing).

HARDWARE CONFIGURATION

The 1-Wire bus has only a single line by definition; it is important that each device on the bus be able to drive it at the appropriate time. To facilitate this, each device attached to the 1-Wire bus must have open drain or 3-state outputs. The 1-Wire port of the DS18B20 (DQ pin) is open drain with an internal circuit equivalent to that shown in Figure 9. A multidrop bus consists of a 1-Wire bus with multiple slaves attached. The 1-Wire bus requires a pullup resistor of approximately 5 kΩ.

HARDWARE CONFIGURATION Figure 9



The idle state for the 1-Wire bus is high. If for any reason a transaction needs to be suspended, the bus MUST be left in the idle state if the transaction is to resume. Infinite recovery time can occur between bits so long as the 1-Wire bus is in the inactive (high) state during the recovery period. If this does not occur and the bus is left low for more than 480 µs, all components on the bus will be reset.

TRANSACTION SEQUENCE

The protocol for accessing the DS18B20 via the 1-Wire port is as follows:

- Initialization
- ROM Function Command
- Memory Function Command
- Transaction/Data

INITIALIZATION

All transactions on the 1-Wire bus begin with an initialization sequence. The initialization sequence consists of a reset pulse transmitted by the bus master followed by presence pulse(s) transmitted by the slave(s).

The presence pulse lets the bus master know that the DS18B20 is on the bus and is ready to operate. For more details, see the "1-Wire Signaling" section.

ROM FUNCTION COMMANDS

Once the bus master has detected a presence, it can issue one of the five ROM function commands. All ROM function commands are 8 bits long. A list of these commands follows (refer to flowchart in Figure 5):

Read ROM [33h]

This command allows the bus master to read the DS18B20's 8-bit family code, unique 48-bit serial number, and 8-bit CRC. This command can only be used if there is a single DS18B20 on the bus. If more than one slave is present on the bus, a data collision will occur when all slaves try to transmit at the same time (open drain will produce a wired AND result).

Match ROM [55h]

The match ROM command, followed by a 64-bit ROM sequence, allows the bus master to address a specific DS18B20 on a multidrop bus. Only the DS18B20 that exactly matches the 64-bit ROM sequence will respond to the following memory function command. All slaves that do not match the 64-bit ROM sequence will wait for a reset pulse. This command can be used with a single or multiple devices on the bus.

Skip ROM [CCh]

This command can save time in a single drop bus system by allowing the bus master to access the memory functions without providing the 64-bit ROM code. If more than one slave is present on the bus and a Read command is issued following the Skip ROM command, data collision will occur on the bus as multiple slaves transmit simultaneously (open drain pulldowns will produce a wired AND result).

Search ROM [F0h]

When a system is initially brought up, the bus master might not know the number of devices on the 1-Wire bus or their 64-bit ROM codes. The search ROM command allows the bus master to use a process of elimination to identify the 64-bit ROM codes of all slave devices on the bus.

Alarm Search [ECh]

The flowchart of this command is identical to the Search ROM command. However, the DS18B20 will respond to this command only if an alarm condition has been encountered at the last temperature measurement. An alarm condition is defined as a temperature higher than TH or lower than TL. The alarm condition remains set as long as the DS18B20 is powered up, or until another temperature measurement reveals a non-alarming value. For alarming, the trigger values stored in EEPROM are taken into account. If an alarm condition exists and the TH or TL settings are changed, another temperature conversion should be done to validate any alarm conditions.

Example of a ROM Search

The ROM search process is the repetition of a simple three-step routine: read a bit, read the complement of the bit, then write the desired value of that bit. The bus master performs this simple, three-step routine on each bit of the ROM. After one complete pass, the bus master knows the contents of the ROM in one device. The remaining number of devices and their ROM codes may be identified by additional passes.

The following example of the ROM search process assumes four different devices are connected to the same 1-Wire bus. The ROM data of the four devices is as shown:

ROM1	00110101...
ROM2	10101010...
ROM3	11110101...
ROM4	00010001...

The search process is as follows:

1. The bus master begins the initialization sequence by issuing a reset pulse. The slave devices respond by issuing simultaneous presence pulses.

2. The bus master will then issue the Search ROM command on the 1-Wire bus.

3. The bus master reads a bit from the 1-Wire bus. Each device will respond by placing the value of the first bit of their respective ROM data onto the 1-Wire bus. ROM1 and ROM4 will place a 0 onto the 1-Wire bus, i.e., pull it low. ROM2 and ROM3 will place a 1 onto the 1-Wire bus by allowing the line to stay high. The result is the logical AND of all devices on the line, therefore the bus master sees a 0. The bus master reads another bit. Since the Search ROM data command is being executed, all of the devices on the 1-Wire bus respond to this second read by placing the complement of the first bit of their respective ROM data onto the 1-Wire bus. ROM1 and ROM4 will place a 1 onto the 1-Wire, allowing the line to stay high. ROM2 and ROM3 will place a 0 onto the 1-Wire, thus it will be pulled low. The bus master again observes a 0 for the complement of the first ROM data bit. The bus master has determined that there are some devices on the 1-Wire bus that have a 0 in the first position and others that have a 1.

   The data obtained from the two reads of the three-step routine have the following interpretations:

00	There are still devices attached which have conflicting bits in this position.
01	All devices still coupled have a 0-bit in this bit position.
10	All devices still coupled have a 1-bit in this bit position.
11	There are no devices attached to the 1-Wire bus.

4. The bus master writes a 0. This deselects ROM2 and ROM3 for the remainder of this search pass, leaving only ROM1 and ROM4 connected to the 1-Wire bus.

5. The bus master performs two more reads and receives a 0-bit followed by a 1-bit. This indicates that all devices still coupled to the bus have 0s as their second ROM data bit.

6. The bus master then writes a 0 to keep both ROM1 and ROM4 coupled.

7. The bus master executes two reads and receives two 0-bits. This indicates that both 1-bits and 0-bits exist as the 3rd bit of the ROM data of the attached devices.

8. The bus master writes a 0-bit. This deselects ROM1, leaving ROM4 as the only device still connected.

9. The bus master reads the remainder of the ROM bits for ROM4 and continues to access the part if desired. This completes the first pass and uniquely identifies one part on the 1-Wire bus.

10. The bus master starts a new ROM search sequence by repeating steps 1 through 7.

11. The bus master writes a 1-bit. This decouples ROM4, leaving only ROM1 still coupled.

12. The bus master reads the remainder of the ROM bits for ROM1 and communicates to the underlying logic if desired. This completes the second ROM search pass, in which another of the ROMs was found.

13. The bus master starts a new ROM search by repeating steps 1 through 3.

14. The bus master writes a 1-bit. This deselects ROM1 and ROM4 for the remainder of this search pass, leaving only ROM2 and ROM3 coupled to the system.

15. The bus master executes two Read time slots and receives two 0s.

16. The bus master writes a 0-bit. This decouples ROM3 leaving only ROM2.

17. The bus master reads the remainder of the ROM bits for ROM2 and communicates to the underlying logic if desired. This completes the third ROM search pass, in which another of the ROMs was found.

18. The bus master starts a new ROM search by repeating steps 13 through 15.

19. The bus master writes a 1-bit. This decouples ROM2, leaving only ROM3.

20. The bus master reads the remainder of the ROM bits for ROM3 and communicates to the underlying logic if desired. This completes the fourth ROM search pass, in which another of the ROMs was found.

NOTE:

The bus master learns the unique ID number (ROM data pattern) of one 1-Wire device on each ROM Search operation. The time required to derive the part's unique ROM code is:

960 µs + (8 + 3 x 64) 61 µs = 13.16 ms

The bus master is therefore capable of identifying 75 different 1-Wire devices per second.

I/O SIGNALING

The DS18B20 requires strict protocols to insure data integrity. The protocol consists of several types of signaling on one line: reset pulse, presence pulse, write 0, write 1, read 0, and read 1. All of these signals, with the exception of the presence pulse, are initiated by the bus master.

The initialization sequence required to begin any communication with the DS18B20 is shown in Figure 11. A reset pulse followed by a presence pulse indicates the DS18B20 is ready to send or receive data given the correct ROM command and memory function command.

The bus master transmits (TX) a reset pulse (a low signal for a minimum of 480 µs). The bus master then releases the line and goes into a receive mode (RX). The 1-Wire bus is pulled to a high state via the 5k pullup resistor. After detecting the rising edge on the DQ pin, the DS18B20 waits 15-60 µs and then transmits the presence pulse (a low signal for 60-240 µs).

MEMORY COMMAND FUNCTIONS

The following command protocols are summarized in Table 4, and by the flowchart of Figure 10.

Write Scratchpad [4Eh]

This command writes to the scratchpad of the DS18B20, starting at the TH register. The next 3 bytes written will be saved in scratchpad memory at address locations 2 through 4. All 3 bytes must be written before a reset is issued.

Read Scratchpad [BEh]

This command reads the contents of the scratchpad. Reading will commence at byte 0 and will continue through the scratchpad until the ninth (byte 8, CRC) byte is read. If not all locations are to be read, the master may issue a reset to terminate reading at any time.

Copy Scratchpad [48h]

This command copies the scratchpad into the E^2 memory of the DS18B20, storing the temperature trigger bytes in nonvolatile memory. If the bus master issues read time slots following this command, the DS18B20 will output 0 on the bus as long as it is busy copying the scratchpad to E^2; it will return a 1 when the copy process is complete. If parasite-powered, the bus master has to enable a strong pullup for at least 10 ms immediately after issuing this command. The DS18B20 EEPROM is rated for a minimum of 50,000 writes and 10 years data retention at T=+55°C.

Convert T [44h]

This command begins a temperature conversion. No further data is required. The temperature conversion will be performed and then the DS18B20 will remain idle. If the bus master issues read time slots following this command, the DS18B20 will output 0 on the bus as long as it is busy making a temperature conversion; it will return a 1 when the temperature conversion is complete. If parasite-powered, the bus master has to enable a strong pullup for a period greater than t_{conv} immediately after issuing this command.

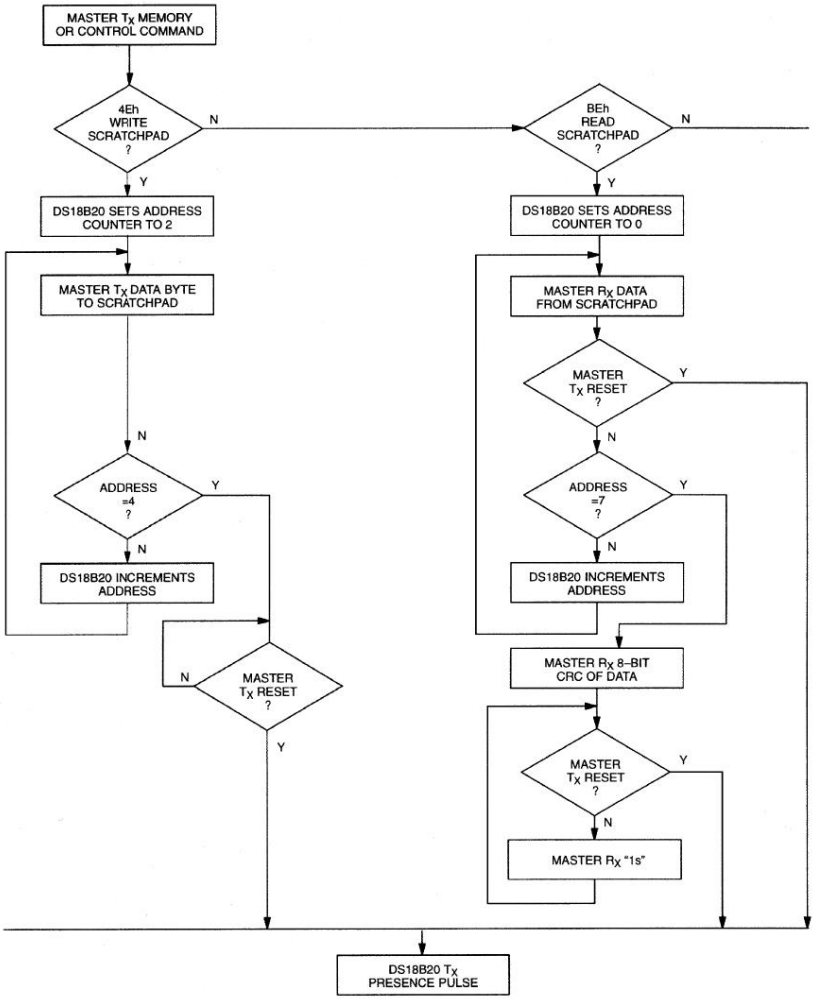
Recall E2 [B8h]

This command recalls the temperature trigger values and configuration register stored in E^2 to the scratchpad. This recall operation happens automatically upon power-up to the DS18B20 as well, so valid data is available in the scratchpad as soon as the device has power applied. With every read data time slot issued after this command has been sent, the device will output its temperature converter busy flag: 0=busy, 1=ready.

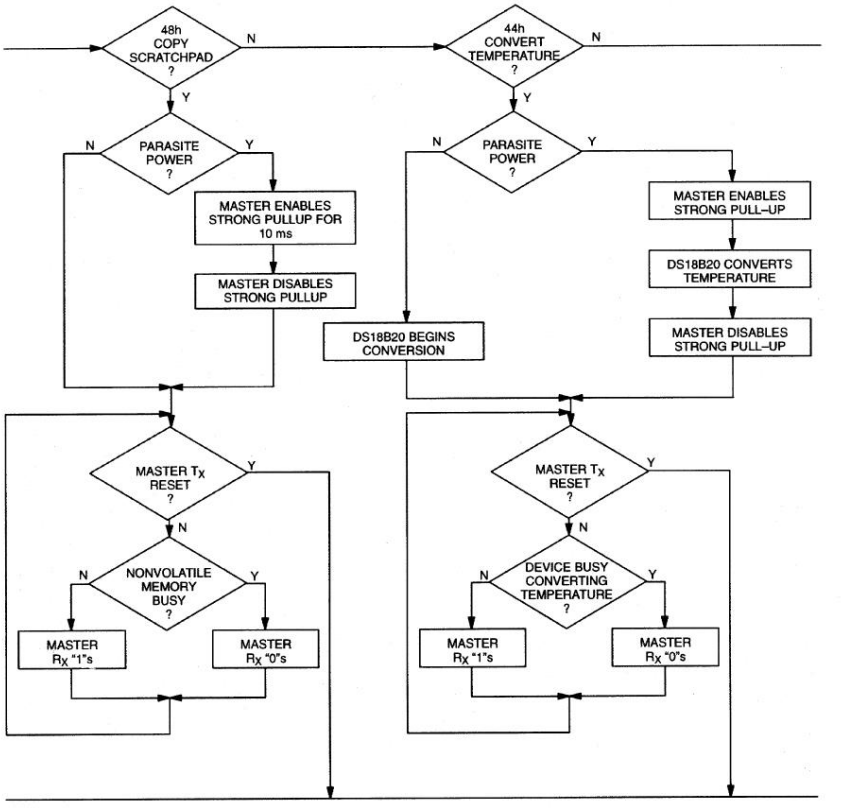
Read Power Supply [B4h]

With every read data time slot issued after this command has been sent to the DS18B20, the device will signal its power mode: 0=parasite power, 1=external power supply provided.

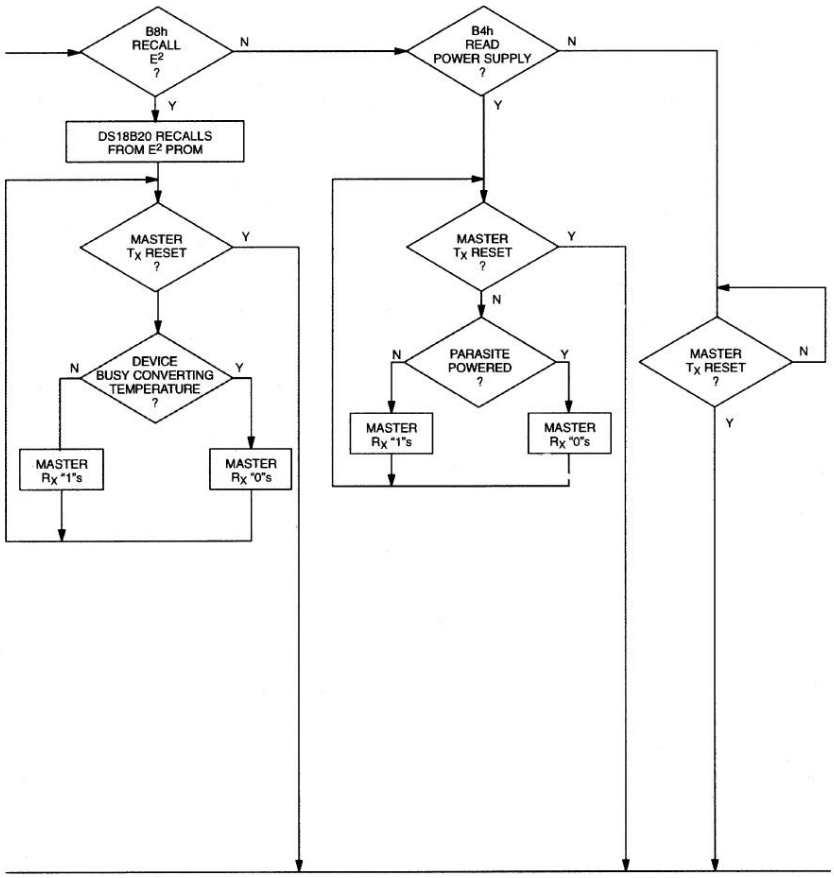
MEMORY FUNCTIONS FLOW CHART Figure 10



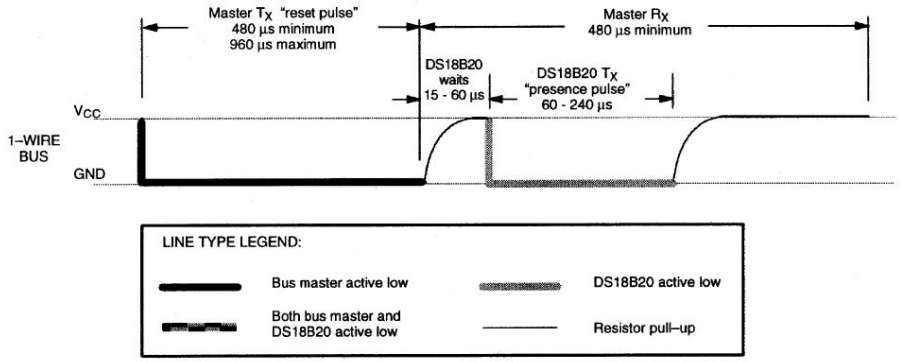
MEMORY FUNCTIONS FLOW CHART Figure 10 (cont'd)



MEMORY FUNCTIONS FLOW CHART Figure 10 (cont'd)



INITIALIZATION PROCEDURE "RESET AND PRESENCE PULSES" Figure 11



DS18B20 COMMAND SET Table 4

INSTRUCTION	DESCRIPTION	PROTOCOL	1-WIRE BUS AFTER ISSUING PROTOCOL	NOTES
TEMPERATURE CONVERSION COMMANDS				
Convert T	Initiates temperature conversion.	44h	<read temperature busy status>	1
MEMORY COMMANDS				
Read Scratchpad	Reads bytes from scratchpad and reads CRC byte.	BEh	<read data up to 9 bytes>	
Write Scratchpad	Writes bytes into scratchpad at addresses 2 through 4 (TH and TL temperature triggers and config).	4Eh	<write data into 3 bytes at addr. 2 through. 4>	3
Copy Scratchpad	Copies scratchpad into nonvolatile memory (addresses 2 through 4 only).	48h	<read copy status>	2
Recall E^2	Recalls values stored in nonvolatile memory into scratchpad (temperature triggers).	B8h	<read temperature busy status>	
Read Power Supply	Signals the mode of DS18B20 power supply to the master.	B4h	<read supply status>	

NOTES:

1. Temperature conversion takes up to 750 ms. After receiving the Convert T protocol, if the part does not receive power from the V_{DD} pin, the DQ line for the DS18B20 must be held high for at least a period greater than t_{conv} to provide power during the conversion process. As such, no other activity may take place on the 1-Wire bus for at least this period after a Convert T command has been issued.

2. After receiving the Copy Scratchpad protocol, if the part does not receive power from the V_{DD} pin, the DQ line for the DS18B20 must be held high for at least 10 ms to provide power during the copy process. As such, no other activity may take place on the 1-Wire bus for at least this period after a Copy Scratchpad command has been issued.

3. All 3 bytes must be written before a reset is issued.

READ/WRITE TIME SLOTS

DS18B20 data is read and written through the use of time slots to manipulate bits and a command word to specify the transaction.

Write Time Slots

A write time slot is initiated when the host pulls the data line from a high logic level to a low logic level. There are two types of write time slots: Write 1 time slots and Write 0 time slots. All write time slots must be a minimum of 60 µs in duration with a minimum of a 1-µs recovery time between individual write cycles.

The DS18B20 samples the DQ line in a window of 15 µs to 60 µs after the DQ line falls. If the line is high, a Write 1 occurs. If the line is low, a Write 0 occurs (see Figure 12).

For the host to generate a Write 1 time slot, the data line must be pulled to a logic low level and then released, allowing the data line to pull up to a high level within 15 µs after the start of the write time slot.

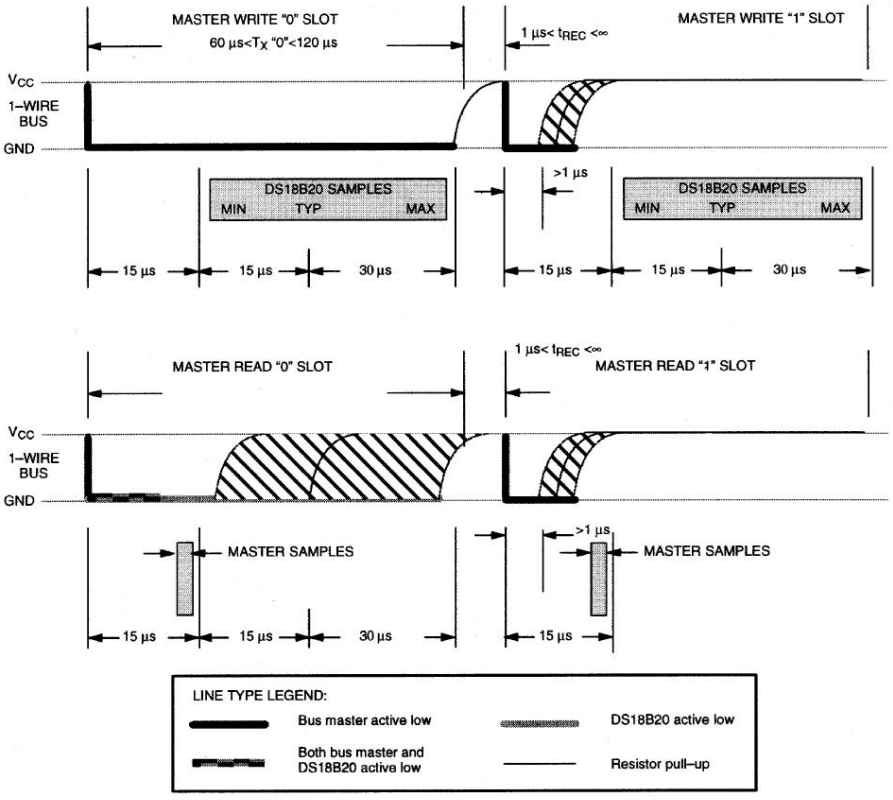
For the host to generate a Write 0 time slot, the data line must be pulled to a logic low level and remain low for 60 µs.

Read Time Slots

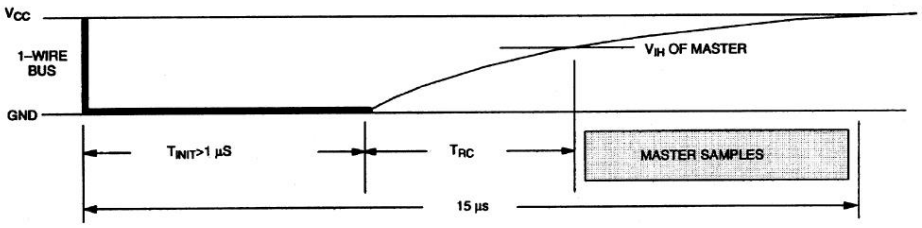
The host generates read time slots when data is to be read from the DS18B20. A read time slot is initiated when the host pulls the data line from a logic high level to logic low level. The data line must remain at a low logic level for a minimum of 1 µs; output data from the DS18B20 is valid for 15 µs after the falling edge of the read time slot. The host therefore must stop driving the DQ pin low in order to read its state 15 µs from the start of the read slot (see Figure 12). By the end of the read time slot, the DQ pin will pull back high via the external pullup resistor. All read time slots must be a minimum of 60 µs in duration with a minimum of a 1-µs recovery time between individual read slots.

Figure 12 shows that the sum of T_{INIT}, T_{RC}, and T_{SAMPLE} must be less than 15 µs. Figure 14 shows that system timing margin is maximized by keeping T_{INIT} and T_{RC} as small as possible and by locating the master sample time towards the end of the 15-µs period.

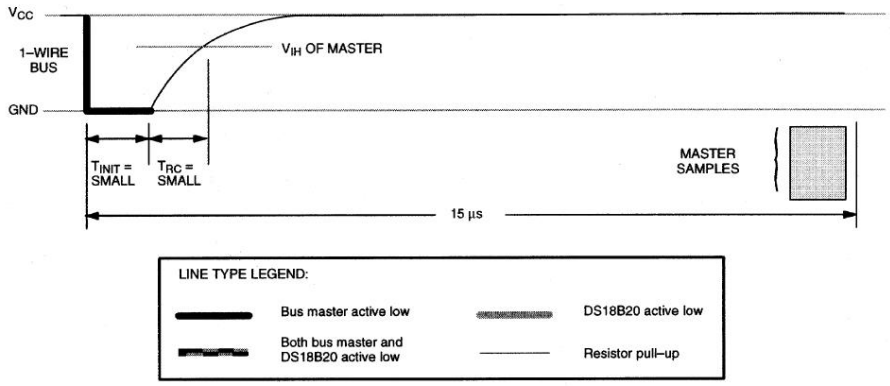
READ/WRITE TIMING DIAGRAM Figure 12



DETAILED MASTER READ 1 TIMING Figure 13



RECOMMENDED MASTER READ 1 TIMING Figure 14



Related Application Notes

The following Application Notes can be applied to the DS18B20. These notes can be obtained from the Dallas Semiconductor "Application Note Book," via our website at http://www.dalsemi.com/.

Application Note 27: "Understanding and Using Cyclic Redundancy Checks with Dallas Semiconductor Touch Memory Product"

Application Note 55: "Extending the Contact Range of Touch Memories"

Application Note 74: "Reading and Writing Touch Memories via Serial Interfaces"

Application Note 104: "Minimalist Temperature Control Demo"

Application Note 106: "Complex MicroLANs"

Application Note 108: "MicroLAN - In the Long Run"

Sample 1-Wire subroutines that can be used in conjunction with AN74 can be downloaded from the website or our Anonymous FTP Site.

MEMORY FUNCTION EXAMPLE Table 5

Example: Bus Master initiates temperature conversion, then reads temperature (parasite power assumed).

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Reset pulse (480-960 µs).
RX	Presence	Presence pulse.
TX	55h	Issue "Match ROM" command.
TX	<64-bit ROM code>	Issue address for DS18B20.
TX	44h	Issue " Convert T" command.
TX	<I/O LINE HIGH>	I/O line is held high for at least a period of time greater than t_{conv} by bus master to allow conversion to complete.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	55h	Issue "Match ROM" command.
TX	<64-bit ROM code>	Issue address for DS18B20.
TX	BEh	Issue "Read Scratchpad" command.
RX	<9 data bytes>	Read entire scratchpad plus CRC; the master now recalculates the CRC of the eight data bytes received from the scratchpad, compares the CRC calculated and the CRC read. If they match, the master continues; if not, this read operation is repeated.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse, done.

MEMORY FUNCTION EXAMPLE Table 6

Example: Bus Master writes memory (parasite power and only one DS18B20 assumed).

MASTER MODE	DATA (LSB FIRST)	COMMENTS
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	CCh	Skip ROM command.
TX	4Eh	Write Scratchpad command.
TX	<3 data bytes>	Writes three bytes to scratchpad (TH, TL, and config).
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	CCh	Skip ROM command.
TX	BEh	Read Scratchpad command.
RX	<9 data bytes>	Read entire scratchpad plus CRC. The master now recalculates the CRC of the eight data bytes received from the scratchpad, compares the CRC and the two other bytes read back from the scratchpad. If data match, the master continues; if not, repeat the sequence.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse.
TX	CCh	Skip ROM command.
TX	48h	Copy Scratchpad command; after issuing this command, the master must wait 10 ms for copy operation to complete.
TX	Reset	Reset pulse.
RX	Presence	Presence pulse, done.

ABSOLUTE MAXIMUM RATINGS*

Voltage on Any Pin Relative to Ground	-0.5V to +6.0V
Operating Temperature	-55°C to +125°C
Storage Temperature	-55°C to +125°C
Soldering Temperature	See J-STD-020A specification

* This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operation sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods of time may affect reliability.

RECOMMENDED DC OPERATING CONDITIONS

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Supply Voltage	V_{DD}	Local Power	3.0		5.5	V	1
Data Pin	DQ		-0.3		+5.5	V	1
Logic 1	V_{IH}		2.2		V_{CC}+ 0.3	V	1,2
Logic 0	V_{IL}		-0.3		+0.8	V	1,3,7

DC ELECTRICAL CHARACTERISTICS (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Thermometer Error	t_{ERR}	-10°C to +85°C			±½	°C	
		-55°C to +125°C			±2		
Input Logic High	V_{IH}	Local Power Parasite Power	2.2 3.0		5.5	V V	1,2 1,2
Input Logic Low	V_{IL}		-0.3		+0.8	V	1,3,7
Sink Current	I_L	$V_{I/O}$=0.4V	-4.0			mA	1
Standby Current	I_{DDS}			750	1000	nA	6,8
Active Current	I_{DD}			1	1.5	mA	4
DQ-Input Load Current	I_{DQ}			5		µA	5

AC ELECTRICAL CHARACTERISTICS: NV MEMORY

(-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
NV Write Cycle Time	t_{WR}			2	10	ms	
EEPROM Writes	N_{EEWR}	-55°C to +55°C	50k			writes	
EEPROM Data Retention	t_{EEDR}	-55°C to +55°C	10			years	

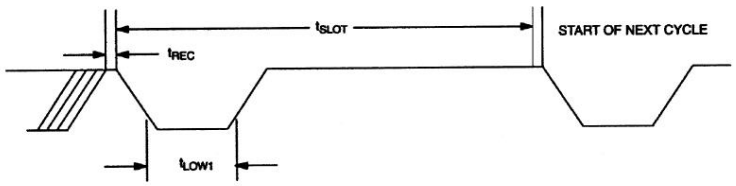
AC ELECTRICAL CHARACTERISTICS: (-55°C to +125°C; V_{DD}=3.0V to 5.5V)

PARAMETER	SYMBOL	CONDITION	MIN	TYP	MAX	UNITS	NOTES
Temperature Conversion Time	t_{CONV}	9 bit			93.75	ms	
		10 bit			187.5		
		11 bit			375		
		12 bit			750		
Time Slot	t_{SLOT}		60		120	µs	
Recovery Time	t_{REC}		1			µs	
Write 0 Low Time	r_{LOW0}		60		120	µs	
Write 1 Low Time	t_{LOW1}		1		15	µs	
Read Data Valid	t_{RDV}				15	µs	
Reset Time High	t_{RSTH}		480			µs	
Reset Time Low	t_{RSTL}		480			µs	9
Presence Detect High	t_{PDHIGH}		15		60	µs	
Presence Detect Low	t_{PDLOW}		60		240	µs	
Capacitance	$C_{IN/OUT}$				25	pF	

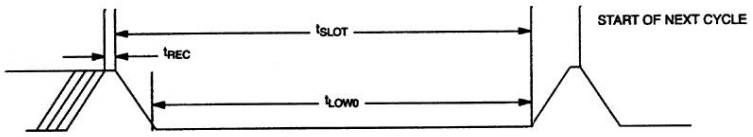
NOTES:

1. All voltages are referenced to ground.
2. Logic one voltages are specified at a source current of 1 mA.
3. Logic zero voltages are specified at a sink current of 4 mA.
4. Active current refers to either temperature conversion or writing to the E^2 memory. Writing to E^2 memory consumes approximately 200 µA for up to 10 ms.
5. Input load is to ground.
6. Standby current specified up to 70°C. Standby current typically is 3 µA at 125°C.
7. To always guarantee a presence pulse under low voltage parasite power conditions, V_{ILMAX} may have to be reduced to as much as 0.5V.
8. To minimize I_{DDS}, DQ should be: GND ≤ DQ ≤ GND +0.3V or V_{DD} – 0.3V ≤ DQ ≤ V_{DD}.
9. Under parasite power, the max t_{RSTL} before a power on reset occurs is 960 µS.

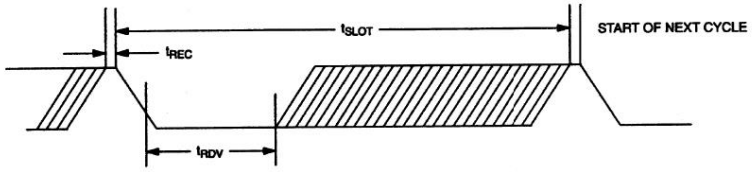
1–WIRE WRITE ONE TIME SLOT



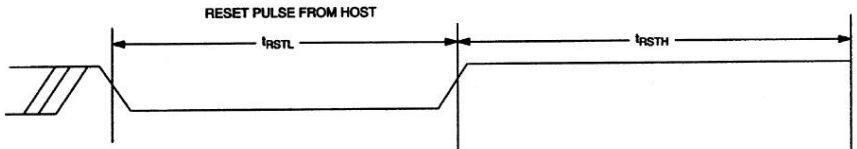
1–WIRE WRITE ZERO TIME SLOT



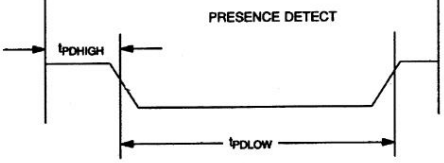
1–WIRE READ ZERO TIME SLOT



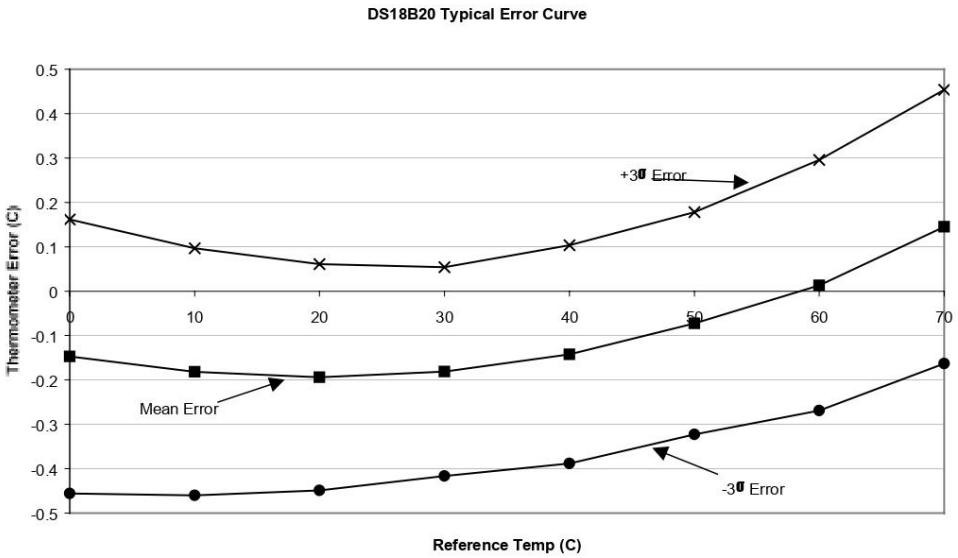
1–WIRE RESET PULSE



1–WIRE PRESENCE DETECT



TYPICAL PERFORMANCE CURVE



16.18. Anexo 18: Datasheet MC14053B

MC14051B, MC14052B, MC14053B

Analog Multiplexers/Demultiplexers

The MC14051B, MC14052B, and MC14053B analog multiplexers are digitally−controlled analog switches. The MC14051B effectively implements an SP8T solid state switch, the MC14052B a DP4T, and the MC14053B a Triple SPDT. All three devices feature low ON impedance and very low OFF leakage current. Control of analog signals up to the complete supply voltage range can be achieved.

Features

- Triple Diode Protection on Control Inputs
- Switch Function is Break Before Make
- Supply Voltage Range = 3.0 Vdc to 18 Vdc
- Analog Voltage Range ($V_{DD} - V_{EE}$) = 3.0 to 18 V
  Note: V_{EE} must be ≤ V_{SS}
- Linearized Transfer Characteristics
- Low−noise − 12 nV/√Cycle, f ≥ 1.0 kHz Typical
- Pin−for−Pin Replacement for CD4051, CD4052, and CD4053
- For 4PDT Switch, See MC14551B
- For Lower R_{ON}, Use the HC4051, HC4052, or HC4053 High−Speed CMOS Devices
- NLV Prefix for Automotive and Other Applications Requiring Unique Site and Control Change Requirements; AEC−Q100 Qualified and PPAP Capable
- These Devices are Pb−Free and are RoHS Compliant

MAXIMUM RATINGS (Voltages Referenced to V_{SS})

Symbol	Parameter	Value	Unit
V_{DD}	DC Supply Voltage Range (Referenced to V_{EE}, $V_{SS} \geq V_{EE}$)	−0.5 to +18.0	V
V_{in}, V_{out}	Input or Output Voltage Range (DC or Transient) (Referenced to V_{SS} for Control Inputs and V_{EE} for Switch I/O)	−0.5 to V_{DD} + 0.5	V
I_{in}	Input Current (DC or Transient) per Control Pin	+10	mA
I_{SW}	Switch Through Current	±25	mA
P_D	Power Dissipation per Package (Note 1)	500	mW
T_A	Ambient Temperature Range	−55 to +125	°C
T_{stg}	Storage Temperature Range	−65 to +150	°C
T_L	Lead Temperature (8−Second Soldering)	260	°C

Stresses exceeding those listed in the Maximum Ratings table may damage the device. If any of these limits are exceeded, device functionality should not be assumed, damage may occur and reliability may be affected.

1. Temperature Derating: "D/DW" Packages: −7.0 mW/°C From 65°C To 125°C

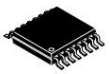
This device contains protection circuitry to guard against damage due to high static voltages or electric fields. However, precautions must be taken to avoid applications of any voltage higher than maximum rated voltages to this high−impedance circuit. For proper operation, V_{in} and V_{out} should be constrained to the range $V_{SS} \leq (V_{in}$ or $V_{out}) \leq V_{DD}$.

Unused inputs must always be tied to an appropriate logic voltage level (e.g., either V_{SS}, V_{EE} or V_{DD}). Unused outputs must be left open.

ON Semiconductor®

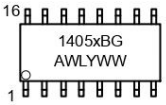
http://onsemi.com

SOIC−16
D SUFFIX
CASE 751B

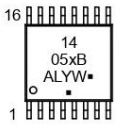


TSSOP−16
DT SUFFIX
CASE 948F

MARKING DIAGRAMS



SOIC−16



TSSOP−16

x = 1, 2, or 3
A = Assembly Location
WL, L = Wafer Lot
Y = Year
WW, W = Work Week
G or ▪ = Pb−Free Package

(Note: Microdot may be in either location)

ORDERING INFORMATION

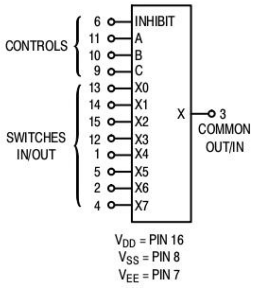
See detailed ordering and shipping information in the package dimensions section on page 9 of this data sheet.

August, 2014 − Rev. 14

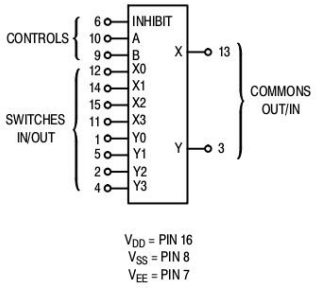
Publication Order Number:
MC14051B/D

MC14051B, MC14052B, MC14053B

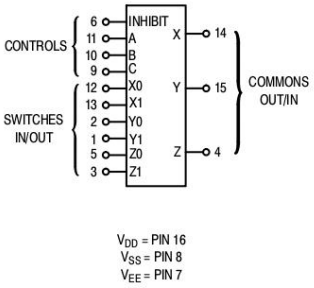
MC14051B
8−Channel Analog
Multiplexer/Demultiplexer



MC14052B
Dual 4−Channel Analog
Multiplexer/Demultiplexer



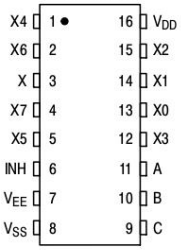
MC14053B
Triple 2−Channel Analog
Multiplexer/Demultiplexer



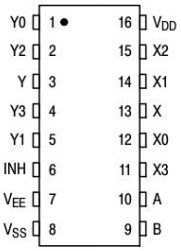
Note: Control Inputs referenced to V_{SS}, Analog Inputs and Outputs reference to V_{EE}. V_{EE} must be ≤ V_{SS}.

PIN ASSIGNMENT

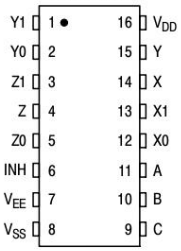
MC14051B



MC14052B



MC14053B



ELECTRICAL CHARACTERISTICS

Characteristic	Symbol	V_{DD}	Test Conditions	−55°C Min	−55°C Max	25°C Min	25°C Typ (Note 2)	25°C Max	125°C Min	125°C Max	Unit
SUPPLY REQUIREMENTS (Voltages Referenced to V_{EE})											
Power Supply Voltage Range	V_{DD}	–	$V_{DD} - 3.0 \geq V_{SS} \geq V_{EE}$	3.0	18	3.0	–	18	3.0	18	V
Quiescent Current Per Package	I_{DD}	5.0 10 15	Control Inputs: $V_{in} = V_{SS}$ or V_{DD}, Switch I/O: $V_{EE} \leq V_{I/O} \leq V_{DD}$, and $\Delta V_{switch} \leq$ 500 mV (Note 3)	– – –	5.0 10 20	– – –	0.005 0.010 0.015	5.0 10 20	– – –	150 300 600	μA
Total Supply Current (Dynamic Plus Quiescent, Per Package	$I_{D(AV)}$	5.0 10 15	$T_A = 25°C$ only (The channel component, $(V_{in} - V_{out})/R_{on}$, is not included.)	Typical			(0.07 μA/kHz) f + I_{DD} (0.20 μA/kHz) f + I_{DD} (0.36 μA/kHz) f + I_{DD}				μA
CONTROL INPUTS — INHIBIT, A, B, C (Voltages Referenced to V_{SS})											
Low−Level Input Voltage	V_{IL}	5.0 10 15	R_{on} = per spec, I_{off} = per spec	– – –	1.5 3.0 4.0	– – –	2.25 4.50 6.75	1.5 3.0 4.0	– – –	1.5 3.0 4.0	V
High−Level Input Voltage	V_{IH}	5.0 10 15	R_{on} = per spec, I_{off} = per spec	3.5 7.0 11	– – –	3.5 7.0 11	2.75 5.50 8.25	– – –	3.5 7.0 11	– – –	V
Input Leakage Current	I_{in}	15	V_{in} = 0 or V_{DD}	–	±0.1	–	±0.00001	±0.1	–	1.0	μA
Input Capacitance	C_{in}	–		–	–	–	5.0	7.5	–	–	pF
SWITCHES IN/OUT AND COMMONS OUT/IN — X, Y, Z (Voltages Referenced to V_{EE})											
Recommended Peak−to−Peak Voltage Into or Out of the Switch	$V_{I/O}$	–	Channel On or Off	0	V_{DD}	0	–	V_{DD}	0	V_{DD}	V_{PP}
Recommended Static or Dynamic Voltage Across the Switch (Note 3) (Figure 5)	ΔV_{switch}	–	Channel On	0	600	0	–	600	0	300	mV
Output Offset Voltage	V_{OO}	–	V_{in} = 0 V, No Load	–	–	–	10	–	–	–	μV
ON Resistance	R_{on}	5.0 10 15	$\Delta V_{switch} \leq$ 500 mV (Note 3) $V_{in} = V_{IL}$ or V_{IH} (Control), and V_{in} = 0 to V_{DD} (Switch)	– – –	800 400 220	– – –	250 120 80	1050 500 280	– – –	1200 520 300	Ω
ΔON Resistance Between Any Two Channels in the Same Package	ΔR_{on}	5.0 10 15		– – –	70 50 45	– – –	25 10 10	70 50 45	– – –	135 95 65	Ω
Off−Channel Leakage Current (Figure 10)	I_{off}	15	$V_{in} = V_{IL}$ or V_{IH} (Control) Channel to Channel or Any One Channel	–	±100	–	±0.05	±100	–	±1000	nA
Capacitance, Switch I/O	$C_{I/O}$	–	Inhibit = V_{DD}	–	–	–	10	–	–	–	pF
Capacitance, Common O/I	$C_{O/I}$	–	Inhibit = V_{DD} (MC14051B) (MC14052B) (MC14053B)	 – – –	 – – –	 – – –	 60 32 17	 – – –	 – – –	 – – –	pF
Capacitance, Feedthrough (Channel Off)	$C_{I/O}$	– –	Pins Not Adjacent Pins Adjacent	– –	– –	– –	0.15 0.47	– –	– –	– –	pF

Product parametric performance is indicated in the Electrical Characteristics for the listed test conditions, unless otherwise noted. Product performance may not be indicated by the Electrical Characteristics if operated under different conditions.

2. Data labeled "Typ" is not to be used for design purposes, but is intended as an indication of the IC's potential performance.
3. For voltage drops across the switch (ΔV_{switch}) > 600 mV (> 300 mV at high temperature), excessive V_{DD} current may be drawn, i.e. the current out of the switch may contain both V_{DD} and switch input components. The reliability of the device will be unaffected unless the Maximum Ratings are exceeded. (See first page of this data sheet.)

ELECTRICAL CHARACTERISTICS (Note 4) (C_L = 50 pF, T_A = 25°C) ($V_{EE} \le V_{SS}$ unless otherwise indicated)

Characteristic	Symbol	$V_{DD} - V_{EE}$ Vdc	Typ (Note 5) All Types	Max	Unit
Propagation Delay Times (Figure 6)	t_{PLH}, t_{PHL}				ns
Switch Input to Switch Output (R_L = 1 kΩ)					
MC14051					
t_{PLH}, t_{PHL} = (0.17 ns/pF) C_L + 26.5 ns		5.0	35	90	
t_{PLH}, t_{PHL} = (0.08 ns/pF) C_L + 11 ns		10	15	40	
t_{PLH}, t_{PHL} = (0.06 ns/pF) C_L + 9.0 ns		15	12	30	
MC14052					ns
t_{PLH}, t_{PHL} = (0.17 ns/pF) C_L + 21.5 ns		5.0	30	75	
t_{PLH}, t_{PHL} = (0.08 ns/pF) C_L + 8.0 ns		10	12	30	
t_{PLH}, t_{PHL} = (0.06 ns/pF) C_L + 7.0 ns		15	10	25	
MC14053					ns
t_{PLH}, t_{PHL} = (0.17 ns/pF) C_L + 16.5 ns		5.0	25	65	
t_{PLH}, t_{PHL} = (0.08 ns/pF) C_L + 4.0 ns		10	8.0	20	
t_{PLH}, t_{PHL} = (0.06 ns/pF) C_L + 3.0 ns		15	6.0	15	
Inhibit to Output (R_L = 10 kΩ, $V_{EE} = V_{SS}$)	t_{PHZ}, t_{PLZ},				ns
Output "1" or "0" to High Impedance, or	t_{PZH}, t_{PZL}				
High Impedance to "1" or "0" Level					
MC14051B		5.0	350	700	
		10	170	340	
		15	140	280	
MC14052B		5.0	300	600	ns
		10	155	310	
		15	125	250	
MC14053B		5.0	275	550	ns
		10	140	280	
		15	110	220	
Control Input to Output (R_L = 1 kΩ, $V_{EE} = V_{SS}$)	t_{PLH}, t_{PHL}				ns
MC14051B		5.0	360	720	
		10	160	320	
		15	120	240	
MC14052B		5.0	325	650	ns
		10	130	260	
		15	90	180	
MC14053B		5.0	300	600	ns
		10	120	240	
		15	80	160	
Second Harmonic Distortion	–	10	0.07	–	%
(R_L = 10KΩ, f = 1 kHz) V_{in} = 5 V_{PP}					
Bandwidth (Figure 7)	BW	10	17	–	MHz
(R_L = 50 Ω, V_{in} = 1/2 (V_{DD}–V_{EE}) p–p, C_L = 50pF					
20 Log (V_{out}/V_{in}) = – 3 dB)					
Off Channel Feedthrough Attenuation (Figure 7)	–	10	–50	–	dB
R_L = 1KΩ, V_{in} = 1/2 (V_{DD} – V_{EE}) p–p					
f_{in} = 4.5 MHz — MC14051B					
f_{in} = 30 MHz — MC14052B					
f_{in} = 55 MHz — MC14053B					
Channel Separation (Figure 8)	–	10	–50	–	dB
(R_L = 1 kΩ, V_{in} = 1/2 (V_{DD}–V_{EE}) p–p,					
f_{in} = 3.0 MHz					
Crosstalk, Control Input to Common O/I (Figure 9)	–	10	75	–	mV
(R_1 = 1 kΩ, R_L = 10 kΩ					
Control $t_{TLH} = t_{THL}$ = 20 ns, Inhibit = V_{SS})					

Product parametric performance is indicated in the Electrical Characteristics for the listed test conditions, unless otherwise noted. Product performance may not be indicated by the Electrical Characteristics if operated under different conditions.

4. The formulas given are for the typical characteristics only at 25°C.
5. Data labelled "Typ" is not lo be used for design purposes but In intended as an indication of the IC's potential performance.

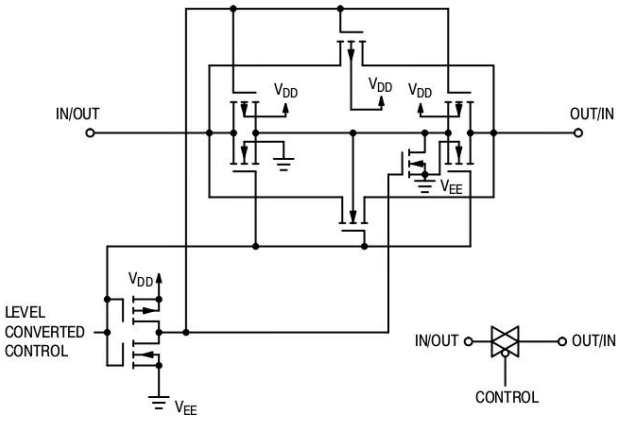


Figure 1. Switch Circuit Schematic

TRUTH TABLE

Control Inputs				ON Switches					
	Select								
Inhibit	C*	B	A	MC14051B	MC14052B		MC14053B		
0	0	0	0	X0	Y0	X0	Z0	Y0	X0
0	0	0	1	X1	Y1	X1	Z0	Y0	X1
0	0	1	0	X2	Y2	X2	Z0	Y1	X0
0	0	1	1	X3	Y3	X3	Z0	Y1	X1
0	1	0	0	X4			Z1	Y0	X0
0	1	0	1	X5			Z1	Y0	X1
0	1	1	0	X6			Z1	Y1	X0
0	1	1	1	X7			Z1	Y1	X1
1	x	x	x	None	None		None		

*Not applicable for MC14052
x = Don't Care

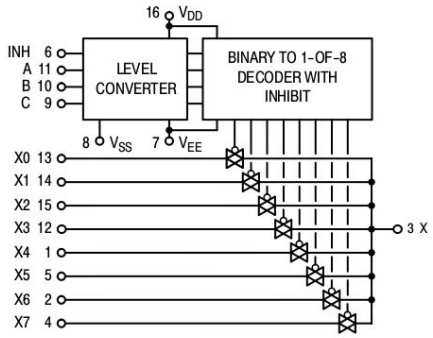


Figure 2. MC14051B Functional Diagram

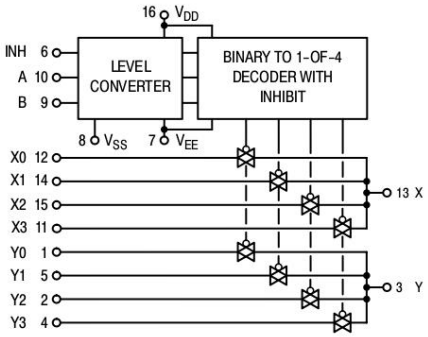


Figure 3. MC14052B Functional Diagram

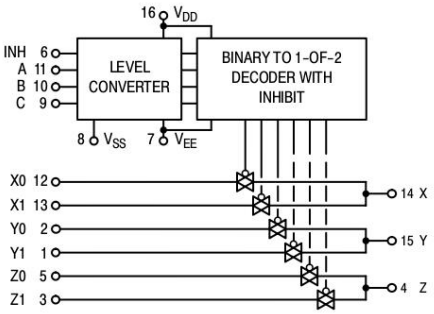


Figure 4. MC14053B Functional Diagram

TEST CIRCUITS

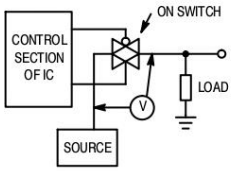


Figure 5. ΔV Across Switch

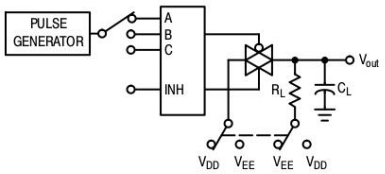


Figure 6. Propagation Delay Times, Control and Inhibit to Output

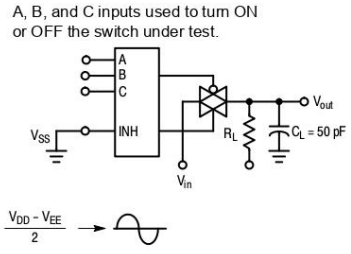


Figure 7. Bandwidth and Off−Channel Feedthrough Attenuation

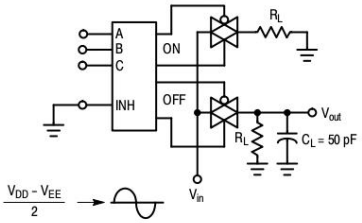


Figure 8. Channel Separation (Adjacent Channels Used For Setup)

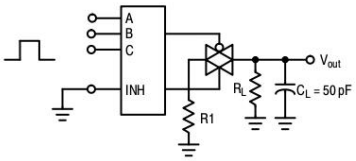


Figure 9. Crosstalk, Control Input to Common O/I

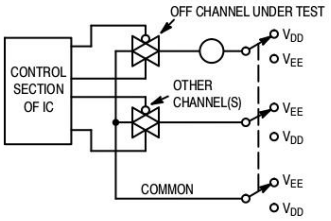


Figure 10. Off Channel Leakage

NOTE: See also Figures 7 and 8 in the MC14016B data sheet.

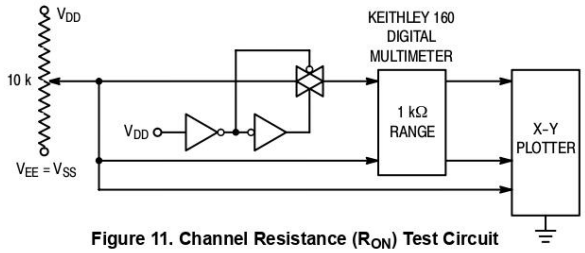


Figure 11. Channel Resistance (R_{ON}) Test Circuit

TYPICAL RESISTANCE CHARACTERISTICS

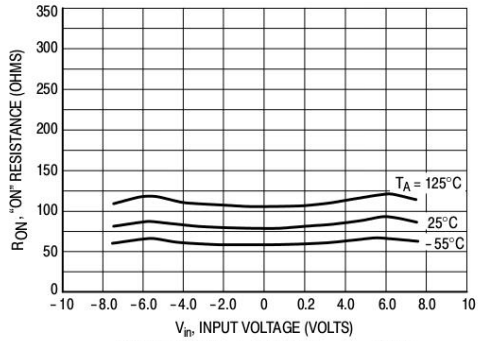


Figure 12. V_{DD} = 7.5 V, V_{EE} = − 7.5 V

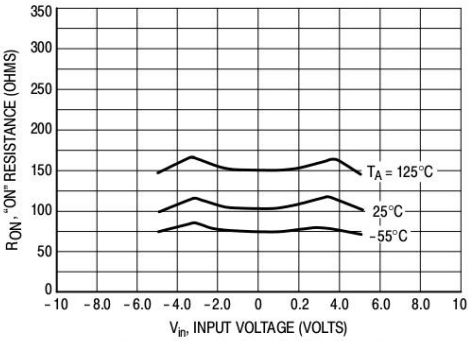


Figure 13. V_{DD} = 5.0 V, V_{EE} = − 5.0 V

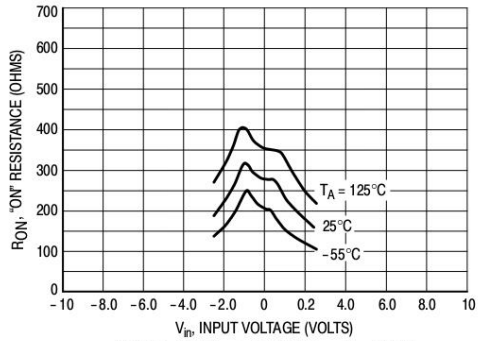


Figure 14. V_{DD} = 2.5 V, V_{EE} = − 2.5 V

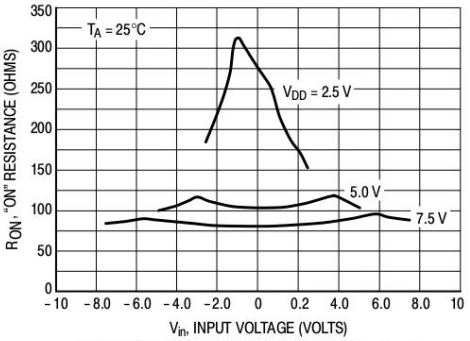


Figure 15. Comparison at 25°C, V_{DD} = −V_{EE}

APPLICATIONS INFORMATION

Figure A illustrates use of the on−chip level converter detailed in Figures 2, 3, and 4. The 0−to−5 V Digital Control signal is used to directly control a 9 V_{p-p} analog signal.

The digital control logic levels are determined by V_{DD} and V_{SS}. The V_{DD} voltage is the logic high voltage; the V_{SS} voltage is logic low. For the example, V_{DD} = +5 V = logic high at the control inputs; V_{SS} = GND = 0 V = logic low.

The maximum analog signal level is determined by V_{DD} and V_{EE}. The V_{DD} voltage determines the maximum recommended peak above V_{SS}. The V_{EE} voltage determines the maximum swing below V_{SS}. For the example, $V_{DD} - V_{SS}$ = 5 V maximum swing above V_{SS}; $V_{SS} - V_{EE}$ = 5 V maximum swing below V_{SS}. The example shows a ±4.5 V signal which allows a 1/2 volt margin at each peak. If voltage transients above V_{DD} and/or below V_{EE} are anticipated on the analog channels, external diodes (Dx) are recommended as shown in Figure B. These diodes should be small signal types able to absorb the maximum anticipated current surges during clipping.

The *absolute* maximum potential difference between V_{DD} and V_{EE} is 18.0 V. Most parameters are specified up to 15 V which is the *recommended* maximum difference between V_{DD} and V_{EE}.

Balanced supplies are not required. However, V_{SS} must be greater than or equal to V_{EE}. For example, V_{DD} = +10 V, V_{SS} = +5 V, and V_{EE} − 3 V is acceptable. See the Table below.

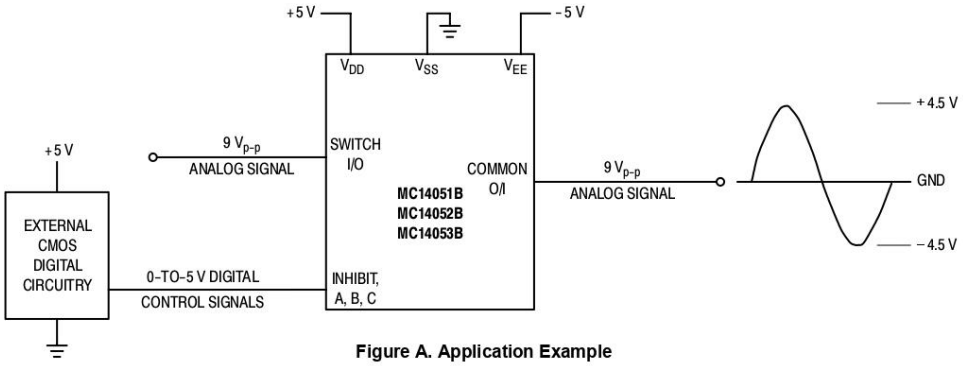


Figure A. Application Example

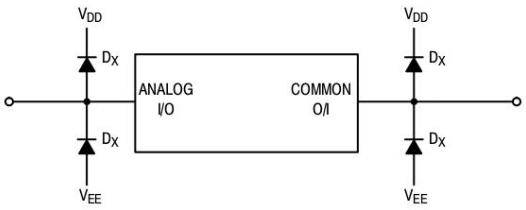


Figure B. External Germanium or Schottky Clipping Diodes

POSSIBLE SUPPLY CONNECTIONS

V_{DD} In Volts	V_{SS} In Volts	V_{EE} In Volts	Control Inputs Logic High/Logic Low In Volts	Maximum Analog Signal Range In Volts
+8	0	–8	+8/0	+8 to –8 = 16 V_{p-p}
+5	0	–12	+5/0	+5 to –12 = 17 V_{p-p}
+5	0	0	+5/0	+5 to 0 = 5 V_{p-p}
+5	0	–5	+5/0	+5 to –5 = 10 V_{p-p}
+10	+5	–5	+10/ +5	+10 to –5 = 15 V_{p-p}

ORDERING INFORMATION

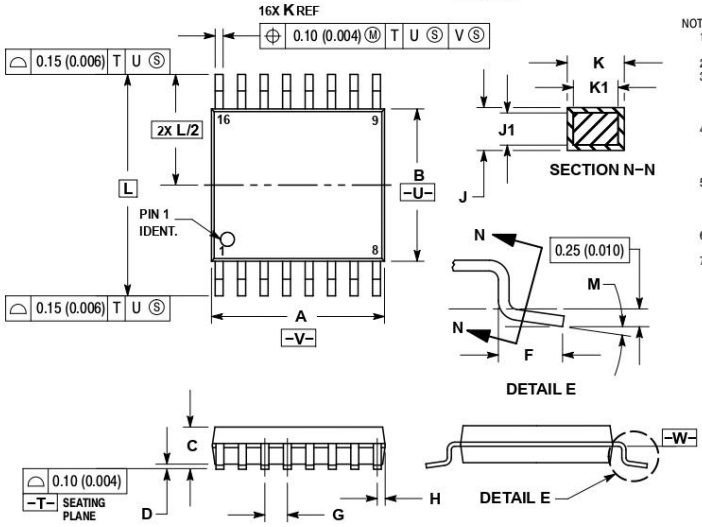
Device	Package	Shipping†
MC14051BDG	SOIC−16 (Pb−Free)	48 Units / Rail
NLV14051BDG*	SOIC−16 (Pb−Free)	48 Units / Rail
MC14051BDR2G	SOIC−16 (Pb−Free)	2500 / Tape & Reel
NLV14051BDR2G*	SOIC−16 (Pb−Free)	2500 / Tape & Reel
MC14051BDTR2G	TSSOP−16 (Pb−Free)	2500 / Tape & Reel
NLV14051BDTR2G*	TSSOP−16 (Pb−Free)	2500 / Tape & Reel
MC14052BDG	SOIC−16 (Pb−Free)	48 Units / Rail
NLV14052BDG*	SOIC−16 (Pb−Free)	48 Units / Rail
MC14052BDR2G	SOIC−16 (Pb−Free)	2500 / Tape & Reel
NLV14052BDR2G*	SOIC−16 (Pb−Free)	2500 / Tape & Reel
MC14052BDTR2G	TSSOP−16 (Pb−Free)	2500 / Tape & Reel
NLV14052BDTR2G*	TSSOP−16 (Pb−Free)	2500 / Tape & Reel
MC14053BDG	SOIC−16 (Pb−Free)	48 Units / Rail
NLV14053BDG*	SOIC−16 (Pb−Free)	48 Units / Rail
MC14053BDR2G	SOIC−16 (Pb−Free)	2500 / Tape & Reel
NLV14053BDR2G*	SOIC−16 (Pb−Free)	2500 / Tape & Reel
MC14053BDTR2G	TSSOP−16 (Pb−Free)	2500 / Tape & Reel
NLV14053BDTR2G*	TSSOP−16 (Pb−Free)	2500 / Tape & Reel

†For information on tape and reel specifications, including part orientation and tape sizes, please refer to our Tape and Reel Packaging Specifications Brochure, BRD8011/D.
*NLV Prefix for Automotive and Other Applications Requiring Unique Site and Control Change Requirements; AEC−Q100 Qualified and PPAP Capable.

MC14051B, MC14052B, MC14053B

PACKAGE DIMENSIONS

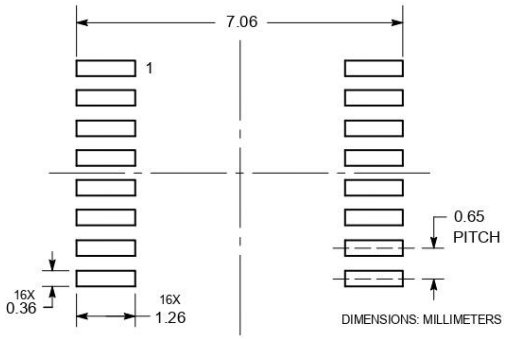
TSSOP−16
DT SUFFIX
CASE 948F
ISSUE B



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DIMENSION A DOES NOT INCLUDE MOLD FLASH. PROTRUSIONS OR GATE BURRS. MOLD FLASH OR GATE BURRS SHALL NOT EXCEED 0.15 (0.006) PER SIDE.
4. DIMENSION B DOES NOT INCLUDE INTERLEAD FLASH OR PROTRUSION. INTERLEAD FLASH OR PROTRUSION SHALL NOT EXCEED 0.25 (0.010) PER SIDE.
5. DIMENSION K DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.08 (0.003) TOTAL IN EXCESS OF THE K DIMENSION AT MAXIMUM MATERIAL CONDITION.
6. TERMINAL NUMBERS ARE SHOWN FOR REFERENCE ONLY.
7. DIMENSION A AND B ARE TO BE DETERMINED AT DATUM PLANE −W−.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	4.90	5.10	0.193	0.200
B	4.30	4.50	0.169	0.177
C	−−−	1.20	−−−	0.047
D	0.05	0.15	0.002	0.006
F	0.50	0.75	0.020	0.030
G	0.65 BSC		0.026 BSC	
H	0.18	0.28	0.007	0.011
J	0.09	0.20	0.004	0.008
J1	0.09	0.16	0.004	0.006
K	0.19	0.30	0.007	0.012
K1	0.19	0.25	0.007	0.010
L	6.40 BSC		0.252 BSC	
M	0 °	8 °	0 °	8 °

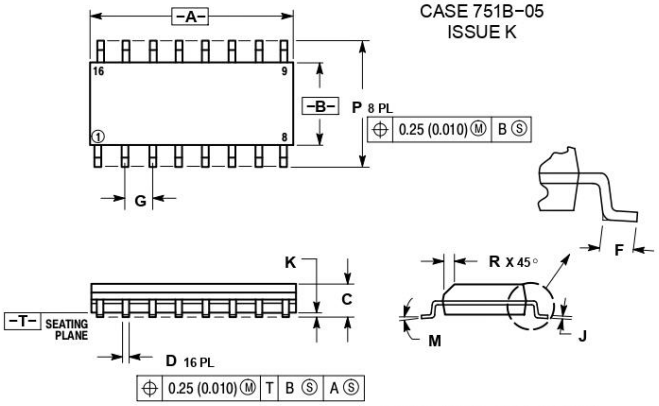
SOLDERING FOOTPRINT*



*For additional information on our Pb−Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

PACKAGE DIMENSIONS

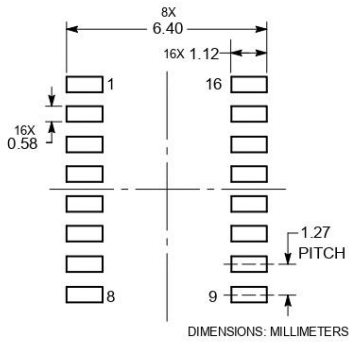
SOIC−16
D SUFFIX
CASE 751B−05
ISSUE K



NOTES:
1. DIMENSIONING AND TOLERANCING PER ANSI Y14.5M, 1982.
2. CONTROLLING DIMENSION: MILLIMETER.
3. DIMENSIONS A AND B DO NOT INCLUDE MOLD PROTRUSION.
4. MAXIMUM MOLD PROTRUSION 0.15 (0.006) PER SIDE.
5. DIMENSION D DOES NOT INCLUDE DAMBAR PROTRUSION. ALLOWABLE DAMBAR PROTRUSION SHALL BE 0.127 (0.005) TOTAL IN EXCESS OF THE D DIMENSION AT MAXIMUM MATERIAL CONDITION.

	MILLIMETERS		INCHES	
DIM	MIN	MAX	MIN	MAX
A	9.80	10.00	0.386	0.393
B	3.80	4.00	0.150	0.157
C	1.35	1.75	0.054	0.068
D	0.35	0.49	0.014	0.019
F	0.40	1.25	0.016	0.049
G	1.27 BSC		0.050 BSC	
J	0.19	0.25	0.008	0.009
K	0.10	0.25	0.004	0.009
M	0°	7°	0°	7°
P	5.80	6.20	0.229	0.244
R	0.25	0.50	0.010	0.019

SOLDERING FOOTPRINT*



DIMENSIONS: MILLIMETERS

*For additional information on our Pb−Free strategy and soldering details, please download the ON Semiconductor Soldering and Mounting Techniques Reference Manual, SOLDERRM/D.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
P.O. Box 5163, Denver, Colorado 80217 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910
Japan Customer Focus Center
Phone: 81−3−5817−1050

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative

MC14051B/D

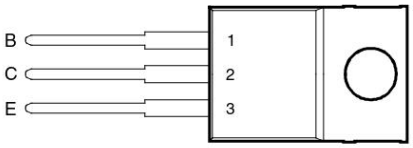
16.19. Anexo 19: Datasheet BD649

BD645, BD647, BD649, BD651
NPN SILICON POWER DARLINGTONS

BOURNS®

- **Designed for Complementary Use with BD646, BD648, BD650 and BD652**
- **62.5 W at 25°C Case Temperature**
- **8 A Continuous Collector Current**
- **Minimum h_{FE} of 750 at 3V, 3 A**

TO-220 PACKAGE
(TOP VIEW)



Pin 2 is in electrical contact with the mounting base.

MDTRACA

absolute maximum ratings at 25°C case temperature (unless otherwise noted)

RATING		SYMBOL	VALUE	UNIT
Collector-base voltage (I_E = 0)	BD645	V_{CBO}	80	V
	BD647		100	
	BD649		120	
	BD651		140	
Collector-emitter voltage (I_B = 0)	BD645	V_{CEO}	60	V
	BD647		80	
	BD649		100	
	BD651		120	
Emitter-base voltage		V_{EBO}	5	V
Continuous collector current		I_C	8	A
Peak collector current (see Note 1)		I_{CM}	12	A
Continuous base current		I_B	0.3	A
Continuous device dissipation at (or below) 25°C case temperature (see Note 2)		P_{tot}	62.5	W
Continuous device dissipation at (or below) 25°C free air temperature (see Note 3)		P_{tot}	2	W
Unclamped inductive load energy (see Note 4)		$½LI_C^2$	50	mJ
Operating junction temperature range		T_j	-65 to +150	°C
Storage temperature range		T_{stg}	-65 to +150	°C
Lead temperature 3.2 mm from case for 10 seconds		T_L	260	°C

NOTES: 1. This value applies for $t_p \leq 0.3$ ms, duty cycle ≤ 10%.
2. Derate linearly to 150°C case temperature at the rate of 0.4 W/°C.
3. Derate linearly to 150°C free air temperature at the rate of 16 mW/°C.
4. This rating is based on the capability of the transistor to operate safely in a circuit of: L = 20 mH, $I_{B(on)}$ = 5 mA, R_{BE} = 100 Ω, $V_{BE(off)}$ = 0, R_S = 0.1 Ω, V_{CC} = 20 V.

PRODUCT INFORMATION

MAY 1993 - REVISED SEPTEMBER 2002
Specifications are subject to change without notice.

1

BD645, BD647, BD649, BD651
NPN SILICON POWER DARLINGTONS

BOURNS®

electrical characteristics at 25°C case temperature (unless otherwise noted)

PARAMETER		TEST CONDITIONS				MIN	TYP	MAX	UNIT
$V_{(BR)CEO}$	Collector-emitter breakdown voltage	I_C = 30 mA	I_B = 0	(see Note 5)	BD645 BD647 BD649 BD651	60 80 100 120			V
I_{CEO}	Collector-emitter cut-off current	V_{CE} = 30 V V_{CE} = 40 V V_{CE} = 50 V V_{CE} = 60 V	I_B = 0 I_B = 0 I_B = 0 I_B = 0		BD645 BD647 BD649 BD651			0.5 0.5 0.5 0.5	mA
I_{CBO}	Collector cut-off current	V_{CB} = 60 V V_{CB} = 80 V V_{CB} = 100 V V_{CB} = 120 V V_{CB} = 40 V V_{CB} = 50 V V_{CB} = 60 V V_{CB} = 70 V	I_E = 0 I_E = 0 I_E = 0 I_E = 0 I_E = 0 I_E = 0 I_E = 0 I_E = 0	 T_C = 150°C T_C = 150°C T_C = 150°C T_C = 150°C	BD645 BD647 BD649 BD651 BD645 BD647 BD649 BD651			0.2 0.2 0.2 0.2 2.0 2.0 2.0 2.0	mA
I_{EBO}	Emitter cut-off current	V_{EB} = 5 V	I_C = 0	(see Notes 5 and 6)				5	mA
h_{FE}	Forward current transfer ratio	V_{CE} = 3 V	I_C = 3 A	(see Notes 5 and 6)		750			
$V_{CE(sat)}$	Collector-emitter saturation voltage	I_B = 12 mA I_B = 50 mA	I_C = 3 A I_C = 5 A	(see Notes 5 and 6)				2 2.5	V
$V_{BE(sat)}$	Base-emitter saturation voltage	I_B = 50 mA	I_C = 5 A	(see Notes 5 and 6)				3	V
$V_{BE(on)}$	Base-emitter voltage	V_{CE} = 3 V	I_C = 3 A	(see Notes 5 and 6)				2.5	V

NOTES: 5. These parameters must be measured using pulse techniques, t_p = 300 µs, duty cycle ≤ 2%.
6. These parameters must be measured using voltage-sensing contacts, separate from the current carrying contacts.

thermal characteristics

PARAMETER		MIN	TYP	MAX	UNIT
$R_{\theta JC}$	Junction to case thermal resistance			2.0	°C/W
$R_{\theta JA}$	Junction to free air thermal resistance			62.5	°C/W

TYPICAL CHARACTERISTICS

TYPICAL DC CURRENT GAIN vs COLLECTOR CURRENT

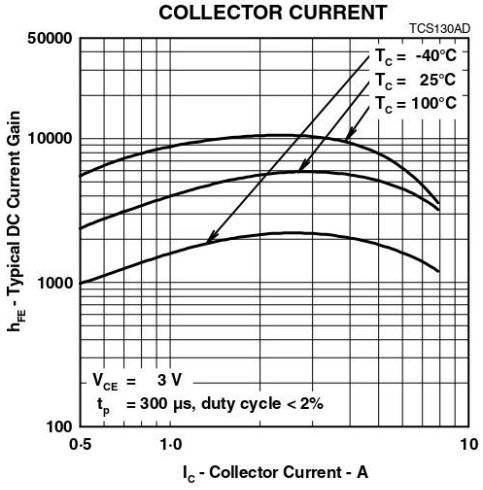


Figure 1.

COLLECTOR-EMITTER SATURATION VOLTAGE vs COLLECTOR CURRENT

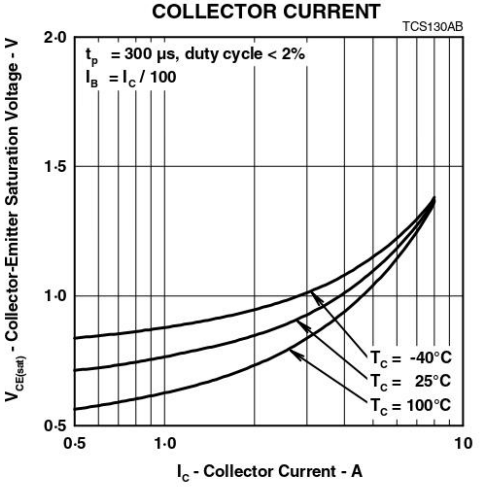


Figure 2.

BASE-EMITTER SATURATION VOLTAGE vs COLLECTOR CURRENT

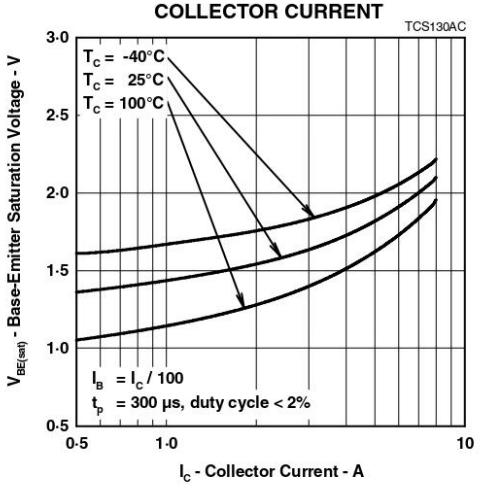


Figure 3.

MAXIMUM SAFE OPERATING REGIONS

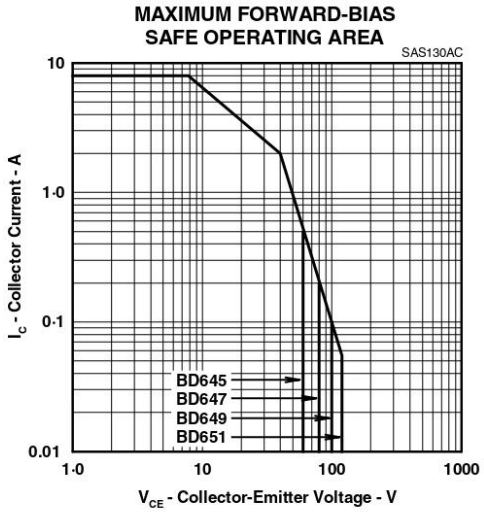


Figure 4.

THERMAL INFORMATION

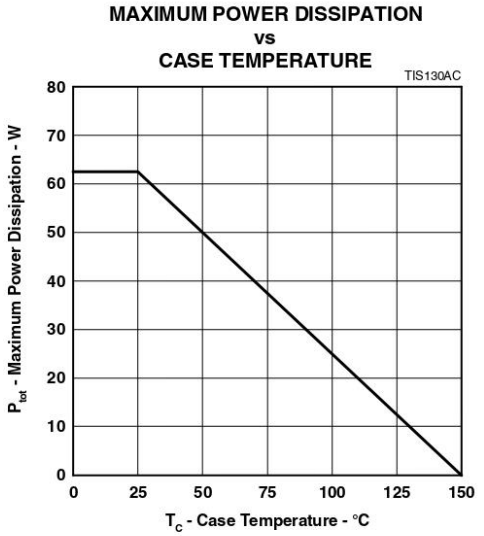


Figure 5.

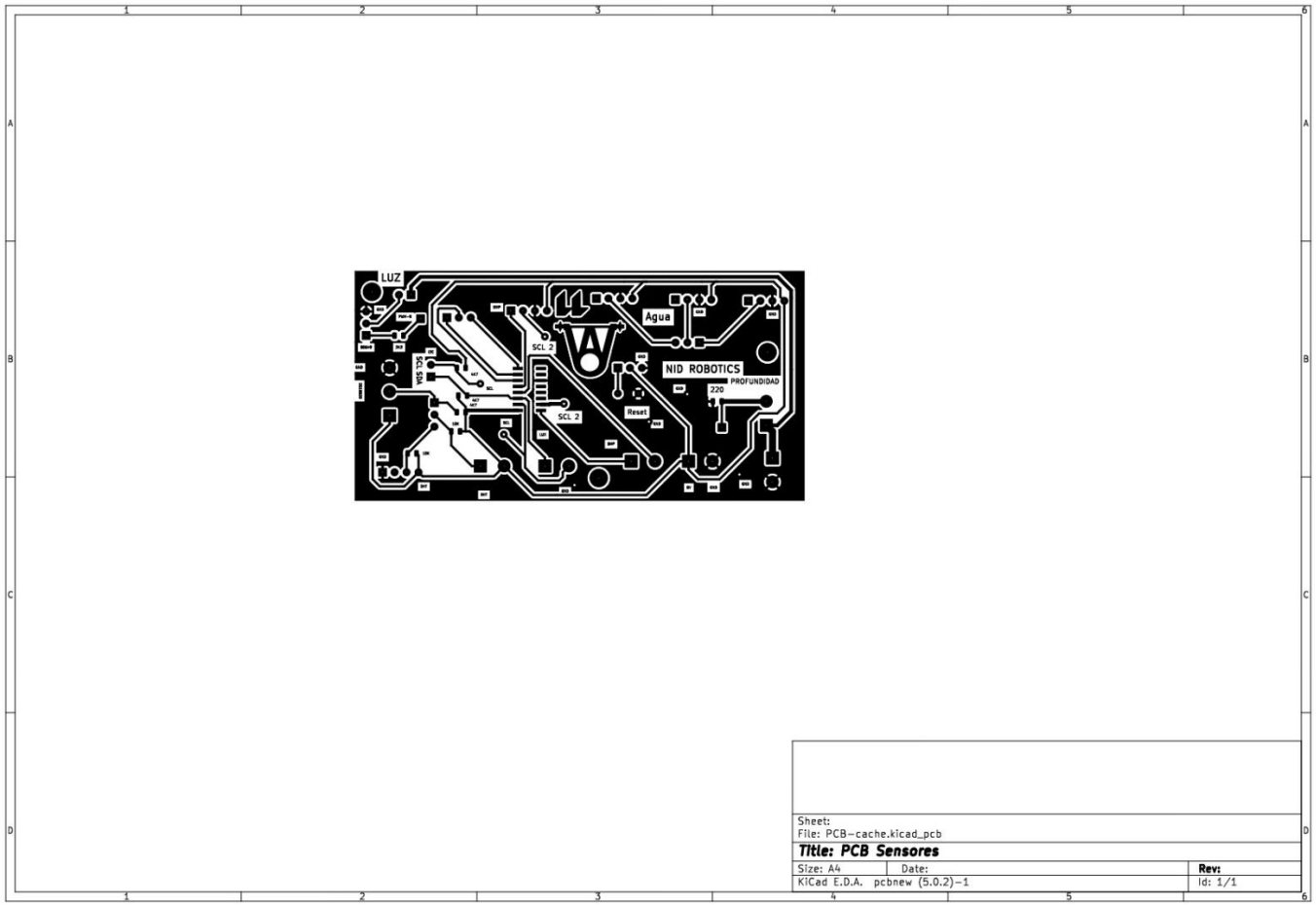
Mouser Electronics

Authorized Distributor

Click to View Pricing, Inventory, Delivery & Lifecycle Information:

Bourns:
BD651 BD649 BD645 BD647 BD645-S BD647-S BD649-S BD651-S

16.20. Anexo 20: Fotolito PCB sensores



16.21. Anexo 21: Fotolito PCB Raspberry Pi