Universidad de Valladolid

ESCUELA DE INGENIERÍAS INDUSTRIALES

Máster en Ingeniería Industrial

MÁSTER EN INGENIERÍA INDUSTRIAL

ESCUELA DE INGENIERÍAS INDUSTRIALES

UNIVERSIDAD DE VALLADOLID

TRABAJO FIN DE MÁSTER

CONTROL DE SISTEMA BOLA-VIGA (BALL & BEAM) A DISTANCIA MEDIANTE PROTOCOLO ETHERNET

Autor: D. Pablo Martín Fernández

Tutor: D. Francisco Javier García Ruiz

Valladolid, septiembre, 2019

ABSTRACT

El sistema “bola y viga” se define como una práctica clásica de control no lineal en la cual se pueden ejecutar distintos algoritmos de control, donde el proceso de diseño matemático está muy presente para que el controlador logre su objetivo. El objetivo principal de este proyecto es servir de base de aprendizaje para los alumnos de la escuela de ingenieras industriales, con el cual puedan aprender metodologías de control y en especial el diseño de controladores en lazo cerrado con algoritmo PID en este caso para equilibrar en una maqueta una bola que se desliza sobre una viga.

Los alumnos podrán modificar parámetros característicos del control PID realizando prácticas y pruebas, mediante el protocolo ethernet TCP/IP de forma telemandada obtenido así la salida del sistema en forma gráfica y de forma visual mediante webcam en tiempo real o de forma presencial mediante protocolo serial-UART.

Este proyecto también pretende ser una experiencia donde los alumnos podrán analizar el sistema desde un modelo matemático y mediante la teoría de control impartida en la escuela de ingenieras industriales obtener los parámetros utilizados en los sistemas de control PID que en la actualidad son empleados en la industria para el control de procesos continuos.

PALABRAS CLAVE

controlador proporcional-integral-derivativo (PID), control remoto, telemando, ethernet , modelado matemático, parámetros transitorios, bola, viga.

Contenido

1 INTRODUCCIÓN 7
1.1 RESUMEN 7
1.2 OBJETIVOS 8
2 DESCRIPCIÓN DE LA PLANTA 9
2.1 DESCRIPCIÓN DE LA PLANTA 9
2.2 FUNCIONAMIENTO BÁSICO 10
2.3 MODELO MATEMÁTICO DEL SISTEMA 12
3 ESTUDIO DEL LUGAR DE LAS RAÍCES 19
4 PLANTEAMIENTO DEL PROBLEMA Y ELEMENTOS NECESARIOS 25
4.1 PROTOCOLO I2C 25
4.2 SOCKET 28
4.3 UART 31
4.4 SERVOMOTOR 35
4.5 CONTROLADOR PID 37
4.5.1 Introducción. 37
4.5.2 Estructura del PID. 37
4.6 RASPBERRY PI 3 40
4.6.1 Introducción. 40
4.6.2 Hardware del raspberry pi y características. 43
4.6.1 software del raspberry pi RASBIAN (Sistema Operativo) 47
4.7 VL53L0X 50
5 IMPLEMENTACIÓN. 55
5.1 IMPLEMENTACIÓN DEL SERVIDOR. 56
5.1.1 Características de Python 56
5.1.2 *Módulo de autentificación* 59
5.1.3 sensor (VL53L0X). 60
5.1.4 Módulo de comunicaciones. 63
5.1.1 Hilos. 65
5.1.2 Funciones auxiliares 66
5.1.1 Programa principal 67

5.1.2 Módulo de PID 69
5.2 IMPLEMENTACIÓN DEL CLIENTE 72
5.2.1 Visual Basic 72
5.2.2 Aplicación cliente basado en Simplot 73
5.2.1 Barra superior 74
5.2.2 Barra lateral 78
5.2.1 Barra baja 79
5.2.2 Área de grafica 80
6 USO DEL DISPOSITIVO 83
6.1 ZIEGLER-NICHOLS EN LAZO CERRADO (RESPUESTA TEMPORAL) [4] 83
6.2 ZIEGLER-NICHOLS EN LAZO ABIERTO (RESPUESTA TEMPORAL) [4] 84
6.3 DISTINTOS TIPOS DE MINIMIZACIÓN INTEGRAL DEL ERROR [4] 85
6.4 MÉTODO MANUAL 86
7 RESULTADOS Y CONCLUSIONES 89
8 BIBLIOGRAFÍA 91
9 INDICE DE IMÁGENES 95
10 INDICE DE ECUACIONES 97

1 INTRODUCCIÓN

1.1 RESUMEN

En el presente proyecto se pretende abordar el sistema clásico viga-bola [Ilustración 1 Esquema de sistema], es decir el control de la distancia de una bola que se desliza sobre un carril o viga. Sobre dicha viga podremos controlar la inclinación frente a la normal. En cuanto al sistema de control de la planta, en este proyecto se podría controlar por distintas metodologías en lazo cerrado tanto in situ como a distancia.

El protocolo de comunicación a nivel local será la comunicación tipo serial y para actuar sobre el sistema de forma remota se utilizará un protocolo ethernet (TCP/IP) mediante socket.

El proyecto se engloba dentro de las prácticas del departamento de automática de la universidad de Valladolid, a fin de poder realizar los ensayos en distintas asignaturas de forma telemandada, es decir, sin necesidad de estar en la misma ubicación de la planta.

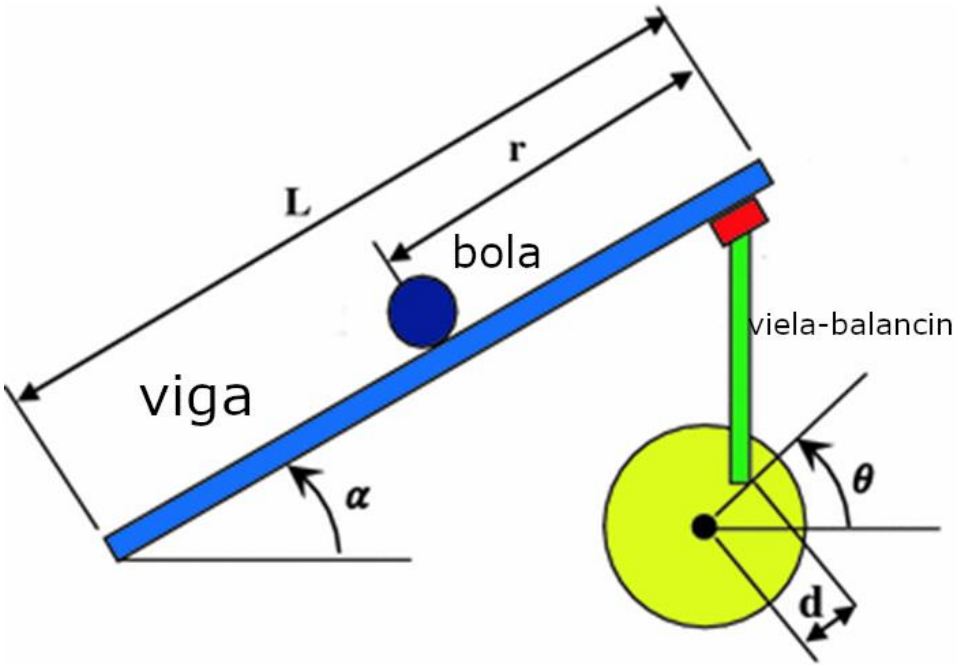


Ilustración 1 Esquema de sistema

1.2 OBJETIVOS

I. Obtener un modelo matemático-físico para poder parametrizar el sistema de manera óptima.
II. Entender el funcionamiento de un sistema controlador automático y los distintos componentes que se compone, sensor, actuador, controlador.
III. Identificación y obtención de los distintos parámetros de un PID mediante distintos métodos como Ziegler – Nichols, MISAE, MITAE y MISE.
IV. Facilitar las distintas modificaciones de los parámetros de control de un sistema, observando las repercusiones de estos cambios sobre el mismo.
V. Proporcionar a los alumnos un sistema de aprendizaje para entender un sistema de control PID.
VI. Poder graficar las distintas señales de entrada, salida de un sistema controlado.
VII. El uso de un programa a distancia mediante internet y uso de protocolos TCP/IP.

2 DESCRIPCIÓN DE LA PLANTA

2.1 DESCRIPCIÓN DE LA PLANTA

La planta es un sistema mecánico conformado por una base la cual posee dos pilares como apoyos: uno de ellos está fijo, con forma de "Y", donde en la parte superior se coloca dos cojinetes atravesado por un eje, y el otro apoyo es móvil pudiéndose desplazar mediante un servo y mecanismos biela-manivela.

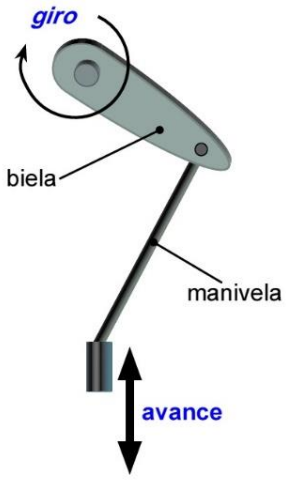


Ilustración 2 Esquema de sistema mecánico biela manivela

Todo el sistema tiene como fin el control de la distancia de bola interior hasta uno de los extremos de la viga. Para ello se usará un sensor del que obtendremos en todo momento este parámetro distancia.

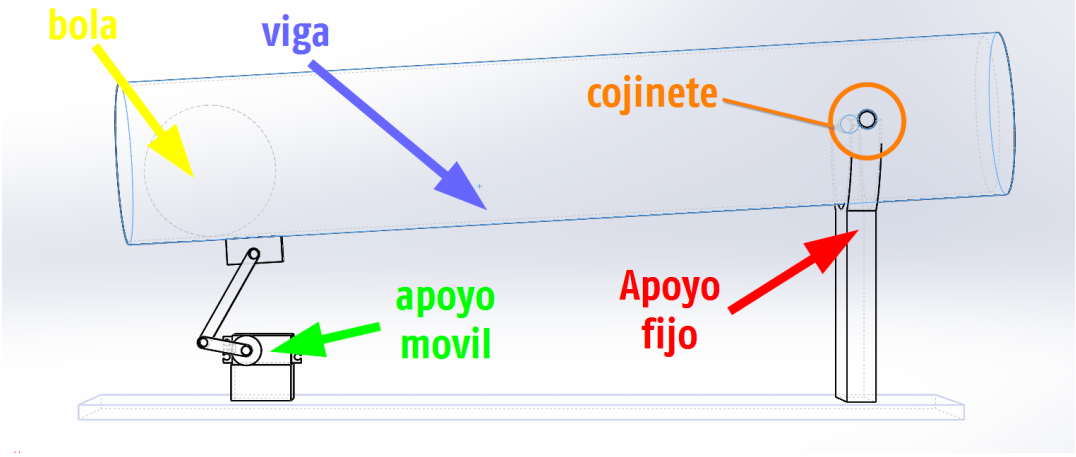


Ilustración 3 Esquema en 3D del sistema inestable

Para poder mover físicamente la bola dentro de la viga se utilizará el servo motor para cambiar la altura del apoyo y nos aprovecharemos de la fuerza gravitatoria. De esta forma, se define una relación entre la posición de servo y la posición de la bola ya que se hace variar la inclinación del tubo, con el fin de que el tubo quede por encima o por debajo de la horizontal del plano y así mover la bola en ambos sentidos.

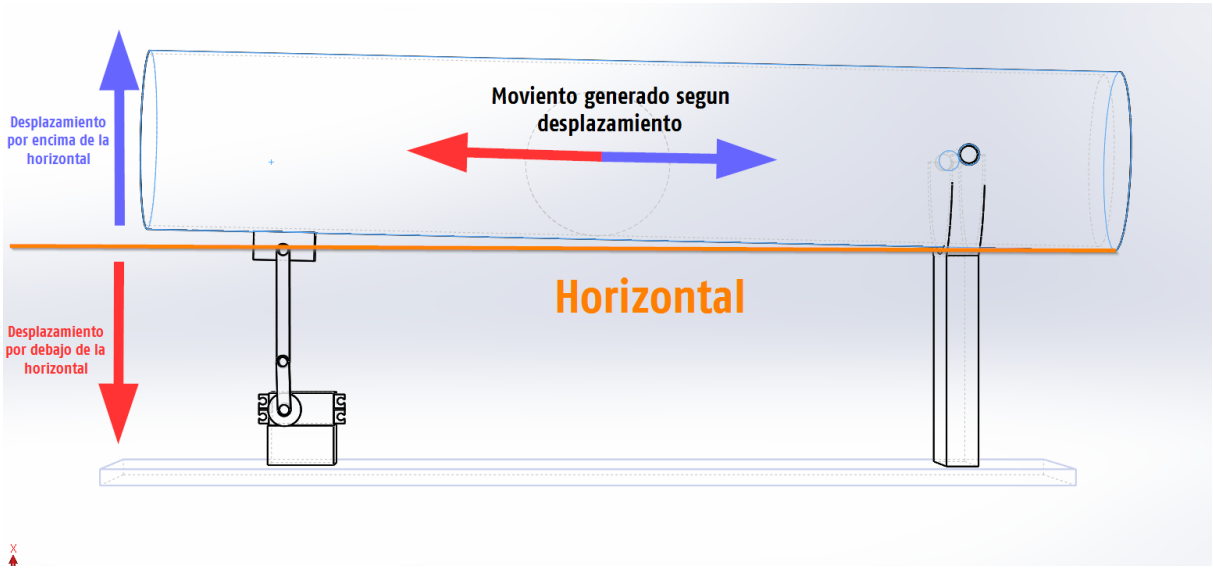


Ilustración 4 Esquema en 3D del sistema estable

2.2 FUNCIONAMIENTO BÁSICO

El funcionamiento del sistema está basado en las relaciones de las fuerzas de dentro del sistema (se utilizará para este cometido una tarjeta Raspberry Pi).

Estas fuerzas se contrarrestan o incrementan dentro del sistema de distintas formas, pero desde el controlador solo se tiene acceso a una sola salida, la distancia, y una única entrada, la posición del servomotor y por relación solidaria el ángulo de la viga sobre la horizontal. Por tanto, nos encontramos ante un sistema SISO.

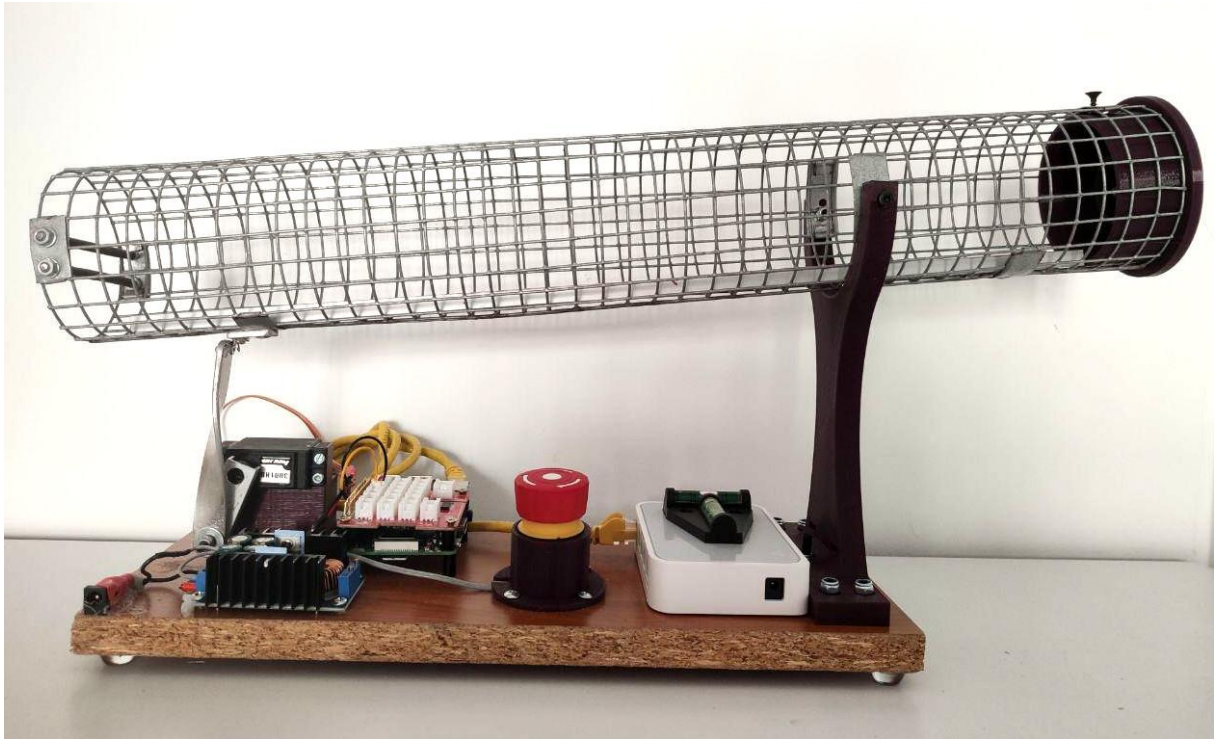


Ilustración 5 Fotografía del modelo del sistema

El sistema tiene como entrada la posición angular del servo mediante la señal digital proporcionada por el controlador (la Raspberry Pi). Para este tipo de señal se usa un PWM con una frecuencia de 20ms, siendo este actuador nuestro limitante en el ciclo de control.

El servo cambia el ángulo de la viga sobre la que se posiciona la bola, la cual a causa de la energía potencial rodará por la viga. Al rodar por la viga cambiará su posición siendo esta distancia controlada por el sensor de distancia (VL53L0X). Este sensor es un sensor de tiempo de vuelo, es decir, envía una señal infrarroja y calcula el tiempo que tarda en volver dicha señal, así puede calcular la distancia a través la diferencia de tiempo entre envió y recepción de esta señal.

La comunicación entre este sensor y el controlador se realiza por medio de un protocolo digital I2C un protocolo complejo multipunto.

El ajuste de los parámetros que necesita el controlador puede ser modificado mediante una aplicación informática diseñada para el sistema operativo de Microsoft. La comunicación se realiza mediante protocolo Ethernet o Serial de forma que posibilita el telemando a distancia de la planta, ejecutándose la parte de control a nivel local por

el controlador (Raspberry Pi).

Para facilitar la comprensión de parámetros y su posible modificación se graficarán los datos obtenidos por el sensor de posición de la bola y el sistema también será visible mediante streaming tipo video. Se realizará un control mediante controlador PID, y tras ajustes sucesivos, se podrá llegar a la estabilidad del sistema.

2.3 MODELO MATEMÁTICO DEL SISTEMA

La dinámica del sistema puede ser convenientemente descrita a partir de las ecuaciones de movimiento derivadas del Lagrangiano,

$$L = K - V$$

Ecuación 1 Ecuación básica de lagrangiano

Donde K es la energía cinética y V es la energía potencial del sistema.

Energía cinética de la viga es,

$$K_v = \frac{1}{2} J_v \dot{\theta}_v^2$$

Ecuación 2 Ecuación cinética de la viga

Donde J_B es el momento de inercia de la viga y θ_B es el ángulo de rotación de la viga.

La energía cinética de la bola es,

$$K_b = \frac{1}{2} J_b \dot{\theta}_b^2 + \frac{1}{2} m {v_b}^2$$

Ecuación 3 Ecuación cinética de la bola

Donde J_b es el momento de inercia de la bola, θ_b es el ángulo de rotación de la bola, v_b es la velocidad lineal de la bola y m es la masa de la bola.

Por último, la energía potencial del sistema puede expresarse como,

$$V = mgy$$

Ecuación 4 energía potencial del sistema

Sin embargo, todas las ecuaciones deben ser escritas en términos de las coordenadas generalizadas, de tal forma que se minimicen el número de variables posibles. El sistema tiene dos grados de libertad, el giro de la viga y la posición de la bola sobre la viga. Por tanto, las coordenadas generalizadas elegidas son $r(t)$, la posición de la bola sobre la viga medido en la dirección que traza la viga a partir del origen de coordenadas, y $\theta_B(t)$, el ángulo de giro de la viga. Estas dos variables son función del tiempo. A continuación, se representan las coordenadas elegidas, junto a las cartesianas, en un esquema:

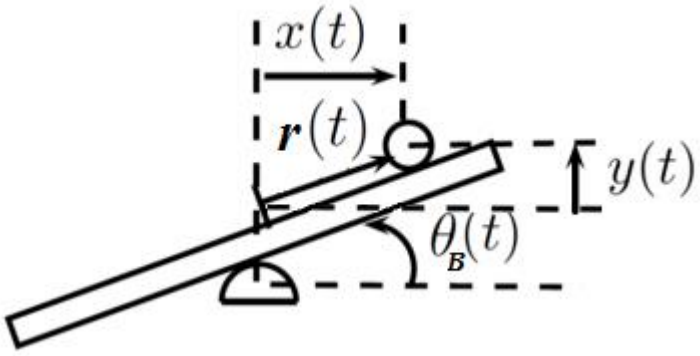


Ilustración 6 Esquema de para el modelo matemático

Ahora, a través de la relación angular, $\dot{\theta}_b$ puede escribirse en términos de las coordenadas generalizadas

$$\dot{\theta}_b = \frac{\dot{r}}{R}$$

Ecuación 5 Ecuación de relación angular

Donde R es el radio de la bola.

Por otro lado, se puede expresar la velocidad lineal de la bola v_b a través de las coordenadas generalizadas usando para ello la relación entre éstas y las cartesianas:

$$r^2 = x^2 + y^2$$

Ecuación 6 velocidad lineal de la bola

Equivalentemente,

$$v_b{}^2 = \dot{x}^2 + \dot{y}^2$$

Ecuación 7 Aceleración lineal de la bola

Las coordenadas cartesianas se relacionan con las generalizadas a través de las relaciones trigonométricas:

$$x = r\cos\theta$$
$$y = r\sin\theta$$

Ecuación 8 Relación coordenadas cartesianas con generalizadas

Como la única coordenada angular que va a utilizarse de este punto en adelante es el ángulo de rotación de la viga, θ_B, vamos a prescindir del subíndice y referirnos a este ángulo simplemente como θ.

Derivando respecto al tiempo,

$$\dot{x} = \dot{r}\cos\theta - r\dot{\theta}\sin\theta$$
$$\dot{y} = \dot{r}\sin\theta + r\dot{\theta}\cos\theta$$

Ecuación 9 Coordenadas angulares

Elevando al cuadrado

$$\dot{x}^2 = \dot{r}^2 \cos^2\theta - 2r\dot{r}\dot{\theta}\cos\theta \sin\theta + r^2\dot{\theta}^2 \sin^2\theta$$
$$\dot{y}^2 = \dot{r}^2 \sin^2\theta + 2r\dot{r}\dot{\theta}\cos\theta \sin\theta + r^2\dot{\theta}^2 \cos^2\theta$$

Ecuación 10 Elevación al cuadrado de coordenadas angulares

Sumando ambas coordenadas y simplificando se obtiene la velocidad lineal de la bola,

$$v_b{}^2 = \dot{r}^2 + r^2\dot{\theta}^2$$

Ecuación 11 Velocidad lineal de la bola

Recuperando las componentes de energía cinética y potencial del lagrangiano, e incorporando las transformaciones a coordenadas generalizadas obtenemos:

$$K_v = \frac{1}{2} J_v \dot{\theta}^2$$
$$K_b = \frac{1}{2} J_b \frac{\dot{r}^2}{R^2} + \frac{1}{2} m(\dot{r}^2 + r^2\dot{\theta}^2)$$
$$V = mgr \sin\theta$$

Ecuación 12 Ecuación general del lagrangiana aplicada

Como puede observarse, ahora el lagrangiano solo dependerá de las coordenadas r y θ, y sus derivadas temporales.

Finalmente, se sustituyen los momentos de inercia de la viga $J_B = \frac{1}{3}ML^2$ (M es la masa de la viga) y de la bola $J_b = \frac{2}{5}mR^2$ y se unifica en el lagrangiano:

$$L = \frac{1}{6}ML^2\dot{\theta}^2 + \frac{1}{5} m\dot{r}^2 + \frac{1}{2} m\left(\dot{r}^2 + r^2\dot{\theta}^2\right) - mgr \sin\theta$$

$$L = \frac{1}{6}ML^2\dot{\theta}^2 + \frac{7}{10}\,m\dot{r}^2 + \frac{1}{2}\,mr^2\dot{\theta}^2 - mgr\,sin\,\theta$$

Ecuación 13 Sustitución en momentos inerciales

Las ecuaciones de movimiento vienen determinadas por las expresiones:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{r}}\right) - \frac{\partial L}{\partial r} = 0$$
$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = \tau$$

Ecuación 14 Ecuaciones del movimiento

La segunda ecuación incluye el momento externo τ producido por el motor para actuar sobre el sistema.

Calculando cada término de la primera ecuación,

$$\frac{\partial L}{\partial \dot{r}} = \frac{7}{5}\,m\dot{r}$$
$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{r}}\right) = \frac{7}{5}\,m\ddot{r}$$
$$\frac{\partial L}{\partial r} = mr\dot{\theta}^2 - \,mg\,sin\,\theta$$

Ecuación 15 cálculo de la primera ecuación

Resulta en la primera ecuación de movimiento:

$$\frac{7}{5}\,m\ddot{r} - mr\dot{\theta}^2 + \,mg\,sin\,\theta = 0$$

Ecuación 16 Resultado del movimiento

Análogamente para la segunda ecuación,

$$\frac{\partial L}{\partial \dot{\theta}} = \frac{1}{3}\,ML^2\dot{\theta} + \,mr^2\dot{\theta} = \left(\frac{1}{3}\,ML^2 + \,mr^2\right)\dot{\theta}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) = \left(\frac{1}{3}\, ML^2 + \, mr^2\right)\ddot{\theta} + 2mr\dot{r}\dot{\theta}$$

$$\frac{\partial L}{\partial \theta} = -mgr\, cos\, \theta$$

Ecuación 17 segunda ecuación

Resulta en la segunda ecuación de movimiento:

$$\left(\frac{1}{3}\, ML^2 + \, mr^2\right)\ddot{\theta} + 2mr\dot{r}\dot{\theta} + \, mgr\, cos\, \theta \, = \, \tau$$

Ecuación 18 Resultado de la segunda ecuación

Centrándonos en la primera ecuación de movimiento, suponiendo que los ángulos de la viga se mantienen relativamente pequeños, podemos despreciar los términos de orden superior, desarrollando la serie de Taylor entorno a $\theta = 0$,

$$\frac{7}{5}\, m\ddot{r} + \, mg\, \theta = 0$$

Ecuación 19 Desarrollo de la serie de Taylor sobre primera ecuación

Por tanto, la ecuación anterior ha sido linealizada para θ.

Aplicando la transformada de Laplace sobre la ecuación anterior obtenemos la siguiente ecuación,

$$\frac{7}{5}\, m\, R(s)s^2 + \, mg\Theta(s) = 0$$

Ecuación 20 aplicación de la transformación de Laplace

Finalmente, la función de transferencia será:

$$P(s) = \frac{R(s)}{\Theta(s)} = \frac{5\, g}{7\ s^2}$$

Ecuación 21 Función de transferencia

3 ESTUDIO DEL LUGAR DE LAS RAÍCES

Para el estudio del lugar de las raíces se usó la aplicación “Matlab”. La idea principal del estudio-diseño del lugar de la raíces es estimar la respuesta en bucle cerrado del sistema a partir del diagrama del lugar de raíz de bucle abierto De esta forma, se establece la respuesta ante salto unitario del sistema. Se utiliza como punto de partida la función de transferencia resultante del estudio matemático modelizado en el apartado anterior es decir :

$$P(s) = \frac{R(s)}{\theta(s)} = \frac{5\ g}{7\ s^2}$$

Ecuación 22 Función de transferencia

Tras ello se utilizará el estudio que nos proporciona Matlab mediante las líneas de código siguiente:

```
1.  >> s = tf ( 's' );
2.  >> P_ball = 5*9.8  / (7* s ^ 2);
3.  >> rlocus (P_ball)
```

Se obtiene el siguiente lugar de las raíces [1] [posición de los polos de lazo cerrado al variar la ganancia del sistema con realimentación unitaria]:

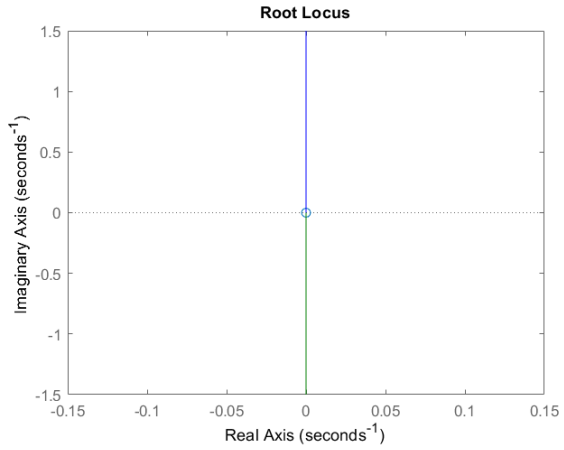


Ilustración 7 posición de los polos de lazo cerrado al variar la ganancia del sistema con realimentación unitaria

Como puede observarse, el sistema tiene dos polos en lazo abierto en el origen. Ahora vamos a realizar el estudio de la función en lazo cerrado para entrada escalón con una ganancia únicamente proporcional [2], mediante el siguiente código:

```
4.  P_ball = (5*9.8)  / (7* s ^ 2);
5.  Kp = 10;
6.  Kd = 0;
7.  Ki = 0;
8.  C = pid(Kp,Ki,Kd);
9.  sys_cl=feedback(C*P_ball,1);
10. t=0:0.01:5;
11. step(0.5*sys_cl)
12. title('funcion solo con KP')
13. axis([0 2 0 1])
```

con estas líneas obtenemos la respuesta del sistema ante tal entrada de forma gráfica [Ilustración 8 Salida ante entrada escalón y controlador KP] :

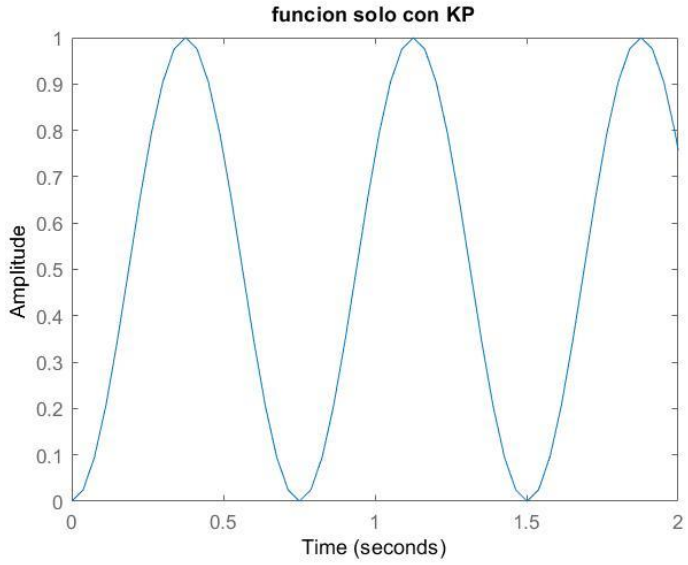


Ilustración 8 Salida ante entrada escalón y controlador KP

En dicha grafica se puede observar que la salida es inestable de tipo oscilante ya que como se puede ver en el lugar de las raíces, al variar la ganancia los polos de lazo cerrado siempre están en el eje imaginario. Para evitar esto, como se comprobará más adelante, se modela un control tipo proporcional derivativo con distintos valores de KD para ello utilizamos el siguiente código de Matlab [Ilustración 9 Respuesta escalón PD (kd bajo)]:

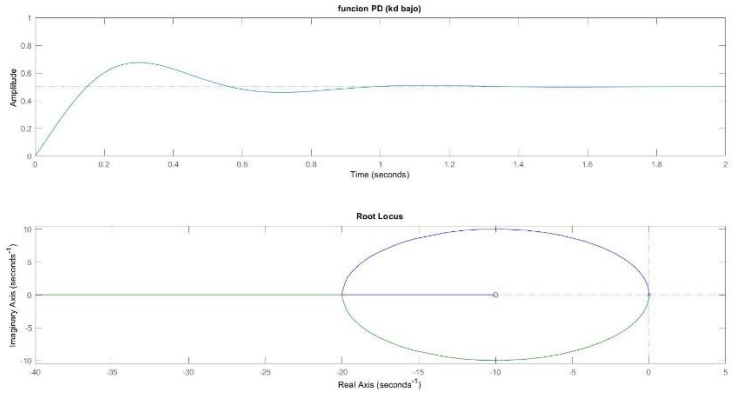


Ilustración 9 Respuesta escalón PD (kd bajo)

Como se puede observar, el sistema se vuelve estable ya que el lugar de las raíces se posiciona en el plano izquierdo de la gráfica. Además podemos observar que al introducir polos dominantes alejados del eje X el sistema generado es su amortiguado lo cual se corrige aumentando el valor de KP o disminuyendo el valor de KD (ya que modifica el cero) moviéndose a través del lugar de las raíces hasta posicionarse sobre el lugar de las raíces que está sobre el eje X[Ilustración 10 Respuesta escalón PD (kd alto)]:

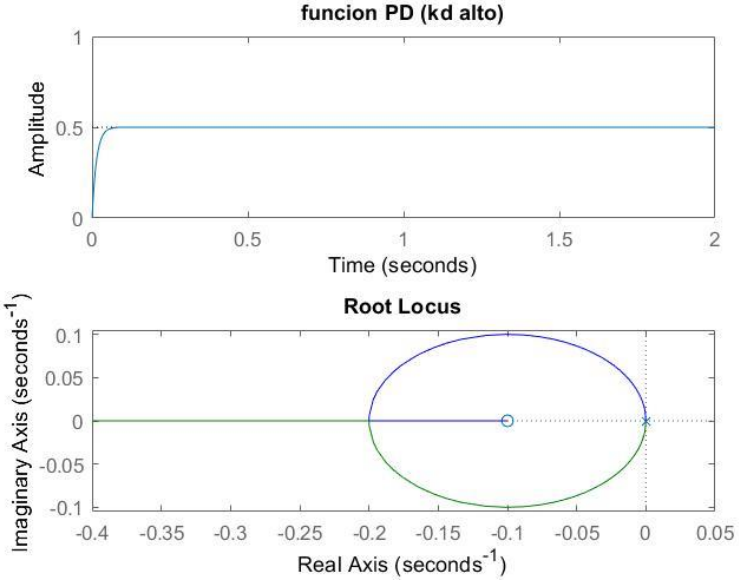


Ilustración 10 Respuesta escalón PD (kd alto)

Se puede inferir al aumentar el valor de KP que el punto sobre las ramas del lugar de

las raíces se posiciona sobre el eje X obteniendo un sistema críticamente amortiguado e incluso sobreamortiguado. Por tanto, se puede resolver que el sistema necesita un controlador de tipo proporcional derivativo, hecho que se demuestra posteriormente de forma experimental.

En caso de que el sistema tenga rozamiento se necesita un controlador PID debido a que de forma matemática no se tuvo en cuenta el rozamiento pero existe en el modelo real. Sin embargo para un caso donde la bola utilizada se deslice fácilmente sobre la viga se necesitará un valor mínimo de integración en las constantes KI para llegar al valor de referencia en estado estacionario [3] , aunque este hecho puede dar lugar a inestabilidades para KP bajos como se puede ver en esta grafica .

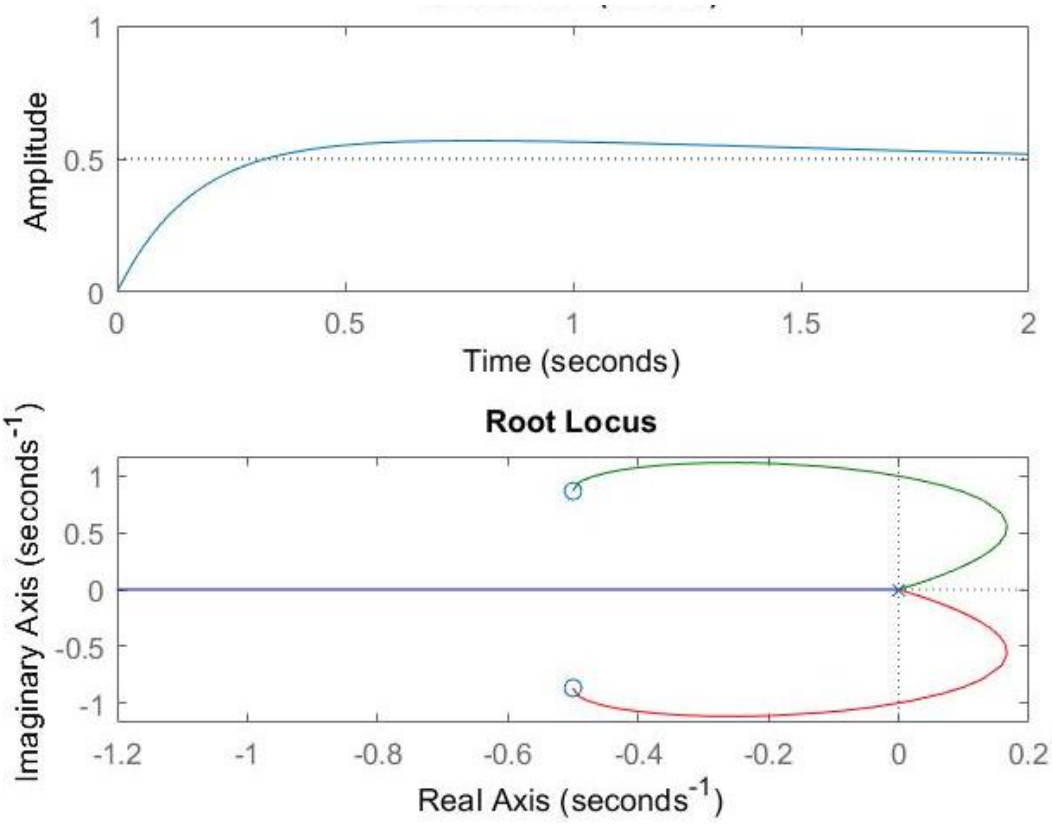


Ilustración 11 sistema con KI

El estudio también se realizó en el dominio de la transformada z para que sea más ajustado a la realidad ya que nuestro controlador es muestreado y se ajusta más a la maqueta propuesta. Para ello utilizamos otra vez Matlab, mediante las siguientes

instrucciones convertimos a dominio z.

```
14. T=20/1000; %Periodo de muestreo de 20ms
15. z=tf('z',T);
16. s=tf('s');
17. P_ball = (5*9.8)  / (7* s ^ 2);
18. P_ball_z=c2d(P_ball,T,'zoh');
```

Tras ello obtenemos la siguiente función:

$$P\_ball\_z = \frac{0.0014\,z + 0.0014}{z\text{^}2 - 2\,z + 1}$$

Podemos realizar el estudio de la estabilidad en lazo cerrando, realimentado el sistema e instalando un controlador con las siguientes líneas.

```
19. Ki=0;
20. Kd=1;
21. Kp=1;
22.
23. N=100; %En Z hay que aÒadir un polo m·s para que el controlador sea realizable
24. % Para que un sistema sea realizable no debe tener mas ceros que polos
25. %El valor de N=100 es orientativo. Es el valor por defecto que da simulink
26. pidz=Kp+Ki*T/(z-1)+Kd*N/(1+N*T/(z-1))
27. rlocus(P_ball_z*pidz)
28.
```

Obteniendo esta grafica de la estabilidad, no variando esencialmente de los resultados del dominio “s” ya que recordamos que en este caso la estabilidad se encuentra dentro del circulo unidad, salvo para valores muy altos de Kp que el sistema se vuelve inestable.

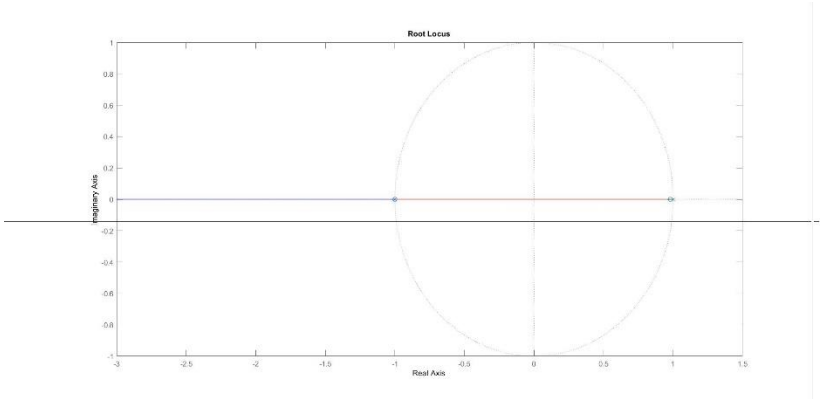


Ilustración 12 Lugar de las raíces en Z

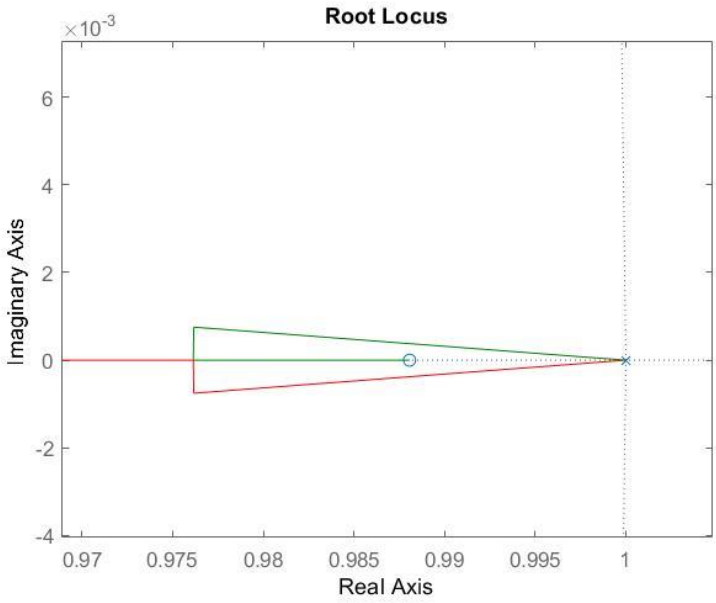


Ilustración 13 Detalle del lugar de las raíces en Z

4 PLANTEAMIENTO DEL PROBLEMA Y ELEMENTOS NECESARIOS

Para la realización del proyecto se han utilizado una serie de herramientas tecnológicas, las cuales nos facilitan el control y nos hacen posible la consecución de los objetivos fijados, las cuales describiremos en este capítulo.

4.1 PROTOCOLO I2C

I²C es un bus de comunicaciones en serie. Es ampliamente utilizado en la industria, principalmente para comunicar microcontroladores con sus periféricos en sistemas integrados (Embedded Systems) y, de manera más general, es usado para comunicar circuitos integrados entre sí, los cuales normalmente están en un mismo circuito impreso. Mediante este protocolo se comunica el sensor de posición VL53L0X con el controlador (la placa Raspberry Pi). El nombre de este protocolo proviene de Inter-Integrated Circuit (Inter-Circuitos Integrados). La velocidad de transmisión de datos es de 400 kbit/s para este sensor, siendo ésta la máxima velocidad aceptada, pero permitiendo velocidades más bajas.

La principal característica de I²C es que utiliza dos líneas para transmitir la información: una para los datos y otra para la señal de reloj. También es necesaria una tercera línea, pero ésta sólo es la referencia (masa). Como suelen comunicarse circuitos de una misma placa que comparten una misma masa, esta tercera línea no suele ser necesaria.

Estas líneas distintas se llaman:

SDA: datos (drenador abierto, por lo que necesitan resistencias de pull-up)

SCL: reloj (drenador abierto, por lo que necesitan resistencias de pull-up)

GND: tierra o masa

Los datos en este protocolo están estipulados previamente y tienen que seguir una serie estricta de “pasos” para el envío y recepción de la información.

El bus está libre cuando SDA y SCL están en estado lógico alto.

En estado bus libre, cualquier dispositivo puede ocupar el bus I²C como maestro.

El maestro comienza la comunicación enviando un patrón llamado "start condition". Esto alerta a los dispositivos esclavos, poniéndolos a la espera de una transmisión.

El maestro se dirige al dispositivo con el que quiere hablar, enviando un byte que contiene los siete bits (A7-A1) que componen la dirección del dispositivo esclavo con

el que se quiere comunicar, y el octavo bit (A0) de menor peso se corresponde con la operación deseada (L/E), lectura=1 (recibir del esclavo) y escritura=0 (enviar al esclavo).

La dirección enviada es comparada por cada esclavo del bus con su propia dirección, y si ambas coinciden, el esclavo se considera direccionado como esclavo-transmisor o esclavo-receptor dependiendo del bit L/E.

El esclavo responde enviando un bit de ACK que le indica al dispositivo maestro que el esclavo reconoce la solicitud y está en condiciones de comunicarse.

Seguidamente comienza el intercambio de información entre los dispositivos.

El maestro envía la dirección del registro interno del dispositivo que se desea leer o escribir.

El esclavo responde con otro bit de ACK.

Ahora el maestro puede empezar a leer o escribir bytes de datos. Todos los bytes de datos deben constar de 8 bits, el número máximo de bytes que pueden ser enviados en una transmisión no está restringido, siendo el esclavo quien fija esta cantidad de acuerdo a sus características.

Cada byte leído o escrito por el maestro debe ser obligatoriamente reconocido por un bit de ACK por el dispositivo maestro/esclavo.

Se repiten los 2 pasos anteriores hasta finalizar la comunicación entre maestro y esclavo.

Aun cuando el maestro siempre controla el estado de la línea del reloj, un esclavo de baja velocidad o que deba detener la transferencia de datos mientras efectúa otra función, puede forzar la línea SCL a nivel bajo. Esto hace que el maestro entre en un estado de espera, durante el cual, no transmite información esperando a que el esclavo esté listo para continuar la transferencia en el punto donde había sido detenida.

Cuando la comunicación finaliza, el maestro transmite una "stop condition" para dejar libre el bus.

Después de la "stop condition", es obligatorio para el bus estar ocupado durante unos microsegundos.

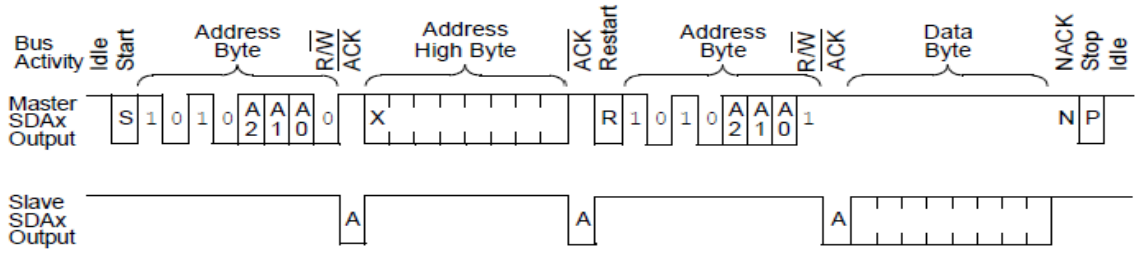


Ilustración 14 Modelo de comunicación I2c

La Raspberry Pi dispone de un módulo de I2C de alta velocidad integrado [Ilustración 15 Pinout de raspberry pi para I2c] siendo así muy simple la configuración, aunque se necesita una adaptación de voltajes ya que toda la placa de este microprocesador funciona a 3,3 voltios mientras que el sensor VL53L0X tiene un voltaje de funcionamiento de 5 voltios (normalmente).

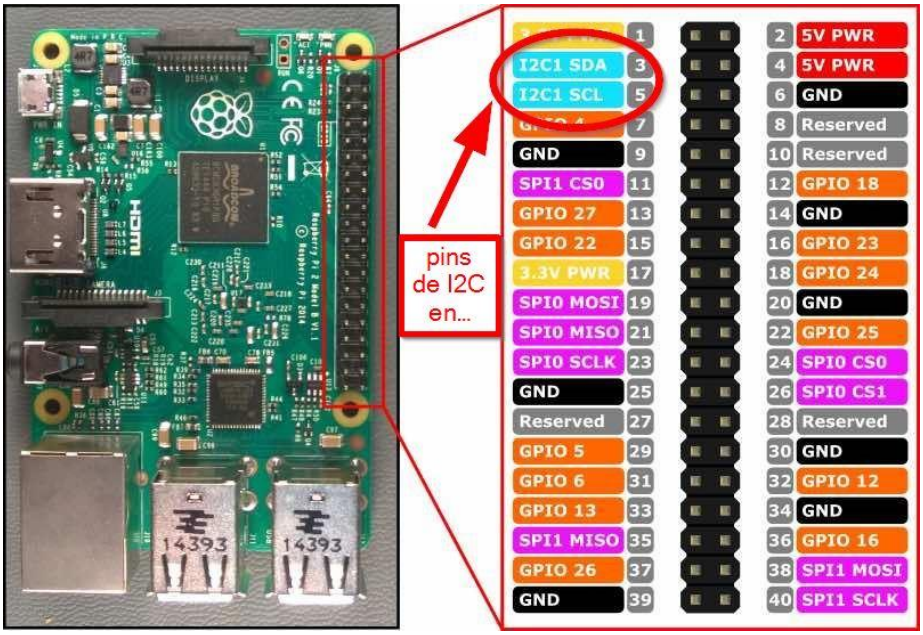


Ilustración 15 Pinout de raspberry pi para I2c

Las líneas de datos y de reloj no están en colector común y para el caso de Raspberry Pi se no recomienda colocar unas resistencias de pull-up de 2.2kΩ como en otro ya que se pueden modificar los voltajes y ensuciar la señal[Ilustración 16 Captura de comunicación I2c con osciloscopio], para crear el protocolo.

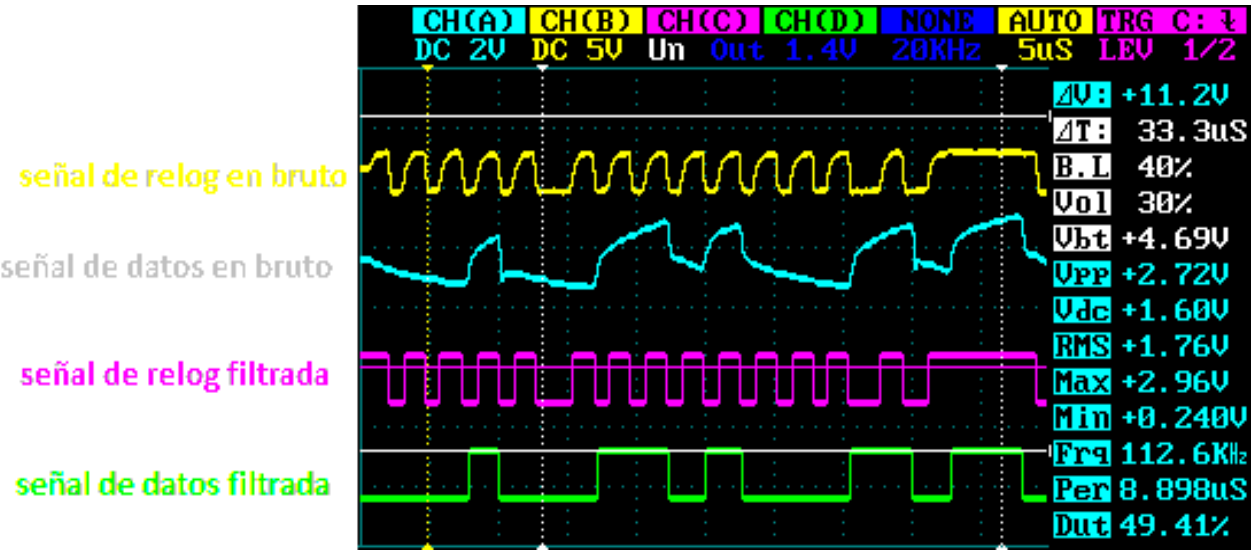


Ilustración 16 Captura de comunicación I2c con osciloscopio

4.2 SOCKET

Un socket (enchufe), es un método para la comunicación entre un programa cliente y un programa del servidor en una red Ethernet, se define, por tanto, como una conexión de tipo punto a punto [4].

Este método de comunicación nace a principio de los 80 a consecuencia del desarrollo de un sistema Unix en la Universidad de California (Berkey) para proporcionar un medio de comunicación entre distintos programas de dos máquinas físicamente separadas.

Los sockets necesitan el uso varios datos para empezar con la comunicación: la IP local de ambas maquinas, el número de puerto de las máquinas y por último el protocolo de transporte que se va a realizar.

Los requisitos de ambos programas para llevar a cabo la comunicación son los siguientes

- Que un programa puedo localizar al otro.
- Que ambos programas sean capaces de poder intercambiar una secuencia de 8 bit, es decir, datos relevantes a su finalidad.

Se necesitas además tres recursos bien diferenciados:

- Un protocolo de comunicación que permita el intercambio de grupos de 8 bit (octetos)
- Un par de direcciones IP cliente-servidor para identificar ambas máquinas
- Un par de número de puerto que identifica dentro de cada máquina el programa a usar.

Los sockets permiten la creación de arquitecturas peer to peer. El servidor queda a la escucha por un puerto específico esperando una petición de conexión por parte de un cliente, el cliente realizará una solicitud de emparejamiento. Tras confirmar los datos queda generado el socket entre los dos dispositivos a la espera de las trasmisiones de datos bidireccionales entre las dos máquinas.

Existen distintos tipos de socket, pero quedan clasificados en dos grandes grupos:

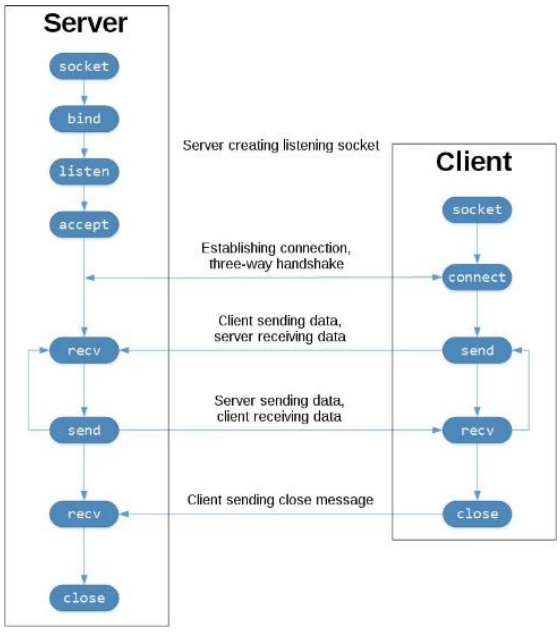
- Sockets de Flujo (SOCK_STREAM): asociado al protocolo TCP, destaca por seguridad en la transmisión de datos, seguridad en la recepción, en la integridad y en la secuencia, entre otros.
- Sockets de Datagramas (SOCK_DGRAM): asociado al protocolo UDP, los paquetes viajarán en tipo datagramas, el cual tiene una comunicación asíncrona en decir sin necesidad de una conexión.

Para este proyecto se ha utilizado SOCK_STREAM ya que este protocolo está libre de errores.

Además, la emisión y recepción de datos es en el mismo orden por el hecho de generar una conexión (asegura el orden de los objetos durante la transmisión) y permite una alta eficiencia a la hora de enviar un número elevado de mensajes y datos.
Tras la explicación del socket en general se abordará las principales funciones y métodos de la API de socket. En nuestro caso utilizaremos Python para la generación del socket por parte del servidor y Visual Basic por parte del cliente, pero las funciones y los métodos son similares en cualquier lenguaje de programación ya que están asociados al protocolo, los cuales se exponen a continuación:

- socket()
- bind()

- listen()
- accept()
- connect()
- connect_ex()
- send()
- recv()
- close()



La primera función por parte del servidor es la generación del socket() que el proceso por el que una máquina empieza a definir los parámetros internos del socket. Tras ello, enlaza el socket propiamente dicho con el programa interno que necesita el traspaso de datos con la función bind() y se queda a la espera de conexión listen(). Tras escuchar una conexión esta es aceptada accept(), y se produce el intercambio de datos bidireccionalmente entre el cliente y el servidor mediante procesos send() y recv().

Al finalizar el intercambio de datos el cliente envía un mensaje close() al servidor este desconecta el socket.

Para realizar una depuración de la conexión creada entre el servidor-cliente podemos realizar consultas en consola con las herramientas típicas de redes tanto de Linux como de Windows obtenido dichas capturas

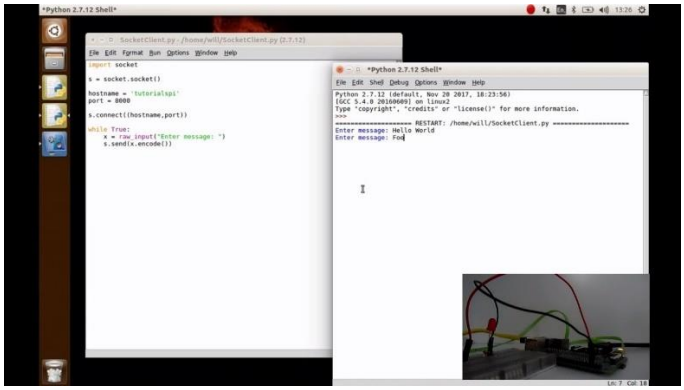


Ilustración 18 Pruebas socket en Ubuntu

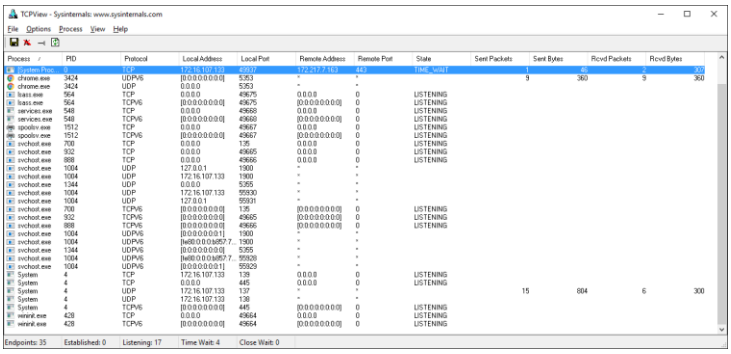


Ilustración 19 Pruebas socket en Windows

4.3 UART

"Universal Asynchronous Receiver-Transmitter" es un protocolo de comunicación utilizado para intercomunicar dispositivos. Este método trocea bytes de datos y transmite los bits individuales de una manera secuencial. En el destino, el segundo UART re-ensambla los bits en bytes completos. Cada UART contiene un registro de desplazamiento. La UART generalmente no genera o recibe las señales externas utilizadas entre los diferentes elementos del equipo. Se utilizan dispositivos de interfaz separados para convertir los niveles lógicos de las señales de la UART hacia y desde los niveles de señalización externos en caso de que hiciera falta, como RS-232 nuestro caso, el cual lo convierte de serial a USB.

RS-232 consiste en un bit de inicio, siete u ocho bits de datos, bits de paridad y bits de parada. Podemos observar un diagrama de tiempos para una trama RS-232, que consiste en un bit de inicio, 7 bits de datos, bits de paridad uno y dos bits de parada. Se debe tener en cuenta que la estructura exacta de la trama debe ser acordada por el transmisor y el receptor, y antes se debe abrir el puerto.

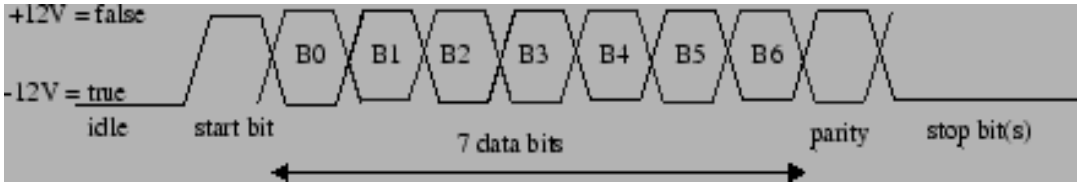


Ilustración 20 Trama de protocolo UART.

El bit de inicio se utiliza para señalar el comienzo de una trama y el bit de parada se utiliza para indicar el final de una trama. La paridad se utiliza para detectar errores de transmisión. Para la comprobación de paridad par, el número de 1 de los datos más el bit de paridad debe ser igual a un número par. Para paridad impar, esta suma debe ser un número impar. Los bits de paridad se utilizan para detectar errores en los datos transmitidos. Antes de enviar una trama, el transmisor establece el bit de paridad de manera que el marco tiene paridad par o impar. El receptor y el transmisor ya han acordado que tipo de comprobación de paridad (par o impar) se está utilizando. Cuando se recibe la trama, entonces el receptor comprueba la paridad de la trama recibida. Si la paridad es incorrecta, entonces el receptor sabe que se ha producido un error en la transmisión, y el receptor puede solicitar a los transmisores el reenvío de la trama.

La Raspberry Pi en su modulo GPIO tiene incorporado un módulo UART, teniendo un voltaje de 3,3 V necesitando placa de adaptación para el uso normal de UART de 5 voltios, por medio de una placa.

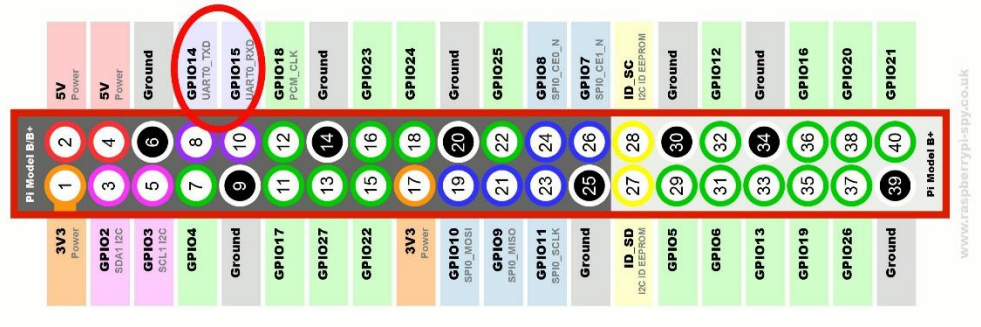


Ilustración 21 Pinout serial de Raspberry Pi

La configuración del UART se realizó con los siguientes parámetros de configuración:

Velocidad de trasmisión 115200 baudios

Bits de stop 1 bit

Bit de paridad 1 bit

Nº de bit trasmitidos 8 bit

Esta conexión se utiliza para intercomunicar el ordenador con el microprocesador (Raspberry pi), por lo cual el envío suele realizarse mediante código ASCII siendo éste por definición “código de caracteres basado en el alfabeto latino, tal como se usa en inglés moderno y en otras lenguas occidentales”. Fue creado en 1963 por el Comité Estadounidense de Estándares (ASA, conocido desde 1969 como el Instituto

Estadounidense de Estándares Nacionales, o ANSI) como una refundición o evolución de los conjuntos de códigos utilizados entonces en telegrafía.

El código ASCII utiliza 7 bits para representar los caracteres, pero normalmente se usa una extensión que utiliza 8 bits para proporcionar caracteres adicionales usados en idiomas distintos al inglés, como por ejemplo el idioma español.

Cada 7 bit se representa un carácter alfanumérico, como se puede ver en la siguiente tabla:

Dec	Hx	Oct	Char	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr	Dec	Hx	Oct	Html	Chr
0	0	000	NUL (null)	32	20	040		Space	64	40	100	@	@	96	60	140	`	`
1	1	001	SOH (start of heading)	33	21	041	!	!	65	41	101	A	A	97	61	141	a	a
2	2	002	STX (start of text)	34	22	042	"	"	66	42	102	B	B	98	62	142	b	b
3	3	003	ETX (end of text)	35	23	043	#	#	67	43	103	C	C	99	63	143	c	c
4	4	004	EOT (end of transmission)	36	24	044	$	$	68	44	104	D	D	100	64	144	d	d
5	5	005	ENQ (enquiry)	37	25	045	%	%	69	45	105	E	E	101	65	145	e	e
6	6	006	ACK (acknowledge)	38	26	046	&	&	70	46	106	F	F	102	66	146	f	f
7	7	007	BEL (bell)	39	27	047	'	'	71	47	107	G	G	103	67	147	g	g
8	8	010	BS (backspace)	40	28	050	(	(	72	48	110	H	H	104	68	150	h	h
9	9	011	TAB (horizontal tab)	41	29	051	)	)	73	49	111	I	I	105	69	151	i	i
10	A	012	LF (NL line feed, new line)	42	2A	052	*	*	74	4A	112	J	J	106	6A	152	j	j
11	B	013	VT (vertical tab)	43	2B	053	+	+	75	4B	113	K	K	107	6B	153	k	k
12	C	014	FF (NP form feed, new page)	44	2C	054	,	,	76	4C	114	L	L	108	6C	154	l	l
13	D	015	CR (carriage return)	45	2D	055	-	-	77	4D	115	M	M	109	6D	155	m	m
14	E	016	SO (shift out)	46	2E	056	.	.	78	4E	116	N	N	110	6E	156	n	n
15	F	017	SI (shift in)	47	2F	057	/	/	79	4F	117	O	O	111	6F	157	o	o
16	10	020	DLE (data link escape)	48	30	060	0	0	80	50	120	P	P	112	70	160	p	p
17	11	021	DC1 (device control 1)	49	31	061	1	1	81	51	121	Q	Q	113	71	161	q	q
18	12	022	DC2 (device control 2)	50	32	062	2	2	82	52	122	R	R	114	72	162	r	r
19	13	023	DC3 (device control 3)	51	33	063	3	3	83	53	123	S	S	115	73	163	s	s
20	14	024	DC4 (device control 4)	52	34	064	4	4	84	54	124	T	T	116	74	164	t	t
21	15	025	NAK (negative acknowledge)	53	35	065	5	5	85	55	125	U	U	117	75	165	u	u
22	16	026	SYN (synchronous idle)	54	36	066	6	6	86	56	126	V	V	118	76	166	v	v
23	17	027	ETB (end of trans. block)	55	37	067	7	7	87	57	127	W	W	119	77	167	w	w
24	18	030	CAN (cancel)	56	38	070	8	8	88	58	130	X	X	120	78	170	x	x
25	19	031	EM (end of medium)	57	39	071	9	9	89	59	131	Y	Y	121	79	171	y	y
26	1A	032	SUB (substitute)	58	3A	072	:	:	90	5A	132	Z	Z	122	7A	172	z	z
27	1B	033	ESC (escape)	59	3B	073	;	;	91	5B	133	[	[	123	7B	173	{	{
28	1C	034	FS (file separator)	60	3C	074	<	<	92	5C	134	\	\	124	7C	174			|
29	1D	035	GS (group separator)	61	3D	075	=	=	93	5D	135	]	]	125	7D	175	}	}
30	1E	036	RS (record separator)	62	3E	076	>	>	94	5E	136	^	^	126	7E	176	~	~
31	1F	037	US (unit separator)	63	3F	077	?	?	95	5F	137	_	_	127	7F	177	&#127;	DEL

Ilustración 22 Tabla de caracteres ASCII

Podemos enviar también datos en otros lenguajes, como se hará en el caso de los datos para usar en la gráfica, donde se usa un protocolo que consume menos bits para enviar la misma cantidad de información, básicamente enviando los datos casi en bruto.

El protocolo usado para graficar es uno específico de la aplicación consistente en el envío de tres bloques: en el primero enviamos una cabecera con 4 valores “CDAB”, en el segundo enviamos el tamaño del valor a enviar y, por último, el valor. La ventaja respecto al sistema anterior es que los datos no van codificados en ASCII lo cual consume muchos menos bits, y por tanto, menos tiempo.

Cabecera	Tamaño del valor	Valor
CDAB	2 bits	Valor de los 2 bits

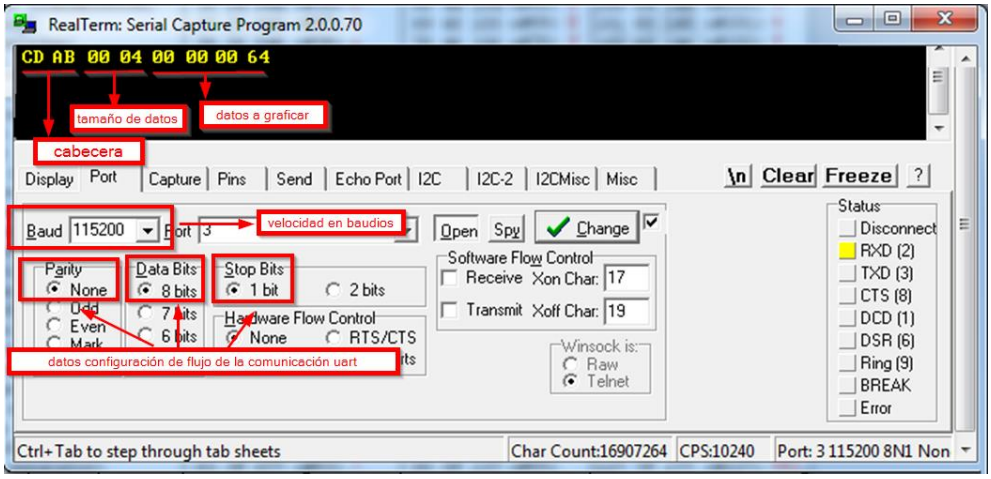


Ilustración 23 Ejemplo de escritura para graficar

Los datos enviados se pueden comprobar con un osciloscopio o con un analizador lógico, viendo así el correcto funcionamiento del protocolo.

En la siguiente imagen se puede ver el envío de la letra “a” en ASCII:

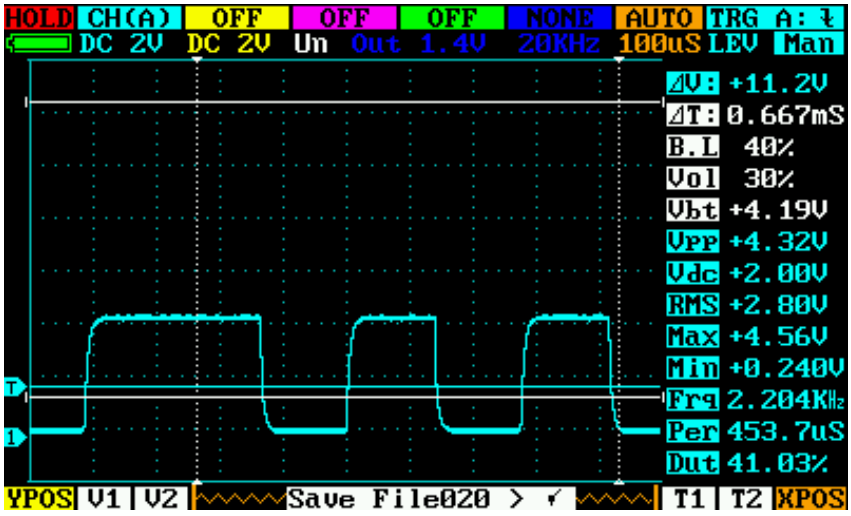


Ilustración 24 Captura con osciloscopio del protocolo UART de la letra “a” en ASCII

4.4 SERVOMOTOR

Un servomotor en un tipo específico de motor con la capacidad especifica de controlar la posición de giro de este. Este dispositivo está formado por componentes eléctricos mecánicos y electromecánicos.

En el interior de este dispositivo se encuentra un motor de DC. Al eje de éste se acopla una caja de engranes la cual transmite el movimiento a otro eje este tipo de engranaje es un engranaje reductor, reduciendo la velocidad del motor de DC y proporcionando más par a la salida.

En el eje del servomotor se encuentra un sensor de posición o encoder normalmente una resistencia variable del giro, pudiendo el dispositivo obtener con facilidad y rapidez la posición del eje en cualquier momento.

Partes de las que se compone [Ilustración 25 Esquema de servomotor]:

- Un motor eléctrico: dispositivo interno encargado de generar el movimiento circular a través de su eje.
- Dispositivo reductor: encargado de reducir la velocidad de la salida del eje del motor a una velocidad apropiada para nuestro sistema.
- Un sistema de control: sistema automático permite controlar el movimiento del eje de salida del servo mediante el uso de pulsos eléctricos.
- Un potenciómetro: Dispositivo eléctrico conectado al eje central o de salida y permite en todo momento saber el ángulo en el que se encuentra posicionado el eje del motor.

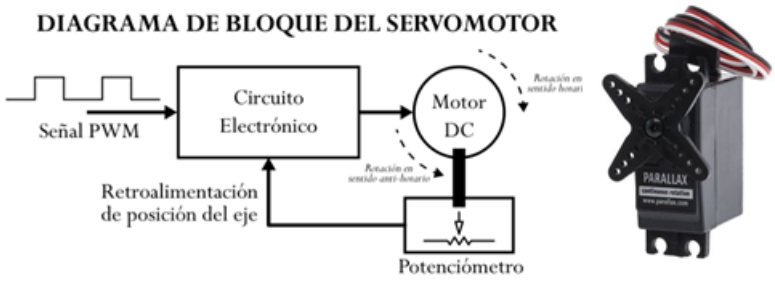


Ilustración 25 Esquema de servomotor

En nuestro caso se ha utilizado un servomotor de aeromodelismo el cual opera con voltajes bajo entre 4 y 6 voltios con un giro de 180 grados. Con esta amplitud de giro nos es suficiente para poder modificar el sistema y poder llegar a un control estable del mismo.

El funcionamiento del servomotor es distinto al de un motor convencional ya que no trabaja con tecnología analógica sino digital. la conexión de un servo está compuesta de 3 cables

Ilustración 26 Código de colores para conexionado de servomotor

Para el cableado conectaremos el conductor positivo y la tierra a la fuente de alimentación, el tercer cable nos sirve para proporcionarle información sobre la posición del eje.

El envío de información se realiza mediante el protocolo PWM a frecuencia de actuación normalmente prefijada de 50 Hz, donde el ancho de pulso de alto proporciona el ángulo en grados del eje, variando este de 0,5 a 2,5 milisegundo y de 0 a 180 grados[Ilustración 27 Señales digitales de control de servos].

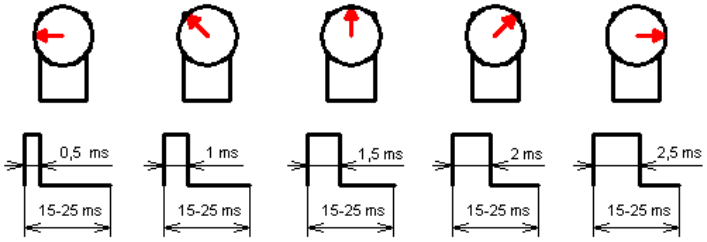


Ilustración 27 Señales digitales de control de servos

4.5 CONTROLADOR PID

4.5.1 Introducción.

Es una estructura de control universalmente utilizada en la industria. Se trata de la familia de controladores de estructura fija, llamada familia de controladores PID. Sirve para contralar cualquier sistema (SISO) y como cualquier el control automático, asienta sus bases esencialmente en el concepto de realimentación.

Estos controladores han demostrado ser robustos y extremadamente beneficiosos (costos, mantenimiento, soporte) en el control de muchas aplicaciones de importancia en la industria.

Históricamente, ya las primeras estructuras de control usaban las ideas del control PID. Sin embargo, no fue hasta el trabajo de Minorsky de 1922, sobre la conducción de barcos, que el control PID cobró verdadera importancia teórica.

Hoy en día, a pesar de la abundancia de sofisticadas herramientas y métodos avanzados de control, el controlador PID es aún el más ampliamente utilizado en la industria moderna, controlando más del 95% de los procesos industriales de lazo cerrado. [5]

4.5.2 Estructura del PID.

El controlador resta la señal de punto actual a la señal de punto de consigna, obteniendo así la señal de error, que determina en cada instante la diferencia que hay entre el valor deseado y el valor medido. La señal de error es utilizada por cada una de las 3 componentes de un controlador PID propiamente tal, para generar las 3 señales que, sumadas, componen la señal de salida que el controlador va a utilizar para gobernar al actuador. La señal resultante de la suma de estas tres señales, se llama variable manipulada y no se aplica directamente sobre el actuador, sino que debe ser transformada para ser compatible con el actuador que usemos.

4.5.2.1 Termino proporcional.

El componente proporcional es uno de los elementos que componen el PID y éste sólo depende de la diferencia entre el punto de consigna y el estado actual del proceso. Esta

diferencia entre estos dos términos se conoce como el término de error o error proporcional. La ganancia proporcional (K_p) determina la proporción de respuesta de salida a la señal de error. Es decir, nos indica el porcentaje de salida que "contrarrestamos" en proporción lineal a la entrada.

Por ejemplo, si el término de error tiene una magnitud de 10 cm en nuestra medida de la distancia viga bola, y nuestra ganancia proporcional es de 5, produciría una respuesta proporcional de 50 en la señal al servomotor proporcionado por la Raspberry y, por lo tanto, aumentará el ángulo del servomotor en dirección opuesta.

En general, el aumento de la ganancia proporcional incrementará la velocidad de la respuesta del sistema de control. Sin embargo, si la ganancia proporcional es demasiado grande, la variable de proceso comenzará a oscilar, por que estaremos en la frontera de estabilidad en lugar de las raíces cero en el eje de las Y.

Si K_p se incrementa aún más, las oscilaciones se hacen más grandes y el sistema se vuelve inestable e incluso puede oscilar fuera de control debido a que nos encontramos en la parte derecha del lugar de las raíces.

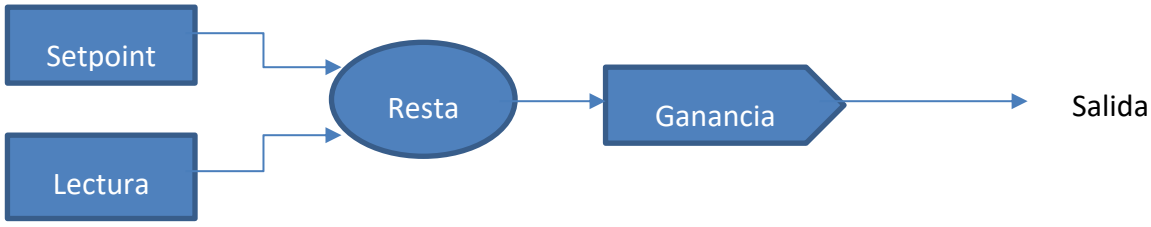


Ilustración 28 Explicación gráfica del actuador proporcional

4.5.2.2 Termino integral

El componente integral reduce el término de error en el tiempo. El resultado es que incluso un pequeño término de error hará que el componente integral aumente lentamente. La respuesta integral aumentará continuamente a través del tiempo a menos que el error es cero, por lo que el efecto es de conducir el error estacionario a cero.

El sistema a controlar es claramente un sistema discretizado, ya que pasa por un microcontrolador y esto hace que la lectura de los sensores se desarrolle en periodos de tiempo establecido. Por lo tanto, se necesita muestrear la señal para que sea similar a la continua, y para ello se utilizaran distintos métodos, llamados métodos de discretización.

Los métodos más importantes son tres:

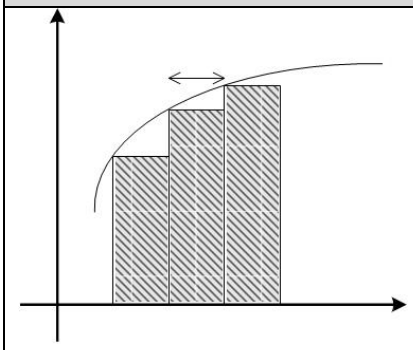
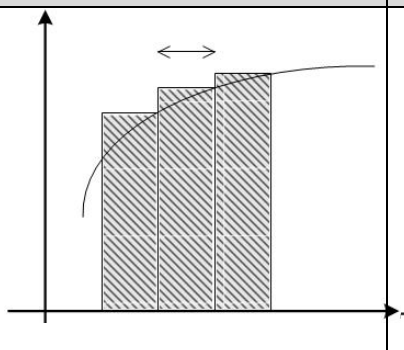
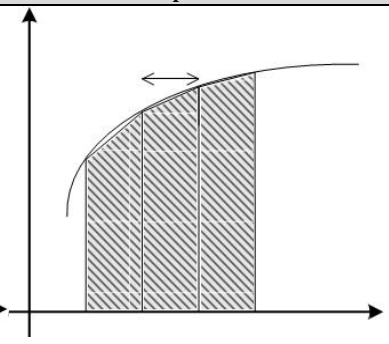
Euler hacia delante	Euler hacia atrás	Trapezoidal
		
$\int_0^t e(\tau)d\tau = \sum_{i=0}^{k-1} e(i)T = \sum_{i=0}^{k-1} Te_i$	$\int_0^t e(\tau)d\tau = \sum_{i=1}^{k} e(i)T = \sum_{i=1}^{k} Te_i$	$\int_0^t e(\tau)d\tau = \sum_{i=1}^{k} T\frac{e_i + e_{i-1}}{2}$

Ilustración 29 Explicación gráfica del actuador proporcional

En nuestro caso, se realizó una aproximación de Euler hacia atrás debido a que nuestro tiempo de muestreo puede ser muy bajo de manera que no se nota de manera real el método utilizado a efectos prácticos. Aunque se realizaron las pruebas de implementación de otros métodos, no se notó mejora. Además, este método tiene la virtud de no cargar el microprocesador con operaciones, lo cual mejora los tiempos de ciclado ajustándose más a un sistema continuo y siendo así más fácil de estabilizar el sistema.

4.5.2.3 Termino derivativo.

El componente derivativo hace que la salida disminuya si la variable de proceso está aumentando rápidamente. La respuesta derivativa es proporcional a la velocidad de cambio de la variable de proceso. El aumento del parámetro tiempo de derivada (D_T) hará que el sistema de control reaccione más rápidamente a los cambios en el término de error y aumentará la velocidad de la respuesta general del sistema de control. La mayoría de los sistemas de control tienen el tiempo de derivada (T_d) muy pequeño, porque la respuesta derivativa es altamente sensible al ruido en la señal de variable de proceso, pero en este caso los filtros ayudan en gran medida a que este término no sea tan reactivo frente al ruido ya que estos están muy filtrados y atenuados. Si la señal de realimentación del sensor es ruidosa, está mal filtrado, o si el tipo de muestreo es demasiado lento, la respuesta derivativa puede hacer que el sistema de control sea

inestable.

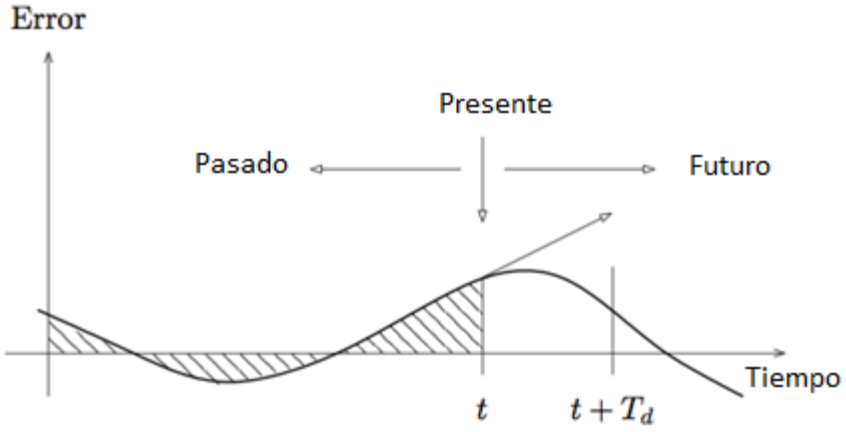


Ilustración 30Gráfica del actuador derivativo

4.6 RASPBERRY PI 3

4.6.1 Introducción.

Raspberry Pi es una placa computadora (SBC) de bajo costo desarrollada en Reino Unido por la Fundación Raspberry Pi, con el objetivo de estimular la enseñanza de ciencias de la computación en las escuelas. En realidad, se trata de una diminuta placa base de 85 x 54 milímetros en el que se aloja un chip Broadcom con procesador ARM de distintas series en función de la versión de la Raspberry Pi que utilicemos de entre 1 y 1,6 GHz y una memoria variable según el modelo. Para este caso se usa la Raspberry Pi, pero la aplicación realizada en Python es ejecutable en cualquier versión de Raspberry Pi.

A continuación se expondrá las distintas versiones de Raspberry Pi y sus características:

Especificaciones	Raspberry Pi 1 modelo A (descontinuada)	Raspberry Pi 1 modelo B (descontinuada)	Raspberry Pi 1 modelo B+	Raspberry Pi 2 modelo B	Raspberry Pi 3 modelo B	Raspberry Pi 3 modelo B+	Raspberry Pi 3 modelo A+	Raspberry Pi 4 modelo B

SoC (CPU, GPU, DSP, RAM y puertos USB)	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2837		Broadcom BCM2711
CPU	ARM 1176JZF-S a 700 MHz (familia ARM11)	900 MHz quad-core ARM Cortex A7	1.2GHz 64-bit quad-core ARMv8	1.4GHz 64-bit quad-core ARMv8	1.5GHz 64-bit quad-core Cortex-A72
Juego de instrucciones	RISC de 32 bits		RISC de 64 bits		
GPU	Broadcom VideoCore IV, OpenGL ES 2.0, MPEG-2 y VC-1 (con licencia), 1080p30 H.264/MPEG-4 AVC				Broadcom VideoCore VI, OpenGL ES 3.0, 1080p30 H.264/MPEG-4 AVC, 4kp60 H.265
Memoria	256 MB (compartidos con la GPU)	512 MiB (compartidos con la GPU)	1 GB (compartidos con la GPU)	512 MiB (compartidos con la GPU)	1 GB, 2 GB o 4 GB (compartidos con la GPU)
Puertos USB 2.0	1	2 (vía hub USB integrado)	4	1	2
Puertos USB 3.0	Ninguno				2
Entradas de vídeo	Conector MIPI CSI que permite instalar un módulo de cámara desarrollado por la Raspberry Pi Foundation				
Salidas de vídeo	Conector RCA (PAL y NTSC), HDMI (rev1.3 y 1.4), Interfaz DSI para panel LCD				Conector RCA (PAL y NTSC), microHDMI rev. 2.0, Interfaz DSI para panel LCD
Salidas de audio	Jack de 3.5 mm, HDMI				Jack de 3.5 mm, 2 puertos microHDMI

Almacenamiento	SD / MMC / ranura para SDIO							MicroSD
Conectividad de red	Ninguna	10/100 Ethernet (RJ-45) vía hub USB		Puerto RJ-45 (ethernet) de 10/100 Mbps vía hub USB Wi-Fi 802.11bgn Bluetooth 4.1	Puerto RJ-45 (Ethernet) de 10/100/1000Mbps vía hub USB limitado a 300Mbit/s Wi-Fi 802.11ac de doble banda Bluetooth 4.2 BLE	Wifi 802.11ac de doble banda Bluetooth 4.2 BLE	Puerto RJ-45 10/100/1000Mbps vía hub USB 3.0 Wi-Fi 802.11ac de doble banda Bluetooth 5.0 BLE	
Periféricos de bajo nivel	8 x GPIO, SPI, I^2C, UART			17 x GPIO y un bus HAT ID				
Consumo energético	500 mA (2.5 W)	700 mA (3.5 W)	600 mA (3.0 W)	800 mA (4.0 W)				Máximo 3A (15.3 W)
Fuente de alimentación	5 V vía Micro USB o puerto GPIO							5 V vía USB-C o puerto GPIO
Dimensiones	85mm x 53mm							
Sistemas operativos soportados	GNU/Linux: Raspbian, Fedora (Pidora), Arch Linux (Arch Linux ARM), Slackware Linux, SUSE Linux Enterprise Server for ARM, RISC OS							GNU/Linux: Raspbian
Especificaciones	Raspberry Pi 1 modelo A (descontinuada)	Raspberry Pi 1 modelo B (descontinuada)	Raspberry Pi 1 modelo B+	Raspberry Pi 2 modelo B	Raspberry Pi 3 modelo B	Raspberry Pi 3 modelo B+	Raspberry Pi 3 modelo A+	Raspberry Pi 4 modelo B

Para que funcione, necesitamos de un medio de almacenamiento (Raspberry Pi utiliza tarjetas de memoria SD o microSD dependiendo del modelo), conectarlo a la corriente utilizando cualquier cargador microUSB de al menos 1000mah para las placas antiguas y de al menos 2500mah para las modernas, (2000mah para la última versión de raspberry versión 4) y si lo deseamos, protegerlo todo utilizando una carcasa en caso

de no necesitar los puertos GPIO. En nuestro caso se le va a colocar una placa para facilitar la adaptación de voltajes y el conexionado con el modelo de viga-bola.
En función del modelo que escojamos, dispondremos de más o menos opciones de conexión, aunque siempre dispondremos de al menos un puerto de salida de vídeo HDMI y otro de tipo RCA, minijack de audio y un puerto USB 2.0 (modelos A y A+, el modelo B dispone de dos USB y B+, Raspberry Pi 2 y Raspberry Pi 3 disponen de 4 USB) al que se puede conectar un teclado y un ratón.
En cuanto a la conexión de red, disponemos de un puerto Ethernet (los modelos A y A+ no disponen de puerto Ethernet) para realizar la conexión de un cable RJ-45 directamente al router o podemos recurrir a utilizar cualquier adaptador inalámbrico WiFi compatible. La versión 3 y siguientes disponen de adaptador WiFi integrado.

4.6.2 Hardware del raspberry pi y características.

4.6.2.1 Puerto GPIO

General Purpose Input Output (GPIO) es un sistema de entrada y salida de propósito general, es decir, consta de una serie de pines o conexiones que se pueden usar como entradas o salidas para múltiples usos. Estos pines están incluidos en todos los modelos de Raspberry Pi aunque con diferencias.

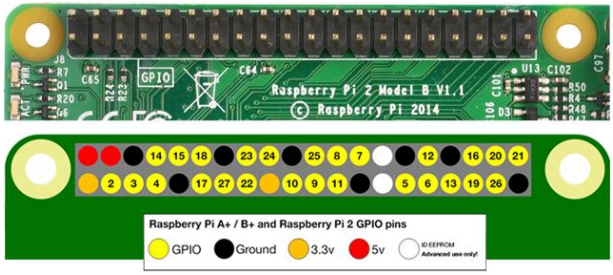


Ilustración 31 Pinout de raspberry pi

Hay que tener en cuenta que dependiendo del modelo de la Raspberry Pi encontramos una cantidad de pines diferentes, por ejemplo, en la versión 1 de Raspberry Pi se tienen 26 pines GPIO mientras que a partir de la versión 2 de Raspberry Pi el número de pines aumentó a 40. Sin embargo, la compatibilidad es total, puesto que los 26 primeros pines

mantienen su función original.

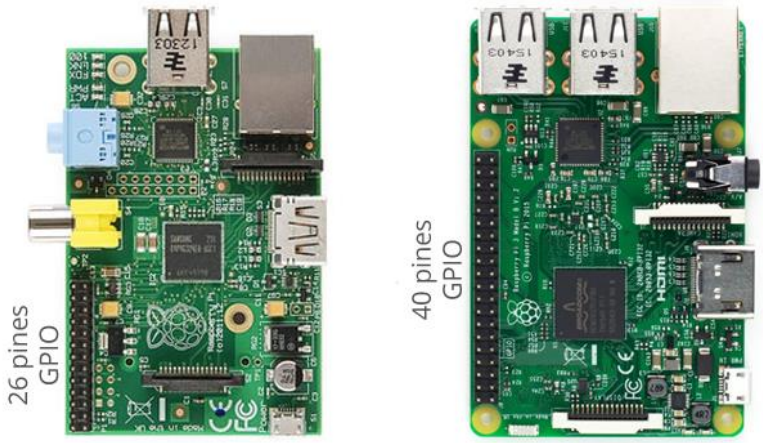


Ilustración 32 Distincion de pinout entre tipos de raspberry pi

Como se observa, el número de pines pasó de 26 a 40 para tener más disponibilidad, aunque volvemos a comentar que los 26 primeros pines son comunes para todas las versiones. Los pines GPIO tienen funciones específicas (aunque algunos comparten funciones) y se pueden agrupar de la siguiente manera:

Amarillo (2): Alimentación a 3.3V.
Rojo (2): Alimentación a 5V.
Naranja (26): Entradas / salidas de propósito general. Pueden configurarse como entradas o salidas. Ten presente que el nivel alto es de 3.3V y no son tolerantes a tensiones de 5V, dato que será importante como veremos a continuación.
Gris (2): Reservados.
Negro (8): Conexión a GND o masa.
Azul (2): Comunicación mediante el protocolo I2C para comunicarse con periféricos como el sensor de este proyecto (VL53L0X).
Verde (2): Destinados a conexión para UART para puerto serie convencional utilizado en caso de no utilizar el protocolo a distancia.
Morado (5): Comunicación mediante el protocolo SPI para comunicarse con periféricos que siguen este protocolo.

Todos los pines son de tipo "unbuffered", es decir, no disponen de buffers de protección

y la placa puede dañarse con un mal uso.

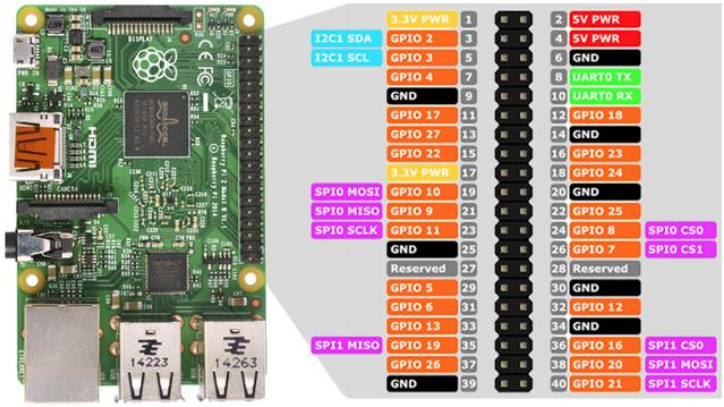


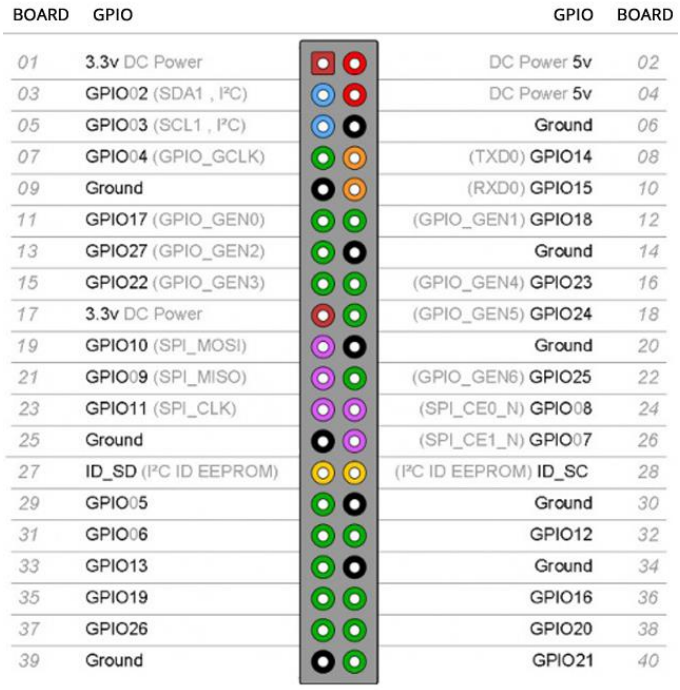
Ilustración 33 Funciones de pinout en Raspberry Pi

Existen 2 formas de numerar los pines de la Raspberry Pi, en modo GPIO o en modo BCM.

En el modo GPIO, los pines se numeran de forma física por el lugar que ocupan en la placa (representados por el color gris), y se mantiene igual para todas las versiones (comenzamos a contar desde arriba a la izquierda y finalizamos abajo a la derecha).

En el modo BCM, los pines se numeran por la correspondencia en el chip Broadcom (que es la CPU de la Raspberry Pi).

Por este mismo motivo se pueden encontrar 2 nomenclaturas a la hora de realizar las conexiones de electrónica con Raspberry Pi, cuando nos refiramos al modo GPIO o al modo BCM. A continuación, mostramos una tabla de equivalencias.



BOARD	GPIO		GPIO	BOARD
01	3.3v DC Power		DC Power 5v	02
03	GPIO02 (SDA1 , I²C)		DC Power 5v	04
05	GPIO03 (SCL1 , I²C)		Ground	06
07	GPIO04 (GPIO_GCLK)		(TXD0) GPIO14	08
09	Ground		(RXD0) GPIO15	10
11	GPIO17 (GPIO_GEN0)		(GPIO_GEN1) GPIO18	12
13	GPIO27 (GPIO_GEN2)		Ground	14
15	GPIO22 (GPIO_GEN3)		(GPIO_GEN4) GPIO23	16
17	3.3v DC Power		(GPIO_GEN5) GPIO24	18
19	GPIO10 (SPI_MOSI)		Ground	20
21	GPIO09 (SPI_MISO)		(GPIO_GEN6) GPIO25	22
23	GPIO11 (SPI_CLK)		(SPI_CE0_N) GPIO08	24
25	Ground		(SPI_CE1_N) GPIO07	26
27	ID_SD (I²C ID EEPROM)		(I²C ID EEPROM) ID_SC	28
29	GPIO05		Ground	30
31	GPIO06		GPIO12	32
33	GPIO13		Ground	34
35	GPIO19		GPIO16	36
37	GPIO26		GPIO20	38
39	Ground		GPIO21	40

Ilustración 34 Diferencias entre GPIO o al modo BCM

De los pines GPIO disponibles, hay una serie de pines con capacidad de PWM (como veremos más adelante, útiles para la salida del servo). Sin embargo, no se dispone de ningún convertidor de analógico a digital. Esto quiere decir que para medir valores de sensores analógicos necesitaremos utilizar un convertidor externo. Habrá que tener en cuenta que cuando se utilizan los pines de GPIO hay que prestar mucha atención para no dañar la propia Raspberry Pi. Es muy importante comprobar los niveles de tensión y la corriente solicitada. Los pines de GPIO pueden generar y consumir tensiones compatibles con los circuitos de 3.3V. No es posible conectar componentes de 5V, puesto que se podría quemar el chip y dañar la Raspberry Pi. Los pines GPIO ofrecen una tensión de 3.3V y no toleran a tensiones de 5V, por ello en este proyecto se utilizará la Crowtail- Base Shield siendo esta una placa de integracion y adaptacion de voltaje, la cual se describirá en los siguientes puntos.

Hay que tener presente que la intensidad de corriente que sale de esos pines proviene de la fuente de 3.3V y esta fue diseñada para una carga de unos 3mA por cada pin GPIO, suficiente para encender diodos led, pero no para dispositivos que soliciten una mínima potencia.

4.6.1 Software de Raspberry Pi RASBIAN (Sistema Operativo)

Según la propia definición dada por los creadores “Raspbian es un sistema operativo gratuito basado en Debian optimizado para el hardware Raspberry Pi. Un sistema operativo es el conjunto de programas y utilidades básicos que hacen que su Raspberry Pi se ejecute”.

Raspbian es un sistema operativo basado en debían puro que viene incorporado más de 35,000 paquetes, software recompilado todo ello con formato agradable, amigable y de fácil manejo.
El desarrollo primigenio de Raspbian se completó en junio de 2012. Sin embargo, Raspbian todavía está en desarrollo activo con un énfasis en mejorar la estabilidad y el rendimiento de la mayor cantidad posible de paquetes de Debian.

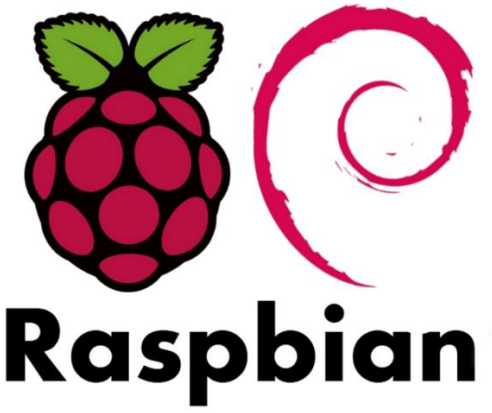


Ilustración 35 Símbolo Raspberry Pi

Una característica que posee es la capacidad de recompilar el código lo cual significa que el código abierto puede ser compilado para su propia arquitectura (armhf), sin tener que utilizar compiladores cruzados. Además, esta distribución, como la mayoría, contiene repositorios de donde el usuario puede descargar multitud de programas como si se tratase de una distribución de GNU/Linux para equipos de escritorio.
En la actualidad Raspbian se proporciona en dos modelos.

- Raspbian lite

Raspbian Lite es un sistema operativo Linux muy básico con solo una interfaz de línea

de comandos (CLI) [Ilustración 36 Conexión a Raspberry pi modo consola]. No hay un escritorio o una GUI de ningún tipo, y no incluye el software educativo o de productividad incluido con la versión de escritorio. Toda la iteración con el sistema operativo es mediante consola. No recomendada para usuarios nuevos y personas no familiarizadas con Linux.

Este formato es el utilizado para el desarrollo de nuestro proyecto debido a que no se necesita interfaz gráfica y su uso puede producir retrasos y consumo de recursos de forma innecesaria.

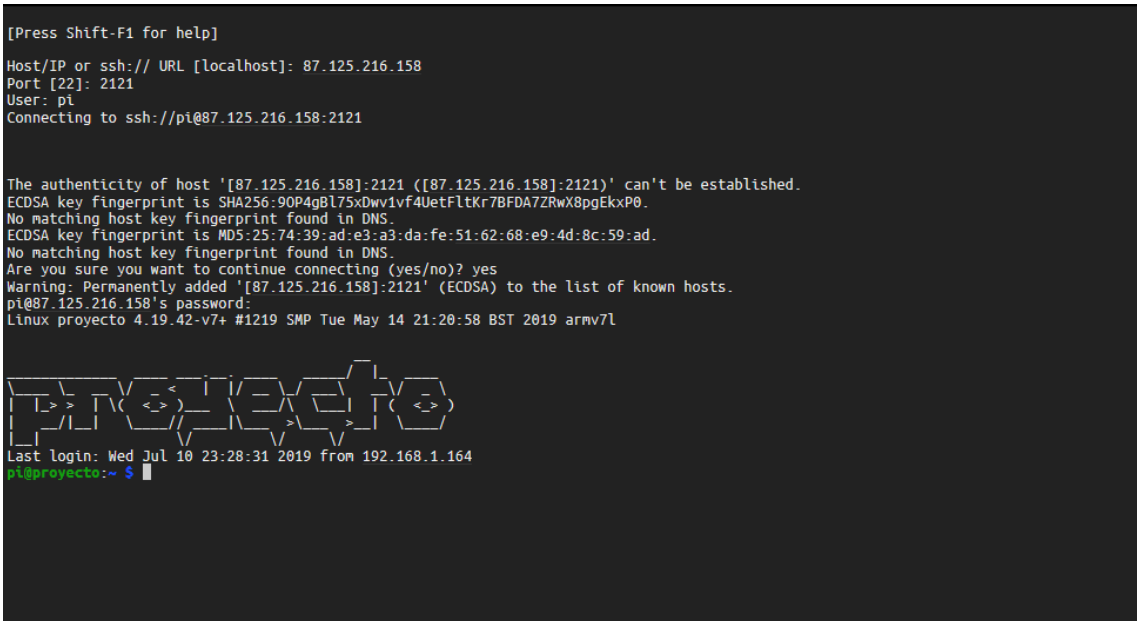


Ilustración 36 Conexión a Raspberry pi modo consola

- Raspbian with desktop

Desktop ofrece un entorno gráfico (LXDE), con múltiples aplicaciones gráficas sobre una GUI [Ilustración 37 Raspberry en modo gráfico], incluyendo software educación y de productividad.
La iteración será con ratón y teclado, recomendada para usuarios nuevos debido a que es amigable e intuitiva.

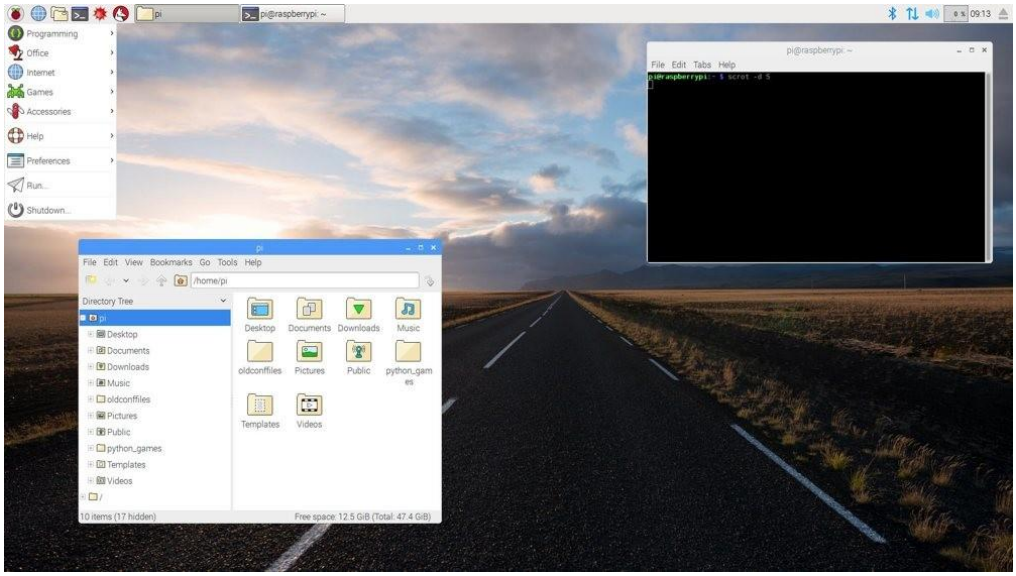


Ilustración 37 Raspberry en modo gráfico

Se utilizará Raspbian Lite ya que nos permite unos menores tiempos de ejecución al no ejecutar parte gráfica la cual proporciona retrasos y es innecesaria para la ejecución del proyecto.

4.6.1.1 Crowtail - Base Shield para Raspberry Pi

Este es un dispositivo adaptador de interfaces y voltajes siendo a su vez un protector de la Raspberry Pi, está diseñado por la compañía Crowtail específicamente para Raspberry Pi con adaptadores para interfaces UART/I2C/Analógico/Digital. Además, esta shield dispone un chip ADC incorporado permitiendo así el uso de entradas y salidas analógicas en la Raspberry Pi.

El chip ADC utilizado es MCP3008. Se comunica con Raspberry Pi mediante el uso de una interfaz SPI, pero esto no se va a utilizar en este proyecto.

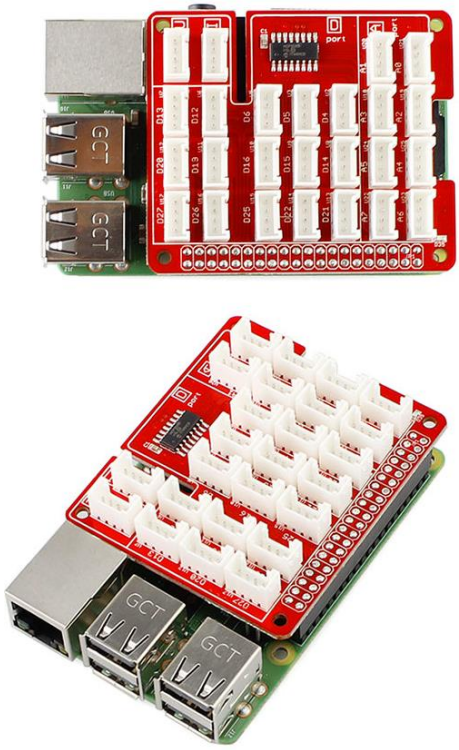


Ilustración 38 Crowtail - Base Shield conectada a Raspberry Pi

4.7 VL53L0X

El VL53L0X es un sensor de tipo “distancia de tiempo de vuelo” [6]. El sensor consta de dos grandes componentes una fuente de láser invisible muy pequeña y un sensor correspondiente a la longitud de onda del láser. El VL53L0X puede detectar el "tiempo de vuelo" o el tiempo que ha tardado la luz en ir y volver al sensor. Se tiene que tener en cuenta el diámetro de la bola ya que la medida será la distancia del sensor más el radio de la bola, dato que se tiene en cuenta en el código. Dado que utiliza una fuente de luz muy estrecha, está especializado en determinar la distancia de solo la superficie directamente delante de ella. A diferencia de los sensores tipo “sónares”, los cuales ya se probaron para este proyecto, las ondas ultrasónicas rebotan obteniendo poca precisión, por el hecho que el cono de emisión es muy grande frente a la tecnología láser en la cual es mucho más estrecho. A diferencia de los sensores de distancia IR

que mide la cantidad de luz que rebota, el VL53L0x es mucho más preciso y no tiene problemas de linealidad o "imágenes dobles" en las que no se puede saber si un objeto está muy lejos o muy cerca.

También se propuso en este proyecto el uso del sensor VL6180X ToF, que puede manejar aproximadamente entre 5 mm a 200 mm de distancia de alcance. Pero para este proyecto se descartó esta opción ya que es necesario más rango como proporciona sin embargo el sensor VL53L0X.

El sensor es de pequeñas dimensiones y fácil de usar en cualquier proyecto de robótica o interactivo. Dado que necesita una potencia y lógica de 2.8 V, se podría usar directamente con la Raspberry Pi en nuestro proyecto, pero es mejor la colocación de algún tipo de separación de voltajes para más seguridad.

Ilustración 39 Dimensión de sensor

La comunicación al sensor se realiza a través de I2C con una API escrita por ST (el fabricante), por lo que se puede portar a cualquier microcontrolador abierto ejecutando las librerías de Python como en nuestro caso dentro del sistema Raspbian de Raspberry Pi.

El sensor puede medir aproximadamente 50 mm a 1,2 metros en el modo predeterminado, como se puede observar en la siguiente gráfica.

Figure 20. Typical ranging (default mode)

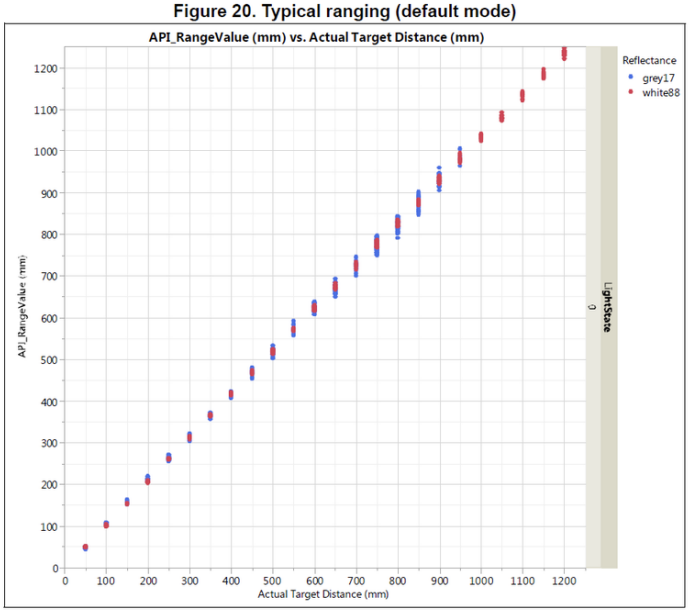


Ilustración 40 Diferencia de detección de sensor en modo predeterminado (gris-blanco)

En el modo de "largo alcance" puede detectar hasta 1,5 a 2 metros en una superficie reflectante blanca, pero no se necesitó debido tanta distancia de detección ya que la longitud del tubo del proyecto es de 40 cm aproximadamente

Figure 21. Typical ranging - Long range mode

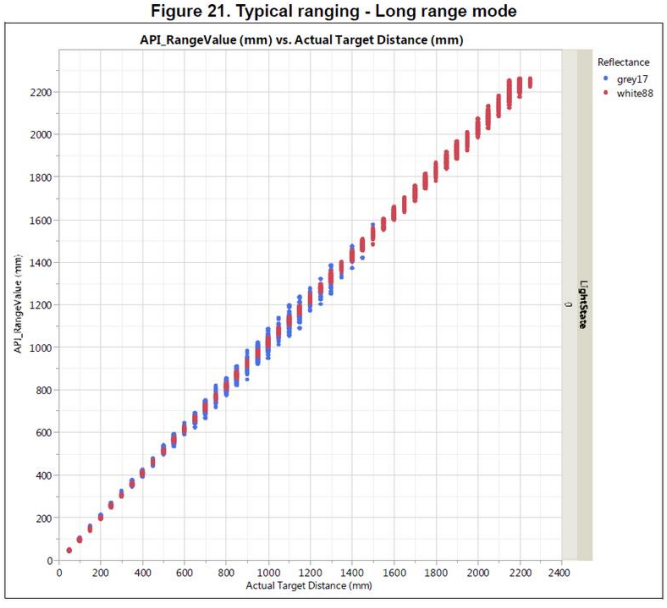


Ilustración 41 Diferencia de detección de sensor en modo largo(gris-blanco)

Dependiendo de la iluminación ambiental y la distancia, este sensor obtendrá precisiones de rango de 3% a 12%; siendo mejor una iluminación y superficies brillantes según la experimentación previa al desarrollo del proyecto. Se necesita algo de experimentación sobre el tipo de bola a usar, ya que si la bola absorbe la luz del láser no obtendrá buenas lecturas.

Table 11. Max ranging capabilities with 33ms timing budget

Target reflectance level (Full FOV)	Conditions	Indoor (2)	Outdoor overcast (2)
White Target (88%)	Typical	200cm+ (1)	80cm
	Minimum	120cm	60cm
Grey Target (17%)	Typical	80cm	50cm
	Minimum	70cm	40cm

Note (1): using long range API profile

Ilustración 42 Medidas máxima de el sensor

Table 12. Ranging accuracy

	Indoor (no infrared)			Outdoor		
Target reflectance level (Full FOV)	Distance	33ms	66ms	Distance	33ms	66ms
White Target (88%)	at 120cm	4%	3%	at 60cm	7%	6%
Grey Target (17%)	at 70cm	7%	6%	at 40cm	12%	9%

Ilustración 43 velocidad y presión máxima del sensor según color

El pinout del VL53L0X es tipo sensor I2C, lo cual quiere decir que utiliza los dos cables de datos/reloj. El protocolo I2C está disponible en la mayoría de los microcontroladores como Raspberry Pi, y puede compartir esos pines con otros sensores siempre que no tengan una colisión de direcciones.

Para futuras referencias, la dirección I2C predeterminada es 0x29 (se puede comprobar este dato en la parte de hardware). Se podría cambiar mediante software de forma temporal para el caso de usar varios sensores en una misma línea de comunicación,

pero esto no es el caso en el proyecto,

Los pines de forma física de la placa del sensor son los siguientes:

- Vin – siendo este pin de alimentación. Dado que el chip utiliza 2.8 VDC, la placa incluye un regulador de voltaje para reducir desde 5 voltios a 2,8 voltios con un mínimo de 3 V dando así una mayor amplitud de alimentación y flexibilidad al compatibilizarlo con otros circuitos y microcontroladores.
- GND – masa o común como referencia
- SCL - Pin de reloj I2C, se conecta a la línea de reloj I2C del microcontrolador.
- SDA - Pin de datos I2C, se conecta a la línea de datos I2C del microcontrolador.
- GPIO: Este es un pin que usa el sensor para indicar que los datos están listos. Es útil para cuando se hace detección continua. En este proyecto no se utiliza esto ya que es suficiente el tiempo de lectura, dado que la limitación viene dada por el tiempo de respuesta del servomotor.
- SHDN: Es el pin de apagado para el sensor. Por defecto está en estado alto. Hay un diodo de cambio de nivel para que pueda usar la lógica de 3-5 V en este pin. Cuando el pin entra en estado bajo el sensor entra en modo de apagado, pero esta utilidad no se usa en el proyecto.

5 IMPLEMENTACIÓN.

La implementación, al igual que las comunicaciones, tiene dos vertientes: la parte de implementación como protocolo serial y la parte de implementación del protocolo Ethernet, aunque fundamentalmente el funcionamiento es el mismo.

La estructura de programación está basada en un modelo servidor-cliente, es decir la Raspberry Pi controla el sistema y genera la posibilidad de conectarte a ella de forma remota para el protocolo Ethernet y de forma local con el protocolo Serial.

El cliente es cualquier programa que se conecta al servidor en un momento puntual para interaccionar con él, cuando termina las operaciones que el cliente necesita terminará la comunicación quedando libre el servidor para otra comunicación, solo se podrá realizar una conexión cliente- servidor

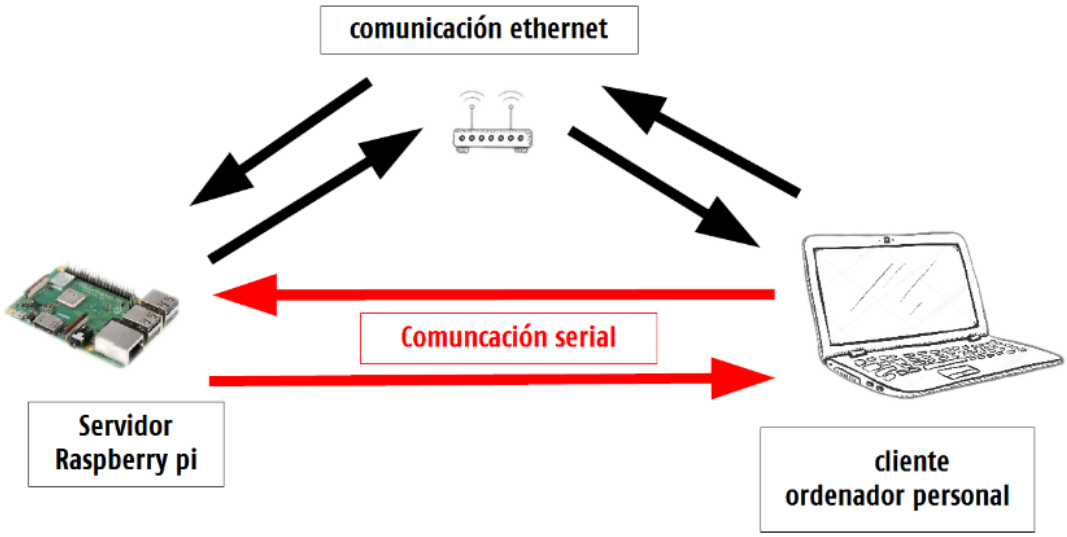


Ilustración 44 Esquema de las conexiones posibles

La tecnología utilizada para implementar los protocolos de comunicación tanto en la parte de servidor como de cliente son distintas pero no respecto a la comunicación en sí, sino no podría existir comunicación. La implementación del cliente en la Raspberry Pi será realizada mediante Python, un lenguaje como después explicaremos óptimo para la configuración de cliente remoto y con mucha versatilidad. Para la parte de servidor se usó un lenguaje con tecnología Microsoft como es Visual Basic ya que la

mayoría de los alumnos en su ordenador personal disponen de un sistema operativo tipo Windows.

La comunicación Ethernet o Serial es independiente de sistemas operativos o de aplicaciones, sino que simplemente son herramientas que nos sirven para que los dispositivos se comuniquen entre sí, por lo cual ya estaban implementada en las librerías de ambos lenguajes de programación, y su aplicación no es difícil.

Por otro lado se generó la posibilidad de un streaming, es decir la posibilidad de mostrar en tiempo real sistema mediante una webcam en video del sistema, para facilitar el control remoto, pero esta tecnología ya estaba implementada en la Raspberry Pi pero no se utilizará en el proyecto.

5.1 IMPLEMENTACIÓN DEL SERVIDOR.

5.1.1 Características de Python

Python es un lenguaje de programación interpretado de tipado dinámico cuya filosofía hace hincapié en una sintaxis que favorezca un código legible. Se trata de un lenguaje de programación multiparadigma y disponible en varias plataformas [7], desglosaremos sus características básicas a continuación:

- Interpretado:

Normalmente los lenguajes de programación se pueden agrupar en Interpretados y Compilados según la forma en la que son traducidos.

Los lenguajes se desarrollaron para facilitar al programador el desarrollo aplicaciones. En el fondo cuando creamos un código lo que estamos haciendo es hablar un lenguaje más fácil de comprender a las personas y que luego será traducido a lenguaje de máquina él cual puede entender el procesador. Entonces, basados en la forma de realizar esta «traducción» distinguiendo en lenguajes en Interpretados o Compilados:

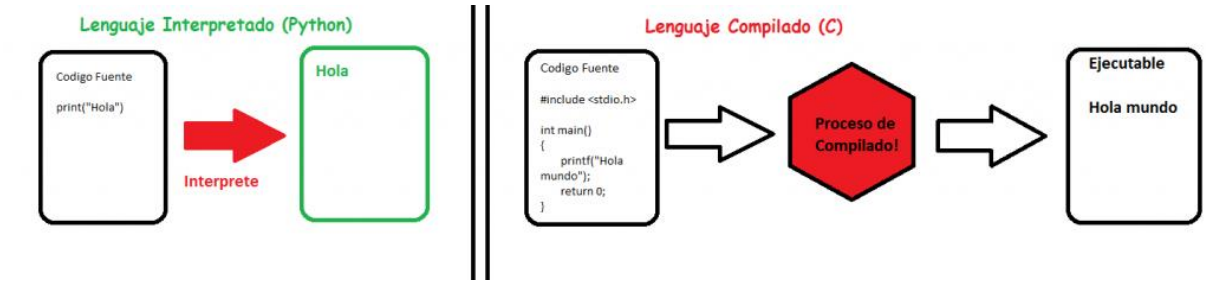


Ilustración 45 Esquemas de tipos de lenguaje

Los lenguajes Compilados son aquellos en los que el código del programador es traducido por completo de una sola vez mediante un proceso llamado «Compilación» para ser ejecutado por un sistema predeterminado. Entre los más comunes encontramos «C», «C++», Java, etc.

Y los lenguajes Interpretados son aquellos en los que el código del programador es traducido mediante un intérprete a medida que es necesario durante su ejecución. Entre los más comunes encontramos «Python«, «Ruby», «Javascript», etc.

El que el lenguaje Python sea interpretado nos presenta ventajas:

- Al ser interpretado se necesita compilar ahorrándonos mucho tiempo en el desarrollo y prueba de una aplicación.
- El código fuente puede ser ejecutado en cualquier software siempre y cuando este disponga del intérprete (Windows, Linux, Mac, Android, Web).

- Multiparadigma:

Python es un lenguaje que soporta más de un paradigma, suponiendo paradigma como modelo de desarrollo (y cada lenguaje tiene el suyo). Los paradigmas de la programación en Python son:

- Imperativo:

Los lenguajes de programación también se pueden agrupar en imperativos y declarativos, los del primer grupo a los que pertenece Python son aquellos que describen el estado del programa y permiten su modificación mediante condiciones o instrucciones de código que le indican al computador cómo realizar una tarea.

Los declarativos son aquellos que solo «declaran» condiciones, ecuaciones, etc. que

describen un problema y detallan su solución.

La principal diferencia es que:

En la programación imperativa se describe paso a paso un conjunto de instrucciones que deben ejecutarse para variar el estado del programa y solucionar el problema. Mientras que en la declarativa sólo se procede a describir el problema que se quiere solucionar.

- Funcional:

La programación funcional es un paradigma de la programación declarativa basada en el uso de funciones matemáticas que permite la variación del programa mediante la mutación de variables. Esto nos va a permitir operar con datos de entrada y salida. Brindándole así la posibilidad al usuario de ingresar datos que serán procesados para darnos otros datos de salida.

Operar a través de funciones recursivas (es decir que pueden volver a recurrirse a ellas) nos ahorrará muchas líneas de código entre otras ventajas.

En sencillas palabras una función es como una operación definida, supongamos una suma (+), los elementos a sumar pueden variar, pero de todas maneras la función será la misma (sumar). Así que sean cuales sean los elementos se sumaran cada vez que nosotros lo necesitemos con la misma instrucción (sumar).

- Orientado a Objetos (POO):

La programación orientada a objetos ofrece la particularidad en la forma de obtener los resultados. Los objetos manipulan los objetos de entrada para la obtención de resultados (salida) específicos donde cada objeto nos ofrece una función específica y también nos permite la agrupación de bibliotecas o librerías.

Los objetos son entidades que tienen un determinado estado, las entidades son propiedades que los diferencian.

- Tipado Dinámico:

El tipado dinámico es aquel tipado en el que una variable puede tomar diferentes valores de distintos tipos en diferentes momentos. En Python las variables son declaradas por su contenido y no por su contenedor, lo que nos va a permitir cambiar el valor y tipo de una variable durante la ejecución sin necesidad de volver a declarar.

5.1.2 *Módulo de autentificación*

El módulo de autentificación es el encargado de permitir la autentificación con usuario y contraseña particulares para el uso de la ampliación de forma remota [8].
Para la encriptación se usa funciones tipo hash, siendo estos algoritmos los que consiguen crear a partir de una entrada de datos, una salida alfanumérica de longitud normalmente fija que representa un resumen de toda la información que se le ha dado.

Un hash es básicamente un proceso criptográfico generado por un algoritmo dedicado a ello, pero se diferencia del resto de los métodos criptográficos en que este no puede descifrarse, es decir con este método no es posible devolver el valor original del valor dado.

Para realizar este proceso se usa una librería llamada “hashlib” en Python, se genera una tabla con duplas como se puede ver en la línea 4, la primera parte de la dupla es el usuario y la segunda en la contraseña, toda la tabla es codificada mediante hashlib y md5.

Tras ello generamos una función (como parámetro de entrada usamos el objeto de la comunicación socket para poder manejar el protocolo), al llamar a dicha función realizara preguntas de usuario y contraseña , el programa cliente enviará el usuario y la contraseña separada por comas , así que tendremos que realizar el “parseo“ de dichos datos mediante funciones propias de la librería string (.rstrip y Split), por ultimo solo nos queda comparar los datos recibido con los ya codificados en caso de que el usuario y contraseña este en nuestra tabla de duplas la función devolverá un “1” en caso contrario un “0”.

```
29.  # -*- coding: utf-8 -*-
30.
31. import hashlib
32. database={'name': '1234', 'name2': '5678', 'name3': '9012'}
33. encrypted = dict([(name,hashlib.md5(pw).hexdigest()) for name, pw in database.items()])

34.
35. def comprobas_user(conn):
36.     print("comprobando user and pass")
37.     conn.send("introduce user y password")
38.     data = conn.recv(1024)
39.     data= data.rstrip('\r\n')
```

```
40.     name, ask=data.split(';')
41.
42.     if name not in encrypted or hashlib.md5(ask).hexdigest() != encrypted.get(name):
43.         return 0
44.     else:
45.         return 1
```

5.1.3 Sensor (VL53L0X).

El sensor de posición es el dispositivo encargado de realizar la medición de la bola dentro de la viga, obtenido la distancia de la bola al final de la viga.

El sensor en el VL53L0X como ya se dijo es un dispositivo que no usa tecnología analógica ya que la cual es muy propensa a ruido y a interferencias externas, usando esta tecnología digital de tipo I2C.

La tecnología I2C tiene librería propia dentro de Python llamada “smbus” por lo cual esta será importada junto con la librería “time” la cual usaremos para realizar cálculos con tiempos, con retrasos o sleep.

Se generará un objeto llamado i2cbus en la línea 13 el cual utilizaremos como manejador para toda la comunicación, siendo este empleado por las funciones i2c_read y i2c_write base para realizar todas las comunicaciones entre el sensor y la Raspberry Pi.

Tras ello importamos la librería que nos proporciona el fabricante ya que es de código privativo en la línea 46 se realiza la operación.

Se genera un descriptor de clase tipo “VL53L0X(object)” [9] el cual usaremos en el programa principal para la creación del objeto. En el constructor de la clase se define la dirección de nuestro sensor en este caso “0x29” siendo esta la utilizada en el dispositivo por defecto. También dentro de la clase se generan unas instancias que son las siguientes:

- def start_ranging(self, mode = VL53L0X_GOOD_ACCURACY_MODE)

Esta función es la encargada de activar el sensor con los parámetros de sensibilidad siguientes:

VL53L0X_GOOD_ACCURACY_MODE = 0 # Good Accuracy mode

VL53L0X_BETTER_ACCURACY_MODE = 1 # Better Accuracy mode

VL53L0X_BEST_ACCURACY_MODE = 2 # Best Accuracy mode

VL53L0X_LONG_RANGE_MODE = 3 # Longe Range mode

VL53L0X_HIGH_SPEED_MODE = 4 # High Speed mode

Los primeros parámetros son muy precisos pero con muy poca velocidad de lectura mientras que en los últimos en caso contrario tienen mucha velocidad, pero son muy poco precisos. Los valores dados por el fabricante son los siguientes:

Perfil de rango	Presupuesto de tiempo de rango
Modo por defecto	30 ms
Alta precisión	200 ms
Largo alcance	33 ms
Alta velocidad	20 ms

	Interior (sin infrarrojos)			Al aire libre		
Tipo de objeto	Distancia	33 ms	66 ms	Distancia	33 ms	66 ms
Blanco blanco (88%)	A 120 cm	4%	3 %	A 60 cm	7 %	6 %
Gray T arg et (17%)	A 70 cm	7%	6%	a 40 cm	12 %	9 %

- def stop_ranging(self):

Esta función deja en una posición de stand-by al sensor en la que no es posible realizar medidas de distancia. En este estado solo es posible volver a activarlos con la función "start_ranging"

- def get_distance(self):

Esta función realiza la medición del sensor obtenido la distancia entre este último y el obstáculo más cercano.

- def get_timing(self):

Función de ejemplo no usada:

```
#!/usr/bin/python

import time
from ctypes import *
import smbus

      VL53L0X_GOOD_ACCURACY_MODE      = 0   # Good Accuracy mode
      VL53L0X_BETTER_ACCURACY_MODE    = 1   # Better Accuracy mode
      VL53L0X_BEST_ACCURACY_MODE      = 2   # Best Accuracy mode
VL53L0X_LONG_RANGE_MODE         = 3   # Longe Range mode
VL53L0X_HIGH_SPEED_MODE         = 4   # High Speed mode

i2cbus = smbus.SMBus(1)

# i2c bus read callback
def i2c_read(address, reg, data_p, length):
    ret_val = 0;
    result = []

    try:
        result = i2cbus.read_i2c_block_data(address, reg, length)
    except IOError:
        ret_val = -1;

    if (ret_val == 0):
        for index in range(length):
            data_p[index] = result[index]

    return ret_val

# i2c bus write callback
def i2c_write(address, reg, data_p, length):
    ret_val = 0;
    data = []

    for index in range(length):
        data.append(data_p[index])
    try:
        i2cbus.write_i2c_block_data(address, reg, data)
    except IOError:
        ret_val = -1;

    return ret_val

# Load VL53L0X shared lib
tof_lib = CDLL("./VL53L0x/vl53l0x_python.so")

# Create read function pointer
READFUNC = CFUNCTYPE(c_int, c_ubyte, c_ubyte, POINTER(c_ubyte), c_ubyte)
read_func = READFUNC(i2c_read)

# Create write function pointer
WRITEFUNC = CFUNCTYPE(c_int, c_ubyte, c_ubyte, POINTER(c_ubyte), c_ubyte)
write_func = WRITEFUNC(i2c_write)

   # pass i2c read and write function pointers to VL53L0X library
   tof_lib.VL53L0X_set_i2c(read_func, write_func)

   class VL53L0X(object):
       """VL53L0X ToF."""

       object_number = 0

       def __init__(self, address=0x29, TCA9548A_Num=255, TCA9548A_Addr=0, **kwargs):
           """Initialize the VL53L0X ToF Sensor from ST"""
```

```
111.            self.device_address = address
112.            self.TCA9548A_Device = TCA9548A_Num
113.            self.TCA9548A_Address = TCA9548A_Addr
114.            self.my_object_number = VL53L0X.object_number
115.            VL53L0X.object_number += 1
116.            self.distancia=0
117.
118.        def start_ranging(self, mode = VL53L0X_GOOD_ACCURACY_MODE):
119.            """Start VL53L0X ToF Sensor Ranging"""
120.            tof_lib.startRanging(self.my_object_number, mode, self.device_address, self.
    TCA9548A_Device, self.TCA9548A_Address)
121.
122.        def stop_ranging(self):
123.            """Stop VL53L0X ToF Sensor Ranging"""
124.            tof_lib.stopRanging(self.my_object_number)
125.
126.        def get_distance(self):
127.            """Get distance from VL53L0X ToF Sensor"""
128.            self.distancia=tof_lib.getDistance(self.my_object_number)
129.            return(self.distancia)
130.
131.
132.        # This function included to show how to access the ST library directly
133.        # from python instead of through the simplified interface
134.        def get_timing(self):
135.            Dev = POINTER(c_void_p)
136.            Dev = tof_lib.getDev(self.my_object_number)
137.            budget = c_uint(0)
138.            budget_p = pointer(budget)
139.            Status =  tof_lib.VL53L0X_GetMeasurementTimingBudgetMicroSeconds(Dev, budget
    _p)
140.            if (Status == 0):
141.                return (budget.value + 1000)
142.            else:
143.                return
```

5.1.4 Módulo de comunicaciones.

Este módulo tiene la función de “parseo” es decir de analizar los datos recibidos por la aplicación cliente y realizar una acción determinada. Para generar este tipo de trascripciones se usa una codificación dentro de los datos recibidos por el cliente y enviados por el server teniendo este la siguiente estructura.

$ “numero de acción”; parámetro1; parámetro2; parámetro3; parámetro “n”….

Ejemplo:

$3;45,50;55

Donde en número de acción es 3 y los parámetros son 45 ,50 ,55.

Describiremos las acciones:

1 →Lanza la ejecución de PID.

2 →Función de reserva pasa futuras ampliaciones de la aplicación.

3 →Lanza la función que realiza la modificación de los parámetros del PID es decir

KP, Kd,Ki, muestreo…

4 → Función de reserva pasa futuras ampliaciones de la aplicación.

5 →Ejecuta la función que se encarga de graficar.

Este proceso de tiene comprobación de fallo de cadena es decir si no se recibe en orden o los caracteres recibidos no son los esperados sale de la función.

```
144.    #! /bin/bash/python3
145.    from ejec_funcion  import Lanzar_PID
146.    from mod_pid  import modificar_pid
147.    from ejec_funcion  import graficar
148.
149.    def acciones(conn,data,s,pid,servo):
150.        elec=data[1:2]
151.        asterisco=data[0:1]
152.        if ( asterisco  ==  "$") :
153.            try:
154.                eleccion=int (data[1:2])
155.
156.                if eleccion is 1:
157.                    vector_parametros=data.split(',')
158.                    Lanzar_PID(float(vector_parametros[2]), float (vector_parametros[1])
    ,conn,pid,servo)
159.                elif eleccion is 2:
160.                    vector_parametros=data.split(',')
161.                    print (" funcion numero 2 ,  numero de parametros %d " % (len(vector
    _parametros)))
162.                    #enviar endendido
163.                elif eleccion is 3:
164.                    vector_parametros=data.split(',')
165.                    print (" funcion numero 3 ,  entre %d " % (len(vector_parametros)))

166.                    modificar_pid(pid,vector_parametros)
167.                elif eleccion is 4:
168.                    vector_parametros=data.split(',')
169.                    print (" funcion numero 4 ,  numero de parametros %d " % (len(vector
    _parametros)))
170.                elif eleccion is 5:
171.                    #print (" peticion")
172.                    graficar(conn, pid)
173.                else:
174.                    #Equivalente a 'default'
175.                    print("Ninguna opcion valida seleccionada")
176.            except ValueError:
177.                #Handle the exception
178.                print("numero no recibido")
179.        else:
180.            print "dolar no recibido"
181.
```

5.1.1 Hilos.

Esta función es la encargada de general un timer que "despierta" cada tiempo determinado una función que se asignemos en la construcción de la clase, siendo en la práctica una especie de interrupción por software la cual despierta de forma cíclica cada tiempo determinado [10].
Esto lo utilizaremos para ejecutar el PID de forma cíclica teniendo así un muestreo y ejecución en tiempo estable evitando en la medida de los posible que dentro de la ejecución del PID se utilicen parámetros temporales, ya que la función integral y derivativa los necesitaría en principio, pero si estos son constantes los descartamos dentro de las constantes.

Se genera el constructor de clases particular heredado de la clase threading en el constructor al cual le "pasamos" los parámetros de la función a atributos internos y generamos un hilo tipo timer y una función manejadora "handle_function".
Como funciones de clase nos encontramos únicamente una función start y una función stop

```
182.    from threading import Timer,Thread,Event
183.    import time
184.    import RPi.GPIO as GPIO
185.    import time
186.    import sys
187.
188.    class perpetualTimer():
189.
190.       def __init__(self,t,hFunction):
191.          self.t=t
192.          self.hFunction = hFunction
193.          self.thread = Timer(self.t,self.handle_function)
194.
195.       def handle_function(self):
196.          self.hFunction()
197.          self.thread = Timer(self.t,self.handle_function)
198.          self.thread.start()
199.
200.       def start(self):
201.          self.thread.start()
202.
203.       def cancel(self):
204.          self.thread.cancel()
```

5.1.2 Funciones auxiliares

Este módulo contiene las funciones auxiliares para poder realizar operaciones no definidas dentro de sus respectivos módulos o que por motivos de acceso no sean posible ejecutar de otra forma, son las siguientes funciones:

- Lanzar_PID

Esta función genera dos hilos: el primero (t) es cíclico y ejecuta la función ejec_pid cada tiempo determinado teniendo así una ejecución exacta y periódica de la instancia update de la clase PID; el segundo hilo (s) ejecuta la función apagar el cual, como veremos más tarde, se encarga de "matar" el primer hilo. Activamos un atributo de la clase PID para poder comprobar en otras partes de le código que se está ejecutando el PID.

- ejec_pid

Esta función es el alma del controlador y tiene únicamente tres líneas: en la línea primera obtiene la distancia la cual es proporcionada por el sensor, la segunda línea (línea 24) actualiza la salida del PID con el valor de la nueva distancia obtenida, y la última línea ejecuta la salida del PID en el servomotor, siendo este el actuador último del sistema.

- Graficar

La función grafica tiene como misión devolver por Ethernet el estado del sensor y el setpoint cada vez que es llamada.

Dichos valores después son graficados por la aplicación cliente. Se realizó un bucle para no duplicar procesos y en caso de que se esté ejecutando el PID se usará la medida del sensor obtenida en la función PID para el ploteo. De esta forma, se elimina el potencial problema de leer el sensor varias veces, ya que la lectura del sensor es un proceso muy lento a nivel ejecución.

- Apagar

Es la función encargada de "matar" los hilos. Es llamada al arrancar el hilo (s)

```
205.    import hilo.hilo
206.    import sys
207.    import PID.PID
208.    import VL53L0x.VL53L0X
209.    from time import time                #Importamos time para poder usar time.sleep
```

```
210.    radio=38
211.
212.    tof = VL53L0x.VL53L0X.VL53L0X()
213.    tof.start_ranging(VL53L0x.VL53L0X.VL53L0X_HIGH_SPEED_MODE)
214.
215.
216.
217.
218.    def Lanzar_PID(ciclo,tiempo,conn, pid,servo):
219.        t = hilo.hilo.perpetualTimer(ciclo),  lambda:ejec_pid(pid,tof,conn,servo))
220.        s = hilo.hilo.perpetualTimer(tiempo, lambda:apagar(s,t,pid))
221.        pid.ejecuta=1
222.        s.start()
223.        t.start()
224.
225.
226.    def ejec_pid(pid,tof,conn,servo):
227.
228.        distance = tof.get_distance()  + radio
229.        pid.update(distance)
230.        servo.ChangeDutyCycle(pid.output)
231.
232.    def graficar (conn, pid):
233.
234.        if ( pid.ejecuta ==  0) :
235.            distance = tof.get_distance()  + radio
236.        else :
237.            distance =tof.distancia
238.        if ( distance <  0) :
239.            distance=-distance
240.
241.        conn.send("@"+";"+str(distance)+";"+str(int (pid.setpoint))+"\r\n" )
242.
243.
244.
245.    def apagar(s,t,pid):
246.        s.cancel()
247.        t.cancel()
248.        pid.ejecuta=0
```

5.1.1 Programa principal

El programa principal es el programa llave para realizar la ejecución de todo el código en principio se encarga de generar los objetos del socket y del PID línea 21 y 22 respectivamente asignándoles los parámetros de los constructores.

Tras la creación del socket inicia el proceso de la creación de conexión como se puede ver entre las líneas 26 a la 44 con los parámetros bind, listen, etc.

Lo siguiente que realiza es la petición de autentificación con la función “comprobas_user” usando como parámetro objeto la propia conexión, y por último entra en la función while(1) y escape por break donde se realiza la lectura de los datos que envía el programa cliente y que son parseados con la función “Tratamiento.acciones” enviando todos los parámetros que necesita para los procesos

```
249.
250.    # -*- coding: utf-8 -*-
251.    import socket
252.    import sys
253.    import time
254.    import Tratamiento.Tratamiento
255.    import  PID.PID
256.    import UserPass.UserPass
257.    from UserPass.UserPass  import comprobas_user
258.    import RPi.GPIO as GPIO    #Importamos la libreria RPi.GPIO
259.    import time                #Importamos time para poder usar time.sleep
260.    a=22
261.    GPIO.setmode(GPIO.BOARD)   #Ponemos la Raspberry en modo BOARD
262.    GPIO.setup(a,GPIO.OUT)    #Ponemos el pin 21 como salida
263.    servo = GPIO.PWM(a,50)        #Ponemos el pin 21 en modo PWM y enviamos 50 pulsos po
    r segundo
264.    servo.start(7.5)              #Enviamos un pulso del 7.5% para centrar el servo
265.
266.    HOST = ''
267.    PORT = 7000
268.    datos = ""
269.    buff_size = 4096
270.    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
271.    pid = PID.PID.PID(0.1, 0.1, 0.1)
272.
273.
274.
275.    print('# conexion creada')
276.    # Create socket on port
277.    try:
278.        s.bind((HOST, PORT))
279.    except socket.error as msg:
280.        print('# conexion fallida ')
281.        sys.exit()
282.    while True:
283.        print('# conexion generada')
284.
285.        # Start listening on socket
286.        s.listen(1)
287.        print('# esperando conexion')
288.
289.        # Wait for client
290.        conn, addr = s.accept()
291.        print('# conectado con ' + addr[0] + ':' + str(addr[1]))
292.        time.sleep(0.1)
293.        s.setblocking(True)
294.
295.        if ( comprobas_user(conn)  ==  0) :
296.            print "Logueado incorrecto"
297.            conn.send("logueado incorrecto")
298.            s.close()
299.            sys.exit(0)
300.        else:
301.            print "logueado correcto"
302.            conn.send("logueado correcto")
303.
304.        while True:
305.            data = conn.recv(1024)
306.            data= data.rstrip('\r\n')
307.            if data == "quit":
```

```
308.                 break
309.             Tratamiento.Tratamiento.acciones(conn,data,s, pid,servo)
310.
311.     s.close()
```

5.1.2 Módulo de PID.

Este módulo es el encargado de crear la clase [11] que controla el sistema mediante en PID , para ello en el constructor de clase se colocan los atributos que posteriormente utilizaremos para las distintas instancias de la clase [12] . Algunos de estos atributos se pasan como parámetros de llamada a la clase, y después se extrapolan como se puede ver en las líneas 306 a 309.

Los atributos básicos son kp, kd, ki, setpoint, bias, output, los cuales serán muy utilizados en distintas partes de la programación.

La instancia clave de la clase es “def update” es la encargada de realizar la operativa matemática del PID así que se explicara más detalladamente.

La primera operación que realiza la instancia es el cálculo de errores en líneas 334 y 335, en él se calculan los errores proporcional y derivativo, y para el caso del error integral se realizan varias operaciones (línea 337 y línea 347) pero se puede comprobar que es el sumatorio acumulado del error anterior. En seudocódigo se podría visualizar con las siguientes ecuaciones [13]:

$$Error\ proporcional = lectura\ de\ distancia - setpoint$$

$$error\ derivativo = error\ proporcional\ actual - error\ proporcional\ ciclo\ anterior$$

$$Error\ integral = \sum error\ proporcional$$

Ecuación 23 cálculo de errores

En la línea 339 se puede observar el reset de la variable integral al cruzar la señal el punto de setpoint, para no inestabilizar el sistema.

La señal se salida será el sumatorio de los productos del error por su correspondiente constante para los tres casos "P" "I" y "D" y su constante correspondiente, pudiéndose observar esto en línea 344 , 350 y 359. Además se le añade un bias o constante, que proporciona un valor al servomotor correspondiente el estado horizontal de la viga, pudiendo así la función variar entorno a los 7.5 ms en la señal de PWM enviada al

servomotor.

La salida del algoritmo llamada ”output” es filtrado en entre la señal que puede admitir el servomotor 10.5 y 4.5 se puede observar en las líneas 362 -365. Por último enviaremos mediante returns esta variable:

```
312.    import time
313.
314.    class PID:
315.        """PID Controller
316.        """
317.
318.        def __init__(self, P=1, I=0, D=0, muestreo=20):
319.
320.            self.Kp = P
321.            self.Ki = I
322.            self.Kd = D
323.            self.muestreo =  muestreo/1000
324.            self.setpoint=0
325.            self.bias = 7.5
326.            self.ErrorPAnt = 0
327.            self.output=0
328.            self.ejecuta=0
329.            self.ErrorAcumulado=0
330.            self.clear()
331.        def clear(self):
332.            """Clears PID computations and coefficients"""
333.            self.SetPoint = 0
334.
335.            self.PTerm = 0
336.            self.ITerm = 0
337.            self.DTerm = 0
338.            self.last_error = 0
339.
340.            # Windup Guard
341.            self.int_error = 0
342.            self.windup_guard = 20
343.
344.            self.output = 0
345.
346.        def update(self, feedback_value):
347.
348.
349.            ErrorP = self.setpoint - feedback_value;
350.            ErrorDT =  self.ErrorPAnt -ErrorP;
351.            self.ErrorPAnt = ErrorP;
352.
353.            if (self.ErrorPAnt>0 and ErrorDT<0 or self.ErrorPAnt<0 and ErrorDT>0):
354.                ErrorAcumulado=0;
355.
356.
357.            # -----calculate P component
358.            PTerm = ErrorP * self.Kp
359.
360.            # -----calculate I component
361.            self.ErrorAcumulado= self.ErrorAcumulado + ErrorP * self.Ki * self.muestreo
362.
363.            # -----calculate D component
364.            DTerm = ErrorDT * self.Kd* self.muestreo;
365.
366.            # -----calculate anti-windup
```

```
367.            # if (ErrorI >= _MaximoI) ErrorI = _MaximoI;
368.            # else if (ErrorI <= _MinimoI) ErrorI = _MinimoI;
369.            # ITerm = ErrorI;
370.
371.            # -----calculate PID
372.
373.            self.output = PTerm + DTerm + self.bias + self.ErrorAcumulado;
374.
375.
376.            if (self.output >= 10.5) :
377.                self.output = 10.5;
378.            if (self.output <= 4.5) :
379.                self.output = 4.5;
380.            # print (" ErrorP ; %f || ErrorDT %f || PTerm %f ||  DTerm %f || output %f "
     % (ErrorP,ErrorDT,PTerm,DTerm,self
381.    .output))
382.            print ("%f ;%f;%f;%f;%f" % (ErrorP,ErrorDT,PTerm,DTerm,self.output))
383.            # print ("**************************************************************
    ***")
384.
385.
386.
387.            return(self.output)
388.
389.        def setKp(self, proportional_gain):
390.            """Determines how aggressively the PID reacts to the current error with sett
    ing Proportional Gain"""
391.            self.Kp = proportional_gain
392.
393.        def setKi(self, integral_gain):
394.            """Determines how aggressively the PID reacts to the current error with sett
    ing Integral Gain"""
395.            self.Ki = integral_gain
396.
397.        def setKd(self, derivative_gain):
398.            """Determines how aggressively the PID reacts to the current error with sett
    ing Derivative Gain"""
399.            self.Kd = derivative_gain
400.
401.        def setWindup(self, windup):
402.            """Integral windup, also known as integrator windup or reset windup,
403.            refers to the situation in a PID feedback controller where
404.            a large change in setpoint occurs (say a positive change)
405.            and the integral terms accumulates a significant error
406.            during the rise (windup), thus overshooting and continuing
407.            to increase as this accumulated error is unwound
408.            (offset by errors in the other direction).
409.            The specific problem is the excess overshooting.
410.            """
411.            self.windup_guard = windup
412.
413.        def setSampleTime(self, sample_time):
414.            """PID that should be updated at a regular interval.
415.            Based on a pre-
    determined sampe time, the PID decides if it should compute or return immediately.
416.            """
417.            self.sample_time = sample_time
```

5.2 IMPLEMENTACIÓN DEL CLIENTE

5.2.1 Visual Basic

El lenguaje de programación Visual Basic es uno de los lenguajes de programación que utiliza una interfaz visual es decir que nos permite programar en un entorno gráfico, nos permite realizar un gran número de tareas sin escribir código, simplemente realizando operaciones con el ratón sobre la pantalla del ordenador.

Este lenguaje de programación es uno de los que más interés despiertan entre los programadores. Las razones son que este lenguaje de programación facilita la realización de tareas complejas en poco tiempo y las personas que están comenzado a programar con Visual Basic ven como son capaces de realizar pequeños programas al poco tiempo de haber comenzado a estudiar este lenguaje de programación.

El Visual Basic es un lenguaje de programación que proviene del BASIC. La primera versión de este lenguaje de programación Visual Basic fue presentada en el año 1991. La intención de este primer programa era simplificar la programación utilizando un entorno de trabajo claro que permitiera crear interfaces gráficas facilitando así la programación.

Las sintaxis que utiliza este lenguaje de programación provienen del conocido BASIC, pero completada con comandos y códigos de otros lenguajes más modernos. Este lenguaje de programación tiene un apartado dedicado a la Programación Orientada a Objetos.

Es un lenguaje muy apropiado para el manejo de bases de datos. Muchas empresas lo utilizan para la gestión de sus bases de datos porque su utilización es sencilla y abundan los programadores de este lenguaje.

De este lenguaje de programación han surgidos algunos derivados como: El VBScript es un lenguaje predeterminado para el Active Server Pages (ASP) que es un lenguaje de programación web. O el Visual Basic.NET que es un lenguaje de similares características a las del C#.

5.2.2 Aplicación cliente basado en Simplot

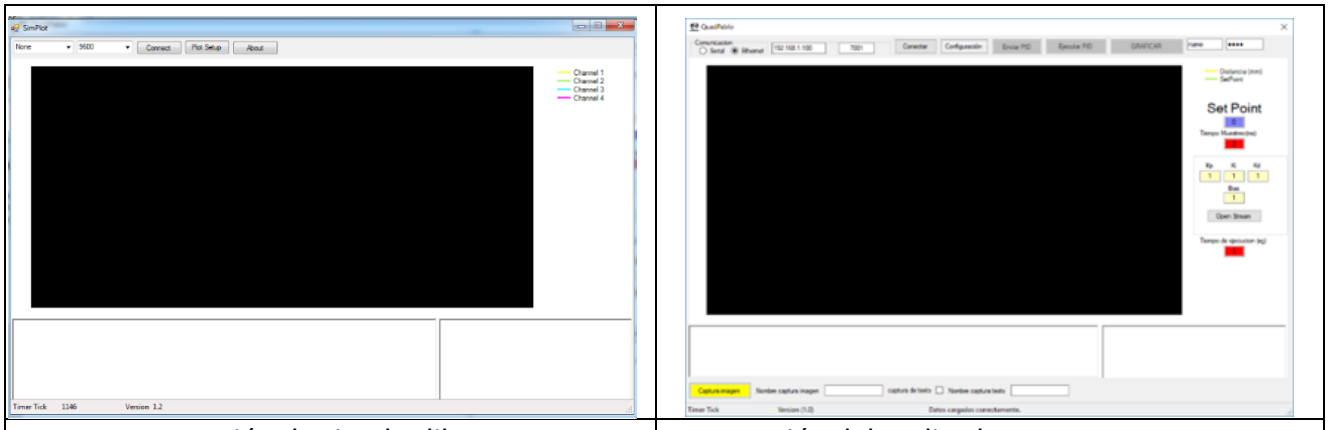
Se ha realizado un programa para poder modificar los parámetros del sistema en el ordenador y que estos pudieran ser enviados, como también graficar los datos requeridos. De esta forma es más fácil el ajuste manual de los parámetros del sistema, dando la posibilidad de una mejor comprensión de estos.

Para realizar graficas después de buscar por la red se encontró un programa que tenía todos los requisitos para poder usar como base de nuestro programa final, su nombre es Simplot [14] y tiene estas características:

- Podía graficar desde el puerto serie
- La velocidad de la gráfica rápida.
- Es software libre
- Tiene muchas posibilidades de modificación y refactorización de código.

Como es software libre tiene una página web donde se puede bajar el código mediante el método mercurial siendo último un tipo de repositorio utilizado en desarrollo informática.

El diseño realizado ha sido como ya comentamos una modificación y tiene este aspecto y distinto a la inicial se puede ver en figura.



Versión de simplot libre	Versión del realizada para este proyecto

Ilustración 46 diferencia entre programa original y modificado

5.2.1 Barra superior

Al diseño original se le añadieron varios tipos de interacciones con el usuario como botones, botones tipo radio, espacio para poder introducir los parámetros, conexiones, configuración …etc.

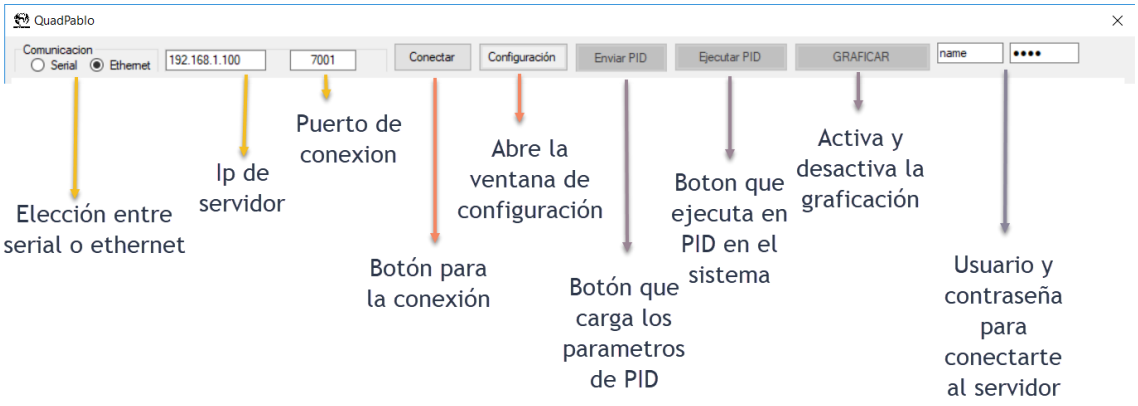


Ilustración 47 Esquema barra superior

En la parte superior podemos encontrar los parámetros y funciones no propias del PID, descritos a continuación

- Radio Botón de tipo de comunicación:

Este botón nos permite elegir entre comunicación Ethernet y comunicación Serial, habilitando o deshabilitando trasparencias dentro de la aplicación, como se puede ver en el código:

```
418.    Private Sub RadioCommSerie_CheckedChanged(sender As Object, e As EventArgs) Handles
    RadioCommSerie.CheckedChanged
419.            IpAddressTextBox.Visible = False
420.            PortIpTextBox.Visible = False
421.            userIpTextBox.Visible = False
422.            pwdIpTextBox.Visible = False
423.
424.            baudRateComboBox.Visible = True
425.            comPortComboBox.Visible = True
426.
427.            OpenWebBtn.Visible = False
428.
429.
430.        End Sub
431.
432.        Private Sub RadioCommEth_CheckedChanged(sender As Object, e As EventArgs) Handle
    s RadioCommEth.CheckedChanged
433.            IpAddressTextBox.Visible = True
```

```
434.            PortIpTextBox.Visible = True
435.            userIpTextBox.Visible = True
436.            pwdIpTextBox.Visible = True
437.
438.            baudRateComboBox.Visible = False
439.            comPortComboBox.Visible = False
440.
441.            OpenWebBtn.Visible = True
442.        End Sub
```

- Ip box y Puerto box o serial velocidad y Botón conectar

En él se colocan los parámetros de IP y puerto o velocidad de la conexión Serial para poder realizar la conexión. Además se encarga de la conexión tipo socket y Serial, adjunto código conexión Socket:

```
443.    If connectDisconnectButton.Text = "Conectar" Then
444.                    cliente = New TcpClient
445.                    Try
446.                        cliente.Connect(IPAddress.Parse(IpAddressTextBox.Text), PortIpTe
    xtBox.Text)
447.                        System.Windows.Forms.Application.DoEvents()
448.                    Catch ex As Exception
449.                        TerminalWindow.Text = ex.Message
450.                    End Try
```

- Botón Configuración

Abre una nueva ventana para realizar una configuración de aspecto de visualización

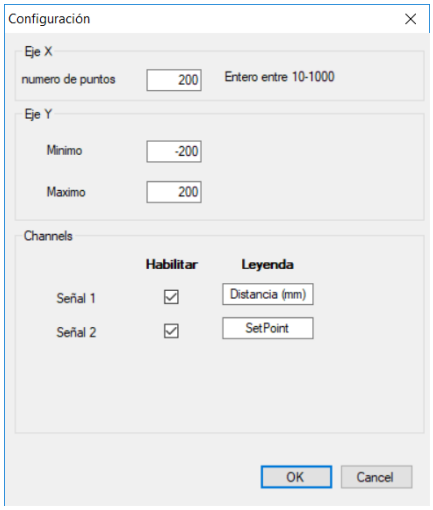


Ilustración 48 ventana de configuración

En él se pueden modificar el tiempo de muestreo y los máximos valores recibidos

además de los nombres y habilitación de las señales.

- Botón envío de PID

Realiza el envío de los parámetros de PID que se encuentran en la posición lateral mediante el método “as string” el cual genera una cadena completa con todos los datos para el envío, comenzando con “$” + “numero de comando” + $dato_1$ + $dato_2$ + … + $dato_n$ +”#”. Tras esto solo queda enviarlo. Después en la parte de servidor solo tendremos que “parsearlo” de forma correcta.

```
451.                    Dim TxtPID As String = "$" + "3," + TxtBox_Kp.Text + "," + TxtBox_Ki
    .Text + "," + TxtBox_Kd.Text + "," + setpoint.Text + "," + TxtBox_Bias.Text
452.
453.                    bytes = Encoding.ASCII.GetBytes(TxtPID)
454.                    leer_escribir = cliente.GetStream
455.                    leer_escribir.Write(bytes, 0, bytes.Length)
456.
457.                    leer_escribir.Flush()
```

- Botón ejecución PID

Envía los parámetros para activar el PID ($1) con el tiempo de muestreo y el tiempo de ejecución del mismo.

```
458.                Dim Txtapagado As String = "$" + "1," + TxtBox_TiempoEjecucion.Text + ",
    " + TxtBox_Ts.Text
459.
460.
461.                bytes = Encoding.ASCII.GetBytes(Txtapagado)
462.                leer_escribir = cliente.GetStream
463.                leer_escribir.Write(bytes, 0, bytes.Length)
464.
465.                leer_escribir.Flush()
466.
467.                bytes = Nothing
```

- Botón graficar:

Este botón activa el temporizador que activa el muestreo o las peticiones de toma de datos con el server.

```
468.            Dim SendDemande As String = "$5"
469.            bytes = Encoding.ASCII.GetBytes(SendDemande)
470.            leer_escribir = cliente.GetStream
471.            leer_escribir.Write(bytes, 0, bytes.Length)
472.            leer_escribir.Flush()
473.
474.            System.Threading.Thread.Sleep(100)
```

```
475.
476.            ReDim bytes(cliente.ReceiveBufferSize)
477.            leer_escribir.Read(bytes, 0, bytes.Length)
478.            cadena = Encoding.ASCII.GetString(bytes, 0, bytes.Length)
479.            Dim cadena2() As String = cadena.Split(vbCrLf)
480.
481.            Dim NuevoPlot = cadena2(0)
482.            byteCount = NuevoPlot.Length
```

- Usuario y contraseña box:

En estos cuadros de texto que controla el acceso a la aplicación. En él se colocan el usuario y la contraseña codificada, al realizar la petición de conexión es lo primero que se envía.

```
483.                        cliente.Connect(IPAddress.Parse(IpAddressTextBox.Text), PortIpTe
    xtBox.Text)
484.                        System.Windows.Forms.Application.DoEvents()
485.                    Catch ex As Exception
486.                        TerminalWindow.Text = ex.Message
487.                    End Try
488.                    If cliente.Connected = True Then
489.                        ' Introducimos Usuario y contrasena
490.                        leer_escribir = cliente.GetStream
491.                        System.Threading.Thread.Sleep(500)


492.                        If leer_escribir.DataAvailable = True Then
493.                            ReDim bytes(cliente.ReceiveBufferSize)
494.                            leer_escribir.Read(bytes, 0, bytes.Length)
495.                            cadena = Encoding.ASCII.GetString(bytes, 0, bytes.Length)
496.                            TerminalWindow.Text = TerminalWindow.Text & cadena & vbNewLi
    ne
497.
498.
499.
500.                            ' Esperamos la recepcion de la cadena para introducir user &
     pwd
501.                            If cadena.Contains("user") Then
502.                                Dim UserTxt As String = userIpTextBox.Text & ";"
503.                                Dim PwdTxt As String = pwdIpTextBox.Text & vbCrLf
504.
505.                                Dim cadena As String = UserTxt & PwdTxt
506.                                bytes = Encoding.ASCII.GetBytes(cadena)
507.
508.                                leer_escribir.Write(bytes, 0, bytes.Length)
509.                                leer_escribir.Flush()
510.
511.                                bytes = Nothing
512.                                PwdTxt = ""
513.                                UserTxt = ""
514.                                System.Threading.Thread.Sleep(500)
515.
516.                                ' Leemos la confirmacion de logueo
517.                                If leer_escribir.DataAvailable = True Then
518.                                    ReDim bytes(cliente.ReceiveBufferSize)
519.                                    leer_escribir.Read(bytes, 0, bytes.Length)
```

```
520.                                    cadena = Encoding.ASCII.GetString(bytes, 0, bytes.Le
    ngth)
521.                                    TerminalWindow.Text = TerminalWindow.Text & cadena &
     vbNewLine
522.
523.                                End If
```

5.2.2 Barra lateral

La barra lateral nos permite realizar la configuración del PID y abrir la ventana de streaming.

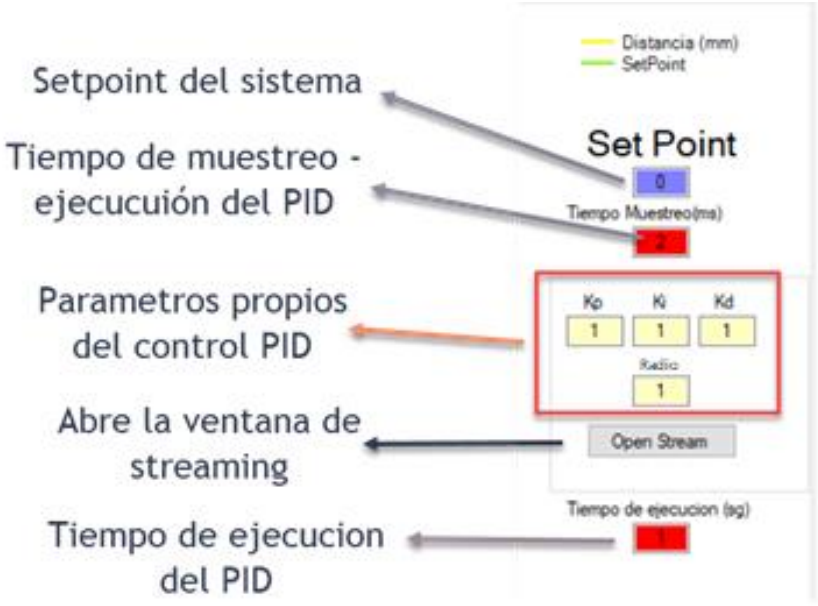


Ilustración 49 Esquema barra lateral

Esta barra tiene todos los parámetros necesarios para modificar los parámetros del PID que son los siguientes:

- Setpoint
- Tiempo de Muestreo
- Kp, kd, ki, y radio de la bola
- Tiempo de ejecución

Por otro lado están limitados y acotados ciertos parametros para no generar problemas o instabilidades con el sistema real.

```
524.        If ts < 2 Or ts > 11 Then
525.     MessageBox.Show("El numero no es correcto solo aceptado en ts entre (2,11)", "Petici
    on no aceptada", MessageBoxButtons.OK, MessageBoxIcon.Error)
```

El ultimo botón es el que nos permite visualizar la cámara conectada al sistema para así poder realizar de forma manual y visible la configuración del PID.

Para ello abre una ventana en nuestro navegador predeterminado con una conexión con la cámara en streaming [Ilustración 50 Navegador con streaming].

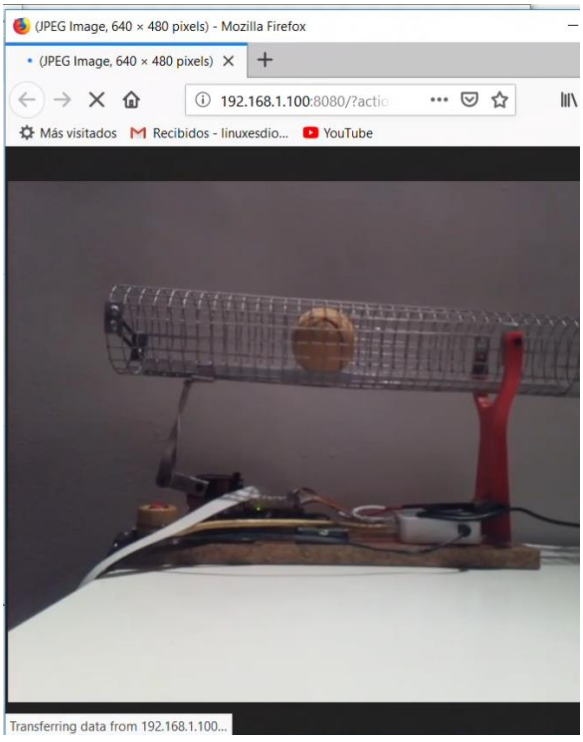


Ilustración 50 Navegador con streaming

5.2.1 Barra baja

En la parte inferior se encuentran los componentes de guardado. En este programa se puede guardar mediante imagen o mediante fichero txt que almacena los datos para procesar con la funcion de exportar de cualquier hoja de calculo .

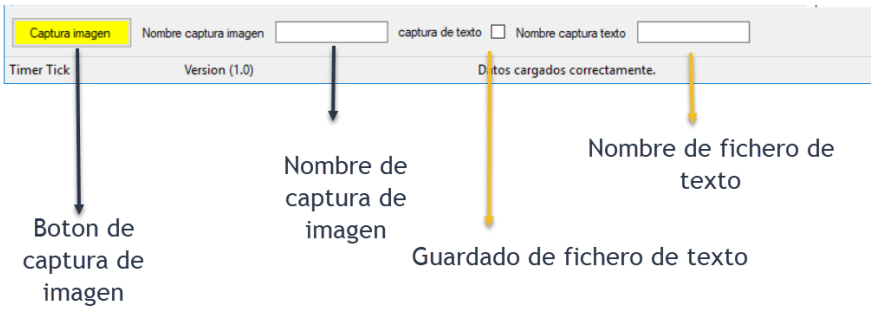


Ilustración 51 Esquema barra inferior

La función de captura de imágenes guarda en un fichero con el nombre que previamente se escribió en la caja a su derecha llamada “nombre captura imagen”.
Se dispone de un botón ok en el que se guardara los fichero de texto cuyo nombre este en el box llamado “nombre de captura de texto“, y quedara guardado a su vez en la parte superior del fichero los parámetros de PID utilizados, en la primera línea.

5.2.2 Área de gráfica

En la parte central de la aplicación de cliente nos encontramos con la parte de gráfica los tamaños y máximos de la misma se pueden modificar desde la ventana de configuración antes explicada.
La línea de color amarillo nos indica la posición recogida por el sensor mientras que la línea verde nos muestra nuestro setpoint en caso de que no se esté ejecutando el PID del sistema.

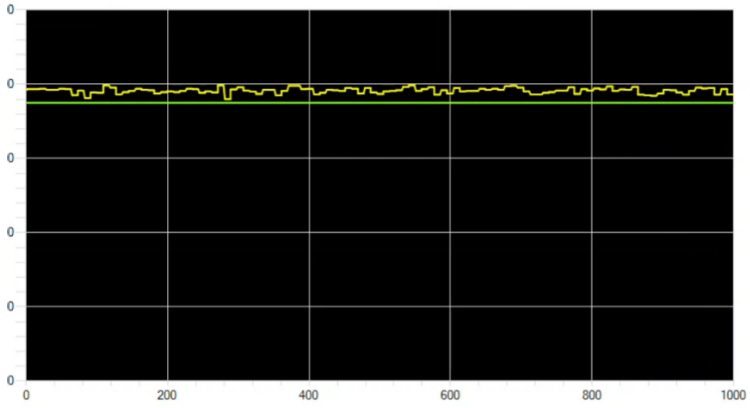


Ilustración 52 Captura de grafica sin ejecución del PID

En el caso de que se halla realizado la ejecución del PID se modificara el color de la línea de setpoint y medidas, pasando esta a azul y blanca respectivamente, con el objetivo de facilitar la visualización del estado del sistema.

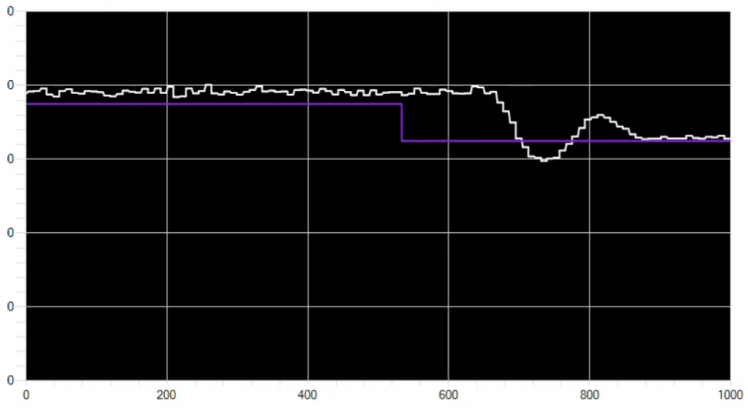


Ilustración 53 Captura de grafica ejecutando PID

6 USO DEL DISPOSITIVO

El dispositivo tiene varios métodos de ajuste del PID. El primer método a tratar es el método matemático, esto es, usando distintas reglas o criterios de sintonía, pero entre los objetivos del proyecto no estaba realizar un ajuste propio, sino que otras personas aprendan los ajustes y como emplearlos. Por ello, este apartado será una guía no demasiado detallada de los distintos métodos [15].

Extendido en la ingeniería está la existencia de tres grandes tipos de criterios de sintonía [15], que son:

- Características de la respuesta temporal.
- Integrales de la señal de error.
- Características de la respuesta en frecuencia.

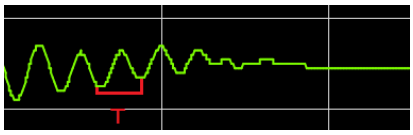
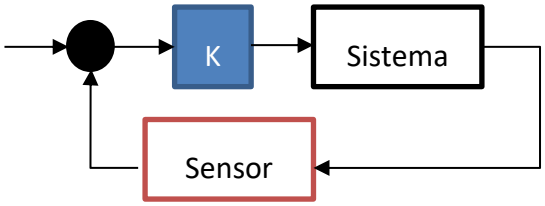
Se intentará explicar alguno de estos métodos con ejemplos puntuales en cada caso.

6.1 ZIEGLER-NICHOLS EN LAZO CERRADO (RESPUESTA TEMPORAL) [16]

Este método es posible usarlo en algunas condiciones especiales, que son las siguientes:

- Amortiguamiento de ¼ ante perturbaciones. (QDR)
- Métodos en lazo abierto y lazo cerrado.
- Válidos para $0.15 < d/\tau < 0.6$ y procesos monótonos.
- Dan valores aproximados: requieren ajuste fino.

La metodología a usar es realizar un experimento utilizando la planta y un controlador proporcional llamado KP en este caso, y forzar el sistema hasta encontrar el punto de estabilidad crítica, tras ello se toman los siguientes datos:

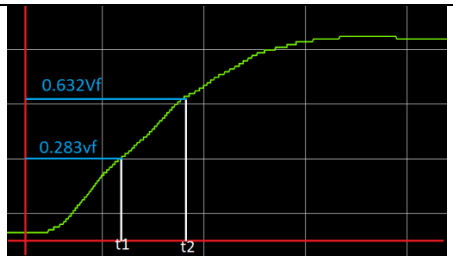
Obtención de T	Obtención de KP
	
T es el periodo de ajuste oscilatorio	K es la máxima ganancia que hace el sistema estable

Y después se podrán ajustar estos parámetros según una tabla hallada mediante método experimental.

Controlador utilizado	Kp	Ki	Kd
P	0.5 K		
PI	0.45 K	T/1.2	
PID paralelo	0.75 K	T/1.6	T/10
PID serie	0.6 K	T/2	T/8

6.2 ZIEGLER-NICHOLS EN LAZO ABIERTO (RESPUESTA TEMPORAL) [16]

Como en el caso anterior, se tendrá que realizar una prueba y tomar parámetros. En este caso son los siguientes:

Parametros necesarios	Fórmulas utilizadas
	$\tau = 1.5(t_2 - t_1)$ $d = t_2 - \tau$ $K = \frac{\Delta y}{\Delta u}$ $\frac{Ke^{-ds}}{\tau S + 1}$

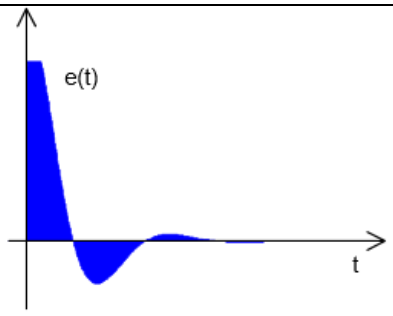
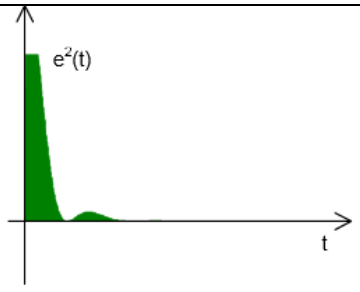
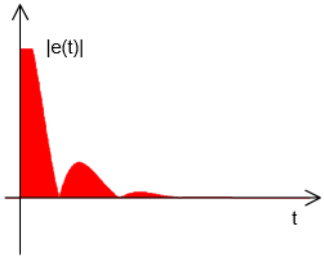
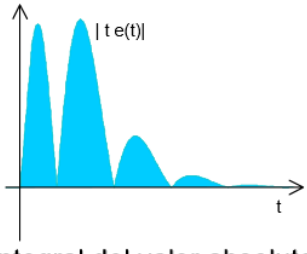
Como en el caso anterior, solo se tendrá que sustituir desde otra tabla experimental:

Controlador utilizado	Kp	KI	Kd
P	τ / (K*d)		
PI	0.9τ /(K*d)	3.33 d	
PID serie	1.2τ /(K*d)	2 d	0.5 d

6.3 DISTINTOS TIPOS DE MINIMIZACIÓN INTEGRAL DEL ERROR [16]

Este método está basado en utilizar el error dinámico como criterio de diseño durante la sintonización de un PID. Varios tipos de sintonizadores han sido desarrollados por distintos investigadores nombrar los más importantes Lopez, Murrill, Smith(1967), Rovira (1969) y Kaya, Scheib (1988).

Los tipos principales de criterios para intentar mejorar la respuesta de los sistemas mediante la sintonización del error integral son:

 IE (Integral del error)	 ISE (Integral del cuadrado del error)
e	e^2
 IAE (Integral del valor absoluto del error)	 ITAE (Integral del valor absoluto del error ponderado en el tiempo)
$\|e\|$	$\|e\|^t$

La metodología experimental es similar al ejemplo anterior y se obtendrá:

$$\tau = 1.5(t_2 - t_1)$$
$$d = t_2 - \tau$$
$$K = \frac{\Delta y}{\Delta u}$$

Tras esto solo tendremos que sustituir en las tablas dadas por cada uno de estos investigadores.

$$K_p K = a\left(\frac{d}{\tau}\right)^b \qquad \frac{\tau}{Ti} = a\left(\frac{d}{\tau}\right)^b \qquad \frac{Td}{\tau} = a\left(\frac{d}{\tau}\right)^b$$

siendo parámetros la “a” y la “b”

Un ejemplo de tabla puede ser, este en este caso, López: [16]

Criterio	Proporcional	Integral	Derivativo
MIAE	a=0.984 b=-0.986	a=0.608 b=-0.707	
MISE	a=1.305 b=-0.959	a=0.492 b=-0.739	
MITAE	a=0.859 b=-0.977	a=0.674 b=-0.68	

6.4 MÉTODO MANUAL

Un método de sintonización normalmente utilizado consiste en sustituir por 0 Kp y Ki, aumentar la Kp hasta que la salida del bucle oscila, pero sigue siendo estable.

Entonces se tiene que la K_p final se debe establecer en aproximadamente la mitad de ese valor.

A continuación, se aumenta K_i hasta que cualquier desviación sea corregida en un tiempo suficientemente bajo para el proceso. Sin embargo, no se aumenta demasiado K_i, puesto que provocará inestabilidad.

Finalmente, se aumenta K_d si es necesario, hasta que la recuperación sea aceptablemente rápida para alcanzar su referencia después de una perturbación, estando en el setpoint. Sin embargo, demasiado K_d hará que la respuesta sea excesiva.
Una sintonización rápida del bucle PID en este caso usualmente sobrepasa ligeramente para alcanzar el punto de ajuste con mayor rapidez.
Una tabla para facilitar el ajuste podría ser así:

Parámetro	Tiempo de subida	Sobrepico	Tiempo de estabilización	Error de estado estacionario	Estabilidad
K_p	Disminución	Aumentar	Pequeño cambio	Disminución	Disminuir
K_i	Disminución	Aumentar	Aumentar	Eliminar	Disminuir
K_d	Modificación ligeramente menor	Disminución	Disminución	No hay efecto en teoría	Mejorar si Kd es pequeño

Como norma general se puede decir que:
Para Kp: Cuanto mayor sea este coeficiente, mayor es la respuesta del sistema, pero parece más sensible y reactivo al cambio angular. Si es demasiado baja el sistema será lento y más difícil de mantener constante. También se debe decir que si es demasiado alto el sistema comienza a oscilar a alta frecuencia.
Para Ki: Este coeficiente puede aumentar la precisión de la posición angular. Sin embargo, cuando el valor de I es demasiado alto, el sistema podría comenzar a tener reacciones lentas y un efecto de disminución de la ganancia proporcional como consecuencia. También podría comenzar a oscilar como consecuencia de tener alta ganancia P, pero con una frecuencia más baja.
Para Kd: Este coeficiente permite al sistema alcanzar más rápidamente la posición deseada. Algunas personas lo llaman el parámetro de aceleración, ya que amplifica la entrada del usuario. También disminuye la acción de control rápidamente cuando el error está disminuyendo rápidamente. En la práctica, se aumentará la velocidad de reacción y en ciertos casos producirá un aumento del efecto de la ganancia de P. La ganancia D hace que el sistema sea más sensible.

7 RESULTADOS Y CONCLUSIONES

Basándonos en los objetivos iniciales del proyecto, en líneas generales se han completado los propósitos de del mismo.
De forma específica el objetivo I que es obtener el modelo matemático se ha cumplido, ya que la respuesta del modelo-maqueta se corresponde a la perfección con los datos obtenidos el análisis matemático y físico.
El segundo objetivo que era entender la forma de las distintas partes de un sistema automático creo que está satisfecho porque se puede entender a la perfección cada objeto del sistema y entender las acciones que este realiza.
El punto III de los objetivos que es la realización del ajuste del PID por distintas metodologías queda suficiente explicado en el proyecto, dando con valores de PID estables mediante estos procedimientos explicados en el punto 6.
Los restantes puntos dentro de los objetivos son de realización de proyecto es decir son de desarrollo técnico y puede comprobarse que la aplicación funciona a la perfección, aunque cabe posibles mejoras no contempladas en este proyecto.

El uso de la Raspberry Pi como sistema de control no era la primera opción propuesta, pero el hecho de realizar un streming mediante webcam hacía necesario un microcontrolador bastante potente, potencia que no podía ser proporcionada con dispositivos más simples a priori como Arduinos , esp8266 o PIC`s.

La parte de desarrollo didáctico, fin último del TFM, que consistía en la generación de una documentación para poder realizar las prácticas a distancia ha sido obtenida con creces. Sin embargo, al depender de parámetros de red o de internet, dichas variables están fuera de control, por lo que el procesamiento de datos y respuesta del PID tenía que estar implementado dentro del servidor. De esta forma se evita generar retrasos en el bucle de control y generando un sistema inestable.

La opción de programar el cliente con Python ha sido una opción más complicada que lo que podría intuirse en un principio ya que es un lenguaje que desconocía a nivel personal, sin embargo, está siendo muy utilizado en universidades de todo el mundo debido a su limpieza y su versatilidad, a su vez también es ampliamente usado para

machine learning, bigdata, etc., a niveles mucho más empresariales dejando atrás a lenguajes clásicos como C o Java.

En la parte de resultado a nivel controlador y parámetros de control ha sido satisfactorio: se ha establecido un alto nivel de control del sistema ya que la motivación pedagógica pasaba por ser "muy fácil ni muy difícil" de realizar la obtención de parámetros. Por ello a nivel personal creo que se ha cumplido los objetivos como se puede ver en las siguientes gráficas donde la respuesta a escalón es exacta para alcanzar el setpoint:

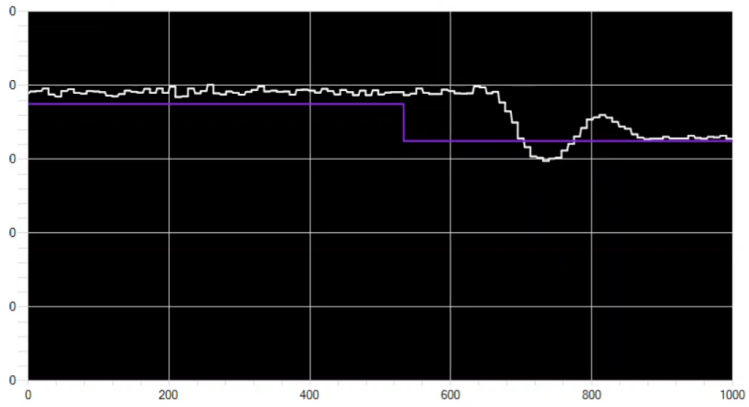


Ilustración 54 respuesta a escalón de sistema

Además, podemos aumentar la dificultad del sistema modificando parámetros físicos reales como el peso o el material de la bola, lo cual crea una gran versatilidad de la práctica dentro de la escuela.

8 BIBLIOGRAFÍA

[1] K. OGATA, INGENIERIA DE CONTROL MODERNA 5ª ED., PRENTICE-HALL, 2010.

[2] K. OGATA, SYSTEM DYNAMICS, Pearson Education Inc.

[3] K. OGATA, DISCRETE-TIME CONTROL SYSTEMS, PRENTICE HALL TRAVEL, 2015.

[4] Server, Socket, «Socket Server,» [En línea]. Available: https://pythonprogramminglanguage.com/socket-server/. [Último acceso: 02 09 2019].

[5] jmirezcontro, «http://jmirezcontrol.files.wordpress.com/2012/01/control_pid_jorge_mirez_2012.pdf, » [En línea]. [Último acceso: 2 9 2019].

[6] robots-argentina.com.ar, «VL53L0X: Sensor de distancia que mide por la velocidad de la luz (Time-of-Fly),» [En línea]. Available: http://robots-argentina.com.ar/didactica/vl53l0x-sensor-de-distancia-que-mide-por-la-velocidad-de-la-luz-time-of-fly/. [Último acceso: 2 9 2019].

[7] Pythones.net, «Qué es Python – Definición, características y ventajas,» [En línea]. Available: https://pythones.net/que-es-python-y-sus-caracteristicas/. [Último acceso: 02 09 2019].

[8] Salcedo, Luis, «Algoritmos Hash - Criptografía con Python y HashLib,» [En línea]. Available: http://www.pythondiario.com/2017/09/algoritmos-hash-criptografia-con-python.html. [Último acceso: 02 09 2019].

[9] johnbryanmoore, «johnbryanmoore/VL53L0X_rasp_python,» [En línea]. Available: https://github.com/johnbryanmoore/VL53L0X_rasp_python/blob/master/python/VL53L0X_example.py. [Último acceso: 02 09 2019].

[10] sankalpjonn, «sankalpjonn/timeloop,» 2 9 2019. [En línea]. Available: https://github.com/sankalpjonn/timeloop. [Último acceso: 2 9 2019].

[11] Ltd., Ivmech Mechatronics Innovation, «ivmech/ivPID,» [En línea]. Available: https://github.com/ivmech/ivPID. [Último acceso: 02 09 2019].

[12] J. L. Cano, «Teoría de control en Python con SciPy (II): Control PID,» [En línea]. Available: https://www.pybonacci.org/2013/11/06/teoria-de-control-en-python-con-scipy-ii-control-pid/. [Último acceso: 2 9 2019].

[13] -the-beginner’s-pid, «http://brettbeauregard.com/blog/2011/04/improving-the-beginner’s-pid-derivative-kick/,» [En línea]. Available: http://brettbeauregard.com/blog/2011/04/improving-the-beginner’s-pid-derivative-

kick/. [Último acceso: 02 09 2019].

[14] simplot, «hg clone https://code.google.com/p/projectsimplot/,» [En línea]. [Último acceso: 2 9 2019].

[15] f. morilla, «http://www.dia.uned.es/~fmorilla/MaterialDidactico/ajuste_empirico.pdf,» [En línea]. [Último acceso: 2 9 2019].

[16] universidad de valladolid, «http://www.isa.cie.uva.es/~maria/pids.pdf,» [En línea]. [Último acceso: 02 09 2019].

[17] l. llamas, «Medir distancia con precisión con Arduino y sensor láser VL53L0X y VL6180X,» [En línea]. Available: https://www.luisllamas.es/arduino-sensor-distancia-vl53l0x/. [Último acceso: 2 9 2019].

[18] tutorialsRaspberryPi, «Raspberry Pi Servo Motor control,» [En línea]. Available: https://tutorials-raspberrypi.com/raspberry-pi-servo-motor-control/. [Último acceso: 02 09 2019].

9 INDICE DE IMÁGENES

Ilustración 1 Esquema de sistema 7
Ilustración 2 Esquema de sistema mecánico biela manivela 9
Ilustración 3 Esquema en 3D del sistema inestable 9
Ilustración 4 Esquema en 3D del sistema estable 10
Ilustración 5 Fotografía del modelo del sistema 11
Ilustración 6 Esquema de para el modelo matemático 13
Ilustración 7 posición de los polos de lazo cerrado al variar la ganancia del sistema con realimentación 19
Ilustración 8 Salida ante entrada escalón y controlador KP 20
Ilustración 9 Respuesta escalón PD (kd bajo) 21
Ilustración 10 Respuesta escalón PD (kd alto) 21
Ilustración 11 sistema con KI 22
Ilustración 12 Lugar de las raíces en Z 23
Ilustración 13 Detalle del lugar de las raíces en Z 24
Ilustración 14 Modelo de comunicación I2c 27
Ilustración 15 Pinout de raspberry pi para I2c 27
Ilustración 16 Captura de comunicación I2c con osciloscopio 28
Ilustración 17 Esquema de comunicación tipo socket 29
Ilustración 18 Pruebas socket en Ubuntu 30
Ilustración 19 Pruebas socket en Windows 31
Ilustración 20 Trama de protocolo UART. 31
Ilustración 21 Pinout serial de Raspberry Pi 32
Ilustración 22 Tabla de caracteres ASCII 33
Ilustración 23 Ejemplo de escritura para graficar 34
Ilustración 24 Captura con osciloscopio del protocolo UART de la letra “a” en ASCII 34
Ilustración 25 Esquema de servomotor 35
Ilustración 26 Código de colores para conexionado de servomotor 36
Ilustración 27 Señales digitales de control de servos 36
Ilustración 28 Explicación gráfica del actuador proporcional 38
Ilustración 29 Explicación gráfica del actuador proporcional 39
Ilustración 30Gráfica del actuador derivativo 40
Ilustración 31 Pinout de raspberry pi 43
Ilustración 32 Distincion de pinout entre tipos de raspberry pi 44
Ilustración 33 Funciones de pinout en Raspberry Pi 45

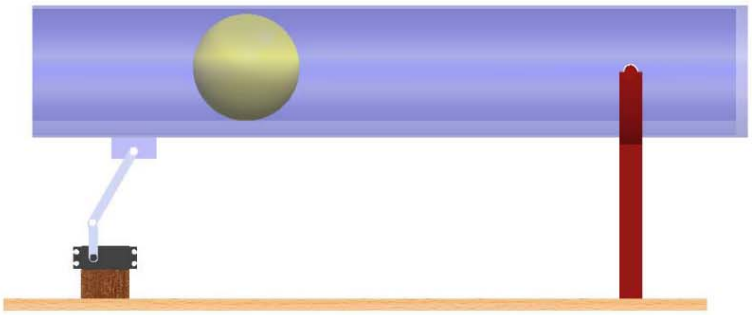
Ilustración 34 Diferencias entre GPIO o al modo BCM 46
Ilustración 35 Símbolo Raspberry Pi..... 47
Ilustración 36 Conexión a Raspberry pi modo consola 48
Ilustración 37 Raspberry en modo gráfico 49
Ilustración 38 Crowtail - Base Shield conectada a Raspberry Pi 50
Ilustración 39 Dimensión de sensor 51
Ilustración 40 Diferencia de detección de sensor en modo predeterminado (gris-blanco)..... 52
Ilustración 41 Diferencia de detección de sensor en modo largo(gris-blanco) 52
Ilustración 42 Medidas máxima de el sensor..... 53
Ilustración 43 velocidad y presión máxima del sensor según color..... 53
Ilustración 44 Esquema de las conexiones posibles 55
Ilustración 45 Esquemas de tipos de lenguaje..... 57
Ilustración 46 diferencia entre programa original y modificado 73
Ilustración 47 Esquema barra superior 74
Ilustración 48 ventana de configuración..... 75
Ilustración 49 Esquema barra lateral 78
Ilustración 50 Navegador con streaming 79
Ilustración 51 Esquema barra inferior..... 80
Ilustración 52 Captura de grafica sin ejecución del PID..... 80
Ilustración 53 Captura de grafica ejecutando PID..... 81
Ilustración 54 respuesta a escalón de sistema..... 90

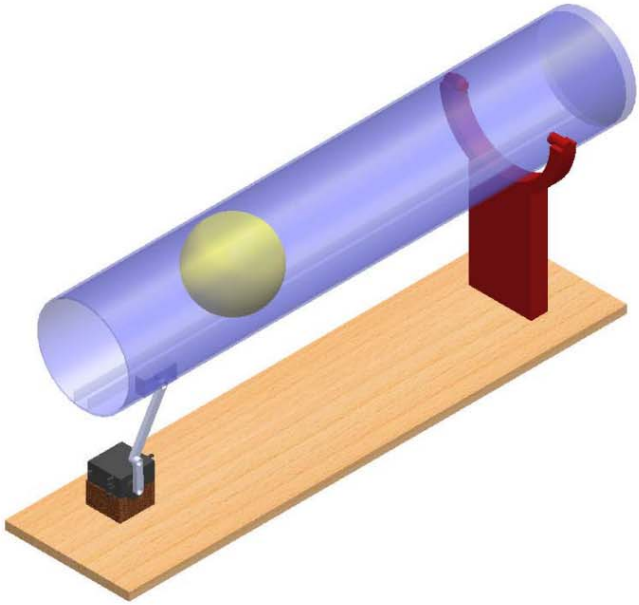
10 INDICE DE ECUACIONES

Ecuación 1 Ecuación básica de lagrangiano 12
Ecuación 2 Ecuación cinética de la viga 12
Ecuación 3 Ecuación cinética de la bola 12
Ecuación 4 energía potencial del sistema 13
Ecuación 5 Ecuación de relación angular 13
Ecuación 6 velocidad lineal de la bola 14
Ecuación 7 Aceleración lineal de la bola 14
Ecuación 8 Relación coordenadas cartesianas con generalizadas 14
Ecuación 9 Coordenadas angulares 14
Ecuación 10 Elevación al cuadrado de coordenadas angulares 15
Ecuación 11 Velocidad lineal de la bola 15
Ecuación 12 Ecuación general del lagrangiana aplicada 15
Ecuación 13 Sustitución en momentos inerciales 16
Ecuación 14 Ecuaciones del movimiento 16
Ecuación 15 cálculo de la primera ecuación 16
Ecuación 16 Resultado del movimiento 16
Ecuación 17 segunda ecuación 17
Ecuación 18 Resultado de la segunda ecuación 17
Ecuación 19 Desarrollo de la serie de Taylor sobre primera ecuación 17
Ecuación 20 aplicación de la transformación de Laplace 17
Ecuación 21 Función de transferencia 18
Ecuación 22 Función de transferencia 19
Ecuación 23 cálculo de errores 69

ANEXO I

PLANOS EN 3D



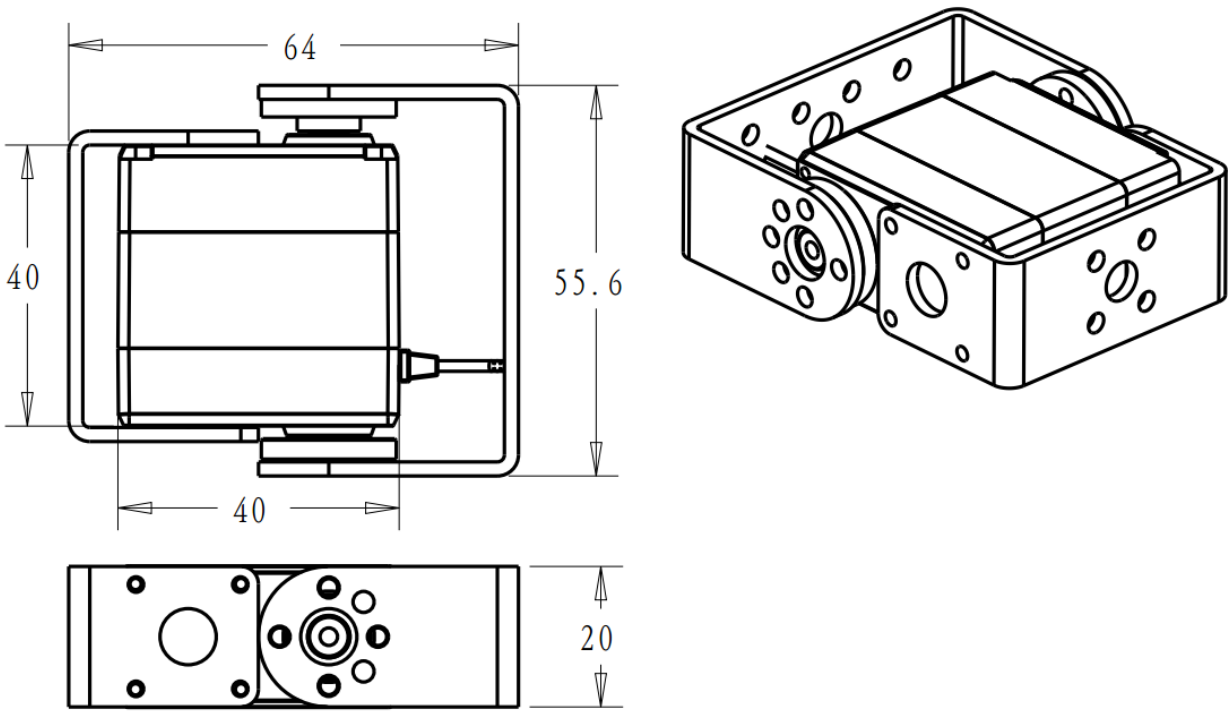


SI NO SE INDICA LO CONTRARIO: LAS COTAS SE EXPRESAN EN MM ACABADO SUPERFICIAL: TOLERANCIAS: LINEAL: ANGULAR:	ACABADO: pablo martin fernandez		REBARBAR Y ROMPER ARISTAS VIVAS	NO CAMBIE LA ESCALA		REVISIÓN 1	
	NOMBRE	FIRMA	FECHA				TÍTULO: viga -bola
DIBUJ.							
VERIF.							
APROB.							
FABR.							
CALID.				MATERIAL:		N.º DE DIBUJO TFMaster	A4
				PESO:		ESCALA:1:10	HOJA 1 DE 1

ANEXO IV

DATASHEET DE SERVOMOTOR RDS3218

产品型号 (Product Name): RDS3218
产品描述 (Product Description): 6V 20kg Robot Digital Servo
产 品 图 (Drawing)



1. 使用环境条件 Apply Environmental Condition

No.	Item	Specification
1-1	存储温度 Storage Temperature Range	-30℃～80℃
1-2	运行温度 Operating Temperature Range	-15℃～70℃
1-3	工作电压范围 Operating Voltage Range	4.8-6.8V

2. 机械特性 Mechanical Specification

No.	Item	Specification
2-1	尺寸 Size	40*20*40mm
2-2	重量 Weight	60g
2-3	齿轮比 Gear ratio	373
2-4	轴承 Bearing	Double bearing
2-5	舵机线 Connector wire	300±5mm
2-6	马达 Motor	3-pole
2-7	防水性能 Waterproof performance	No

更多的舵机规格书和 3D 模型，请到网站下载(For more servo datasheet and 3D files, please go to the website to download) www.dsservo.com

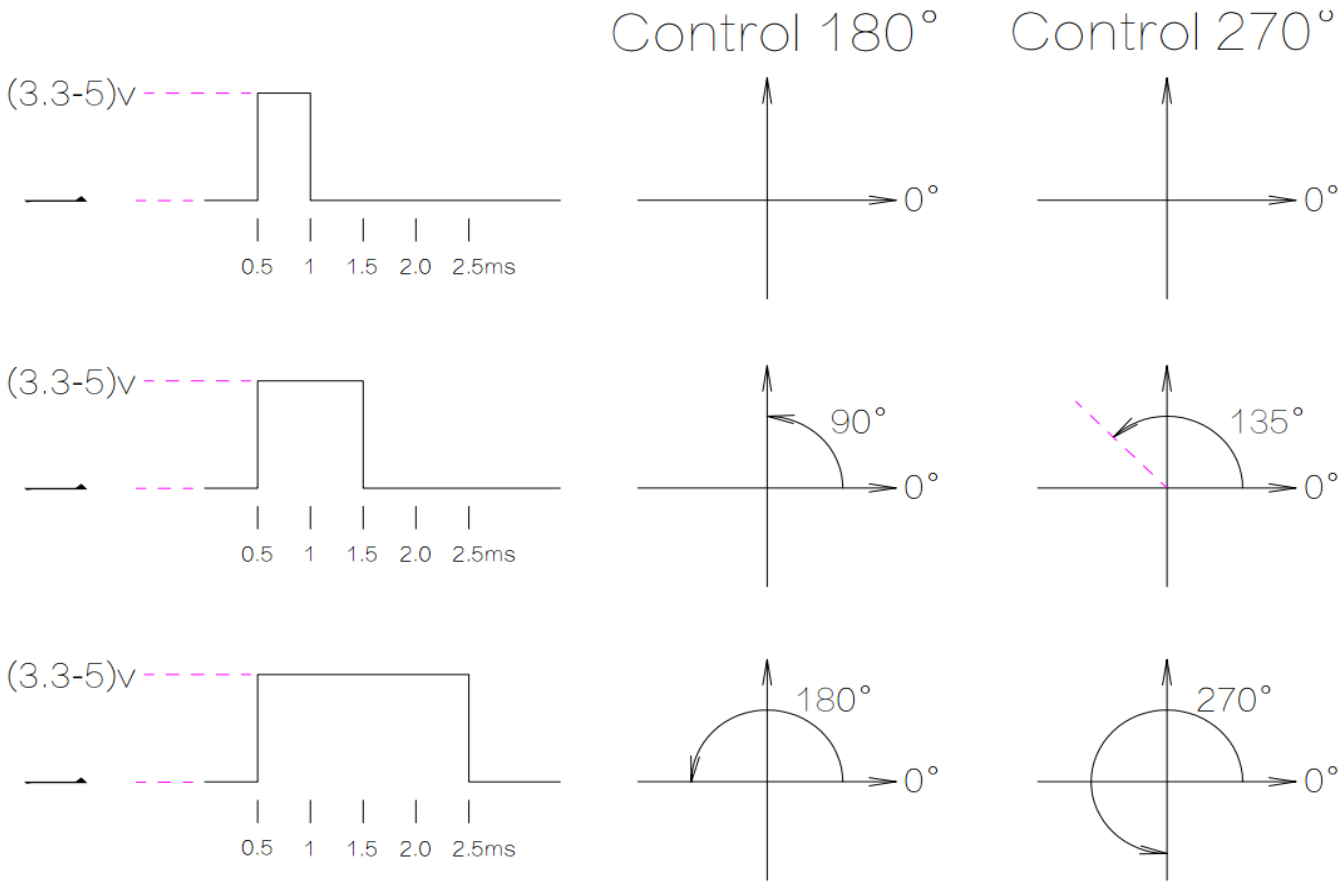
3. 电气特性 Electrical Specification

No.	工作电压 Operating Voltage	5V	6.8V
3-1	待机电流 Idle current(at stopped)	4mA	5mA
3-2	空载转速 Operating speed (at no load)	0.19 sec/60°	0.17sec/60°
3-3	堵转扭矩 Stall torque (at locked)	18.5 kg-cm	21 kg-cm
3-4	堵转电流 Stall current (at locked)	1.8A	2.2A

4. 控制特性 Control Specification

No.	Item	Specification
4-1	驱动方式 Control System	PWM(Pulse width modification)
4-2	脉宽范围 Pulse width range	500～2500μsec
4-3	中点位置 Neutral position	1500μsec
4-4	控制角度 Running degree	180°or 270° (when 500～2500 μ sec)
4-5	控制精度 Dead band width	3 μsec
4-6	控制频率 Operating frequency	50-330Hz
4-7	旋转方向 Rotating direction	Counterclockwise (when 500～2500 μsec)

5. 关于 PWM 控制说明 About PWM Control

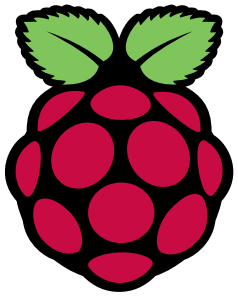


更多的舵机规格书和 3D 模型，请到网站下载(For more servo datasheet and 3D files, please go to the website to download) www.dsservo.com

ANEXO III

DATASHEET DE RASPBERRY PI

DATASHEET



Raspberry Pi Compute Module 3+

Raspberry Pi Compute Module 3+ Lite

Release 1, January 2019

Table 1: Release History

Release	Date	Description
1	28/01/2019	First release

The latest release of this document can be found at `https://www.raspberrypi.org`

Contents

1 Introduction **5**

2 Features **6**
2.1 Hardware . . . 6
2.2 Peripherals . . . 6
2.3 Software . . . 6

3 Block Diagram **7**

4 Mechanical Specification **8**

5 Pin Assignments **9**

6 Electrical Specification **11**

7 Power Supplies **13**
7.1 Supply Sequencing . . . 14
7.2 Power Requirements . . . 14

8 Booting **14**

9 Peripherals **15**
9.1 GPIO . . . 15
9.1.1 GPIO Alternate Functions . . . 16
9.1.2 Secondary Memory Interface (SMI) . . . 17
9.1.3 Display Parallel Interface (DPI) . . . 17
9.1.4 SD/SDIO Interface . . . 18
9.2 CSI (MIPI Serial Camera) . . . 18
9.3 DSI (MIPI Serial Display) . . . 18
9.4 USB . . . 18
9.5 HDMI . . . 18
9.6 Composite (TV Out) . . . 19

10 Thermals **19**
10.1 Temperature Range . . . 19

11 Availability **19**

12 Support **19**

List of Figures

1 CM3+ Block Diagram 7
2 CM3+ Mechanical Dimensions 8
3 Digital IO Characteristics 13

List of Tables

1 Release History 1
2 Compute Module 3+ SODIMM Connector Pinout 9
3 Pin Functions 10
4 Absolute Maximum Ratings 11
5 DC Characteristics 12
6 Digital I/O Pin AC Characteristics 12
7 Power Supply Operating Ranges 13
8 Mimimum Power Supply Requirements 14
9 GPIO Bank0 Alternate Functions 16
10 GPIO Bank1 Alternate Functions 17

1 Introduction

The Raspberry Pi Compute Module 3+ (CM3+) is a range of DDR2-SODIMM-mechanically-compatible System on Modules (SoMs) containing processor, memory, eMMC Flash (on non-Lite variants) and supporting power circuitry. These modules allow a designer to leverage the Raspberry Pi hardware and software stack in their own custom systems and form factors. In addition these modules have extra IO interfaces over and above what is available on the Raspberry Pi model A/B boards, opening up more options for the designer.

The CM3+ contains a BCM2837B0 processor (as used on the Raspberry Pi 3B+), 1Gbyte LPDDR2 RAM and eMMC Flash. The CM3+ is currently available in 4 variants, CM3+/8GB, CM3+/16GB, CM3+/32GB and CM3+ Lite, which have 8, 16 and 32 Gigabytes of eMMC Flash, or no eMMC Flash, respectively.

The CM3+ Lite product is the same as CM3+ except the eMMC Flash is not fitted, and the SD/eMMC interface pins are available for the user to connect their own SD/eMMC device.

Note that the CM3+ is electrically identical and, with the exception of higher CPU z-height, physically identical to the legacy CM3 products.

CM3+ modules require a software/firmware image dated November 2018 or newer to function correctly.

2 Features

2.1 Hardware

- Low cost
- Low power
- High availability
- High reliability
  - Tested over millions of Raspberry Pis Produced to date
  - Module IO pins have 15 micro-inch hard gold plating over 2.5 micron Nickel

2.2 Peripherals

- 48x GPIO
- 2x I2C
- 2x SPI
- 2x UART
- 2x SD/SDIO
- 1x HDMI 1.3a
- 1x USB2 HOST/OTG
- 1x DPI (Parallel RGB Display)
- 1x NAND interface (SMI)
- 1x 4-lane CSI Camera Interface (up to 1Gbps per lane)
- 1x 2-lane CSI Camera Interface (up to 1Gbps per lane)
- 1x 4-lane DSI Display Interface (up to 1Gbps per lane)
- 1x 2-lane DSI Display Interface (up to 1Gbps per lane)

2.3 Software

- ARMv8 Instruction Set
- Mature and stable Linux software stack
  - Latest Linux Kernel support
  - Many drivers upstreamed
  - Stable and well supported userland
  - Full availability of GPU functions using standard APIs

3 Block Diagram

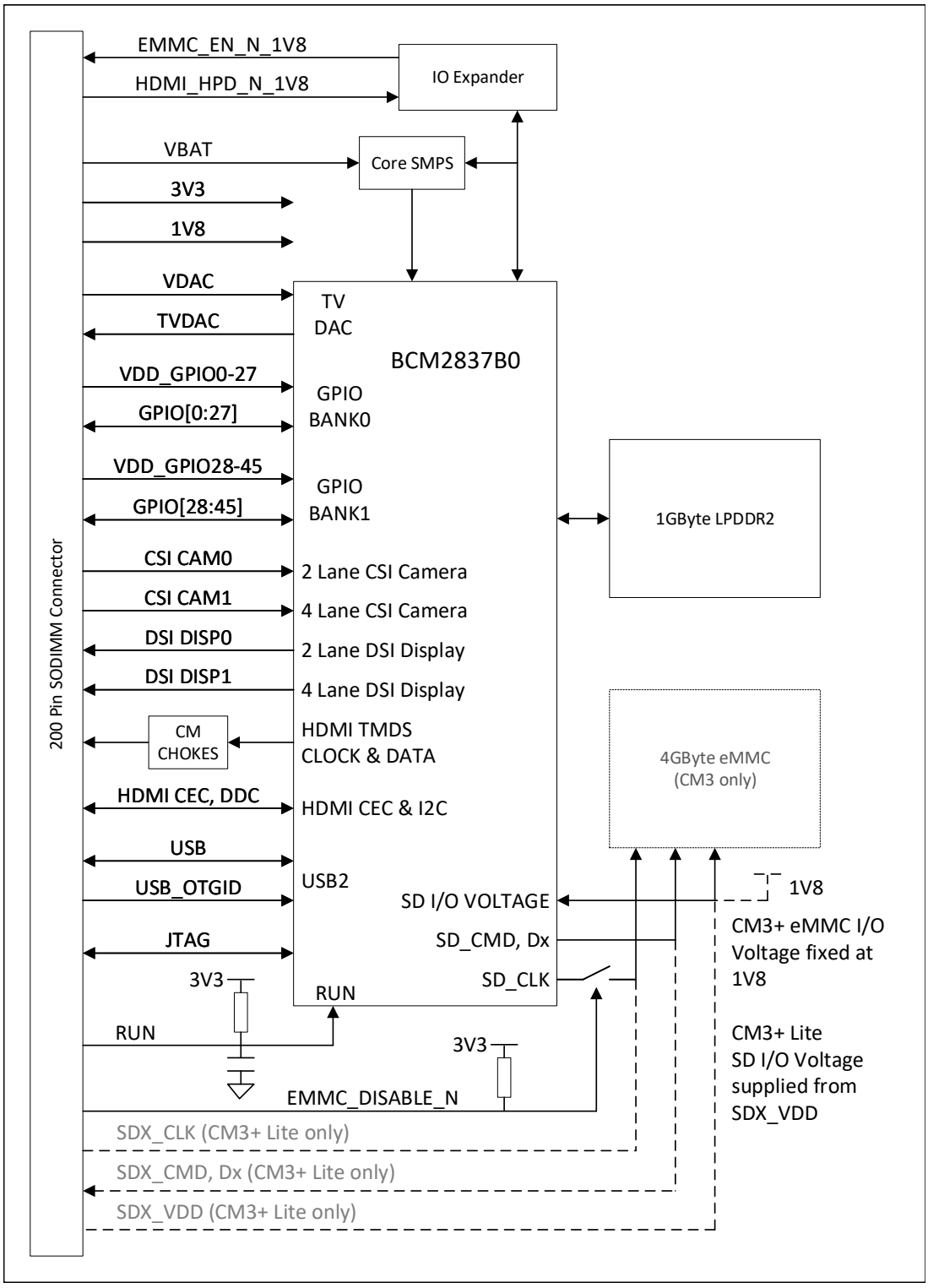


Figure 1: CM3+ Block Diagram

4 Mechanical Specification

The CM3+ modules conform to JEDEC MO-224 mechanical specification for 200 pin DDR2 (1.8V) SODIMM modules and therefore should work with the many DDR2 SODIMM sockets available on the market. **(Please note that the pinout of the Compute Module is not the same as a DDR2 SODIMM module; they are not electrically compatible.)**

The SODIMM form factor was chosen as a way to provide the 200 pin connections using a standard, readily available and low cost connector compatible with low cost PCB manufacture.

The maximum component height on the underside of the Compute Module is 1.2mm.

The maximum component height on the top side of the Compute Module is 2.5mm.

The Compute Module PCB thickness is 1.0mm +/- 0.1mm.

Note that the location and arrangement of components on the Compute Module may change slightly over time due to revisions for cost and manufacturing considerations; however, maximum component heights and PCB thickness will be kept as specified.

Figure 2 gives the CM3+ mechanical dimensions.

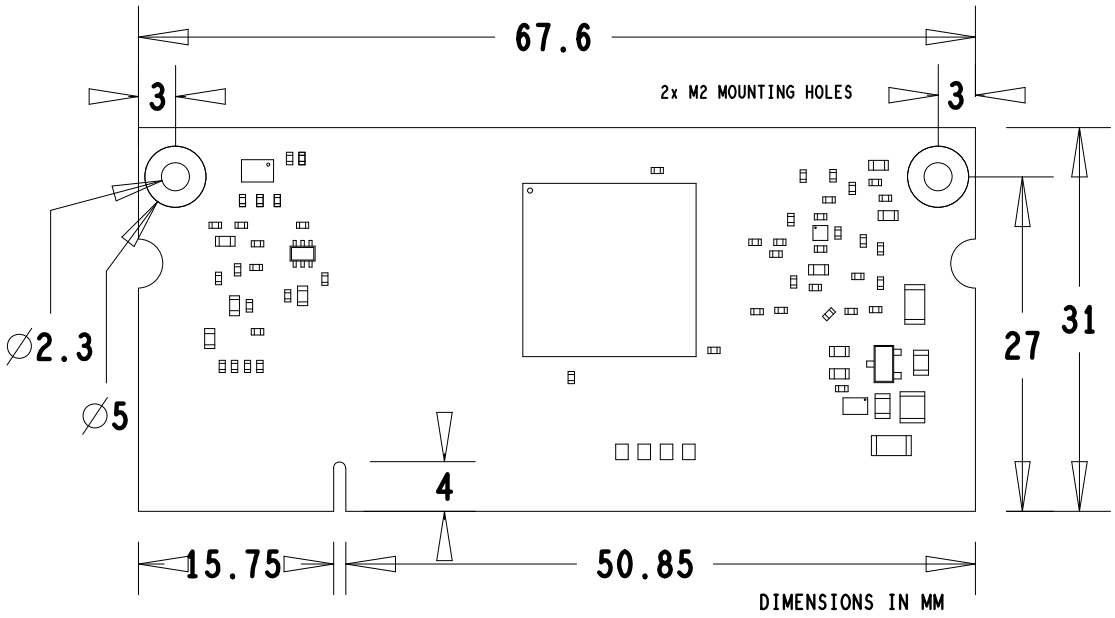


Figure 2: CM3+ Mechanical Dimensions

5 Pin Assignments

CM3+	CM3+ Lite	PIN		PIN	CM3+	CM3+ Lite
GND		1		2	EMMC_DISABLE_N	
GPIO0		3		4	NC	SDX_VDD
GPIO1		5		6	NC	SDX_VDD
GND		7		8	GND	
GPIO2		9		10	NC	SDX_CLK
GPIO3		11		12	NC	SDX_CMD
GND		13		14	GND	
GPIO4		15		16	NC	SDX_D0
GPIO5		17		18	NC	SDX_D1
GND		19		20	GND	
GPIO6		21		22	NC	SDX_D2
GPIO7		23		24	NC	SDX_D3
GND		25		26	GND	
GPIO8		27		28	GPIO28	
GPIO9		29		30	GPIO29	
GND		31		32	GND	
GPIO10		33		34	GPIO30	
GPIO11		35		36	GPIO31	
GND		37		38	GND	
GPIO0-27_VDD		39		40	GPIO0-27_VDD	
		KEY				
GPIO28-45_VDD		41		42	GPIO28-45_VDD	
GND		43		44	GND	
GPIO12		45		46	GPIO32	
GPIO13		47		48	GPIO33	
GND		49		50	GND	
GPIO14		51		52	GPIO34	
GPIO15		53		54	GPIO35	
GND		55		56	GND	
GPIO16		57		58	GPIO36	
GPIO17		59		60	GPIO37	
GND		61		62	GND	
GPIO18		63		64	GPIO38	
GPIO19		65		66	GPIO39	
GND		67		68	GND	
GPIO20		69		70	GPIO40	
GPIO21		71		72	GPIO41	
GND		73		74	GND	
GPIO22		75		76	GPIO42	
GPIO23		77		78	GPIO43	
GND		79		80	GND	
GPIO24		81		82	GPIO44	
GPIO25		83		84	GPIO45	
GND		85		86	GND	
GPIO26		87		88	HDMI_HPD_N_1V8	
GPIO27		89		90	EMMC_EN_N_1V8	
GND		91		92	GND	
DSI0_DN1		93		94	DSI1_DP0	
DSI0_DP1		95		96	DSI1_DN0	
GND		97		98	GND	
DSI0_DN0		99		100	DSI1_CP	
DSI0_DP0		101		102	DSI1_CN	
GND		103		104	GND	
DSI0_CN		105		106	DSI1_DP3	
DSI0_CP		107		108	DSI1_DN3	
GND		109		110	GND	
HDMI_CLK_N		111		112	DSI1_DP2	
HDMI_CLK_P		113		114	DSI1_DN2	
GND		115		116	GND	
HDMI_D0_N		117		118	DSI1_DP1	
HDMI_D0_P		119		120	DSI1_DN1	
GND		121		122	GND	
HDMI_D1_N		123		124	NC	
HDMI_D1_P		125		126	NC	
GND		127		128	NC	
HDMI_D2_N		129		130	NC	
HDMI_D2_P		131		132	NC	
GND		133		134	GND	
CAM1_DP3		135		136	CAM0_DP0	
CAM1_DN3		137		138	CAM0_DN0	
GND		139		140	GND	
CAM1_DP2		141		142	CAM0_CP	
CAM1_DN2		143		144	CAM0_CN	
GND		145		146	GND	
CAM1_CP		147		148	CAM0_DP1	
CAM1_CN		149		150	CAM0_DN1	
GND		151		152	GND	
CAM1_DP1		153		154	NC	
CAM1_DN1		155		156	NC	
GND		157		158	NC	
CAM1_DP0		159		160	NC	
CAM1_DN0		161		162	NC	
GND		163		164	GND	
USB_DP		165		166	TVDAC	
USB_DM		167		168	USB_OTGID	
GND		169		170	GND	
HDMI_CEC		171		172	VC_TRST_N	
HDMI_SDA		173		174	VC_TDI	
HDMI_SCL		175		176	VC_TMS	
RUN		177		178	VC_TDO	
DD_CORE (DO NOT CONNEC		179		180	VC_TCK	
GND		181		182	GND	
1V8		183		184	1V8	
1V8		185		186	1V8	
GND		187		188	GND	
VDAC		189		190	VDAC	
3V3		191		192	3V3	
3V3		193		194	3V3	
GND		195		196	GND	
VBAT		197		198	VBAT	
VBAT		199		200	VBAT	

Table 2: Compute Module 3+ SODIMM Connector Pinout

Table 2 gives the Compute Module 3+ pinout and Table 3 gives the pin functions.

Pin Name	DIR	Voltage Ref	PDN[a] State	If Unused	Description/Notes
RUN and Boot Control (see text for usage guide)					
RUN	I	3V3[b]	Pull High	Leave open	Has internal 10k pull up
EMMC_DISABLE_N	I	3V3[b]	Pull High	Leave open	Has internal 10k pull up
EMMC_EN_N_1V8	O	1V8	Pull High	Leave open	Has internal 2k2 pull up
GPIO					
GPIO[27:0]	I/O	GPIO0-27_VDD	Pull or Hi-Z[c]	Leave open	GPIO Bank 0
GPIO[45:28]	I/O	GPIO28-45_VDD	Pull or Hi-Z[c]	Leave open	GPIO Bank 1
Primary SD Interface[d,e]					
SDX_CLK	O	SDX_VDD	Pull High	Leave open	Primary SD interface CLK
SDX_CMD	I/O	SDX_VDD	Pull High	Leave open	Primary SD interface CMD
SDX_Dx	I/O	SDX_VDD	Pull High	Leave open	Primary SD interface DATA
USB Interface					
USB_Dx	I/O	-	Z	Leave open	Serial interface
USB_OTGID	I	3V3		Tie to GND	OTG pin detect
HDMI Interface					
HDMI_SCL	I/O	3V3[b]	Z[f]	Leave open	DDC Clock (5.5V tolerant)
HDMI_SDA	I/O	3V3[b]	Z[f]	Leave open	DDC Data (5.5V tolerant)
HDMI_CEC	I/O	3V3	Z	Leave open	CEC (has internal 27k pull up)
HDMI_CLKx	O	-	Z	Leave open	HDMI serial clock
HDMI_Dx	O	-	Z	Leave open	HDMI serial data
HDMI_HPD_N_1V8	I	1V8	Pull High	Leave open	HDMI hotplug detect
CAM0 (CSI0) 2-lane Interface					
CAM0_Cx	I	-	Z	Leave open	Serial clock
CAM0_Dx	I	-	Z	Leave open	Serial data
CAM1 (CSI1) 4-lane Interface					
CAM1_Cx	I	-	Z	Leave open	Serial clock
CAM1_Dx	I	-	Z	Leave open	Serial data
DSI0 (Display 0) 2-lane Interface					
DSI0_Cx	O	-	Z	Leave open	Serial clock
DSI0_Dx	O	-	Z	Leave open	Serial data
DSI1 (Display 1) 4-lane Interface					
DSI1_Cx	O	-	Z	Leave open	Serial clock
DSI1_Dx	O	-	Z	Leave open	Serial data
TV Out					
TVDAC	O	-	Z	Leave open	Composite video DAC output
JTAG Interface					
TMS	I	3V3	Z	Leave open	Has internal 50k pull up
TRST_N	I	3V3	Z	Leave open	Has internal 50k pull up
TCK	I	3V3	Z	Leave open	Has internal 50k pull up
TDI	I	3V3	Z	Leave open	Has internal 50k pull up
TDO	O	3V3	O	Leave open	Has internal 50k pull up

[a] The PDN column indicates power-down state (when RUN pin LOW)
[b] Must be driven by an open-collector driver
[c] GPIO have software enabled pulls which keep state over power-down
[d] Only available on Lite variants
[e] The CM will always try to boot from this interface first
[f] Requires external pull-up resistor to 5V as per HDMI spec

Table 3: Pin Functions

6 Electrical Specification

Caution! Stresses above those listed in Table 4 may cause permanent damage to the device. This is a stress rating only; functional operation of the device under these or any other conditions above those listed in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

Symbol	Parameter	Minimum	Maximum	Unit
VBAT	Core SMPS Supply	-0.5	6.0	V
3V3	3V3 Supply Voltage	-0.5	4.10	V
1V8	1V8 Supply Voltage	-0.5	2.10	V
VDAC	TV DAC Supply	-0.5	4.10	V
GPIO0-27_VDD	GPIO0-27 I/O Supply Voltage	-0.5	4.10	V
GPIO28-45_VDD	GPIO28-45 I/O Supply Voltage	-0.5	4.10	V
SDX_VDD	Primary SD/eMMC Supply Voltage	-0.5	4.10	V

Table 4: Absolute Maximum Ratings

DC Characteristics are defined in Table 5

Symbol	Parameter	Conditions	Minimum	Typical	Maximum	Unit
V_{IL}	Input low voltage[a]	VDD_IO = 1.8V	-	-	0.6	V
		VDD_IO = 2.7V	-	-	0.8	V
		VDD_IO = 3.3V	-	-	0.9	V
V_{IH}	Input high voltage[a]	VDD_IO = 1.8V	1.0	-	-	V
		VDD_IO = 2.7V	1.3	-	-	V
		VDD_IO = 3.3V	1.6	-	-	V
I_{IL}	Input leakage current	TA = +85°C	-	-	5	μA
C_{IN}	Input capacitance	-	-	5	-	pF
V_{OL}	Output low voltage[b]	VDD_IO = 1.8V, IOL = -2mA	-	-	0.2	V
		VDD_IO = 2.7V, IOL = -2mA	-	-	0.15	V
		VDD_IO = 3.3V, IOL = -2mA	-	-	0.14	V
V_{OH}	Output high voltage[b]	VDD_IO = 1.8V, IOH = 2mA	1.6	-	-	V
		VDD_IO = 2.7V, IOH = 2mA	2.5	-	-	V
		VDD_IO = 3.3V, IOH = 2mA	3.0	-	-	V
I_{OL}	Output low current[c]	VDD_IO = 1.8V, VO = 0.4V	12	-	-	mA
		VDD_IO = 2.7V, VO = 0.4V	17	-	-	mA
		VDD_IO = 3.3V, VO = 0.4V	18	-	-	mA
I_{OH}	Output high current[c]	VDD_IO = 1.8V, VO = 1.4V	10	-	-	mA
		VDD_IO = 2.7V, VO = 2.3V	16	-	-	mA
		VDD_IO = 3.3V, VO = 2.3V	17	-	-	mA
R_{PU}	Pullup resistor	-	50	-	65	kΩ
R_{PD}	Pulldown resistor	-	50	-	65	kΩ

[a] Hysteresis enabled
[b] Default drive strength (8mA)
[c] Maximum drive strength (16mA)

Table 5: DC Characteristics

AC Characteristics are defined in Table 6 and Fig. 3.

Pin Name	Symbol	Parameter	Minimum	Typical	Maximum	Unit
Digital outputs	t_{rise}	10-90% rise time[a]	-	1.6	-	ns
Digital outputs	t_{fall}	90-10% fall time[a]	-	1.7	-	ns
GPCLK	t_{JOSC}	Oscillator-derived GPCLK cycle-cycle jitter (RMS)	-	-	20	ps
GPCLK	t_{JPLL}	PLL-derived GPCLK cycle-cycle jitter (RMS)	-	-	48	ps

[a] Default drive strength, CL = 5pF, VDD_IOx = 3.3V

Table 6: Digital I/O Pin AC Characteristics

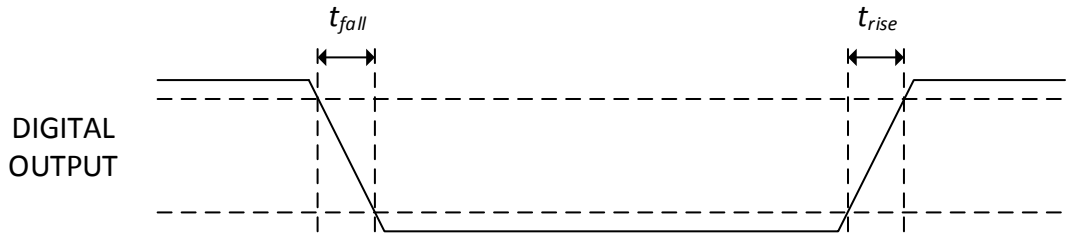


Figure 3: Digital IO Characteristics

7 Power Supplies

The Compute Module 3+ has six separate supplies that must be present and powered at all times; you cannot leave any of them unpowered, even if a specific interface or GPIO bank is unused. The six supplies are as follows:

1. VBAT is used to power the BCM2837 processor core. It feeds the SMPS that generates the chip core voltage.
2. 3V3 powers various BCM2837 PHYs, IO and the eMMC Flash.
3. 1V8 powers various BCM2837 PHYs, IO and SDRAM.
4. VDAC powers the composite (TV-out) DAC.
5. GPIO0-27_VREF powers the GPIO 0-27 IO bank.
6. GPIO28-45_VREF powers the GPIO 28-45 IO bank.

Supply	Descripion	Minimum	Typical	Maximum	Unit
VBAT	Core SMPS Supply	2.5	-	5.0 + 5%	V
3V3	3V3 Supply Voltage	3.3 - 5%	3.3	3.3 + 5%	V
1V8	1V8 Supply Voltage	1.8 - 5%	1.8	1.8 + 5%	V
VDAC	TV DAC Supply[a]	2.5 - 5%	2.8	3.3 + 5%	V
GPIO0-27_VDD	GPIO0-27 I/O Supply Voltage	1.8 - 5%	-	3.3 + 5%	V
GPIO28-45_VDD	GPIO28-45 I/O Supply Voltage	1.8 - 5%	-	3.3 + 5%	V
SDX_VDD	Primary SD/eMMC Supply Voltage	1.8 - 5%	-	3.3 + 5%	V

[a] Requires a clean 2.5-2.8V supply if TV DAC is used, else connect to 3V3

Table 7: Power Supply Operating Ranges

7.1 Supply Sequencing

Supplies should be staggered so that the highest voltage comes up first, then the remaining voltages in descending order. This is to avoid forward biasing internal (on-chip) diodes between supplies, and causing latch-up. Alternatively supplies can be synchronised to come up at exactly the same time as long as at no point a lower voltage supply rail voltage exceeds a higher voltage supply rail voltage.

7.2 Power Requirements

Exact power requirements will be heavily dependent upon the individual use case. If an on-chip subsystem is unused, it is usually in a low power state or completely turned off. For instance, if your application does not use 3D graphics then a large part of the core digital logic will never turn on and need power. This is also the case for camera and display interfaces, HDMI, USB interfaces, video encoders and decoders, and so on.

Powerchain design is critical for stable and reliable operation of the Compute Module 3+. We strongly recommend that designers spend time measuring and verifying power requirements for their particular use case and application, as well as paying careful attention to power supply sequencing and maximum supply voltage tolerance.

Table 8 specifies the recommended minimum power supply outputs required to power the Compute Module 3+.

Supply	Minimum Requirement	Unit
VBAT (CM1)	2000[a]	mW
VBAT (CM3,3L)	3500[a]	mW
3V3	250	mA
1V8	250	mA
VDAC	25	mA
GPIO0-27_VDD	50[b]	mA
GPIO28-45_VDD	50[b]	mA
SDX_VDD	50[b]	mA

[a] Recommended minimum. Actual power drawn is very dependent on use-case
[b] Each GPIO can supply up to 16mA, aggregate current per bank must not exceed 50mA

Table 8: Mimimum Power Supply Requirements

8 Booting

The eMMC Flash device on CM3+ is directly connected to the primary BCM2837 SD/eMMC interface. These connections are not accessible on the module pins. On CM3+ Lite this SD interface is available on the SDX_ pins.

When initially powered on, or after the RUN pin has been held low and then released, the BCM2837 will try to access the primary SD/eMMC interface. It will then look for a file called bootcode.bin on the primary partition (which must be FAT) to start booting the system. If it cannot access the SD/eMMC device or the boot code cannot be found, it will fall back to waiting for boot code to be written to it over USB; in other words, its USB port is in slave mode waiting to accept boot code from a suitable host.

A USB boot tool is available on Github which allows a host PC running Linux to write the BCM2837 boot code over USB to the module. That boot code then runs and provides access to the SD/eMMC as a USB mass storage device, which can then be read and written using the host PC. Note that a Raspberry Pi can be used as the host machine. For those using Windows a precompiled and packeged tool is available. For more information see here.

The Compute Module has a pin called EMMC_DISABLE_N which when shorted to GND will disable the SD/eMMC interface (by physically disconnecting the SD_CMD pin), forcing BCM2837 to boot from USB. Note that when the eMMC is disabled in this way, it takes a couple of seconds from powering up for the processor to stop attempting to talk to the SD/eMMC device and fall back to booting from USB.

Note that once booted over USB, BCM2837 needs to re-enable the SD/eMMC device (by releasing EMMC_DISABLE_N) to allow access to it as mass storage. It expects to be able to do this by driving the EMMC_EN_N_1V8 pin LOW, which at boot is initially an input with a pull up to 1V8. If an end user wishes to add the ability to access the SD/eMMC over USB in their product, similar circuitry to that used on the Compute Module IO Board to enable/disable the USB boot and SD/eMMC must be used; that is, EMMC_DISABLE_N pulled low via MOSFET(s) and released again by MOSFET, with the gate controlled by EMMC_EN_N_1V8. **Ensure you use MOSFETs suitable for switching at 1.8V (i.e. use a device with gate threshold voltage, Vt, suitable for 1.8V switching).**

9 Peripherals

9.1 GPIO

BCM2837 has in total 54 GPIO lines in 3 separate voltage banks. All GPIO pins have at least two alternative functions within the SoC. When not used for the alternate peripheral function, each GPIO pin may be set as an input (optionally as an interrupt) or an output. The alternate functions are usually peripheral I/Os, and most peripherals appear twice to allow flexibility on the choice of I/O voltage.

GPIO bank2 is used on the module to connect to the eMMC device and for an on-board I2C bus (to talk to the core SMPS and control the special function pins). On CM3+ Lite most of bank2 is exposed to allow a user to connect their choice of SD card or eMMC device (if required).

Bank0 and 1 GPIOs are available for general use. GPIO0 to GPIO27 are bank0 and GPIO28-45 make up bank1. GPIO0-27_VDD is the power supply for bank0 and GPIO28-45_VDD is the power supply for bank1. SDX_VDD is the supply for bank2 on CM3+ Lite. These supplies can be in the range 1.8V-3.3V (see Table 7) and are not optional; each bank must be powered, even when none of the GPIOs for that bank are used.

Note that the HDMI_HPD_N_1V8 and EMMC_EN_N_1V8 pins are 1.8V IO and are used for special functions (HDMI hot plug detect and boot control respectively). Please do not use these pins for any other purpose, as the software for the module will always expect these pins to have these special functions. If they are unused please leave them unconnected.

All GPIOs except GPIO28, 29, 44 and 45 have weak in-pad pull-ups or pull-downs enabled when the device is powered on. It is recommended to add off-chip pulls to GPIO28, 29, 44 and 45 to make sure they never float during power on and initial boot.

9.1.1 GPIO Alternate Functions

GPIO	Default Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
0	High	SDA0	SA5	PCLK	-	-	-
1	High	SCL0	SA4	DE	-	-	-
2	High	SDA1	SA3	LCD_VSYNC	-	-	-
3	High	SCL1	SA2	LCD_HSYNC	-	-	-
4	High	GPCLK0	SA1	DPI_D0	-	-	ARM_TDI
5	High	GPCLK1	SA0	DPI_D1	-	-	ARM_TDO
6	High	GPCLK2	SOE_N	DPI_D2	-	-	ARM_RTCK
7	High	SPI0_CE1_N	SWE_N	DPI_D3	-	-	-
8	High	SPI0_CE0_N	SD0	DPI_D4	-	-	-
9	Low	SPI0_MISO	SD1	DPI_D5	-	-	-
10	Low	SPI0_MOSI	SD2	DPI_D6	-	-	-
11	Low	SPI0_SCLK	SD3	DPI_D7	-	-	-
12	Low	PWM0	SD4	DPI_D8	-	-	ARM_TMS
13	Low	PWM1	SD5	DPI_D9	-	-	ARM_TCK
14	Low	TXD0	SD6	DPI_D10	-	-	TXD1
15	Low	RXD0	SD7	DPI_D11	-	-	RXD1
16	Low	FL0	SD8	DPI_D12	CTS0	SPI1_CE2_N	CTS1
17	Low	FL1	SD9	DPI_D13	RTS0	SPI1_CE1_N	RTS1
18	Low	PCM_CLK	SD10	DPI_D14	-	SPI1_CE0_N	PWM0
19	Low	PCM_FS	SD11	DPI_D15	-	SPI1_MISO	PWM1
20	Low	PCM_DIN	SD12	DPI_D16	-	SPI1_MOSI	GPCLK0
21	Low	PCM_DOUT	SD13	DPI_D17	-	SPI1_SCLK	GPCLK1
22	Low	SD0_CLK	SD14	DPI_D18	SD1_CLK	ARM_TRST	-
23	Low	SD0_CMD	SD15	DPI_D19	SD1_CMD	ARM_RTCK	-
24	Low	SD0_DAT0	SD16	DPI_D20	SD1_DAT0	ARM_TDO	-
25	Low	SD0_DAT1	SD17	DPI_D21	SD1_DAT1	ARM_TCK	-
26	Low	SD0_DAT2	TE0	DPI_D22	SD1_DAT2	ARM_TDI	-
27	Low	SD0_DAT3	TE1	DPI_D23	SD1_DAT3	ARM_TMS	-

Table 9: GPIO Bank0 Alternate Functions

GPIO	Default Pull	ALT0	ALT1	ALT2	ALT3	ALT4	ALT5
28	None	SDA0	SA5	PCM_CLK	FL0	-	-
29	None	SCL0	SA4	PCM_FS	FL1	-	-
30	Low	TE0	SA3	PCM_DIN	CTS0	-	CTS1
31	Low	FL0	SA2	PCM_DOUT	RTS0	-	RTS1
32	Low	GPCLK0	SA1	RING_OCLK	TXD0	-	TXD1
33	Low	FL1	SA0	TE1	RXD0	-	RXD1
34	High	GPCLK0	SOE_N	TE2	SD1_CLK	-	-
35	High	SPI0_CE1_N	SWE_N	-	SD1_CMD	-	-
36	High	SPI0_CE0_N	SD0	TXD0	SD1_DAT0	-	-
37	Low	SPI0_MISO	SD1	RXD0	SD1_DAT1	-	-
38	Low	SPI0_MOSI	SD2	RTS0	SD1_DAT2	-	-
39	Low	SPI0_SCLK	SD3	CTS0	SD1_DAT3	-	-
40	Low	PWM0	SD4	-	SD1_DAT4	SPI2_MISO	TXD1
41	Low	PWM1	SD5	TE0	SD1_DAT5	SPI2_MOSI	RXD1
42	Low	GPCLK1	SD6	TE1	SD1_DAT6	SPI2_SCLK	RTS1
43	Low	GPCLK2	SD7	TE2	SD1_DAT7	SPI2_CE0_N	CTS1
44	None	GPCLK1	SDA0	SDA1	TE0	SPI2_CE1_N	-
45	None	PWM1	SCL0	SCL1	TE1	SPI2_CE2_N	-

Table 10: GPIO Bank1 Alternate Functions

Table 9 and Table 10 detail the default pin pull state and available alternate GPIO functions. Most of these alternate peripheral functions are described in detail in the Broadcom Peripherals Specification document and have Linux drivers available.

9.1.2 Secondary Memory Interface (SMI)

The SMI peripheral is an asynchronous NAND type bus supporting Intel mode80 type transfers at 8 or 16 bit widths and available in the ALT1 positions on GPIO banks 0 and 1 (see Table 9 and Table 10). It is not publicly documented in the Broadcom Peripherals Specification but a Linux driver is available in the Raspberry Pi Github Linux repository (bcm2835_smi.c in linux/drivers/misc).

9.1.3 Display Parallel Interface (DPI)

A standard parallel RGB (DPI) interface is available on bank 0 GPIOs. This up-to-24-bit parallel interface can support a secondary display. Again this interface is not documented in the Broadcom Peripherals Specification but documentation can be found here.

9.1.4 SD/SDIO Interface

The BCM283x supports two SD card interfaces, SD0 and SD1.

The first (SD0) is a proprietary Broadcom controller that does not support SDIO and is the primary interface used to boot and talk to the eMMC or SDX_x signals.

The second interface (SD1) is standards compliant and can interface to SD, SDIO and eMMC devices; for example on a Raspberry Pi 3 B+ it is used to talk to the on-board CYW43455 WiFi device in SDIO mode.

Both interfaces can support speeds up to 50MHz single ended (SD High Speed Mode).

9.2 CSI (MIPI Serial Camera)

Currently the CSI interface is not openly documented and only CSI camera sensors supported by the official Raspberry Pi firmware will work with this interface. Supported sensors are the OmniVision OV5647 and Sony IMX219.

It is recommended to attach other cameras via USB.

9.3 DSI (MIPI Serial Display)

Currently the DSI interface is not openly documented and only DSI displays supported by the official Raspberry Pi firmware will work with this interface.

Displays can also be added via the parallel DPI interface which is available as a GPIO alternate function - see Table 9 and Section 9.1.3

9.4 USB

The BCM2837 USB port is On-The-Go (OTG) capable. If using either as a fixed slave or fixed master, please tie the USB_OTGID pin to ground.

The USB port (Pins USB_DP and USB_DM) must be routed as 90 ohm differential PCB traces.

Note that the port is capable of being used as a true OTG port however there is no official documentation. Some users have had success making this work.

9.5 HDMI

BCM283x supports HDMI V1.3a.

It is recommended that users follow a similar arrangement to the Compute Module IO Board circuitry for HDMI output.

The HDMI CK_P/N (clock) and D0-D2_P/N (data) pins must each be routed as matched length 100 ohm differential PCB traces. It is also important to make sure that each differential pair is closely phase matched. Finally, keep HDMI traces well away from other noise sources and as short as possible.

Failure to observe these design rules is likely to result in EMC failure.

9.6 Composite (TV Out)

The TVDAC pin can be used to output composite video (PAL or NTSC). Please route this signal away from noise sources and use a 75 ohm PCB trace.

Note that the TV DAC is powered from the VDAC supply which must be a clean supply of 2.5-2.8V. It is recommended users generate this supply from 3V3 using a low noise LDO.

If the TVDAC output is not used VDAC can be connected to 3V3, but it must be powered even if the TV-out functionality is unused.

10 Thermals

The BCM2837 SoC employs DVFS (Dynamic Voltage and Frequency Scaling) on the core voltage. When the processor is idle (low CPU utilisation), it will reduce the core frequency and voltage to reduce current draw and heat output. When the core utilisation exceeds a certain threshold the core votlage is increased and the core frequency is boosted to the maximum working frerquency of 1.2GHz. The voltage and frequency are throttled back when the CPU load reduces back to an 'idle' level OR when the silicon temperature as mesured by the on-chip temperature sensor exceeds 80C (thermal throttling).

A designer must pay careful attention to the thermal design of products using the CM3+ so that performance is not artificially curtailed due to the processor thermal throttling, as the Quad ARM complex in the BCM2837 can generate significant heat output under load.

10.1 Temperature Range

The operating temperature range of the module is set by the lowest maximum and highest minimum of any of the components used.

The eMMC and LPDDR2 have the narrowest range, these are rated for -25 to +80 degrees Celsius. Therefore the nominal range for the CM3+ and CM3+ Lite is -25C to +80C.

However, this range is the maximum for the silicon die; therefore, users would have to take into account the heat generated when in use and make sure this does not cause the temperature to exceed 80 degrees Celsius.

11 Availability

Raspberry Pi guarantee availability of CM3+ and CM3+ Lite until at least January 2026.

12 Support

For support please see the hardware documentation section of the Raspberry Pi website and post questions to the Raspberry Pi forum.

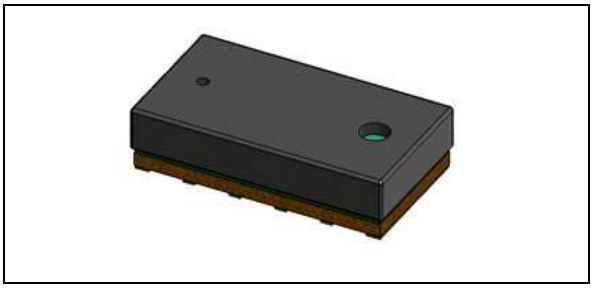
ANEXO II

DATASHEET DE VI5310X

VL53L0X

World’s smallest Time-of-Flight ranging and gesture detection sensor

Datasheet - production data



Features

- Fully integrated miniature module
  - 940 nm laser VCSEL
  - VCSEL driver
  - Ranging sensor with advanced embedded micro controller
  - 4.4 x 2.4 x 1.0 mm
- Fast, accurate distance ranging
  - Measures absolute range up to 2 m
  - Reported range is independent of the target reflectance
  - Advanced embedded optical cross-talk compensation to simplify cover glass selection
- Eye safe
  - Class 1 laser device compliant with latest standard IEC 60825-1:2014 - 3rd edition
- Easy integration
  - Single reflowable component
  - No additional optics
  - Single power supply
  - I2C interface for device control and data transfer
  - Xshutdown (reset) and interrupt GPIO
  - Programmable I2C address

Applications

- User detection for personal computers/ laptops/tablets and IoT (energy saving)
- Robotics (obstacle detection)
- White goods (hand detection in automatic faucets, soap dispensers etc.)
- 1D gesture recognition.
- Laser assisted autofocus. Enhances and speeds up camera autofocus system performance, especially in difficult scenes (low light levels, low contrast) or fast moving video mode.

Description

The VL53L0X is a new generation Time-of-Flight (ToF) laser-ranging module housed in the smallest package on the market today, providing accurate distance measurement whatever the target reflectances unlike conventional technologies. It can measure absolute distances up to 2m, setting a new benchmark in ranging performance levels, opening the door to various new applications.

The VL53L0X integrates a leading-edge SPAD array (Single Photon Avalanche Diodes) and embeds ST’s second generation FlightSense™ patented technology.

The VL53L0X’s 940 nm VCSEL emitter (Vertical Cavity Surface-Emitting Laser), is totally invisible to the human eye, coupled with internal physical infrared filters, it enables longer ranging distances, higher immunity to ambient light, and better robustness to cover glass optical crosstalk.

Contents

1 Overview 6

1.1 Technical specification 6

1.2 System block diagram 6

1.3 Device pinout 7

1.4 Application schematic 8

2 Functional description 9

2.1 System functional description 9

2.2 Firmware state machine description 10

2.3 Customer manufacturing calibration flow 11

2.3.1 SPAD and temperature calibration 12

2.3.2 Ranging offset calibration 12

2.3.3 Cross-talk calibration 13

2.4 Ranging operating modes 13

2.5 Ranging profiles 14

2.6 Ranging profile phases 14

2.6.1 Initialization and load calibration data phase 15

2.6.2 Ranging phase 15

2.6.3 Digital housekeeping 15

2.7 Getting the data: interrupt or polling 16

2.8 Device programming and control 16

2.9 Power sequence 16

2.9.1 Power up and boot sequence 16

2.10 Ranging sequence 17

3 Control interface 18

3.1 I^2C interface - timing characteristics 20

3.2 I^2C interface - reference registers 21

4 Electrical characteristics 22

4.1 Absolute maximum ratings 22

4.2 Recommended operating conditions 22

4.3 ESD 22
4.4 Current consumption 23
4.5 Electrical characteristics 24

5 Performance 25
5.1 Measurement conditions 25
5.2 Max ranging distance 26
5.3 Ranging accuracy 27
5.3.1 Standard deviation 27
5.3.2 Range profile examples 28
5.3.3 Ranging offset error 28

6 Outline drawing 29

7 Laser safety considerations 32

8 Packaging and labeling 33
8.1 Product marking 33
8.2 Inner box labeling 33
8.3 Packing 33
8.3.1 Tape outline drawings 34
8.4 Pb-free solder reflow process 34
8.5 Handling and storage precautions 36
8.5.1 Shock precaution 36
8.5.2 Part handling 36
8.5.3 Compression force 36
8.5.4 Moisture sensitivity level 36
8.6 Storage temperature conditions 36

9 Ordering information 37

10 Acronyms and abbreviations 37

11 ECOPACK® 38

12 Revision history 39

List of tables

Table 1. Technical specification . . . 6
Table 2. VL53L0X pin description . . . 7
Table 3. I^2C interface - timing characteristics . . . 20
Table 4. Reference registers . . . 21
Table 5. 32-bit register example . . . 21
Table 6. Absolute maximum ratings . . . 22
Table 7. Recommended operating conditions . . . 22
Table 8. ESD performances . . . 22
Table 9. Consumption at ambient temperature . . . 23
Table 10. Digital I/O electrical characteristics . . . 24
Table 11. Max ranging capabilities with 33ms timing budget . . . 26
Table 12. Ranging accuracy . . . 27
Table 13. Range profiles . . . 28
Table 14. Ranging offset . . . 28
Table 15. Recommended solder profile . . . 35
Table 16. Recommended storage conditions . . . 36
Table 17. Ordering information . . . 37
Table 18. Acronyms and abbreviations . . . 37
Table 19. Document revision history . . . 39

List of figures

Figure 1. VL53L0X block diagram . . . 6
Figure 2. VL53L0X pinout (bottom view) . . . 7
Figure 3. VL53L0X schematic . . . 8
Figure 4. VL53L0X system functional description . . . 9
Figure 5. Firmware state machine . . . 10
Figure 6. Customer manufacturing calibration flow . . . 11
Figure 7. Range offset . . . 12
Figure 8. Cross-talk compensation . . . 13
Figure 9. Typical initialization / ranging / housekeeping phases . . . 14
Figure 10. Power up and boot sequence . . . 16
Figure 11. Power up and boot sequence with XSHUT not controlled . . . 17
Figure 12. Ranging sequence . . . 17
Figure 13. Data transfer protocol . . . 18
Figure 14. VL53L0X I2C device address: 0x52 . . . 18
Figure 15. VL53L0X data format (write) . . . 19
Figure 16. VL53L0X data format (read) . . . 19
Figure 17. VL53L0X data format (sequential write) . . . 19
Figure 18. VL53L0X data format (sequential read) . . . 20
Figure 19. I^2C timing characteristics . . . 21
Figure 20. Typical ranging (default mode) . . . 25
Figure 21. Typical ranging - long range mode . . . 26
Figure 22. Outline drawing (page 1/3) . . . 29
Figure 23. Outline drawing (page 2/3) . . . 30
Figure 24. Outline drawing - with liner (page 3/3) . . . 31
Figure 25. Class 1 laser product label . . . 32
Figure 26. Example of marking . . . 33
Figure 27. Tape outline drawing . . . 34
Figure 28. Solder profile . . . 35

1 Overview

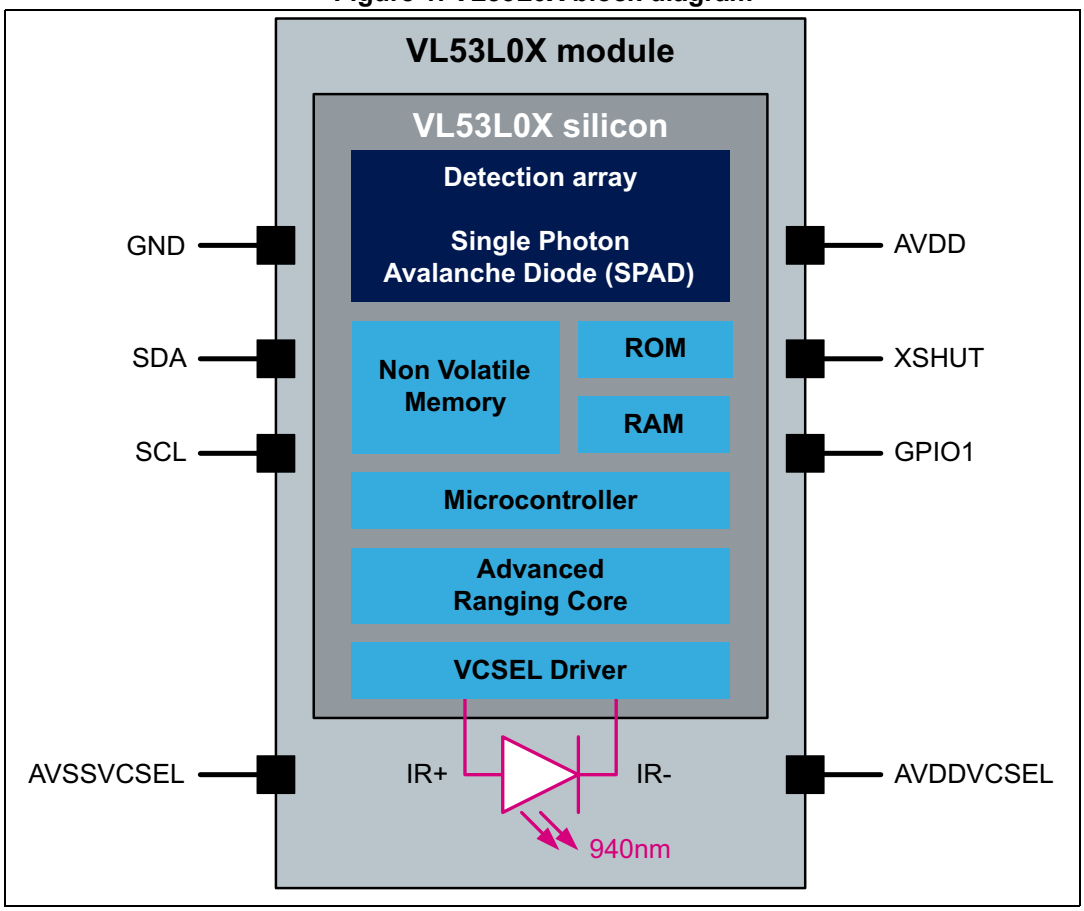
1.1 Technical specification

Table 1. Technical specification

Feature	Detail
Package	Optical LGA12
Size	4.40 x 2.40 x 1.00 mm
Operating voltage	2.6 to 3.5 V
Operating temperature:	-20 to 70°C
Infrared emitter	940 nm
I^2C	Up to 400 kHz (FAST mode) serial bus Address: 0x52

1.2 System block diagram

Figure 1. VL53L0X block diagram



1.3 Device pinout

Figure 2 shows the pinout of the VL53L0X (see also *Figure 22*).

Figure 2. VL53L0X pinout (bottom view)

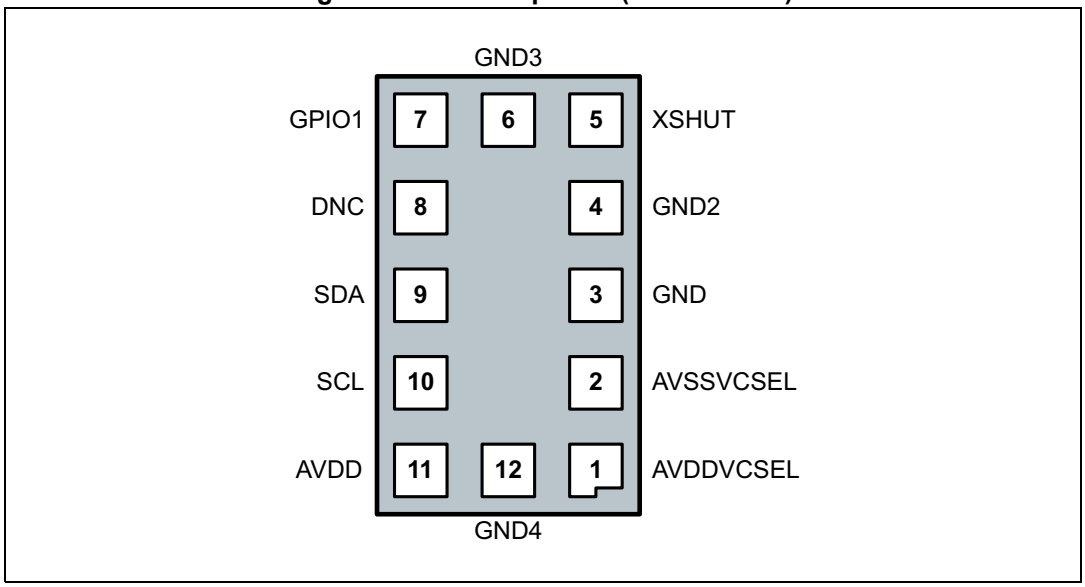


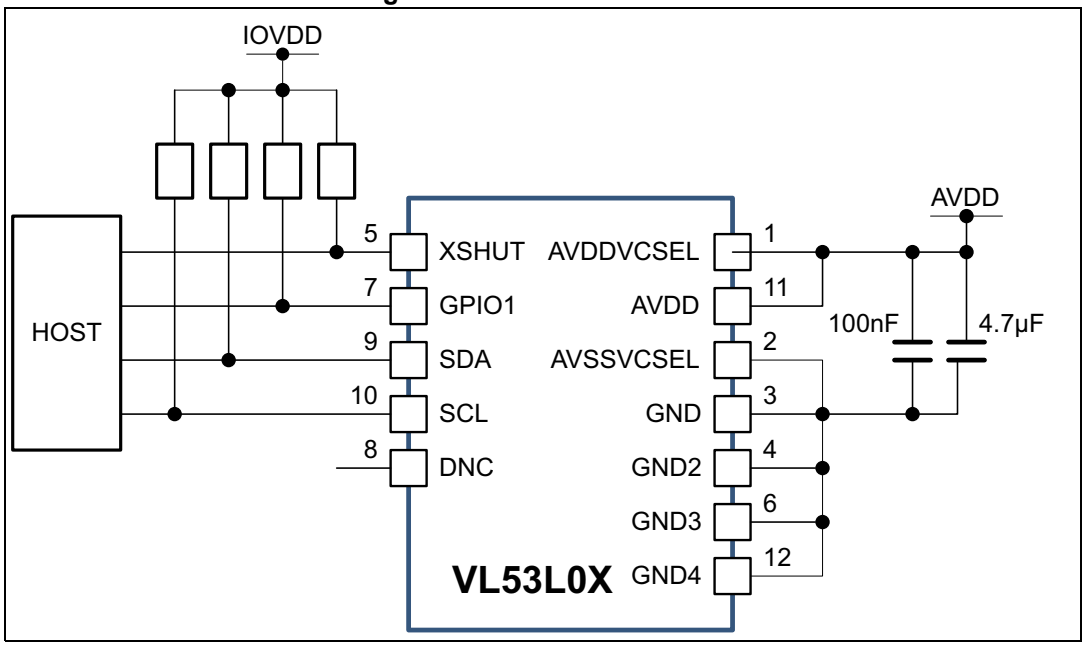
Table 2. VL53L0X pin description

Pin number	Signal name	Signal type	Signal description
1	AVDDVCSEL	Supply	VCSEL Supply, to be connected to main supply
2	AVSSVCSEL	Ground	VCSEL Ground, to be connected to main ground
3	GND	Ground	To be connected to main ground
4	GND2	Ground	To be connected to main ground
5	XSHUT	Digital input	Xshutdown pin, Active LOW
6	GND3	Ground	To be connected to main ground
7	GPIO1	Digital output	Interrupt output. Open drain output.
8	DNC	Digital input	Do Not Connect, must be left floating.
9	SDA	Digital input/output	I^2C serial data
10	SCL	Digital input	I^2C serial clock input
11	AVDD	Supply	Supply, to be connected to main supply
12	GND4	Ground	To be connected to main ground

1.4 Application schematic

Figure 3 shows the application schematic of the VL53L0X.

Figure 3. VL53L0X schematic



Note: Capacitors on external supply AVDD should be placed as close as possible to the AVDDVCSEL and AVSSVCSEL module pins.

Note: External pull-up resistors values can be found in I2C-bus specification. Pull-up are typically fitted only once per bus, near the host.
Recommended values for pull-up resistors for an AVDD of 2.8V and 400KHz I^2C clock would be 1.5k to 2k Ohms.

Note: XSHUT pin must always be driven to avoid leakage current. Pull-up is needed if the host state is not known.

XSHUT is needed to use HW standby mode (no I^2C comm).

Note: XSHUT and GPIO1 pull up recommended values are 10k Ohms

Note: GPIO1 to be left unconnected if not used

2 Functional description

2.1 System functional description

Figure 4 shows the system level functional description. The host customer application is controlling the VL53L0X device using an API (Application Programming Interface).

The API is exposing to the customer application a set of high level functions that allows control of the VL53L0X Firmware (FW) like initialization/calibration, ranging Start/Stop, choice of accuracy, choice of ranging mode.

The API is a turnkey solution, it consists of a set of C functions which enables fast development of end user applications, without the complication of direct multiple register access. The API is structured in a way that it can be compiled on any kind of platform through a well isolated platform layer.

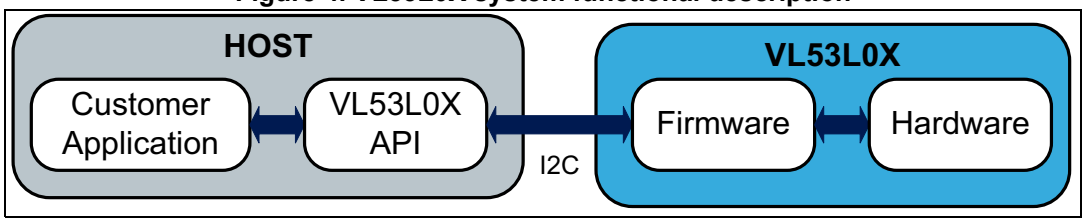
The API package allows the user to take full benefit of VL53L0X capabilities.

A detailed description of the API is available in the VL53L0X API User Manual (separate document, DocID029105).

VL53L0X FW fully manages the hardware (HW) register accesses.

Section 2.2: Firmware state machine description details the Firmware state machine.

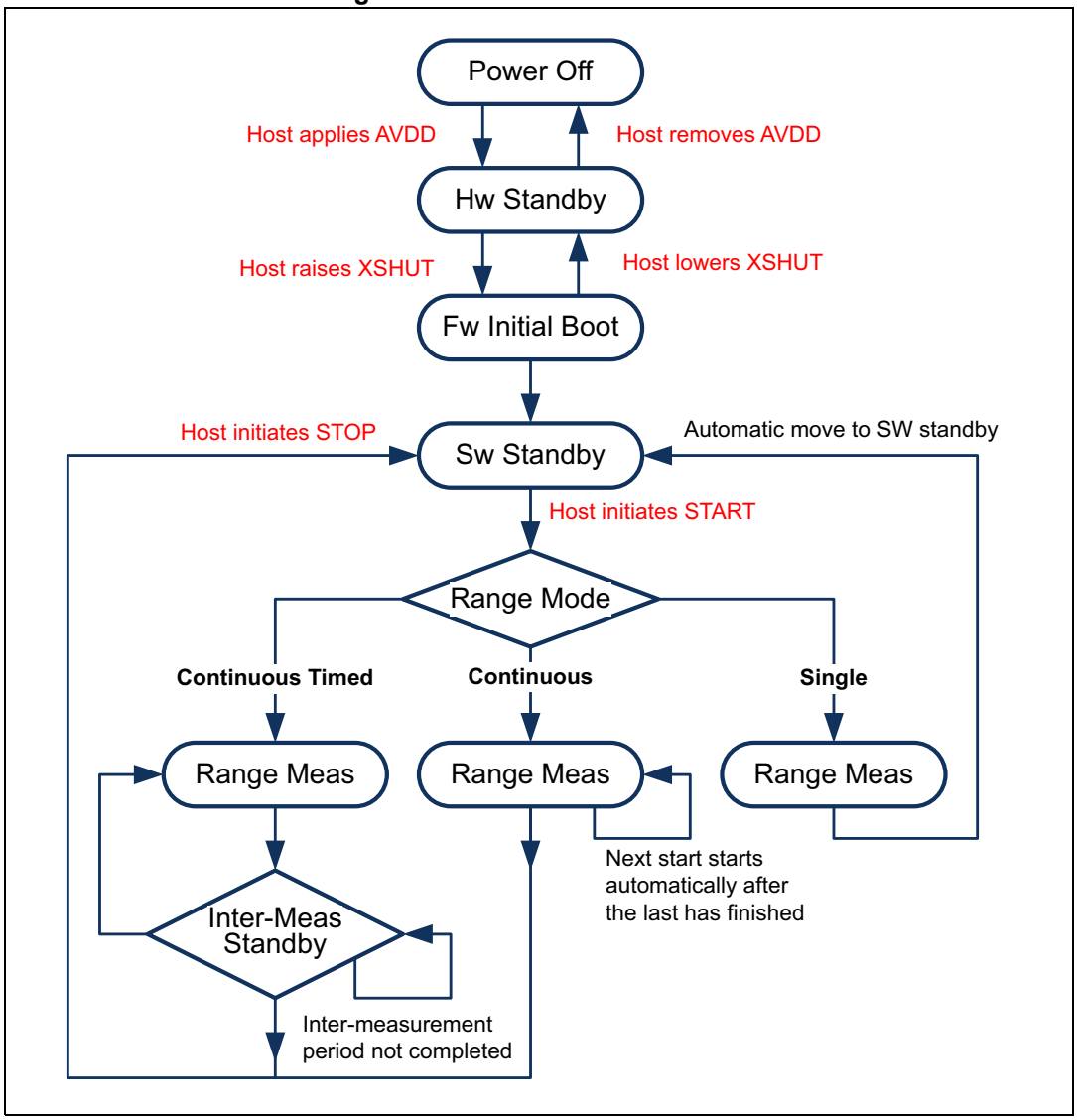
Figure 4. VL53L0X system functional description



2.2 Firmware state machine description

Figure 5 shows the Firmware state machine.

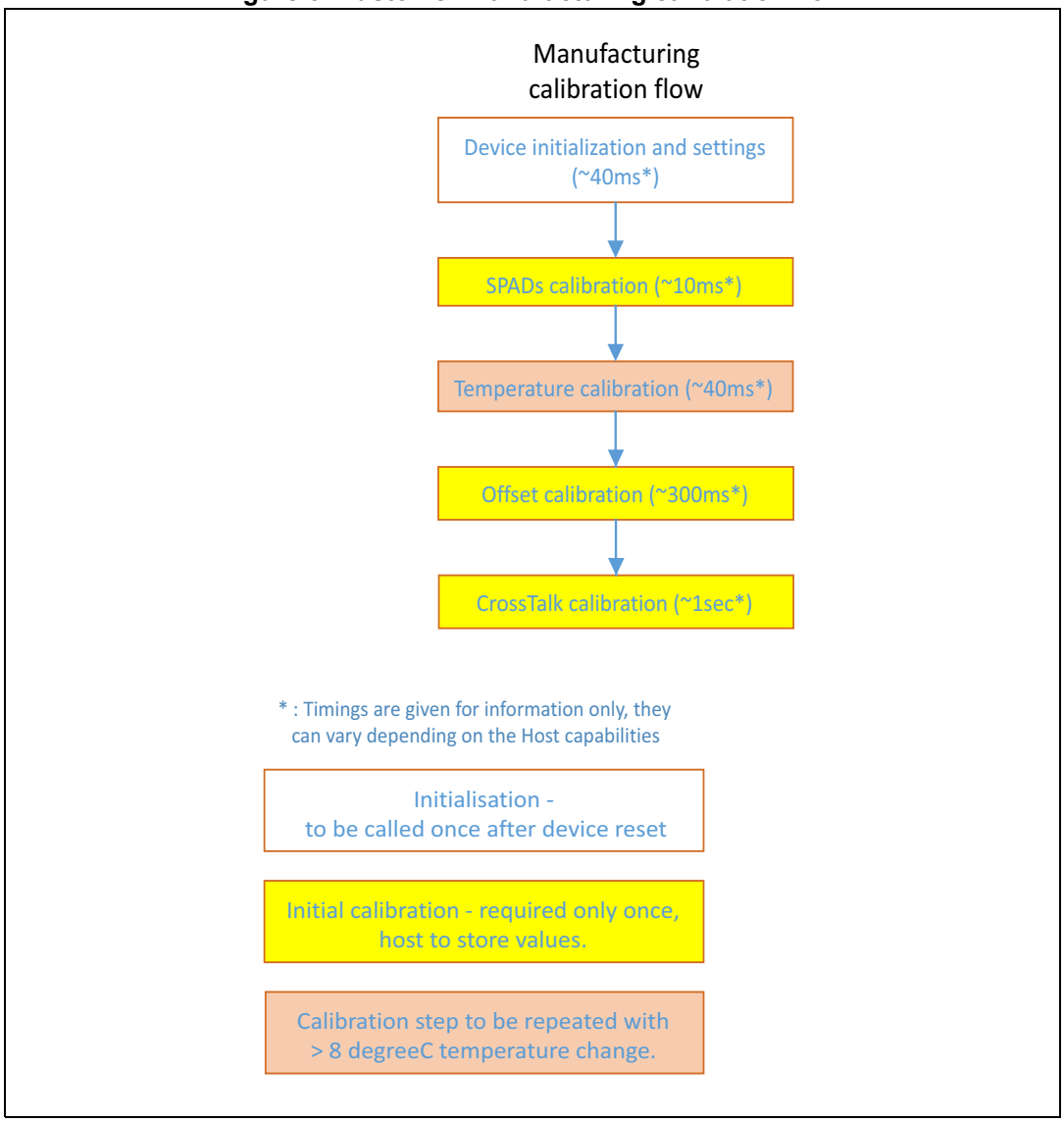
Figure 5. Firmware state machine



2.3 Customer manufacturing calibration flow

Figure 6 shows the recommended calibration flow that should be applied at customer level, at factory, once only. This flow takes into account all parameters (cover glass, temperature & voltage) from the application.

Figure 6. Customer manufacturing calibration flow



2.3.1 SPAD and temperature calibration

In order to optimize the dynamic of the system, the reference SPADs have to be calibrated. Reference SPAD calibration needs to be done only once during the initial manufacturing calibration, the calibration data should then be stored on the Host.

Temperature calibration is the calibration of two parameters (VHV and phase cal) which are temperature dependent. These two parameters are used to set the device sensitivity. Calibration should be performed during initial manufacturing calibration, it must be performed again when temperature varies more than 8degC compared to the initial

calibration temperature.

For more details on SPAD and temperature calibration please refer to the VL53L0X API User Manual.

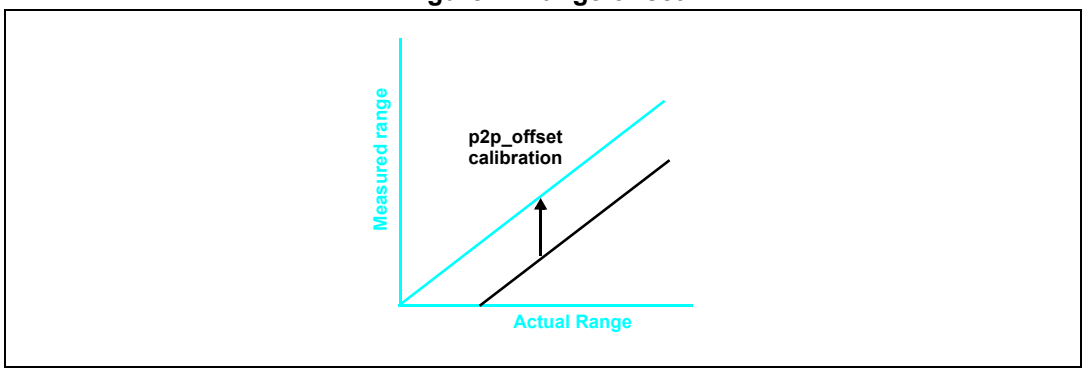
2.3.2 Ranging offset calibration

Ranging offset can be characterized by the mean offset, which is the centering of the measurement versus the real distance.

Offset calibration should be performed at factory for optimal performances (recommended at 10cm). The offset calibration should take into account:

- Supply voltage and temperature
- Protective cover glass above VL53L0X module

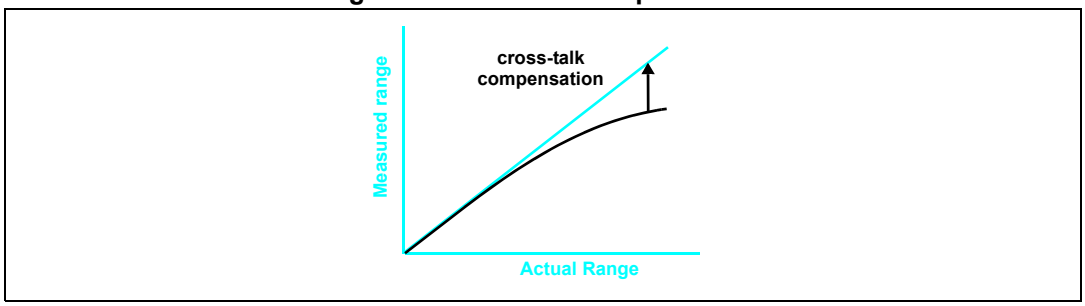
Figure 7. Range offset



2.3.3 Cross-talk calibration

Cross-talk is defined as the signal return from the cover glass. The magnitude of the cross-talk depends on the type of glass and air gap. Cross-talk results in a range error which is proportional to the ratio of the cross-talk to the signal return from the target.

Figure 8. Cross-talk compensation



Full offset and cross-talk calibration procedure is described in the VL53L0X API User Manual.

2.4 Ranging operating modes

There are 3 ranging modes available in the API:

1. Single ranging

   Ranging is performed only once after the API function is called.
   System returns to SW standby automatically.

2. Continuous ranging

   Ranging is performed in a continuous way after the API function is called. As soon as the measurement is finished, another one is started without delay.
   User has to stop the ranging to return to SW standby. The last measurement is completed before stopping.

3. Timed ranging

   Ranging is performed in a continuous way after the API function is called. When a measurement is finished, another one is started after a user defined delay.
   This delay (inter-measurement period) can be defined through the API.

User has to stop the ranging to return to SW standby.

If the stop request comes during a range measurement, the measurement is completed before stopping. If it happens during an inter-measurement period, the range measurement stops immediately.

2.5 Ranging profiles

There are 4 different ranging profiles available via API example code. Customers can create their own ranging profile dependent on their use case performance requirements.
For more details please refer to the VL53L0X API User Manual.

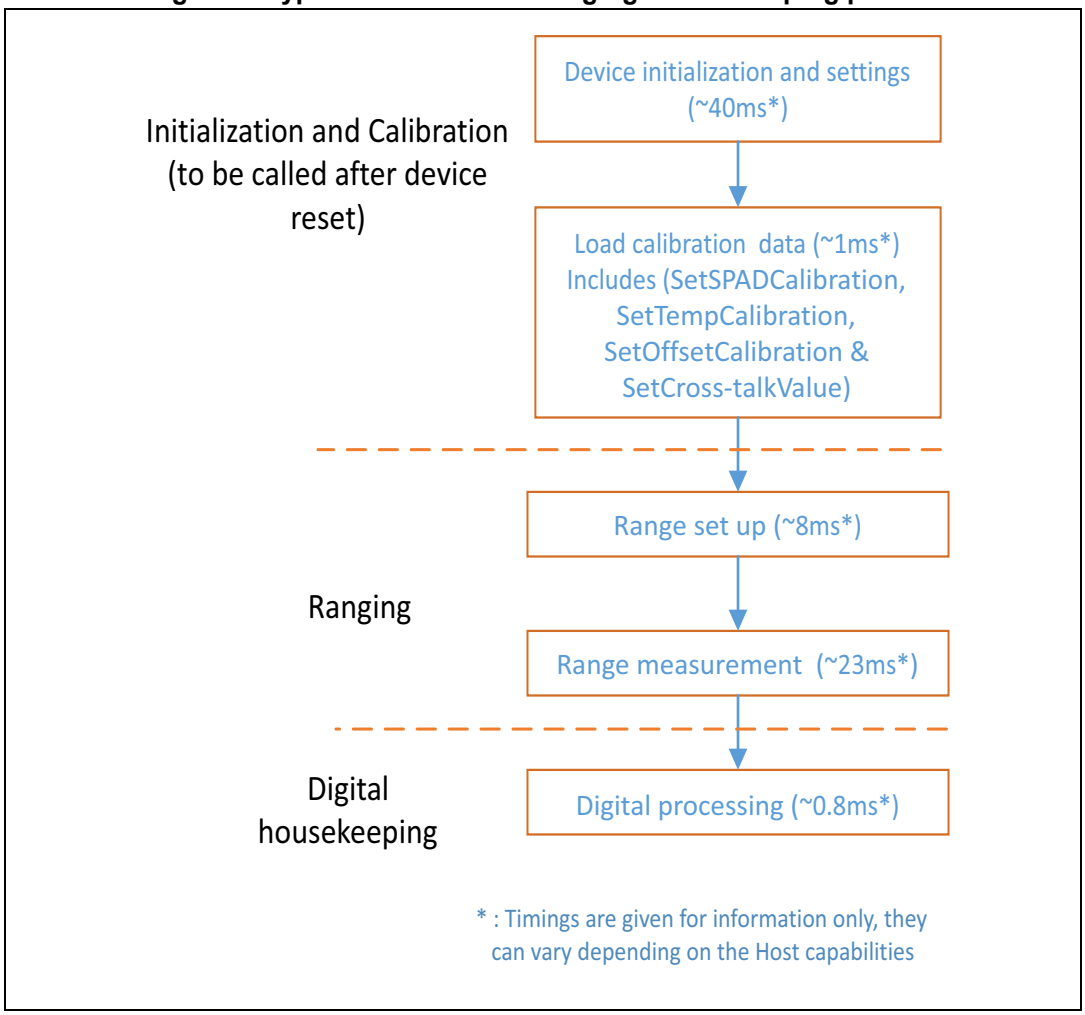
1. Default mode
2. High speed
3. High accuracy
4. Long range

2.6 Ranging profile phases

Each range profile consists of 3 consecutive phases:

- Initialization and load calibration data
- Ranging
- Digital housekeeping

Figure 9. Typical initialization / ranging / housekeeping phases



2.6.1 Initialization and load calibration data phase

Initialization and calibration phase is performed before the first ranging or after a device reset, see *Figure 9*.
The user may then have to repeat the temperature calibration phase in a periodic way, depending on the use case.

For more details on the calibration functions please refer to the VL53L0X API User Manual.

2.6.2 Ranging phase

The ranging phase consists of a range setup then range measurement.

During the ranging operation, several VCSEL infrared pulses are emitted, then reflected back by the target object, and detected by the receiving array. The photo detector used inside VL53L0X is using advanced ultra-fast SPAD technology (Single Photon Avalanche Diodes), protected by several patents.

The typical timing budget for a range is 33ms (init/ranging/housekeeping), see *Figure 12*, with the actual range measurement taking 23ms, see *Figure 9*. The minimum range measurement period is 8ms.

2.6.3 Digital housekeeping

Digital processing (housekeeping) is the last operation inside the ranging sequence that computes, validates or rejects a ranging measurement. Part of this processing is performed internally while the other part is executed on the Host by the API.

At the end of the digital processing, the ranging distance is computed by VL53L0X itself. If the distance could not be measured (weak signal, no target…), a corresponding error code is provided.

The following functions are performed on the device itself:

- Signal value check (weak signal)
- Offset correction
- Cross-talk correction (in case of cover glass)
- Final ranging value computation

While the API performs the following:

- Return Ignore Threshold RIT check (Signal check versus cross talk)
- Sigma check (accuracy condition)
- Final ranging state computation

If the user wants to enhance the ranging accuracy, some extra processing (not part of the API) can be carried out by the host, for example, rolling average, hysteresis or any kind of filtering.

2.7 Getting the data: interrupt or polling

User can get the final data using a polling or an interrupt mechanism.

Polling mode: user has to check the status of the ongoing measurement by polling an API function.

Interrupt mode: An interrupt pin (GPIO1) sends an interrupt to the host when a new measurement is available.

The description of these 2 modes is available in the VL53L0X API User Manual.

2.8 Device programming and control

Device physical control interface is I^2C, described in *Section 3: Control interface*.

A software layer (API) is provided to control the device. The API is described in the VL53L0X API User Manual.

2.9 Power sequence

2.9.1 Power up and boot sequence

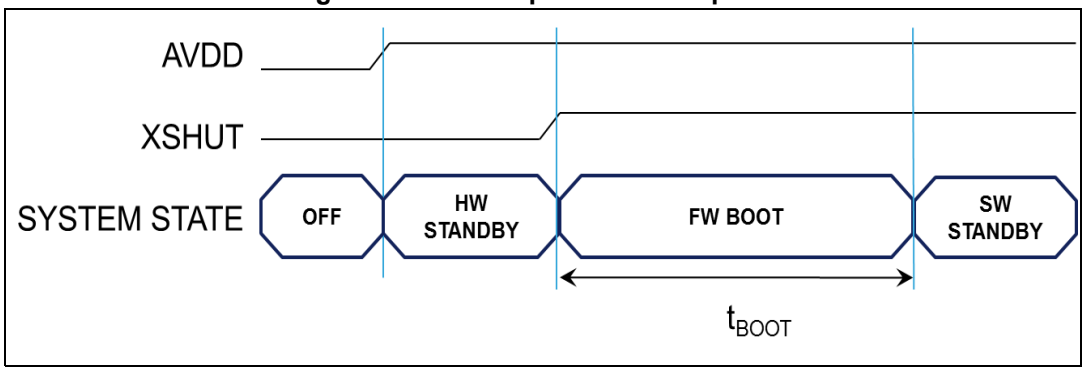
There are two options available for device power up/boot.

Option 1: XSHUT pin connected and controlled from host.

This option helps to optimize power consumption as the VL53L0X can be completely powered off when not used, and then woken up through host GPIO (using XSHUT pin).

HW Standby mode is defined as the period when AVDD is present and XSHUT is low.

Figure 10. Power up and boot sequence

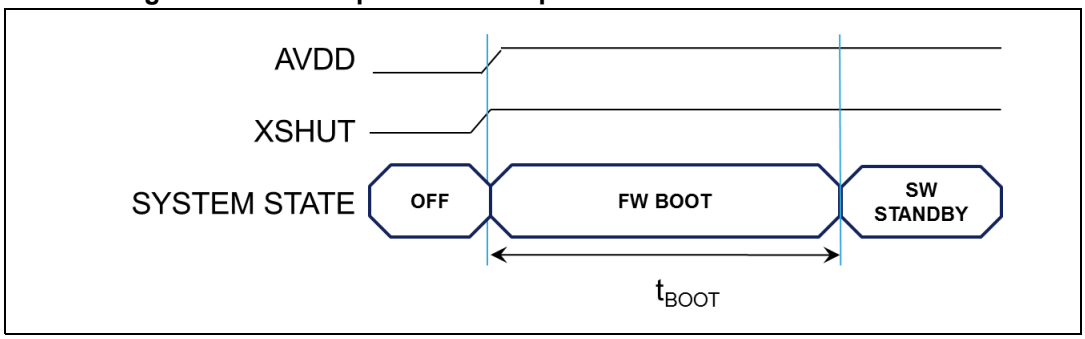


t_{BOOT} is 1.2ms max.

Option 2: XSHUT pin not controlled by host, and tied to AVDD through pull-up resistor.

In case XSHUT pin is not controlled, the power up sequence is presented in *Figure 11*. In this case, the device is going automatically in SW STANDBY after FW BOOT, without entering HW STANDBY.

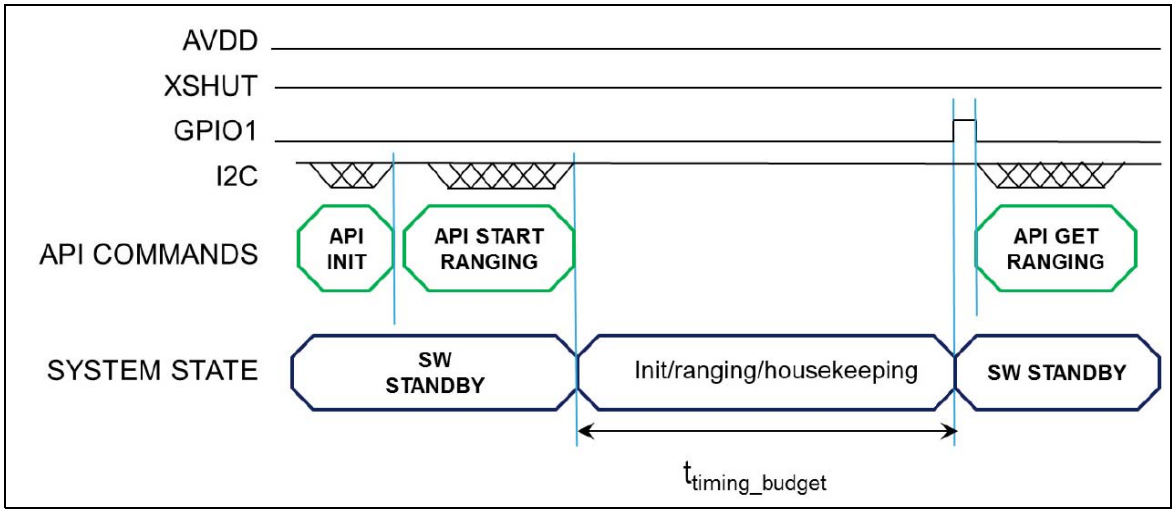
Figure 11. Power up and boot sequence with XSHUT not controlled



t_{BOOT} is 1.2ms max.

2.10 Ranging sequence

Figure 12. Ranging sequence



t_{timing_budget} is a parameter set by the user, using a dedicated API function.

Default value is 33ms.

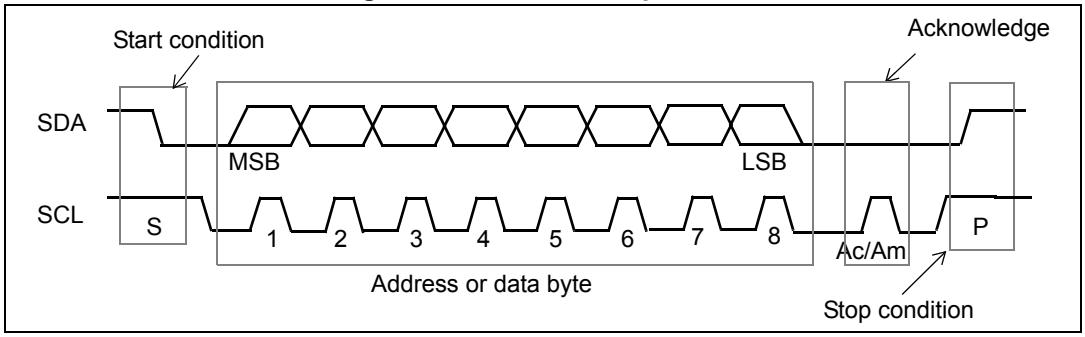
3 Control interface

This section specifies the control interface. The I^2C interface uses two signals: serial data line (SDA) and serial clock line (SCL). Each device connected to the bus is using a unique address and a simple master / slave relationships exists.

Both SDA and SCL lines are connected to a positive supply voltage using pull-up resistors located on the host. Lines are only actively driven low. A high condition occurs when lines are floating and the pull-up resistors pull lines up. When no data is transmitted both lines are high.

Clock signal (SCL) generation is performed by the master device. The master device initiates data transfer. The I^2C bus on the VL53L0X has a maximum speed of 400 kbits/s and uses a device address of 0x52.

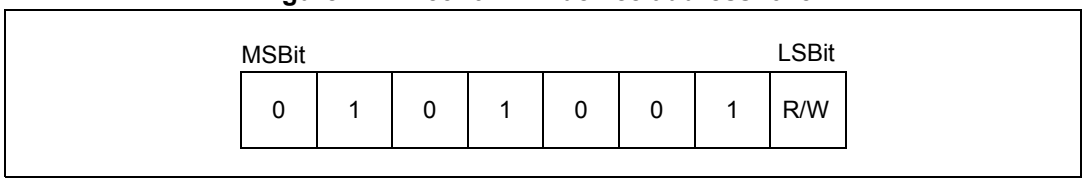
Figure 13. Data transfer protocol



Information is packed in 8-bit packets (bytes) always followed by an acknowledge bit, Ac for VL53L0X acknowledge and Am for master acknowledge (host bus master). The internal data is produced by sampling SDA at a rising edge of SCL. The external data must be stable during the high period of SCL. The exceptions to this are start (S) or stop (P) conditions when SDA falls or rises respectively, while SCL is high.

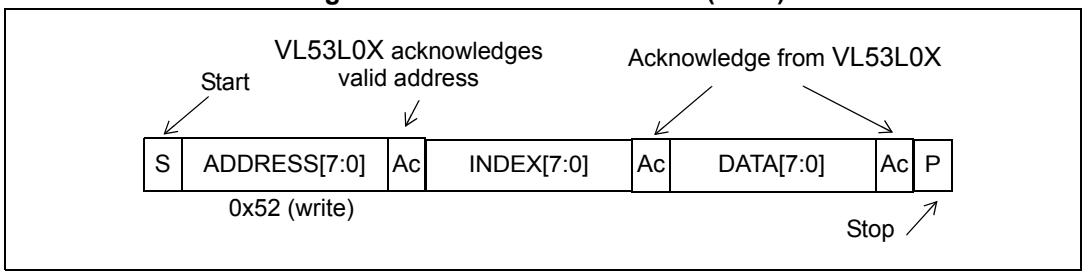
A message contains a series of bytes preceded by a start condition and followed by either a stop or repeated start (another start condition but without a preceding stop condition) followed by another message. The first byte contains the device address (0x52) and also specifies the data direction. If the least significant bit is low (that is, 0x52) the message is a master write to the slave. If the lsb is set (that is, 0x53) then the message is a master read from the slave.

Figure 14. VL53L0X I2C device address: 0x52



All serial interface communications with the camera module must begin with a start condition. The VL53L0X module acknowledges the receipt of a valid address by driving the SDA wire low. The state of the read/write bit (lsb of the address byte) is stored and the next byte of data, sampled from SDA, can be interpreted. During a write sequence the second byte received provide a 8-bit index which points to one of the internal 8-bit registers.

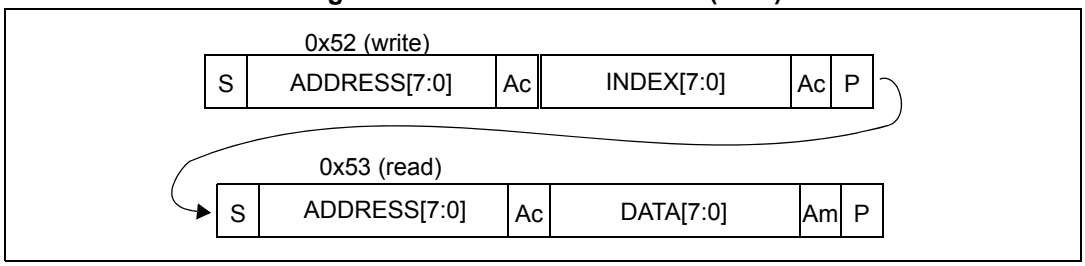
Figure 15. VL53L0X data format (write)



As data is received by the slave it is written bit by bit to a serial/parallel register. After each data byte has been received by the slave, an acknowledge is generated, the data is then stored in the internal register addressed by the current index.

During a read message, the contents of the register addressed by the current index is read out in the byte following the device address byte. The contents of this register are parallel loaded into the serial/parallel register and clocked out of the device by the falling edge of SCL.

Figure 16. VL53L0X data format (read)



At the end of each byte, in both read and write message sequences, an acknowledge is issued by the receiving device (that is, the VL53L0X for a write and the host for a read).

A message can only be terminated by the bus master, either by issuing a stop condition or by a negative acknowledge (that is, **not** pulling the SDA line low) after reading a complete byte during a read operation.

The interface also supports auto-increment indexing. After the first data byte has been transferred, the index is automatically incremented by 1. The master can therefore send data bytes continuously to the slave until the slave fails to provide an acknowledge or the master terminates the write communication with a stop condition. If the auto-increment feature is used the master does **not** have to send address indexes to accompany the data bytes.

Figure 17. VL53L0X data format (sequential write)

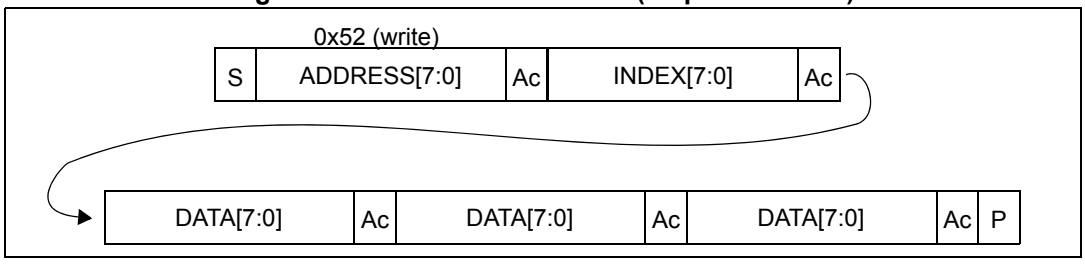
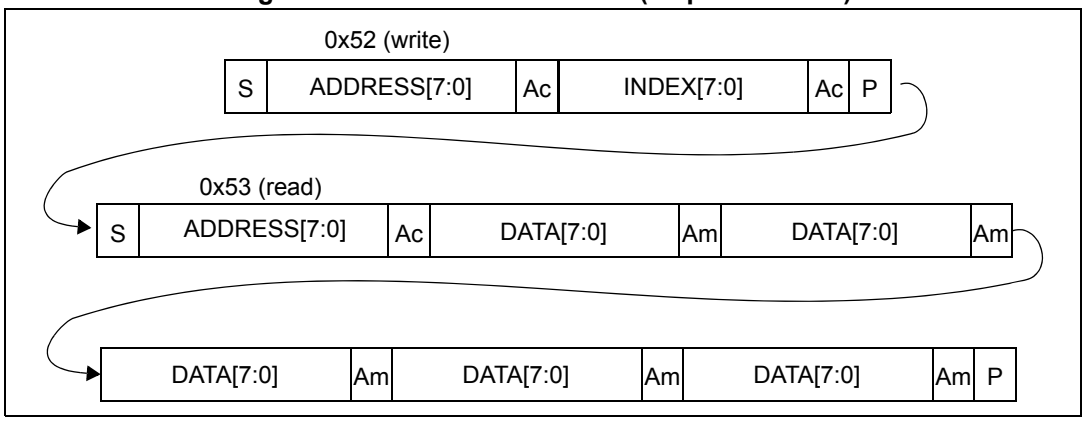


Figure 18. VL53L0X data format (sequential read)



3.1 I²C interface - timing characteristics

Timing characteristics are shown in *Table 3*. Please refer to *Figure 19* for an explanation of the parameters used.

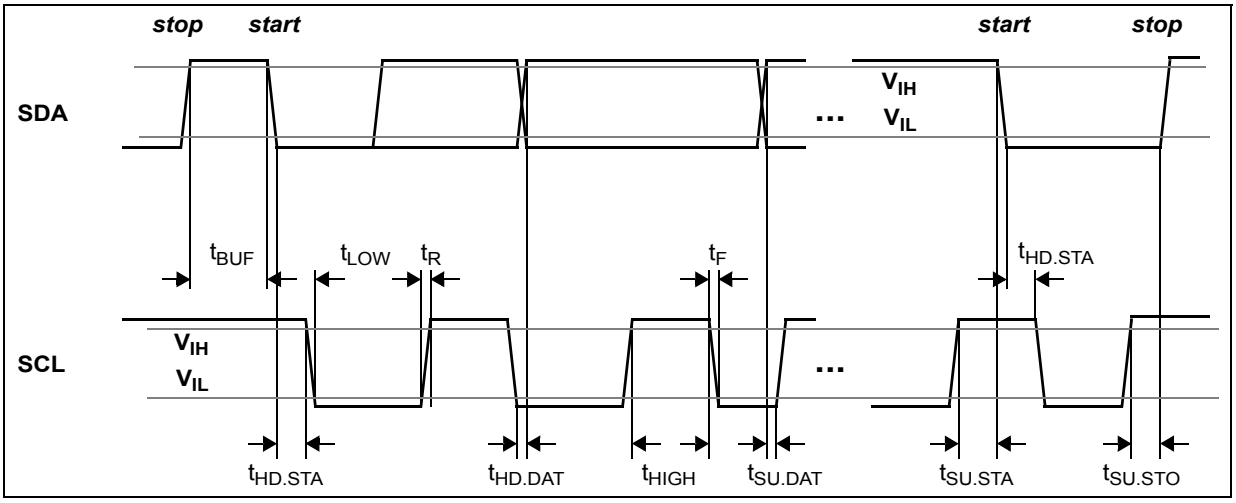
Timings are given for all PVT conditions.

Table 3. I²C interface - timing characteristics

Symbol	Parameter	Minimum	Typical	Maximum	Unit
F_{I2C}	Operating frequency (Standard and Fast mode)	0	-	400(1)	kHz
t_{LOW}	Clock pulse width low	1.3	-	-	µs
t_{HIGH}	Clock pulse width high	0.6	-	-	µs
t_{SP}	Pulse width of spikes which are suppressed by the input filter	-	-	50	ns
t_{BUF}	Bus free time between transmissions	1.3	-	-	ms
$t_{HD.STA}$	Start hold time	0.26	-	-	µs
$t_{SU.STA}$	Start set-up time	0.26	-	-	µs
$t_{HD.DAT}$	Data in hold time	0	-	0.9	µs
$t_{SU.DAT}$	Data in set-up time	50	-	-	ns
t_R	SCL/SDA rise time	-	-	120	ns
t_F	SCL/SDA fall time	-	-	120	ns
$t_{SU.STO}$	Stop set-up time	0.6	-	-	µs
Ci/o	Input/output capacitance (SDA)	-	-	10	pF
Cin	Input capacitance (SCL)	-	-	4	pF
C_L	Load capacitance	-	125	400	pF

1. The maximum bus speed is also limited by the combination of 400pF load capacitance and pull-up resistor. Please refer to the I²C specification for further information.

Figure 19. I²C timing characteristics



All timings are measured from either V_{IL} or V_{IH}.

3.2 I²C interface - reference registers

The registers shown in the table below can be used to validate the user I²C interface.

Table 4. Reference registers

Address	(After fresh reset, without API loaded)
0xC0	0xEE
0xC1	0xAA
0xC2	0x10
0x51	0x0099
0x61	0x0000

Note: I2C read/writes can be 8,16 or 32-bit. Multi-byte reads/writes are always addressed in ascending order with MSB first as shown in Table 5.

Table 5. 32-bit register example

Register address	Byte
Address	MSB
Address + 1	..
Address + 2	..
Address + 3	LSB

4 Electrical characteristics

4.1 Absolute maximum ratings

Table 6. Absolute maximum ratings

Parameter	Min.	Typ.	Max.	Unit
AVDD	-0.5	-	3.6	V
SCL, SDA, XSHUT and GPIO1	-0.5	-	3.6	V

Note: Stresses above those listed in Table 6. may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of the specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.

4.2 Recommended operating conditions

Table 7. Recommended operating conditions(1)

Parameter		Min.	Typ.	Max.	Unit
Voltage (AVDD)		2.6	2.8	3.5	V
IO (IOVDD)(2)	Standard mode	1.6	1.8	1.9	V
	2V8 mode(3)(4)	2.6	2.8	3.5	V
Temperature (normal operating)		-20		+70	°C

1. There are no power supply sequencing requirements. The I/Os may be high, low or floating when AVDD is applied. The I/Os are internally failsafe with no diode connecting them to AVDD
2. XSHUT should be high level only when AVDD is on.
3. SDA, SCL, XSHUT and GPIO1 high levels have to be equal to AVDD in 2V8 mode.
4. The default API mode is 1V8.
   2V8 mode is programmable using device settings loaded by the API. For more details please refer to the VL53L0X API User Manual.

4.3 ESD

VL53L0X is compliant with ESD values presented in *Table 8*

Table 8. ESD performances

Parameter	Specification	Conditions
Human Body Model	JS-001-2012	+/- 2kV, 1500 Ohms, 100pF
Charged Device Model	JZSD22-C101	+/- 500V

4.4 Current consumption

Table 9. Consumption at ambient temperature(1)

Parameter	Min.	Typ.	Max.	Unit
HW STANDBY	3	5	7	uA
SW STANDBY (2V8 mode)(2)	4	6	9	uA
Timed ranging Inter measurement		16		uA
Active Ranging average consumption (including VCSEL) (3)(4)		19		mA
Average power consumption at 10Hz with 33ms ranging sequence			20	mW

1. All current consumption values include silicon process variations. Temperature and Voltage are at nominal conditions (23degC and 2.8V).

   All values include AVDD and AVDDVCSEL.

2. In standard mode (1V8), pull-ups have to be modified, then SW STANDBY consumption is increased by +0.6uA.
3. Active ranging is an average value, measured using default API settings (33ms timing budget).
4. Peak current (including VCSEL) can reach 40mA.

4.5 Electrical characteristics

Table 10. Digital I/O electrical characteristics

Symbol	Parameter	Minimum	Typical	Maximum	Unit
Interrupt pin (GPIO1)					
V_{IL}	Low level input voltage	-	-	0.3 IOVDD	V
V_{IH}	High level input voltage	0.7 IOVDD	-	-	V
V_{OL}	Low level output voltage (I_{OUT} = 4 mA)	-	-	0.4	V
V_{OH}	High level output voltage at (I_{OUT} = 4 mA)	IOVDD-0.4	-	-	V
F_{GPIO}	Operating frequency (C_{LOAD} = 20 pF)	0	-	108	MHz
I^2C interface (SDA/SCL)					
V_{IL}	Low level input voltage	-0.5	-	0.6	V
V_{IH}	High level input voltage	1.12	-	IOVDD+0.5	V
V_{OL}	Low level output voltage (I_{OUT} = 4 mA in Standard and Fast modes)	-	-	0.4	V
I_{IL}/I_{IH}	Leakage current(1)	-	-	10	µA
	Leakage current(2)	-	-	0.15	µA

1. AVDD = 0 V
2. AVDD = 2.85 V; I/O voltage = 1.8 V

5 Performance

5.1 Measurement conditions

In all measurement tables in the document, it is considered that the full Field Of View (FOV) is covered.

VL53L0X system FOV is 25degrees.

Reflectance targets are standard ones (Grey 17% N4.74 and White 88% N9.5 Munsell charts).

Unless mentioned, device is controlled through the API using the default settings (refer to VL53L0X API User Manual for API settings description).

Figure 20. Typical ranging (default mode)

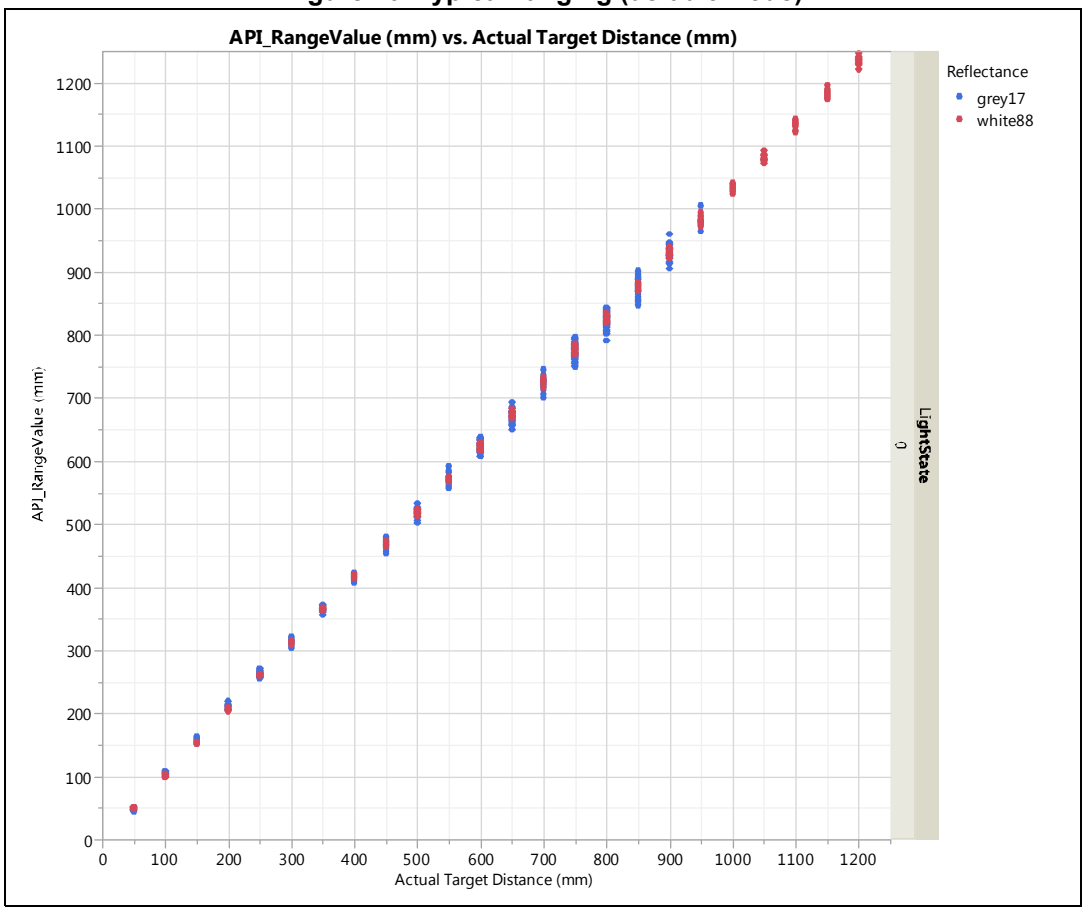
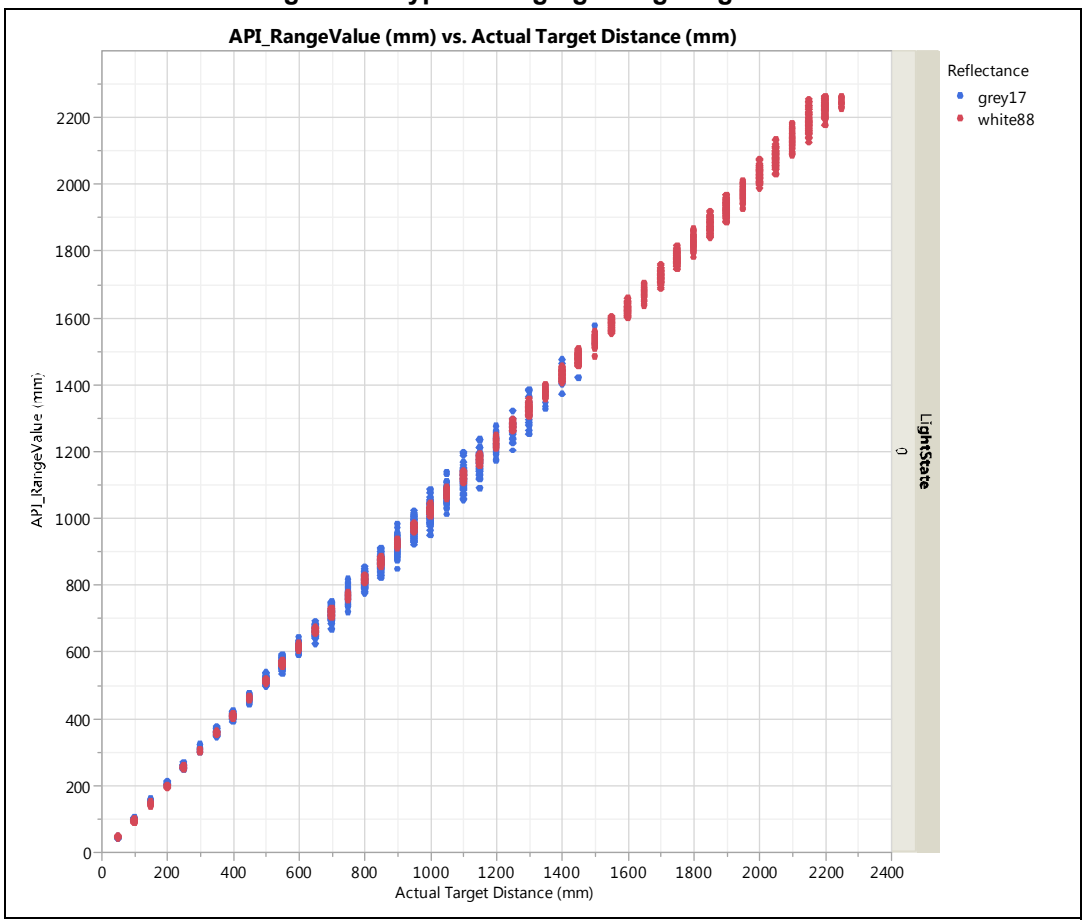


Figure 21. Typical ranging - long range mode



5.2 Max ranging distance

Table 11 presents the ranging specification for VL53L0X bare module, without cover glass, at room temperature (23degreesC) and with nominal voltage (2.8Volts).

Table 11. Max ranging capabilities with 33ms timing budget

Target reflectance level (full FOV)	Conditions	Indoor (2)	Outdoor overcast (2)
White target (88%)	Typical	200cm+ (1)	80cm
	Minimum	120cm	60cm
Grey target (17%)	Typical	80cm	50cm
	Minimum	70cm	40cm

Note (1): using long range API profile

Note (2):

- Indoor: no infrared
- Outdoor overcast corresponds to a parasitic noise of 10kcps/SPAD for VL53L0X module. For reference, this corresponds to a 1.2W/m² at 940nm, and is equivalent to 5kLux daylight, while ranging on a grey 17% chart at 40cm

Measurement conditions:

- Targets reflectance used : Grey (17%), White (88%)
- Nominal Voltage (2.8V) and Temperature (23degreesC)
- All distances are for a complete Field of View covered (FOV = 25degrees)
- 33ms timing budget

All distances mentioned in this table are guaranteed for a minimum detection rate of 94% (up to 100%). Detection rate is the worst case percentage of measurements that will return a valid measurement when target is detected.

5.3 Ranging accuracy

5.3.1 Standard deviation

Ranging accuracy can be characterized by standard deviation. It includes Measure-to-Measure and Part-to-Part (silicon) dispersion.

Table 12. Ranging accuracy

	Indoor (no infrared)			Outdoor		
Target reflectance level (full FOV)	**Distance**	**33 ms**	**66 ms**	**Distance**	**33 ms**	**66 ms**
White Target (88%)	At 120 cm	4 %	3 %	At 60 cm	7 %	6 %
Grey Target (17%)	At 70 cm	7 %	6%	at 40 cm	12 %	9 %

Measurement conditions:

- Targets reflectance used: Grey (17 %), White (88 %)
- Offset correction done at 10 cm from sensor
- Indoor: no infrared / Outdoor: eq. 5 kLux equivalent sunlight (10 kcps/SPAD)
- Nominal voltage (2v8) and Temperature (23 degreesC)
- All distances are for a complete FOV covered (FOV = 25 degrees)
- Detection rate is considered at 94 % minimum

5.3.2 Range profile examples

Table 13 details typical performance for the four example ranging profiles, as per measurement conditions in *Section 5.3: Ranging accuracy*.

Table 13. Range profiles

Range profile	Range timing budget	Typical performance	Typical application
Default mode	30 ms	1.2 m, accuracy as per *Table 12*	Standard
High accuracy	200 ms	1.2 m, accuracy < +/- 3 %	Precise measurement
Long range	33 ms	2 m, accuracy as per *Table 12*	Long ranging, only for dark conditions (no IR)
High speed	20 ms	1.2 m, accuracy +/- 5 %	High speed where accuracy is not priority

5.3.3 Ranging offset error

The table below shows how range offset may drift over distance, voltage and temperature.

Assumes offset calibrated at 10cm. See VL53L0X API User Manual for details on offset calibration.

Table 14. Ranging offset

	Nominal conditions	Measure point	Typical offset from nominal	Maximum offset from nominal
Ranging distance	Offset calibration at 10 cm (“zero”)	White 120 cm (indoor) Grey 70 cm (indoor) White 60 cm (outdoor) Grey 40 cm (outdoor)		< 3 %
Voltage drift	2.8 V	2.6 V to 3.5 V	+/- 10 mm	+/- 15 mm
Temperature drift	23 °C	-20°C to +70°C	+/- 10 mm	+/- 30 mm

6 Outline drawing

Figure 22. Outline drawing (page 1/3)

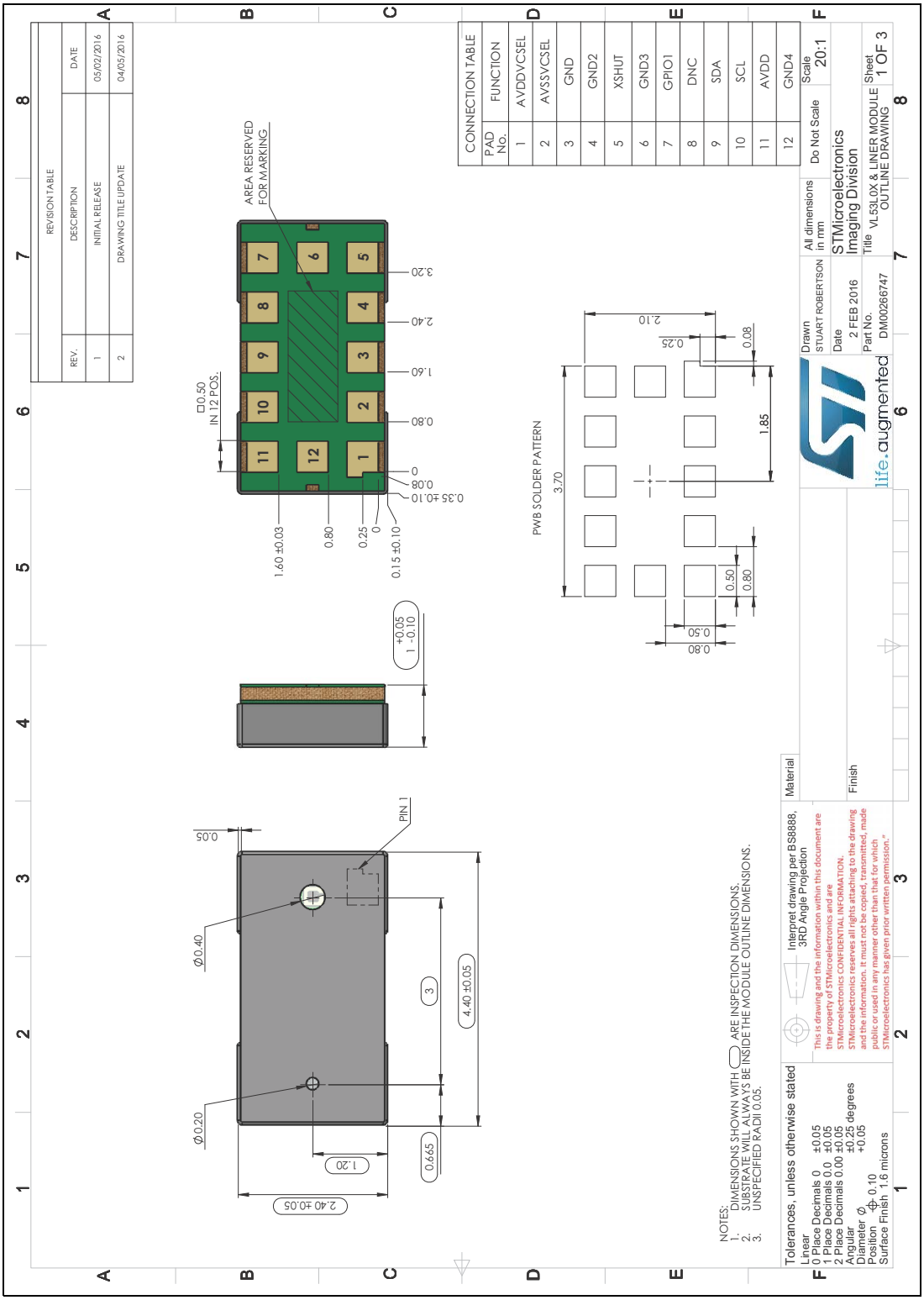


Figure 23. Outline drawing (page 2/3)

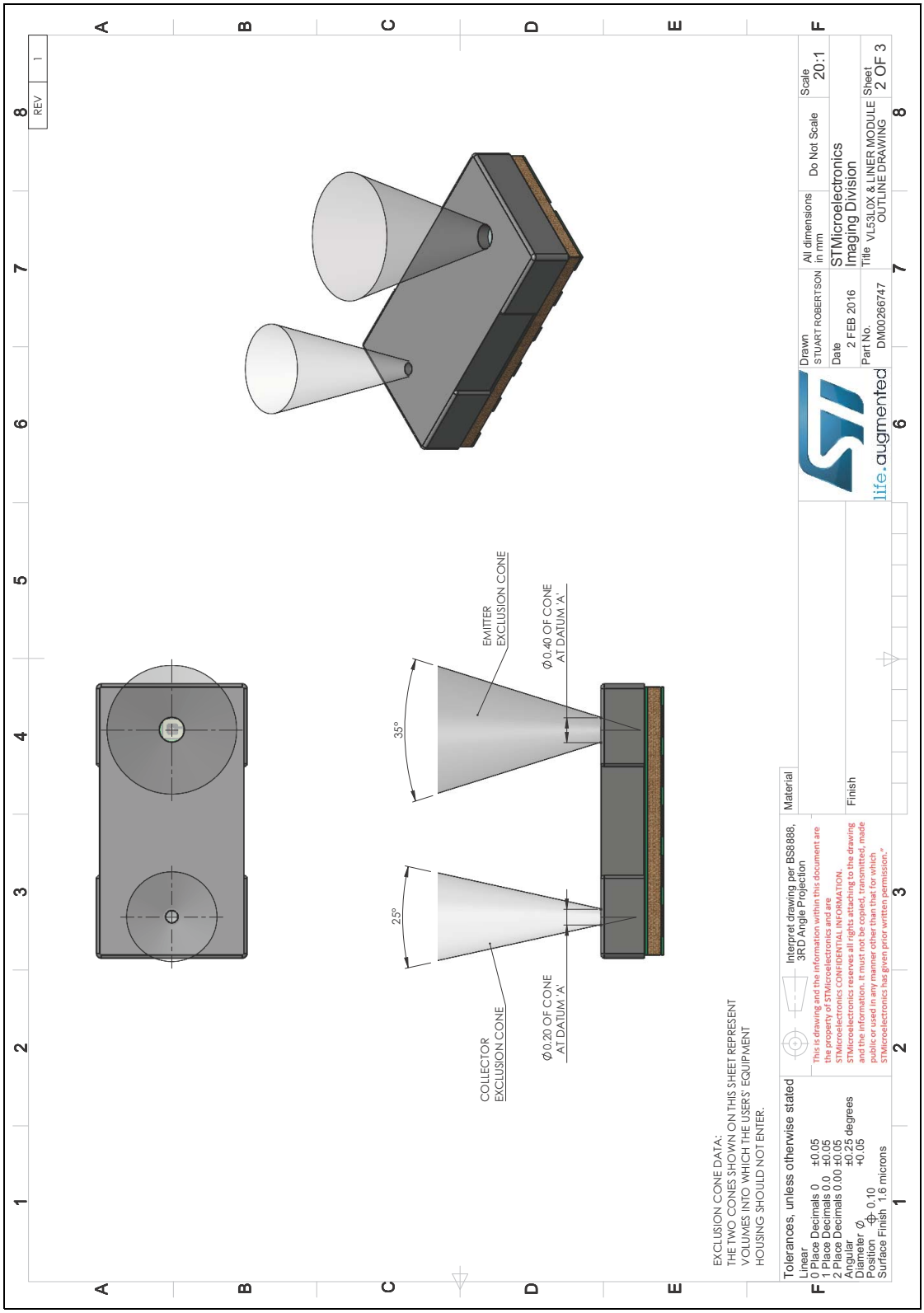
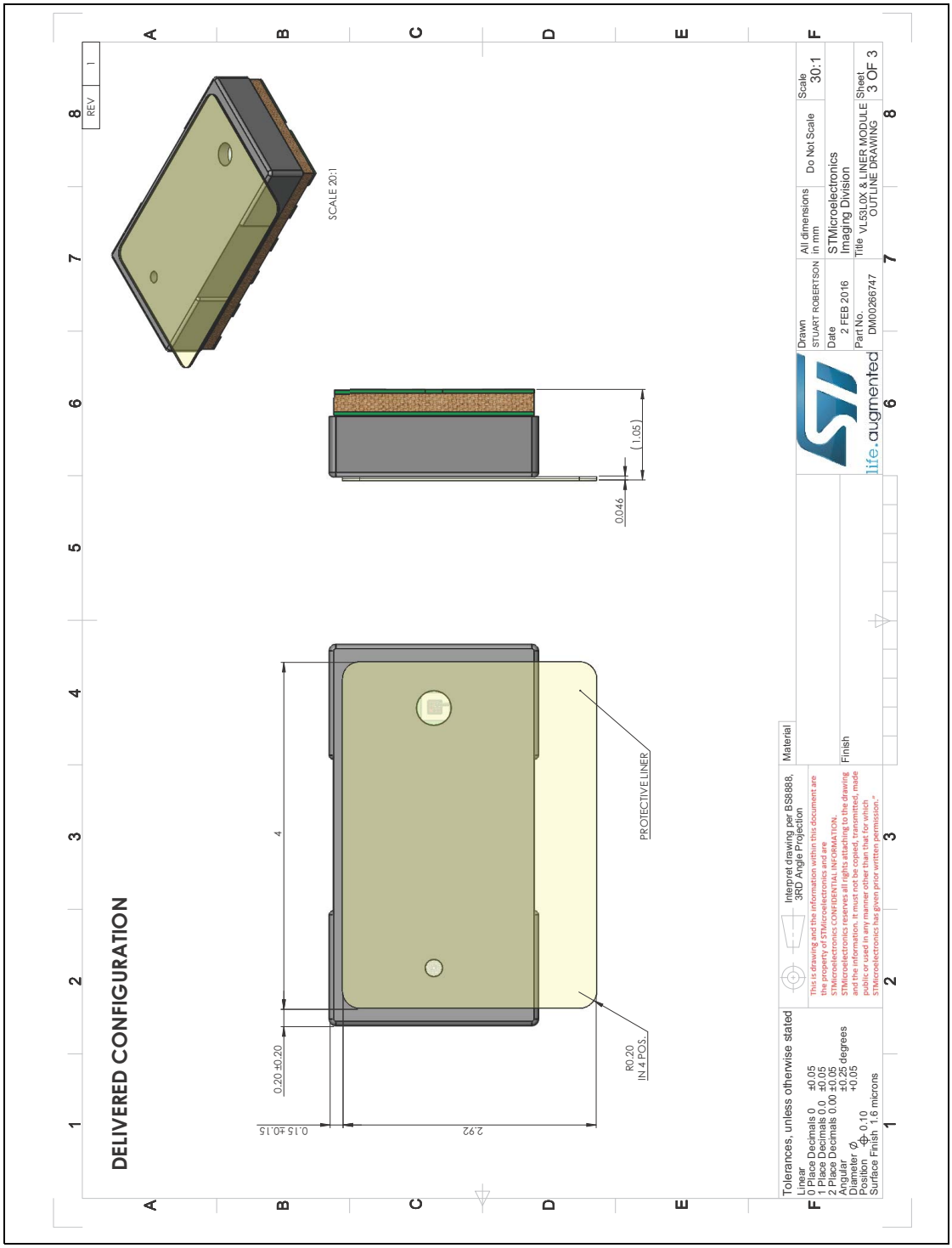


Figure 24. Outline drawing - with liner (page 3/3)

The VL53L0X module is delivered with a protective liner covering the top of the cap to protect the sensor from foreign material during the assembly process. It must be removed by the customer just before mounting the cover glass



7 Laser safety considerations

The VL53L0X contains a laser emitter and corresponding drive circuitry. The laser output is designed to remain within Class 1 laser safety limits under all reasonably foreseeable conditions including single faults in compliance with IEC 60825-1:2014 (third edition).

The laser output will remain within Class 1 limits as long as the STMicroelectronics recommended device settings (API settings) are used and the operating conditions specified are respected.

The laser output power must not be increased by any means and no optics should be used with the intention of focusing the laser beam.

Caution: Use of controls or adjustments or performance of procedures other than those specified herein may result in hazardous radiation exposure.

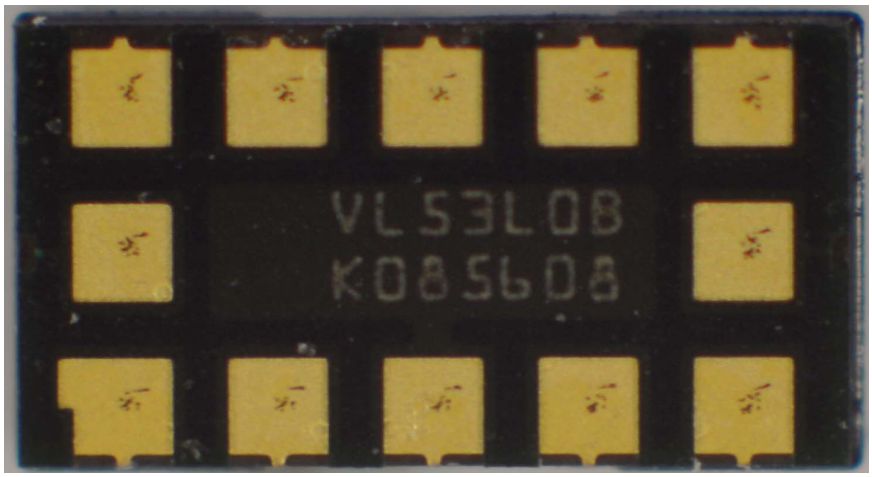
Figure 25. Class 1 laser product label

8 Packaging and labeling

8.1 Product marking

A 2-line product marking is applied on the backside of the module (i.e. on the substrate). The first line is the silicon product code, and the second line, the internal tracking code.

Figure 26. Example of marking



8.2 Inner box labeling

The labeling follows the ST standard packing acceptance specification.

The following information will be on the inner box label:

- assembly site
- sales type
- quantity
- trace code
- marking
- bulk ID number

8.3 Packing

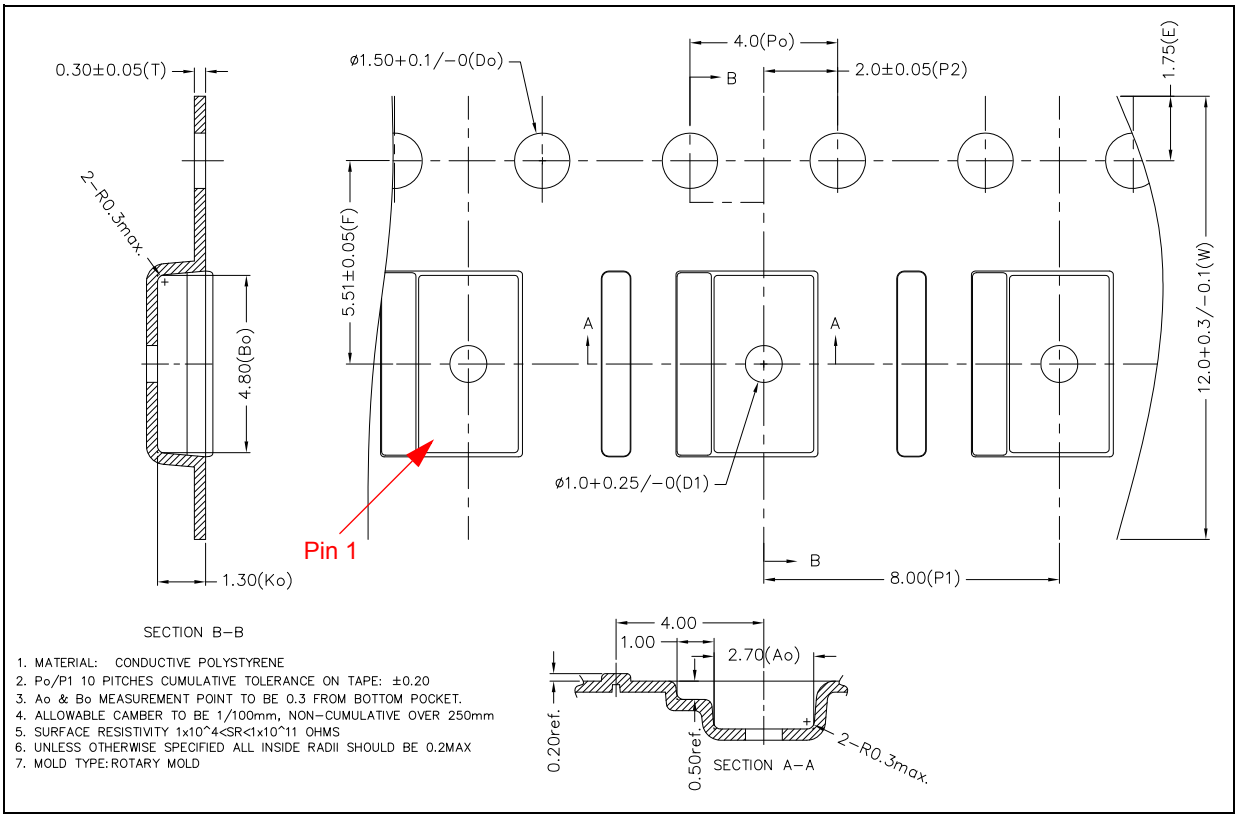
At customer / subcontractor level, it is recommended to mount theVL53L0X in a clean environment to avoid foreign material deposition.

To help avoid any foreign material contamination at phone assembly level the modules will be shipped in a tape and reel format with a protective liner. The packaging will be vacuum-sealed and include a desiccant.

The liner is compliant with reflow at 260°C. It must be removed during assembly of the customer device, just before mounting the cover glass.

8.3.1 Tape outline drawings

Figure 27. Tape outline drawing



8.4 Pb-free solder reflow process

Figure 28 and *Table 15* shows the recommended and maximum values for the solder profile.

Customers will have to tune the reflow profile depending on the PCB, solder paste and material used.

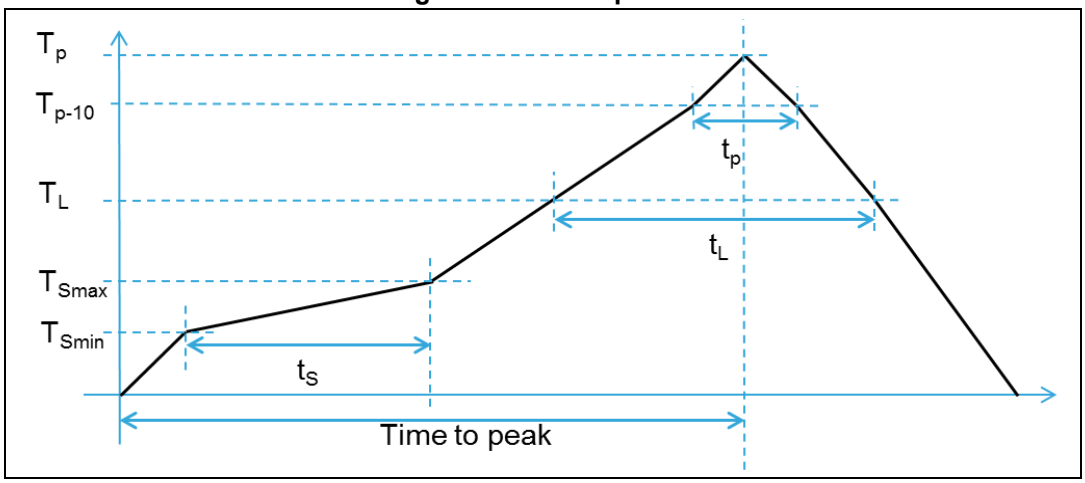
We expect customers to follow the “recommended” reflow profile, which is specifically tuned for VL53L0X package.

For any reason if a customer must perform a reflow profile which is different from “recommended” one (especially peak >240°C), this new profile must be qualified by the customer at its own risk. In any case, the profile have to be within the “maximum” profile limit described in *Table 15*.

Table 15. Recommended solder profile

Parameters	Recommended	Maximum	Units
Minimum temperature (T_S min) Maximum temperature (T_S max) Time t_s (T_S min to T_S max)	130 200 90-110	150 200 60 - 120	°C °C seconds
Temperature (T_L) Time (t_L) Ramp up	217 55-65 +2	217 55 - 65 +3	°C seconds °C/second
Temperature (T_{p-10}) Time (t_{p-10}) Ramp up	- - -	250 10 +3	°C seconds °C/second
Peak temperature (Tp)	240	260 max	°C
Time to peak	300	300	seconds
Ramp down (peak to T_L)	-4	-6	°C/second

Figure 28. Solder profile



Note: Temperature mentioned in Table 15 is measured at the top of VL53L0X package.

Note: The component should be limited to a maximum of 3 passes through this solder profile.

8.5 Handling and storage precautions

8.5.1 Shock precaution

Proximity sensor modules house numerous internal components that are susceptible to shock damage. If a unit is subject to excessive shock, is dropped onto the floor, or a tray/reel of units is dropped onto the floor, it must be rejected, even if no apparent damage is visible.

8.5.2 Part handling

Handling must be done with non-marring ESD safe carbon, plastic, or Teflon tweezers. Ranging module are susceptible to damage or contamination. A clean assembly process is advised at customer after un-taping the parts, and until a protective cover glass is mounted.

8.5.3 Compression force

A maximum compressive load of 25N shall be applied on the module.

8.5.4 Moisture sensitivity level

Moisture sensitivity is level 3 (MSL) as described in IPC/JEDEC JSTD-020-C

8.6 Storage temperature conditions

Table 16. Recommended storage conditions

Parameter	Min.	Typ.	Max.	Unit
Temperature (storage)	-40		+85	°C

9 Ordering information

Table 17. Ordering information

Sales type	Package	Packing
VL53L0CXV0DH/1	Optical LGA12 with liner	Tape and reel

10 Acronyms and abbreviations

Table 18. Acronyms and abbreviations

Acronym/ abbreviation	Definition
ESD	Electrostatic discharge
I^2C	Inter-integrated circuit (serial bus)
NVM	Non volatile memory
RIT	Return Ignore Threshold
SPAD	Single photon avalanche diode
VCSEL	Vertical cavity surface emitting laser

11 ECOPACK®

In order to meet environmental requirements, ST offers these devices in different grades of ECOPACK® packages, depending on their level of environmental compliance. ECOPACK® specifications, grade definitions and product status are available at: *www.st.com*.
ECOPACK® is an ST trademark.

12 Revision history

Table 19. Document revision history

Date	Revision	Changes
30-May-2016	1.0	Initial release.
09-Apr-2018	2	Updated title Updated *Features* Small text changes to *Description* Removed note from *Section 2.6.2: Ranging phase* Added text before *Figure 24*, *Section 6: Outline drawing*