UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

Actualización de la operación de un robot humanoide utilizando un sistema embebido

TESIS

Que para obtener el título de

Ingeniero Eléctrico Electrónico

P R E S E N T A

ARAUJO PINO ALAN EMIR

DIRECTOR DE TESIS

Mtro. Adalberto Joel Durán Ortega

Ciudad Universitaria, Cd. Mx., 2017

Agradecimientos

A la memoria de mis abuelos que siempre me apoyaron incondicionalmente.

A mi hermano Andrés Araujo por la ayuda con el formato de los códigos.

A Sandra Méndez por la motivación y ayuda en la corrección.

A mis padres quienes han estado siempre apoyando.

Al Maestro Joel Durán por la paciencia y dedicación a este texto.

Al Doctor Víctor Lomas por las críticas y correcciones acertadas.

Al Doctor Mario Peña por la oportunidad de mi participación en el equipo UNAMoids.

Al Doctor Jorge Cuevas, que, aunque comenzó su participación con el escrito bastante avanzado, proporciono la atención necesaria.

Y a todas las personas que a lo largo de este trabajo me motivaron a terminarlo.

Prefacio

En los últimos semestres de mis estudios universitarios me di a la tarea de participar con el equipo UNAMoids del IIMMAS, encargados entre otras labores, de investigar y poner en marcha robots para tener material dispuesto a la investigación. Entre otras tareas en las que me vi inmerso estuvo la de investigar sobre la posibilidad de actualizar el robot Ullama. Este robot se encontraba en uso, más dada las capacidades de computo, los experimentos que se lograban realizar eran limitados.

Los principales objetivos en aquel momento eran, realizar las conexiones físicas y probar parte por parte los dispositivos periféricos de los cuales el robot humanoide no podía prescindir. Además, de instalar el software necesario para poder realizar pruebas de funcionamiento para algoritmos más avanzados.

Llegado el término de la investigación, nos dimos cuenta de la gran cantidad de información que se había reunido sobre el proyecto, así como de algunos programas y utilidades creados para realizar pruebas de funcionamiento, por lo que se decidió seguir desarrollando algunas utilidades y reorganizar la información de tal forma que existiese documentación formal sobre el funcionamiento, configuración y programación de la nueva versión del robot humanoide. Así entonces, dado que me era necesario tomar una forma de titulación dada la terminación de mis estudios, decidí darme a la tarea de redactar este trabajo en el cual se ven lo más claro que me fue posible, mis ideas y parte de las metodologías que seguí para obtener la actualización del humanoide Ullama.

Índice

1. Generalidades

La robótica es conocida por el común de personas como un tema cultural y no tanto tecnológico, que ha inspirado a escritores y directores de cine para crear sus obras. Aun así, la robótica como tema de estudio tecnológico tiene implicaciones importantes a nivel social y personal, ejemplos claros de esto pueden ser apreciados en la rehabilitación de pacientes con discapacidad o en los quirófanos robotizados (Cortés, 2011).

Así como otras áreas de investigación, la robótica ha ido y sigue cambiando conforme al tiempo dependiente de varios campos de conocimiento como la física, matemáticas, control, electrónica y computación. Esto es un factor a tener en cuenta en el ejercicio de la investigación de este tema, debido a que actualmente la tecnología (sobre todo la basada en la electrónica) se encuentra cambiando vertiginosamente, lo cual apunta a que el desarrollo actual de robots se dispone a quedar rezagada en cuestión de décadas (Cortés, 2011). A pesar de que el objeto desarrollado quedará rezagado (tecnológicamente hablando), la información sobre su desarrollo, así como el conocimiento sobre la resolución de problemas específicos y gran parte de la programación (si esta es requerida) puede ser transferida a proyectos basados en nuevas tecnologías.

1.1 Antecedentes históricos

A lo largo de la historia, el ser humano se ha interesado por replicar características propias de entes animales y sobre todo de ellos mismos (Barrientos, 2007). Existen ejemplos históricos claros desde el año 270 a.C. con las clepsidras de Ctesibius que eran relojes de agua, pasando por aves mecánicas como el Gallo de la catedral de Estrasburgo (1353), el pato que come y grazna de Jaques de Vaucanson (1738), un león creado por Leonardo Da Vinci (1500), las muñecas para servir el té y dibujar de Hanzo Hosokawa (1796) y La familia Maillardet (1805), hasta los nuevos inventos utilizados para servicio y exploración espacial (Barrientos, 2007).

A todas las invenciones anteriormente mencionadas se les da el nombre de autómatas que viene del latín *autómata* y este a su vez viene del griego *autómata* que significa ingenios mecánicos que obran por sí mismos (Española, 2016). Este término fue usado durante gran parte de la historia de la humanidad y fue hasta la obra de teatro de Karel Capek en el año de 1922 que se usó el término de robot proveniente de la palabra checa *robota* que significa trabajo y se usó para designar a máquinas antropomórficas autosuficientes que argumentan con humanos (Cortés, 2011). Actualmente su significado es más extenso, la Academia Mexicana de la Lengua dice "máquina electrónica diseñada para hacer operaciones y movimientos automáticamente" (Academia Mexicana de la Lengua, 2015), dado a los productos introducidos en el mercado en los últimos años basados en la robótica como Darwin-OP utilizado para la investigación o el BB-8 de la saga de Star wars con propósitos de entretenimiento (ROBOTIS, TechSupport_ENG, 2010) (sphero, 2016)

La robótica moderna basada en semiconductores para su control y programación como la conocemos actualmente tiene pocos años de existencia, para darnos una idea, los circuitos integrados fueron inventados en el año 1958 y al poco tiempo se comenzaron a construir los primeros robots que utilizaban estos dispositivos para su funcionamiento, lo cual quiere decir que la robótica moderna ha existido sólo durante poco más de cinco décadas, aunque su existencia ha sido corta en comparación a otras áreas de estudio como la minería o la electricidad ha tenido increíbles contribuciones sin los cuales nuestra calidad o estilo de vida no podría ser sustentada (Barrientos, 2007 y Cortés, 2011).

En la actualidad los robots que más se fabrican son los que están enfocados a la industria automotriz con alrededor 98,900 (noventa y ocho mil novecientas unidades) vendidas solamente en el año 2014 que equivale al 43.18% de las unidades fabricadas en el mismo año (Robotics, 2016). Por otra parte, los robots de servicio que son aquellos que interactúan directamente con personas, se han hecho cada vez más famosos debido a las tecnologías móviles con las cuales mediante un dispositivo uno puede controlar o configurar el comportamiento de estos robots.

1.2 Descripción del problema

El equipo UNAMoids del IIMAS posee un conjunto de robots para hacer experimentos, de entre estos se encuentra el robot Ullama, que es un humanoide integrado totalmente por este mismo equipo. Ullama, es un prototipo cuyo principal propósito es el de llegar a ser una plataforma enfocada a la experimentación.

Una de las principales carencias que tenía Ullama, era la limitada potencia de cálculo con la que contaba para poder procesar video, así también la tarjeta principal que utilizaba para su funcionamiento (sistema embebido) ya no se fabricaba ni se le seguía desarrollando software, lo que significada un estancamiento en el uso de librerías actualizadas para la investigación (como OpenCV). Por otra parte, el sistema operativo, la documentación sobre su uso y el software para su operación eran insuficientes y poco prácticos para personas que recién se encontraban con este robot. (OpenCV_Developers_Team, 2016)

1.3 Objetivo

Diseñar y construir el sistema general de alimentación del robot, usando una fuente "step down", buscando la mejor configuración entre conexiones y baterías respecto a su operación, teniendo en cuenta los siguientes componentes: una tarjeta controladora de servomotores, un sensor de posición y un sistema embebido actual de la marca PCDuino, el cual a su vez sería capaz de utilizar librerías para el procesamiento de imagen y video a través de una cámara web. Así también, se realizará investigación, interconexión, configuración y/o programación de los dispositivos anteriormente mencionados, con el fin de poder hacer nuevamente experimentos con el robot Ullama.

1.4 Hipótesis

Es posible poner en funcionamiento al robot experimental Ullama empleando un nuevo sistema embebido (PCDuino) que permita coordinar el uso de los sensores periféricos como son: una cámara USB, un sensor de orientación I^2C y una tarjeta controladora de servomotores vinculada al sistema embebido a

través de un puerto serie. Mediante esta modificación se pretende aumentar las capacidades de procesamiento del humanoide, permitiendo los movimientos con una realimentación de postura y el procesamiento de video.

1.5 Alcances

Poner en funcionamiento un sistema embebido, el cual este comunicado con tres dispositivos electrónicos enfocados a la visión, movimiento y orientación del robot.

Realizar el diseño tomando en cuenta la portabilidad entre plataformas que ocupen el mismo sistema operativo, así como a los cambios tecnológicos referentes a sistemas embebidos.

Realizar los análisis y consideraciones pertinentes para futuros trabajos en los cuales se agreguen nuevas funcionalidades o actualizaciones.

Propiciar el diseño de un conjunto de librerías que sirvan como plataforma para la experimentación futura con el robot.

En cuanto a las partes estructural, cinemática y dinámica no están contempladas en este trabajo dado que el objetivo es únicamente poner el robot en funcionamiento.

1.6 Sobre este trabajo

EL presente trabajo está dividido en cuatro capítulos, el primero habla de la información necesaria para poder comprender los demás temas vistos, esto incluye una revisión sobre las comunicaciones electrónicas, así como una introducción al software requerido. El segundo capítulo está enfocado a la definición de la arquitectura usada, comenzado por los puertos físicos de comunicación del sistema embebido. Después sigue la implementación y desarrollo donde se habla de las consideraciones y análisis tomas en cuenta para la puesta en marcha del robot. Y, por último, están las conclusiones y perspectivas a futuro, donde se mencionan las aportaciones que tiene este trabajo y una visión propia del potencial que tiene este robot en la investigación.

Al final del trabajo se anexaron tanto las librerías desarrolladas como los ejemplos del uso de las mismas, en cuanto a la escrita en el lenguaje C++, esta se encuentra dividida en dos archivos, un *header* (cabecera) y una clase donde se implementan todos los métodos mencionados en el header, la librería en el lenguaje "C" contiene únicamente métodos y Macros, y finalmente el código para el test de video en el lenguaje C++ con el uso de las librerías OpenCV (Todo esto será explicado a fondo).

Cabe destacar que en este trabajo se instalaron y desarrollaron aplicaciones únicamente con el objetivo de poner al robot en funcionamiento, por lo que para una aplicación que requiera características como confiabilidad o incluso seguridad (no solo para personas sino para los mismos dispositivos) es necesario hacer un previo análisis y hacer las configuraciones pertinentes, tanto en código como en cableado. Por lo anterior se puede entender que existen ciertas condiciones en las que el robot no debe estar por seguridad de los elementos eléctricos (visto más adelante).

2 Marco teórico

Es necesario aclarar que todo el desarrollo de este proyecto fue enfocado a la actualización de un robot humanoide llamado Ullama, el cual se encontraba obsoleto, por lo que el título decidido para este trabajo se refiere a la integración de nuevos y viejos componentes (sensores y actuadores) cuya tarjeta principal es un nuevo sistema embebido con la capacidad de procesar video. Adicionalmente a la integración de los componentes se instalaron librerías para el procesamiento de video, se programaron nuevas clases y funciones enfocadas a la adquisición de datos, y automatización de la generación de información para el manejo de los motores.

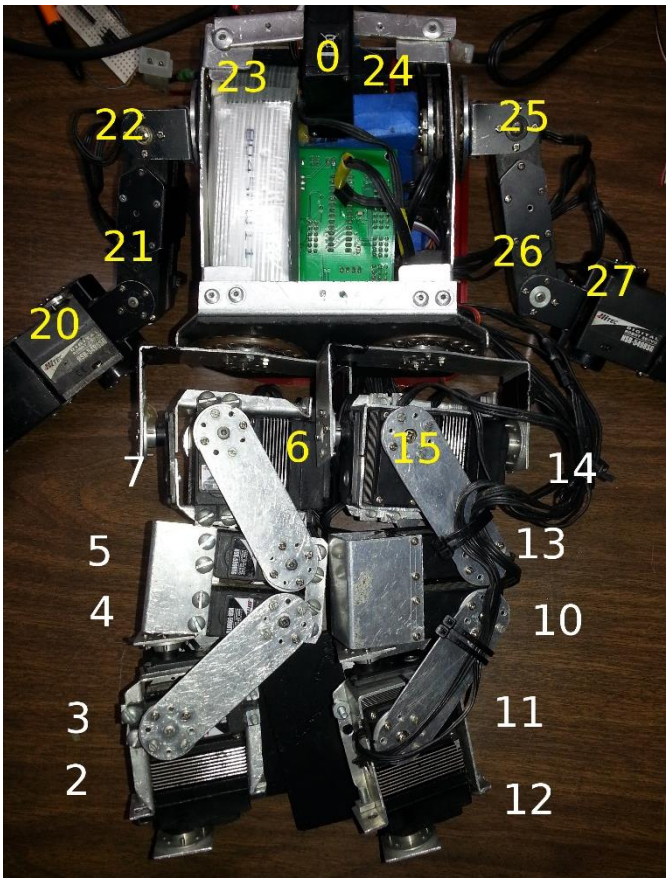


Ilustración 2-1 identificación del número de motor en el Robot Ullama

En la ilustración 2-1 se encuentra el robot Ullama acostado de tal maneja que se alcanza a ver cada uno de los motores que lo componen, así como su actual numeración para su manejo desde el sistema operativo. Los motores 23 y

24 no son totalmente visibles puesto que estos se encuentran dentro del pecho del robot. Además de lo anterior, se puede apreciar también en la zona central del pecho, parte de la tarjeta encargada de controlar los motores y justo junto a esta una de las dos baterías que energizan los circuitos de Ullama.

Para poder entender mejor tanto la arquitectura del robot como el desarrollo del software, en este capítulo se muestra el funcionamiento de las comunicaciones electrónicas usadas para la transferencia de información entre los dispositivos, así como una introducción a lo que es el sistema operativo GNU/Linux para sistemas embebidos y una breve descripción de lo que es el procesamiento de imagen y video enfocado al reconocimiento de bordes.

2.1 Antecedentes del software

Para hacer más entendible la situación en la que se encontraba el robot se presenta la siguiente información en la cual se expone la forma en la cual se manejaban los motores y se obtenían mediciones de los sensores. Así también, se pone un ejemplo de una captura de posiciones que se alcanzó a realizar con los programas que usaba anteriormente Ullama.

Anteriormente el robot tenía con un conjunto de programas, cuya finalidad era comunicarse con cada uno de los sensores y actuadores que se conectaban al sistema operativo anterior y eso se hacía mediante librerías específicas de cada uno de los fabricantes de los componentes. Estos programas además contaban con un auto instalador escrito para la consola de GNU/Linux BASH, el cual realizaba la compilación del código y organización en diferentes carpetas para su posterior ejecución.

Entre otras características que se lograron probar con el conjunto de programas que tenía el humanoide, estaba la de poner al robot en una cierta posición mediante el uso de los servomotores. Para lograr el movimiento y alcanzar la posición deseada, la información de cada uno de estos motores se almacenaba en un texto plano (archivo que se puede leer directamente sin necesidad de un traductor) de tal forma que la posición de cada renglón en el texto correspondía a el número de motor a mover. Así, el programa leía línea a

línea cada una de las posiciones y finalmente agregaba la velocidad y tiempo de activación de los motores (justo en las ultimas 4 líneas). Cabe notar que estos valores (velocidad y tiempo) se aplicaban a todas las instrucciones antes de ser mandados a la tarjeta que controlaba los motores.

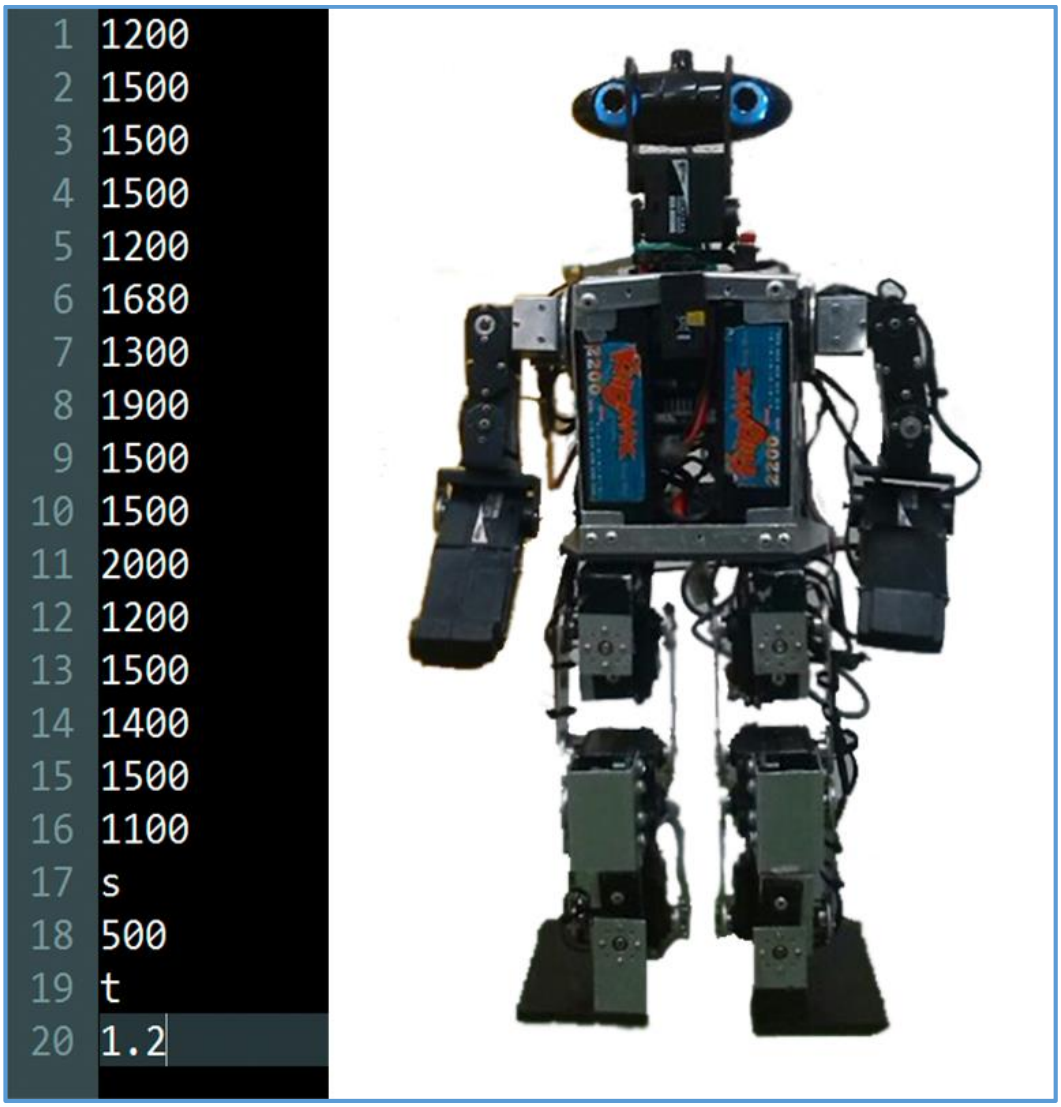


Ilustración 2-2 Ilustración del robot Alpha1 Pro en posición neutral o de descanso. Imagen encontrada en el catálogo de búsqueda de la página Aliexpress (UBTECH, s.f.) (Aliexpress, 2016)

En la ilustración 2-2 (encontrada en un catálogo de compras de internet) se puede notar en paralelo, la posición neutra (piernas estiradas y brazos colgando a los costados) de un robot humanoide y sus supuestas instrucciones para los servomotores (haciendo caso a lo explicado anteriormente). Como anteriormente es mencionado, existe una correspondencia entre el renglón y la posición del motor siendo así, el motor numero 0 tendrá la posición 1200 (explicado en el apartado de manipulación de motores), el motor numero 1

tendrá la 1500 y así consecutivamente hasta llegar al renglón 17, 18, 19 y 20 donde se aprecia una “S” y una “t” seguidos cada cual de un numero el cual indica la velocidad y el tiempo del movimiento de todos los motores respectivamente.

En cuanto al uso de giroscopio en el robot, anteriormente ya se tenían programadas algunas funciones, pero debido al cambio de plataforma estas librerías a pesar de poder ser compiladas en el nuevo sistema operativo, no llegaban a reconocer el giroscopio. Dado este problema se decidió a programar unas nuevas funciones básicas para lectura y escritura en I^2C que puedan ser configurables para comunicarse con diferentes dispositivos.

Entre la información y los archivos que usaba anteriormente el robot Ullama, no se encontró nada referente al procesamiento de imagen y video. Por lo que uno se optó por buscar un sistema embebido que fuese capaz de realizar este tipo de tareas.

2.2 Revisión de las comunicaciones electrónicas

Para la realización del proyecto fue necesario comunicar la tarjeta controladora de motores, la cámara, el sensor de orientación y el sistema embebido, todos estos dispositivos electrónicos utilizan las comunicaciones electrónicas para la transferencia de información de un dispositivo a otro. Dado que todos los dispositivos poseen diferentes tipos de comunicación se decidió hacer una revisión de las más importantes.

Las comunicaciones electrónicas se definen como la transmisión de información mediante sistemas a base de semiconductores, los medios de propagación de dicha información pueden ser mediante el uso de diferentes fenómenos físicos. Estos fenómenos físicos son relativos al electromagnéticos y muchas de sus aplicaciones utilizan cables metálicos, fibra de vidrio o incluso la atmosfera terrestre (Tomasi, 2003).

En esta revisión no se tomó en cuenta la comunicación USB puesto que, ante el sistema operativo, esta funciona como *plug and play* (conecta y usa).

2.2.1 Comunicación UART

La comunicación UART es un acrónimo en inglés (Universal Asynchronous receiver / transmitter) el cual fue diseñado en gran parte por Gordon Bell, un reconocido ingeniero informático del MIT (Massachusetts Institute of Technology). Esta comunicación fue introducida para poder comunicar la primera serie de computadoras electónicas llamas PDP a distancias cortas, es una comunicación *peer to peer* o uno a uno y tiene diferentes configuraciones para asegurar una comunicación estable (Allison, 2015).

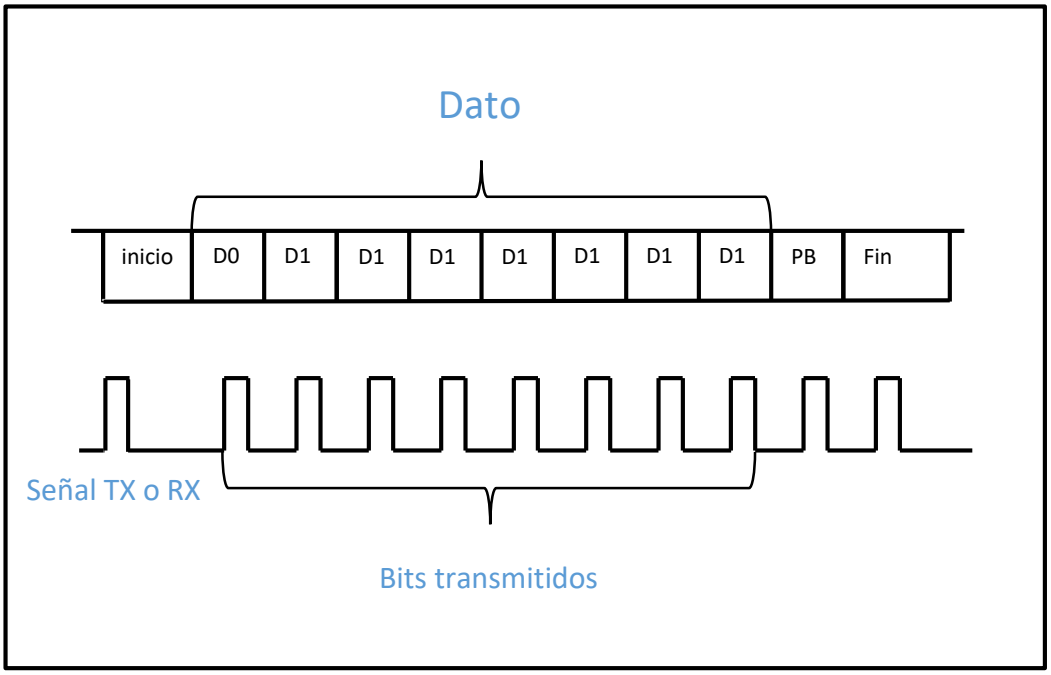


Figura 2-1 Trama de datos contra señal eléctrica TX o RX (Tomasi, 2003)

La comunicación empieza con un bit en alto por parte del transmisor, para esto el receptor se encuentra en modo de espera, una vez que se ha mandado ese bit en alto se comienza a transmitir secuencialmente bit por bit hasta completar los ocho bits que crean el byte de información que se necesita mandar, finalmente para terminar de transmitir, se manda un bit de stop y así se termina la transmisión. Una de las configuraciones que tiene la transmisión serial UART es un bit de paridad que sirve para comprobar que la información recibida es correcta; La función del bit de paridad es completar la suma de los bits trasmitidos de tal manera que sea igual a un número par o impar (depende de la configuración), para saber si es par o impar tanto el receptor como el transmisor deben estar configurados para aceptar dicha configuración. Entre otras

configuraciones que tiene la comunicación UART es la asignación de la velocidad de conexión que se le conoce como *Baudage* o *Baud rate* y se refiere al número de paquetes de información que se mandarán por segundo, esto es muy importante puesto que uno puede transmitir y recibir erróneamente valores debido a una mala configuración. Además de la velocidad en la mayoría de los dispositivos modernos se puede configurar el número de bits que se pueden transmitir, el tipo de bit de paridad o incluso la longitud del bit de stop (Tomasi, 2003).

2.2.2 Comunicación I^2C (inter-integrated circuit)

La comunicación *I^2C* (i cuadrada c) fue diseñada por la compañía Philips en el año de 1982 viene de una deformación de su acrónimo en inglés (inter-integrated circuit), es una comunicación serial en la cual hay maestros y esclavos (Philips Semiconductors, 2003), comúnmente el maestro es el encargado de coordinar los procesos principales de cierta tarea, este se puede ver ejemplificado como un sistema embebido, microcontrolador o tarjeta de desarrollo, mientras que el esclavo (regularmente) es el encargado de aportar cierta información adicional proveniente de sensores o de ampliar la características propias de maestro funcionando como ejemplo como memoria externa.

Se le llama maestro a todo dispositivo que sea capaz de comenzar la comunicación a través del bus (canal de comunicación electrónico) y esclavos a los circuitos integrados que están a la espera de indicaciones ya sea para leer o escribir en su memoria interna.

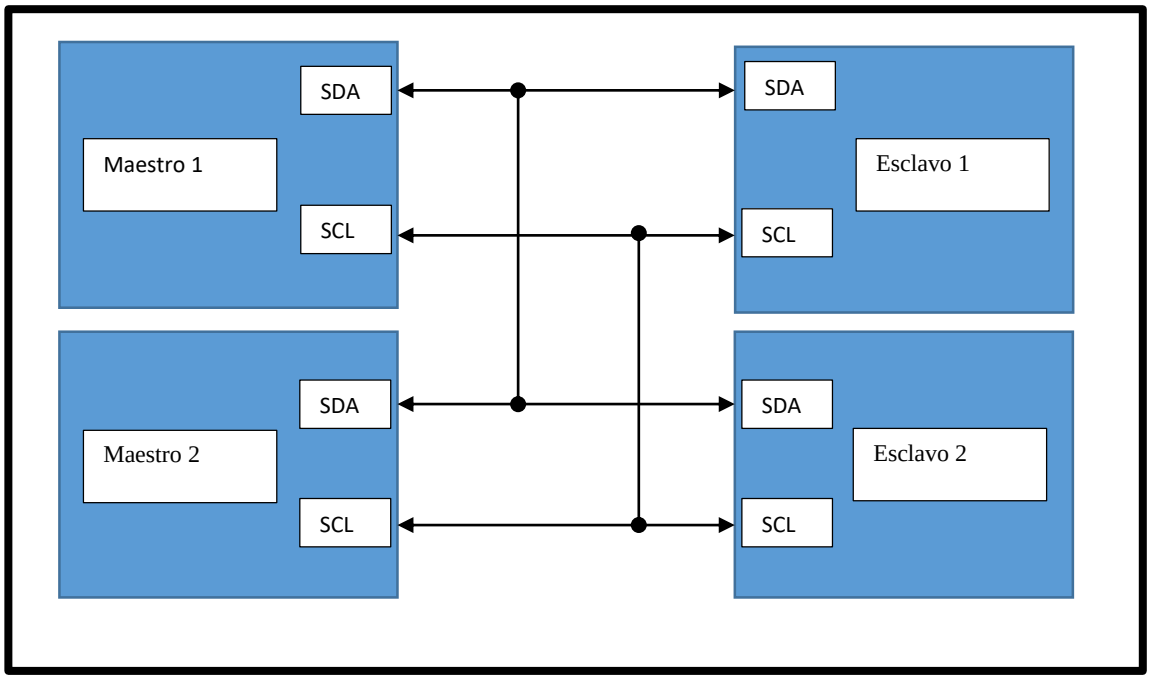


Figura 2-3 Esquema de ejemplo en las conexiones entre maestros y esclavos inspirado en (sparkfun, I2C - learn.sparkfun.com, 2016)

Este tipo de comunicación se realiza con solo 2 hilos de conexión entre circuitos y un hilo extra para la referencia de voltaje no mostrada en la ilustración. Su funcionamiento es el siguiente.

1. El maestro espera hasta no ver actividad en el bus de comunicación (SDA) y solo cuando el reloj (SCL) y el bus están en altos está libre.
2. Pone un mensaje en el bus avisando que está por comenzar. Todos los otros dispositivos escuchan al bus de datos esperando que se marque su nombre (dirección del dispositivo).
3. La conexión con el reloj valida la información que se transmite por el bus o en otras palabras la información que tenga el bus solo será aceptada cuando el reloj cambie de estado lógico bajo a alto (cero a cinco volts respectivamente).
4. Se arroja el nombre (dirección) del dispositivo con el cual se quiere comunicar al bus de comunicación.
5. Manda un bit al bus diciendo si quiere mandar o recibir información del otro chip.

6. Se pregunta al otro circuito integrado sí reconoce la dirección y está listo para comunicarse.
7. Una vez que el otro circuito ha sido avisado (mediante un bit) que todo está bien la información puede ser transmitida.
8. El primer circuito integrado manda o recibe tantos bytes como sean necesarios. Además, después de cada byte se retransmite otro byte avisando que la transferencia se realizó con éxito.
9. Cuando toda la información se ha terminado de transmitir el primer chip debe liberar el bus y esto se hace con un mensaje especial llamado 'STOP'. Este es solo un bit de información transferida por en un modo especial donde participan tanto el reloj como el bus.

(Philips Semiconductors, 2003) "Trabajo original publicado en 2003"

Una condicionante física que tiene esta forma de comunicación para lograr realizar transferencias exitosas entre los dispositivos interconectados es que no exista una capacitancia mayor ni cercana a los 400 [pF] (cuatrocientos picos Faradios). Si bien esta consideración es importante tenerla en cuenta, para este proyecto no fue necesaria tomarla, dado que las distancias entre dispositivos que llegan a generar esta capacitancia son mayores a los de 5 [m]; tomando en cuenta la aproximación descrita en la datasheet (hoja de datos del fabricante) donde la capacitancia de un alambre es de 80pF/m (ochenta picos Faradios por cada metro) (Inc. I. , RM-MPU-6000A, 2012, pág. 25).

2.3 Introducción a Linux embebido

El núcleo Linux inicialmente fue diseñado para la arquitectura x86 de Intel, pero poco a poco con los avances que ha tenido la computación electrónica han salido cada vez más conjuntos de instrucciones. Entre otros conjuntos de instrucciones actuales para el núcleo Linux están ARM, x86, x64; estos últimos dos son de tipo CISC (del acrónimo en inglés Complex Instruction Set Computer) y la primera es tipo RISC (por sus siglas en inglés Reduced Instruction Set Computer).

Muchos de los microprocesadores y microcontroladores modernos utilizan la arquitectura ARM debido a su alta optimización de cómputo y energía y por esto mismo se ha necesitado un sistema operativo para poder aprovechar toda su capacidad de computo (King, 2015).

GNU/Linux es un excelente sistema para aplicaciones embebidas no solo por su desarrollo de código abierto (open source) que permite conocer más detalladamente la programación y funcionamiento del sistema, ni por la gran cantidad de personas que contribuyen a la documentación y programación si no por la forma en la que está programado el sistema. La característica más importante por lo que el sistema GNU/Linux es tan útil para aplicaciones embebidas es que permite programar de tal forma que es posible tener acceso directo a cualquier parte del hardware, lo que es una ventaja increíblemente grande para el desarrollo de aplicaciones que involucran puertos de comunicación como I2C, UART o puertos lógicos, entre otros.

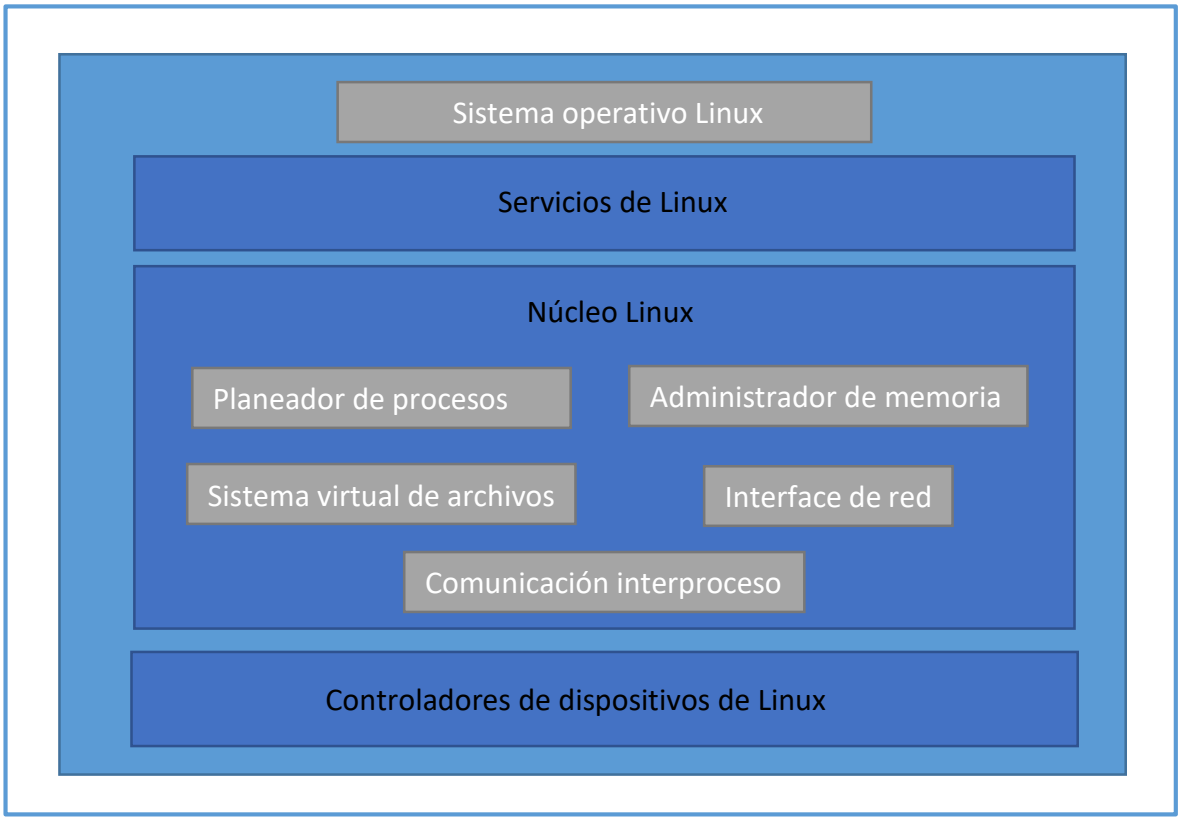


Figura 2-4 Diagrama a bloques del sistema operativo Linux (Noergaard, 2005).

En la figura 2-4 se muestra que el sistema operativo Linux está concentrado en tres partes, los servicios, el núcleo y los controladores de

dispositivos. A diferencia con otras arquitecturas, como la que se muestra en la figura 2-5, es apreciable notar que el acceso a los dispositivos puede hacerse directamente en cualquier parte sin necesidad de comunicarse con múltiples intermediarios, lo que representa una gran ventaja para este proyecto. Otra gran ventaja es el sistema virtual de archivos el cual permite manejar los mismos dispositivos como archivos.

Además de lo anterior, GNU/Linux es un sistema portable, multitarea y multiusuario Unix (Vaduva, 2015) y gracias al sistema virtual de archivos, tanto los dispositivos de almacenamiento extraíbles como los discos duros se pueden manejar también como archivos.

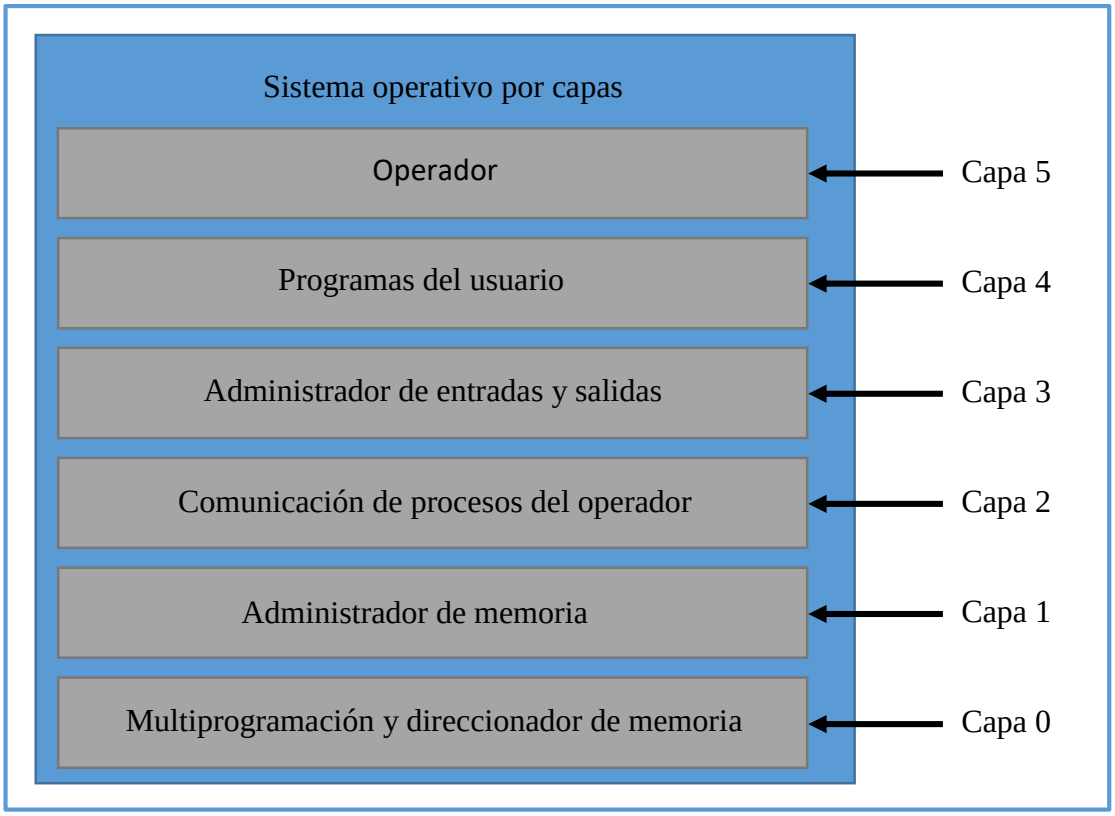


Ilustración 2-5 Diagrama a bloques de un sistema operativo por capas (Noergaard, 2005, p. 386).

Por parte del diagrama de la Ilustración 2-5, se muestra lo que es un sistema operativo programado por capas. En esta arquitectura existe una dependencia contante de los bloques superiores a los inferiores de tal manera que para poder realizar ciertas operaciones es necesario comunicarse con cada una de las partes intermedias del sistema operativo antes de llegar a realizar la tarea desecada (Noergaard, 2005) (Vaduva, 2015).

2.4 Introducción al procesamiento de Imagen y video

El procesamiento de video es una disciplina ampliamente aplicable a distintas ramas del conocimiento, entre otras esta la sismología, la astronomía, la medicina, y la ingeniería y dado el incremento en la potencia de computo de los procesadores en los últimos años, las aplicaciones y campos en los cuales se ve inmersa ha ido en aumento. Entre otras aplicaciones desarrolladas en las últimas décadas están el IRM (Imagen por resonancia magnética), el reconocimiento de patrones para automatización industrial, la monitorización de los astros y la video-vigilancia (Bovik, 2000).

En lo que se refiere a este trabajo el video es una secuencia de imágenes digitales, (perteneciente al campo de los números discretos) cuya información a su vez, se encuentra ordenada en arreglos matriciales. Así mismo estos, se constituyen por medio de pixeles, que son un conjunto de valores relativos a los colores y cantidad de luz que percibe el ser humano.

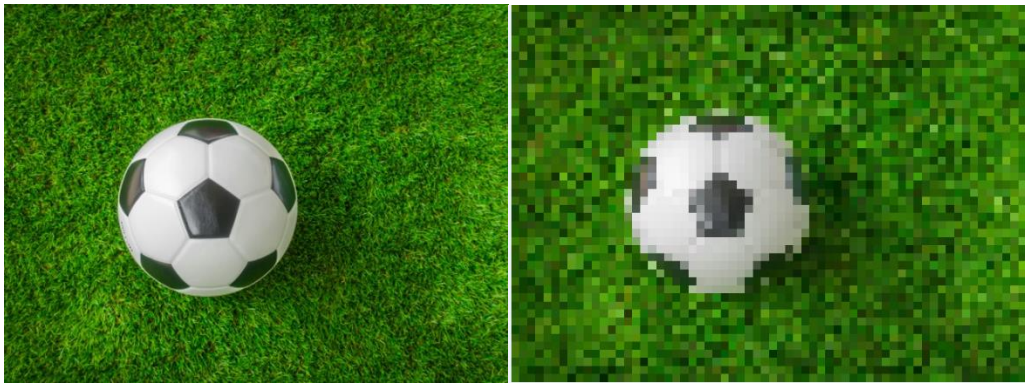


Ilustración 2-6 Imagen de un balón en el pasto, en resoluciones distintas donde se aprecian los pixeles.

Sea así, un ejemplo de una imagen digital es el mostrado anteriormente. La imagen de lado izquierdo tiene un tamaño de 640 x 480 pixeles mientras que la de la derecha 58 x 43 pixeles, a esta última se le ha bajado la resolución (cantidad de pixeles por área) para ejemplificar la información almacenada en la imagen. Ambas están compuestas a su vez de los tres colores primarios rojo, verde y azul.

El procesamiento del video es un tema decisivo para ciertas aplicaciones y eventos en las cuales se necesita tener información de la posición, color o textura de los objetos que le rodean. Un ejemplo de esto puede ser visto en la

competencia RoboCup en la cual se usa procesamiento de imagen y/o video para obtener mayor información del entorno con la cual tomar mejores decisiones (RoboCup, 2016).

2.4.1 Espacio de color

Los colores son percepciones humanas de un cierto rango del espectro electromagnético. Si bien, es cierto que la visión humana puede percibir un rango continuo del espectro, existen ciertas longitudes de onda o colores a los cuales el ojo humano es más sensible. A estos colores (Rojo, verde y azul) se les conoce como los colores primarios (Bovik, 2000).

En lo que se refiere a la computación actual el espacio de color, es la forma en que se encuentra almacenada la información de cada uno de los pixeles (siendo la de los colores primarios la usada en este trabajo), comúnmente para denominar a los colores primarios se utiliza el acrónimo en inglés RGB (red, green and blue) (Gloria Bueno García, 2015).

Ilustración 2-7Descomposición de la imagen de un balón en los colores primarios RGB

En la ilustración anterior se puede apreciar la imagen un balón que ha sido separada en sus tres canales de color (rojo, verde y azul), cabe destacar que, dado que los elementos de la imagen pertenecen más a los colores verdes, los canales rojo y azul se ven opacos puesto que la los valores promedio de esta imagen son mínimos.

Así también además del espacio de color RGB donde cada pixel contiene la información perteneciente a cada color existen otros espacios de color como la escala de grises donde el valor de cada pixel es representado en capas de grises.

Ilustración 2-8Imagen de un balón en escala de grises

En la ilustración anterior se aprecia la misma imagen de un balón donde se han combinado los tres canales y ha sido expuestos en escalas de grises.

2.4.2 Filtros de imagen

Un filtro de imagen digital se entiende como un procesamiento en el cual se reducen o se mejoran ciertas características presentes a la información de una imagen, algunas de estas características pueden ser debidas a el ruido inherente en los sensores que capturaron la imagen (como en una cámara digital), por descuidos en la configuración de captura (alto contraste, baja exposición entre otros efectos de cámaras), texturas o detalles debidos al contenido. Además de poder mitigar o potenciar los efectos anteriormente descritos, los filtros también son usados para adaptar imágenes como por ejemplo en el cambio de resolución, o también en el cambio cromático (de imagen de colores a blanco y negro).

El ruido en la fotografía, es un elemento indeseable en la mayoría de las aplicaciones donde se requiere un cálculo exacto para la toma de decisiones mediante el procesamiento de imágenes por lo que existen distintas técnicas (no solo mediante software) para disminuirlo.

Así, dado el uso de métodos de obtención de imágenes susceptibles al ruido, se han hecho más usuales los filtros supresores de ruido. Y entre los filtros supresores está el gaussiano que es el más común para el uso de reconocimiento de bordes el (visto más adelante) (DAVIES, 2012).

Ilustración 2-9Imagen de un balón alterada mediante un filtro gaussiano

En la imagen anterior se puede apreciar la imagen de un balón a la cual se le ha cambiado de espacio de color y finalmente ha sido procesada con un filtro gaussiano. Esta última imagen se puede apreciar borrosa a comparación de la original (expuesta en el subíndice anterior).

Otro filtro bastante utilizado por los principiantes en procesamiento de imagen y video es el filtro Canny el cual sirve para el reconocimiento de bordes en imágenes con bajo ruido.

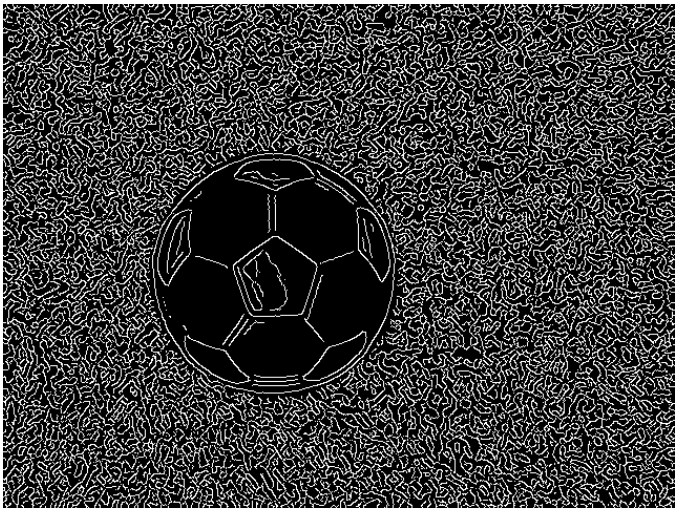


Ilustración 2-10Imagen de un balón después de ser procesada con un cambio de espacio de color, un filtro gaussiano y un filtro de reconocimiento de bordes Canny

En la imagen anterior se alcanza a notar el contorno del balón que se encuentra sobre un pastizal, se puede apreciar también los bordes que genera el

algoritmo sobre el pasto; este último en muchos casos se considera inservible puesto que para procesamientos posteriores no aporta información relevante.

Para conocer la teoría a fondo y tener una explicación formal se puede consultar el libro “Computer and Machine Vision” de DAVIES (DAVIES, 2012).

3 Arquitectura del robot

Para un mejor entendimiento del presente trabajo se expone a continuación la arquitectura que constituye al robot con la implementación del sistema embebido. El primer acercamiento a este tema es que el sistema embebido o PCDuino es quien orquesta todas las acciones en los demás dispositivos (más adelante se explicara a fondo). A su vez cada uno de los dispositivos conectados a la PCDuino hacen un procesamiento con la información proporcionada y generan ciertas acciones ya sean físicas o lógicas (las cuales no son necesarias explicar).

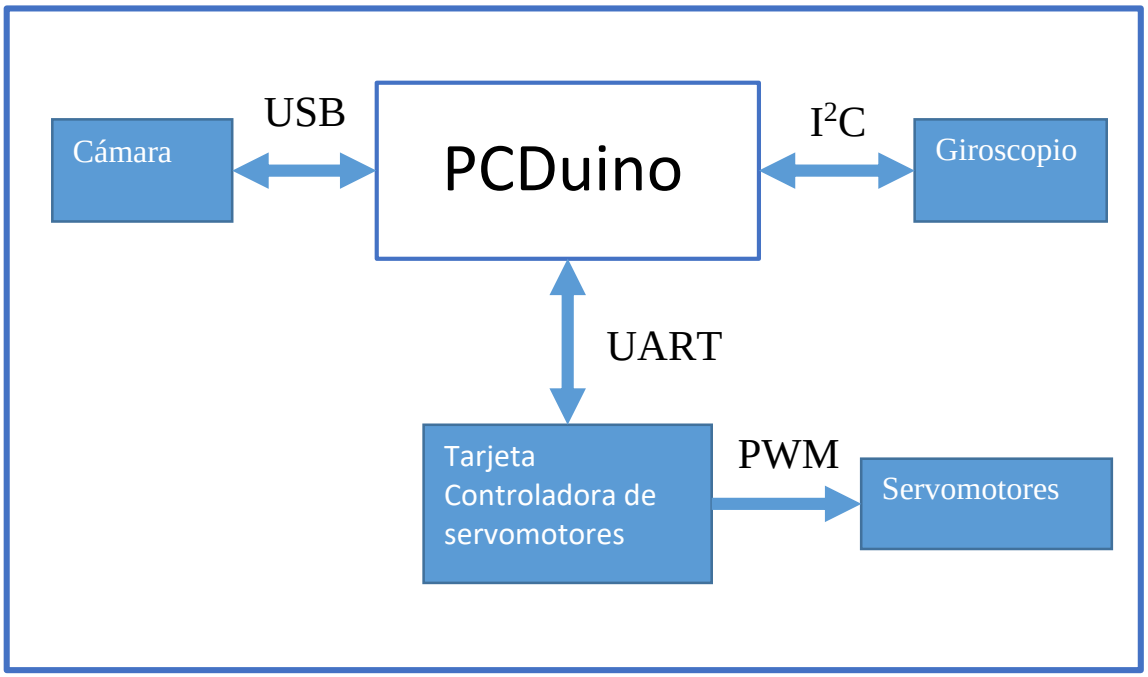


Ilustración 3-1Arquitectura del robot Ullama

En la ilustración 2-1 se tiene al sistema embebido al centro, conectado mediante diferentes formas de comunicación electrónica a tres dispositivos. Estos dispositivos tienen diferentes funciones, siendo así, la cámara sirve para la adquisición de imágenes, el giroscopio obtiene información de la orientación del robot y la tarjeta controladora de servomotores convierte la información transmitida a señales eléctricas PWM (explicado en la manipulación de motores) que finalmente serán transformadas por los servomotores a posiciones de cada uno de los motores. El número de servomotores es totalmente dependiente de

la tarjeta que los administra, en este caso con la tarjeta que se tiene en uso actualmente, se tiene la capacidad de soportar hasta 32 motores funcionando al mismo tiempo.

Por parte de las comunicaciones, se puede observar que para todos los dispositivos exceptuado los motores, la información puede fluir en ambos sentidos (cada parte puede funcionar como emisores o receptores según se necesite).

3.1 PCDuino

Las computadoras de una sola tarjeta o *single board computer* (SBC por su acrónimo en inglés) son parecidas al pc de escritorio con la diferencia de que las SBC están armadas de tal manera que en una sola placa de silicio se encuentran montados el procesador, la memoria y los puertos de entrada y salida necesarios para comunicarse con el medio (Newark, 2014).

Debido al decremento en el precio y a su vez el aumento en las prestaciones de las SBC, estas se han vuelto cada vez más populares entre personas interesadas en la electrónica, un ejemplo claro es la Rasberry pi creada en el laboratorio de computo de la Universidad de Cambridge, que aún conserva el costo de $35 dólares en algunas tiendas y cuyo objetivo fue tener una plataforma que le permitiera a la gente joven aprender a programar (Newark, 2014).

Estas tarjetas están diseñadas para diferentes necesidades por ejemplo las tarjetas de desarrollo Rasberry pi fueron inicialmente diseñadas para enseñar a las personas como está construida y programada a nivel básico una computadora, Atmel's SAMA5D3 fue usada para prototipaje rápido, RIoTboard se centra en el desarrollo con Android y el desarrollo del internet de las cosas IoT (internet of things) (Newark, 2014).

En el caso de este proyecto se utilizó el sistema embebido PCDuino. La principal de las razones por lo cual se decidió esto, fue debido a su disponibilidad en el laboratorio de automatización y control del IIMAS.

PCDuino es distribuido por, LinkSprite una empresa australiana que se dedica a la enseñanza, desarrollo y venta de sistemas embebidos (linksprite, 2015). Además, de los sistemas embebidos que esta empresa vende, también pone a disposición a través de internet, algunos códigos de ejemplo, que demuestran el uso y configuración de los dispositivos periféricos con los que cuentan estas tarjetas.

Especificaciones del Hardware	
CPU	1GHz ARM Cortex A10
GPU	OpenVG 1.1 Mali 400 core
DRAM	1GB
Almacenamiento	2GB Flash, microSD hasta 32GB
Salida de video	HDMI
SO	ubuntu 12.04
Interface	Headers de 2.54mm
Interface de red	10/100Mbps RJ45 y extensión USB Wifi
Alimentación	5V a 2000mA
Tamaño	125mm X 52mm

Tabla 3-1 Especificaciones de la tarjeta PCDuino (LinkSprite, 2015) (Liu, 2013, pág. 2)

En la tabla 2-1 se pueden apreciar las especificaciones con que cuenta la tarjeta PCDuino v1. Esta tarjeta cuenta con una memoria flash interna la cual puede almacenar un sistema operativo Linux básico (sin interface gráfica amigable) o se le puede agregar una tarjeta microSD con Ubuntu (sistema operativo del que se habla más adelante) ya instalado previamente (SFUPTOWNMAKER, s.f.).

3.1.1 Puertos de PCDuino

Los puertos de comunicación en los sistemas embebidos son parte medular puesto que es mediante estos puertos que el sistema interacciona para poder realizar la función que se le pide. Por lo que en este apartado se exponen todos los puestos de comunicación del sistema embebido usado.

Existen muchos tipos de puertos de comunicación digitales principalmente divididos en dos grupos: los de comunicación serial y los de comunicación paralelo. La tarjeta PCDuino pose puestos de comunicación de ambas categorías y pueden ser vistos en la siguiente tabla.

Puertos de PCDuino	
GPIO*	14
PWM*	6
ADC	6
UART*	1
I2C	1
SPI*	1
USB	3
ETHERNET	1
HDMI	1
*Los pines pueden compartir funciones por lo que se debe configurar antes la tarjeta	

Tabla 3-2 Lista con la cantidad y tipo de puertos de la tarjeta PCDuino (Liu, 2013, pág. 2 y 6)

En la tabla 2-2 podemos apreciar un listado de los puertos de los que dispone la tarjeta. El sistema embebido PCDuino tiene puertos GPIO (entradas y salidas de propósito general por su acrónimo en inglés), PWM (modulación por ancho de pulso por su acrónimo de ingles), ADC (convertidor analógico digital por su acrónimo en inglés) y USB (canal de comunicación universal por su acrónimo en inglés). Estos deben de ser configurados directamente por el sistema operativo o en su defecto por el usuario para su correcto funcionamiento.

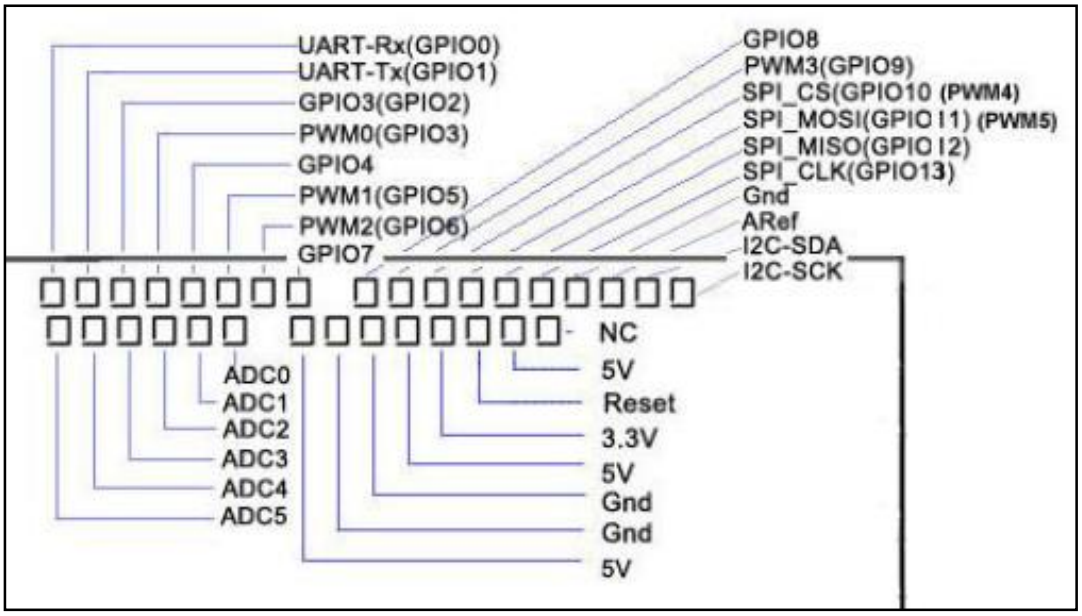


Figura 3-1 Pines de conexión de la tarjeta PCDuino. Vista desde la esquina superior derecha de la tarjeta (Liu, 2013, pág. 10)

En la Figura 2-1 se muestra la posición en la que se encuentran ubicados los puertos de comunicación en la PCDuino, aunque pareciera que esta tiene muchos, realmente no todos pueden estar activados al mismo tiempo. Esto es debido a que existen pines que comparten funciones, como ejemplo tenemos los pines del UART (receptor y transmisor asíncrono universal por su acrónimo en inglés) que comparten la posición un GPIOs (Liu, 2013, pág. 62).

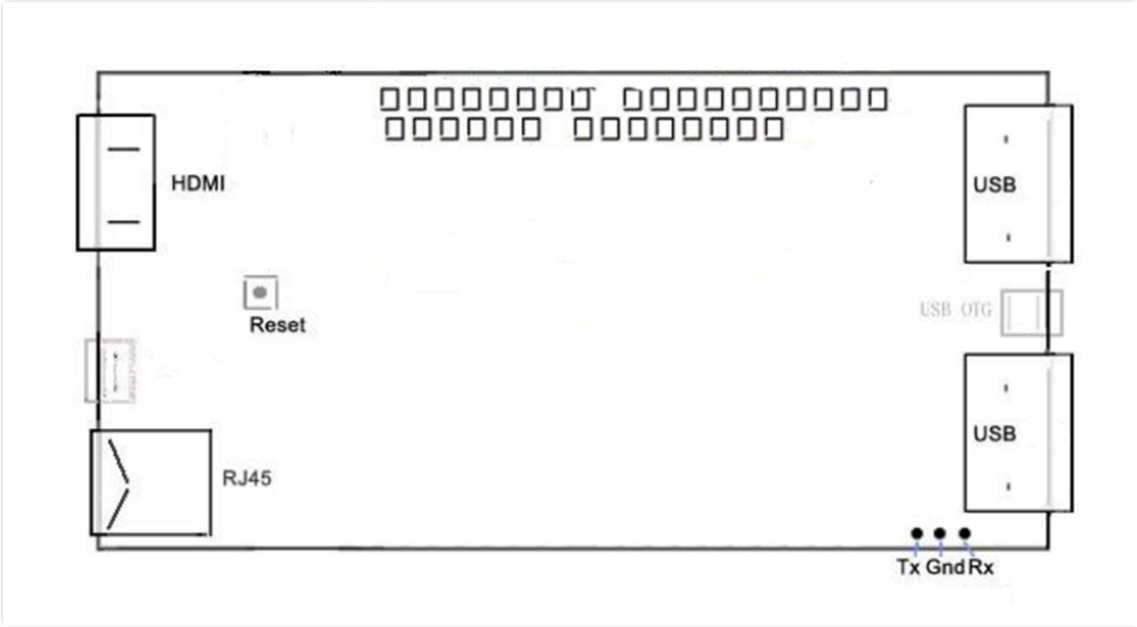


Figura 3-2 Pines de la tarjeta PCDuino. Vista frontal al procesador (Liu, 2013, pág. 10)

En la parte superior central de la figura 2-2 se puede aprecia la ubicación física de los pines, de los cuales, se habla en párrafos anteriores. Además de los ya mencionados se tienen puertos de comunicación para dispositivos periféricos como HDMI, USB y Ethernet de los cuales se tiene dos USB, un micro USB, un USB OTG y un RJ45 respectivamente. En la siguiente lista se da una breve descripción de su uso.

- HDMI: *High-Definition Multimedia Interface (interface multimedia de alta definición),* conector que va a una pantalla para su uso con raton y o teclado.
- USB y micro USB: *Universal Serial Bus (canal universal de comunicación serial),* puertos de alimentación e intercambio de datos (ejemplo archivos con memorias extraíbles).
- RJ45: *registered Jack;* conector para enlaces con tarjetas de ethernet, usado para generar o conectarse a una red local.

En cuanto al botón de *reset*, este sirve para reiniciar el SO. Además de lo anterior, se tiene un puerto para depuración, que se puede apreciar en la parte inferior derecha de la imagen, este es serial tipo UART y se usa para entrar directamente al sistema operativo de la tarjeta en tal modo.

3.2 Servo motores

Los servomotores son motores que mediante un sistema de control (normalmente electromecánico) mantiene una cierta posición respecto a una señal dada. Estos tienen diversas aplicaciones y entre otras se encuentran la robótica.

Ilustración 3-2 Servomotor de modelismo (González, 2016)

El humanoide cuenta con servomotores de modelismo para realizar todos sus movimientos, y estos se encuentran en la siguiente disposición:

· 4 en cada brazo

·6 en cada pierna

·1 para el cuello

·1 para la cabeza

Cada uno de los motores es controlado con una señal PWM (*pulse-width modulation*) la cual se explicará más adelante (ROBOTIS, Overview of Communication, s.f.).

Los primeros sistemas fueron desarrollados en el siglo XIX ya inmersos en la revolución industrial y el servomotor de DC como tal fue desarrollado poco después de la invención de dicho motor cerca del año 1976. Inicialmente los servomotores solo tenían control de posición y fue cerca de 1980 cuando se agregó el control para velocidad (NAKAMURA, 1998).

El primer robot con actuadores totalmente eléctricos fue creado por ABB en 1974, por lo que se puede deducir que la robótica como la conocemos comúnmente en imágenes y medios culturales realmente tiene poco más de cuatro décadas (Hughes, 2006).

El nombre servo motor viene del tipo de control llamado servocontrolado o de ciclo cerrado. El control específico para este sistema de control toma una o dos variables para realizar la realimentación que son posición y velocidad. Los servomotores para modelismo actuales (en su mayoría) utilizan sistemas digitales para el control, tomando la señal de realimentación mediante un *encoder* (sensor de posición hecho por un arreglo óptico y un disco con marcas cercanas al borde) o una entrada analógica (Barrientos, 2007).

Los sistemas de control de posición son ampliamente utilizados actualmente en la industria. Muchas fábricas en la actualidad utilizan robots que en su mayoría tienen actuadores eléctricos y entre ellos hay servomotores.

Actualmente los servomotores para aplicaciones en robótica son totalmente digitales salvo en casos educacionales donde pueden ser más ilustrativos y entendibles su estudio. Estos servomotores tienen ya un sistema embebido inmerso a base de un microcontrolador el cual se encarga de medir la posición y velocidad, ya sea mediante un *encoder* o por un convertidor analógico digital. Para la obtención de la variable de velocidad el sistema digital mide el tiempo que pasa entre una medición a la anterior, ya sea analógica o digital y finalmente divide la relación de la posición obtenida del *encoder* entre el tiempo de las mediciones. En cuanto a la alimentación del motor de DC, esta es alternada (en polaridad) para dirigir el giro en el sentido deseado y dado que el microcontrolador tiene una señal de salida de poca potencia, es necesario amplificarla para poder mover el motor. Así tenemos la siguiente ecuación que ejemplifica la relación entre ganancia y las señales de entrada y salida.

$$G = \frac{Hs}{Hi}$$

Donde “G” es la ganancia de amplificación dada la entrada “Hi” con la salida “Hs”.

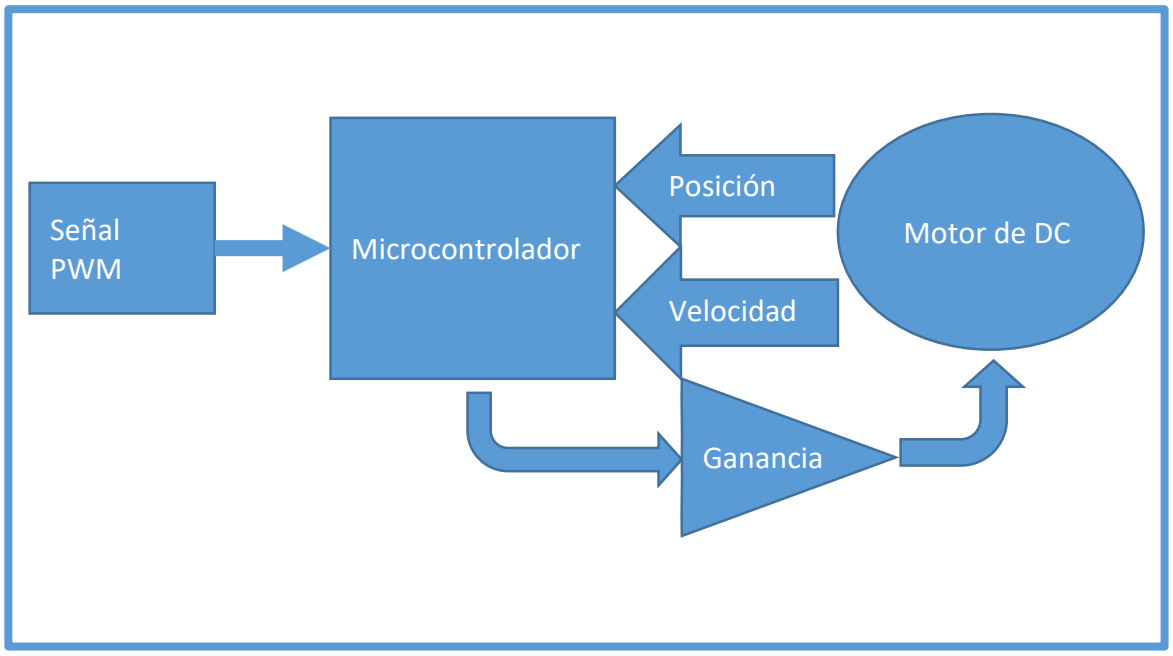


Figura 3-3Esquematico a bloques del control general de un servomotor digital

En el caso de este esquema tenemos una señal PWM la cual lleva un cierto ancho de pulso el cual está relacionado con la posición deseada, inmediatamente el microcontrolador, con base a la información que obtiene de los sensores de posición y velocidad (este es omitido en algunos casos) manda pulso los cuales pasaran a través de un amplificador para finalmente llegar al motor el cual cambia de posición, cambiando en consecuencia los valores de los sensores y provocando que el microcontrolador deba hacer un reajuste en la señal que llegara al motor a través del amplificador…; y esto continua hasta que el motor se encuentra en la posición deseada, la cual surge cuando el microcontrolador no debe hacer ningún reajuste.

3.2.1 Manejo de posición por PWM

Actualmente, existen servomotores que utilizan una señal digital PWM (*pulse-width modulation*) para que se les informe que posición tomar en cada momento. Entre otras aplicaciones que tienen estos están la robótica, el aeromodelismo y la automatización.

El protocolo de comunicación más usado para servomotores dedicados a la robótica es mediante una señal de onda cuadrada de longitud de pulso variable(PWM) a una frecuencia de 50[Hz] y cuyos rangos de operación usuales van desde ,5[ms] o 2,5[ms] de ancho de pulso (Tomasi, 2003).

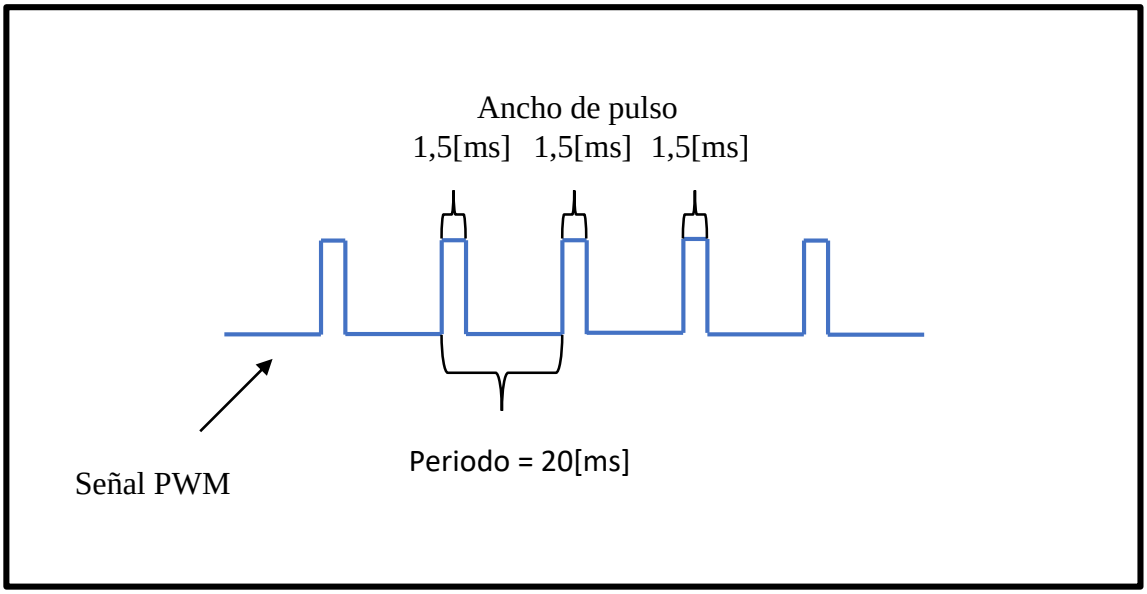


Figura 3-3 Señal PWM de ejemplo con ancho de pulso a 1500[us] para 90º en el motor HSR-5498SG (HITEC RCD USA, 2016)

El manejo de la posición de los motores del humanoide se realiza mediante una señal PWM como se muestra en la imagen anterior. En esta imagen se tiene un ancho de pulso de 1,5[ms] lo que equivale a 90º en el servomotor.

En la puesta en marcha, normalmente se trata de tener al robot fuera de los extremos, puesto que normalmente los servomotores para robótica tienen un limitador de giro por el cual, al estar cerca de los extremos, el motor junto con el conjunto de engranes tiende a esforzarse dando la posibilidad a desgastes innecesarios por parte del mecanismo.

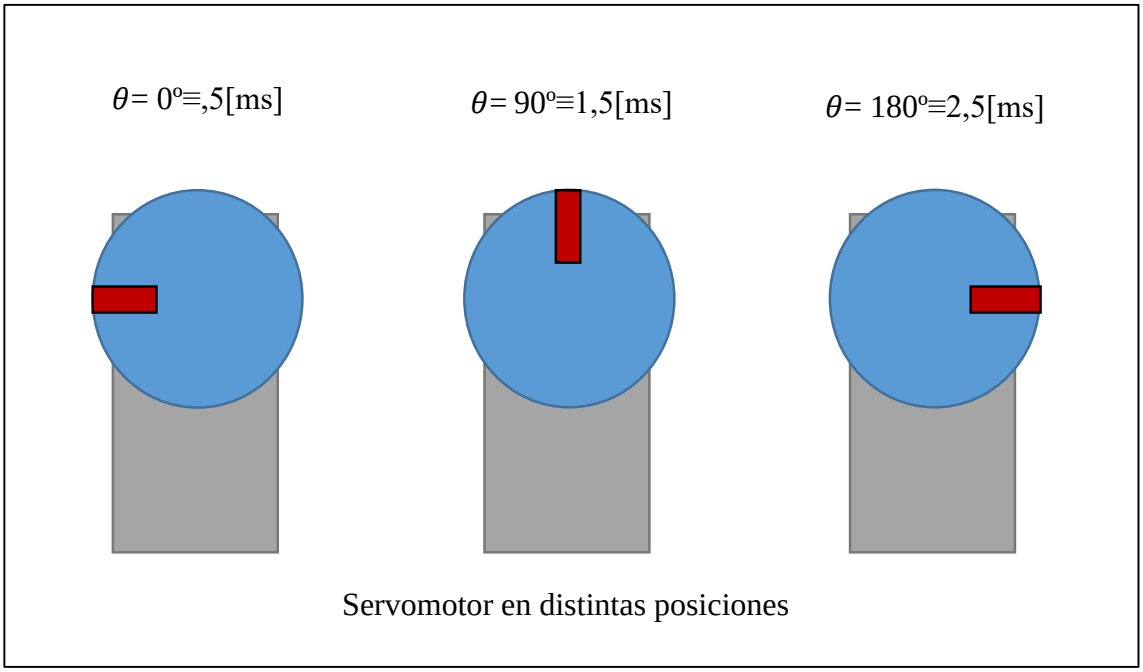


Figura 3-4 Servomotor en 0º, 90º y 180º

La correspondencia que tiene la señal PWM con el ángulo del motor es lineal y depende de la resolución de la señal PWM iniciándose con la marca de posición apuntando al lado izquierdo (viendo el motor con el disco de frente), tomando como referencia 5[ms] o cero grados 0º al servomotor visto de frente con el disco en su límite izquierdo. Siendo la resolución de la tarjeta controladora de motores de 2000 unidades (dos mil unidades), esta se puede mover con un paso mínimo de ,09º por cada unidad (Lynxmotion I. , 2005).

En cuando al manejo de ángulos directos se pueden obtener diferentes modelos dependiendo de la necesidad que se tenga, ya sea manejar solo ángulos positivos, en grados o radianes.

Modelo para radianes positivos

$$\theta[rad] = \pi * \frac{ancho - 500}{2000}$$

Modelo tomando como referencia de 0º (cero grados) en 1,5[ms]

$$\theta[º] = (180º) * \frac{ancho - 1500}{2000}$$

3.2.2 Tarjeta controladora de motores Lynxmotion SSC-32

En este proyecto de opto por utilizar la tarjeta SSC-32 de Lynxmotion, la razón principal por la cual se usó esta, fue por su buen funcionamiento en el mismo robot anteriormente a que se le instalara el sistema embebido presentado en este trabajo (PCDuino).

La tarjeta Lynxmotion SSC-32 es una solución integral para el control de múltiples servomotores puesto que genera hasta 32 PWM con salidas independientes. Esta pose un microcontrolador Atmel, modelo ATMEGA168-20PU programado con la plataforma de desarrollo de ARDUINO (Arduino, Arduino, 2016). En las siguientes ilustraciones se muestran las conexiones del microcontrolador con un registro de corrimiento (memoria electrónica que va moviendo bits en cierto sentido) explicando de manera muy general su funcionamiento.

Si bien el robot funcionaba anteriormente con la tarjeta SSC-32 y muchas de las conexiones y configuraciones estaban realizadas, estas no se encontraban documentadas. Por lo que en el presente apartado se pretende dar una breve introducción a lo que es el funcionamiento de la tarjeta SSC-32 y así, tener un mejor entendimiento en posteriores capítulos. Donde se realizará un análisis para la creación de librerías que faciliten su uso.

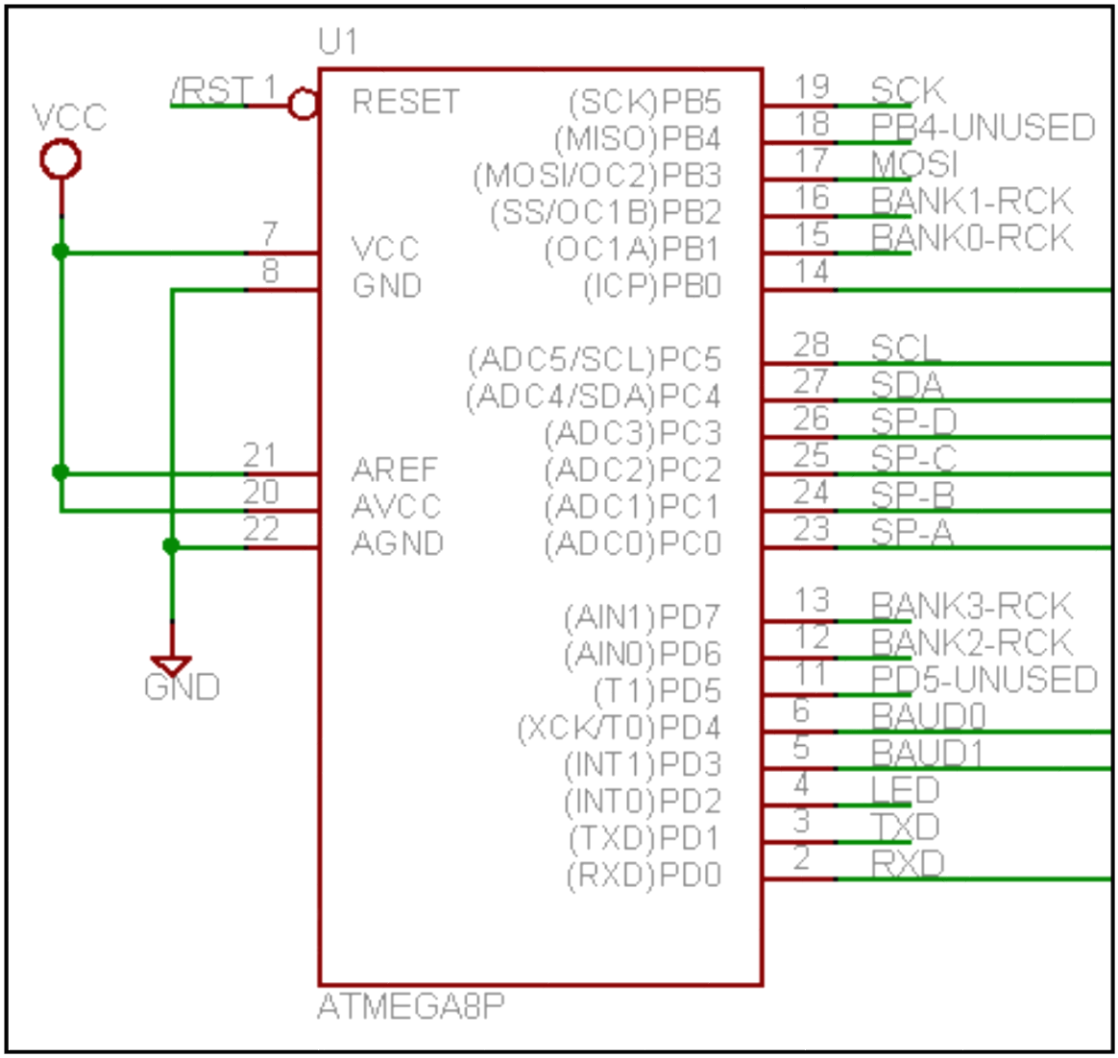


Figura 3-5 Vista de las conexiones con el microcontrolador usado en la ssc-32. Circuito de Mike Dvorsky (Tweed, 2016)

El proceso para poder mover un motor usado la tarjeta SSC-32 comienza cuando se manda un mensaje mediante UART el cual contiene información referente a la posición, velocidad y tiempo. Este mensaje es recibido por el microcontrolador (mostrado en la figura 2-6) y justo después procesada de tal manera que este produce señales digitales (cambios en el nivel de voltaje entre dos posibles valores, normalmente 0[V] y 5[V]) las cuales llegan a los registros de corrimiento 74HC595 (mostrados en la figura 2-7) mediante los pines MOSI, BANK#-RCK y SCK (Lynxmotion I. , 2005).

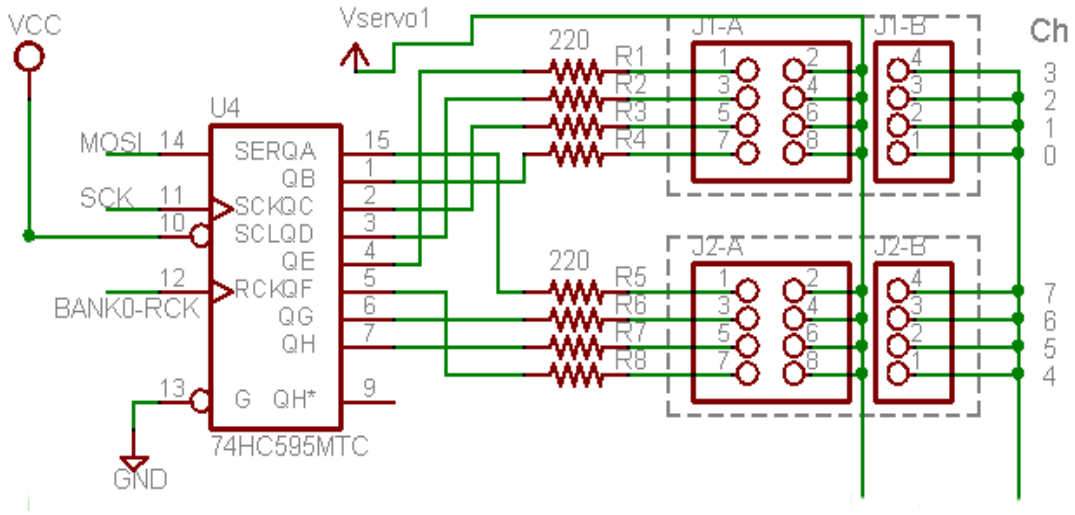


Figura 3-6 Vista las conexiones con el circuito de corrimiento 74HC595. Circuito de Mike Dvorsky (Tweed, 2016)

A los registros de corrimiento les llegan las tres señales descritas en el párrafo anterior, MOSI que es la entrada serial, SCK reloj de entrada para corrimiento del registro, BANK0-RCK entrada del reloj del registro de almacenamiento (semiconductors, 2015) (Tweed, 2016).

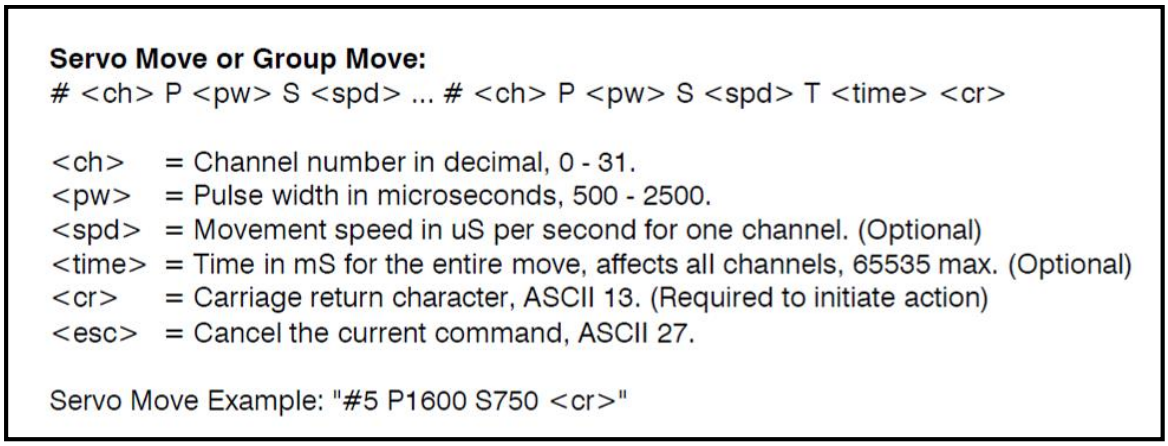
En conjunto el microcontrolador y los cuatro registros de corrimiento que posee el circuito, producen los 32 pulsos PWM mediante un programa que pone al alcance el fabricante Lynxmotion. Este programa puede ser descargado para la actualización de la tarjeta.

3.2.2.1 *Protocolo de comunicación de la tarjeta controladora de motores*

Un protocolo de comunicación es un conjunto de reglas que se deben seguir para lograr la transmisión de información entre receptor y emisor. En lo referente a las comunicaciones electrónicas, para lograr el intercambio de información dentro dos sistemas digitales es necesario seguir las mismas reglas y tener el hardware necesario para poder transmitir y recibir mensajes de manera adecuada (Tomasi, 2003).

En cuanto a esta tarjeta SSC-32, la comunicación es mediante el puerto UART y su protocolo comienza por una letra inicial que indica el tipo de valor a cargar en el dispositivo, seguido del valor que se le dará a tal para su posterior ejecución, toda esta información se encuentra en el manual de la tarjeta (Lynxmotion I. , 2005).

Existen diferentes instrucciones que se le pueden mandar a este dispositivo, las cuales están organizadas en 12 categorías, más para fines de este trabajo solo es necesario conocer la forma *Servo Move* o Move Grupo que es la siguiente:



Servo Move or Group Move:
<ch> P <pw> S <spd> ... # <ch> P <pw> S <spd> T <time> <cr>

<ch> = Channel number in decimal, 0 - 31.
<pw> = Pulse width in microseconds, 500 - 2500.
<spd> = Movement speed in uS per second for one channel. (Optional)
<time> = Time in mS for the entire move, affects all channels, 65535 max. (Optional)
<cr> = Carriage return character, ASCII 13. (Required to initiate action)
<esc> = Cancel the current command, ASCII 27.

Servo Move Example: "#5 P1600 S750 <cr>"

Figura 3-7 Sintaxis de las instrucciones Mover servomotor "Servo Move o Group Move" obtenido de (Lynxmotion I. , 2005, pág. 5)

En la figura 2-8 se muestra como están constituidas las instrucciones que permiten mover a los motores mediante el uso del puerto UART. Siendo así, el carácter '#' es el inicio que conforma una nueva orden para un motor, 'ch' es efectivamente el número del motor al que se le dará la orden, 'pw' es la longitud del pulso pwm en alto cuyo valor está en microsegundos y puede valer entre 500 y 2500 unidades (visto en el manejo de servomotores), 'spd' velocidad en uS por segundo (micro steps por su acrónimo en inglés), 'time' tiempo de traslado de la posición inicial a la posición final, 'cr' carácter de retorno referente a la codificación ascii, el cual se usa para iniciar las ordenes que se le han mandado al microcontrolador y finalmente 'esc' (también referente a la codificación ascii) que sirve para limpiar las ordenes que le han llegado al microcontrolador aún no ejecutadas (Organización Internacional para la Estandarización, 1975).

3.2.3 Servomotor HSR-5498SG

En este pequeño apartado se muestran las características de los servomotores HSR-5498SG los cuales se encuentran actualmente funcionando en el robot. Estos mismos motores se encontraban instalados antes de que se hiciera el cambio de sistema embebido y se decidió seguir usándolos puesto que fue la mejor solución encontrada al momento.

El robot usa servomotores del modelo HSR-5498SG marca HiTEC, se usó este modelo de servomotores debido a que el voltaje nominal de entrada corresponde al de la batería y la tarjeta controladora de motores puede utilizar el mismo voltaje de alimentación de la batería como la alimentación para los motores, de tal forma que los tres dispositivos pueden funcionar al mismo nivel de voltaje sin necesidad de agregar un circuito para acoplarlos. Este nivel de voltaje corresponde al máximo soportado por el servomotor y el nominal de la batería que es de 7.4 [V].

Condiciones operativas del servomotor HSR-5498SG	Valor mínimo	Valor máximo
Voltaje operativo	6 v	7.4v
Velocidad de operación	,22 seg/60º	,19 seg/60º
Torque	11 Kg*cm	13 Kg*cm
Corriente operativa sin carga	200mA	240mA
Corriente operativa con carga	1200mA	1480mA
Peso	59,8g	

Tabla 3-3 Tabla informativa de las características del servomotor HSR-5498SG (Lee, 2006) (HITEC RCD USA, 2016)

En la tabla 2-13 se muestran los rangos de operación de los servomotores descritos anteriormente, cabe destacar que estos valores se usaran en los capítulos siguientes para hacer un análisis de energía o de operación.

Adicionalmente cabe mencionar que, la información técnica encontrada antes de realizar este proyecto sobre el posicionamiento de los motores en la tarjeta controladora de motores era insuficiente. Por lo que dado el cambio de sistema embebido fue necesario generar nueva documentación, así como reacomodar la localización de los motores en otros puertos sobre la misma tarjeta ssc-32. Esta nueva información es usada en el presente trabajo y de igual manera, una documentación más técnica queda como respaldo con el equipo UANMoids.

3.3 Sensor de orientación

El humanoide cuenta con una pequeña tarjeta de color azul de 2 por 1.5[cm] en la cual se encuentran integrados el acelerómetro y un giroscopio, los cuales en conjunto pueden funcionar como un sensor de orientación, esta tarjeta se encuentra en la parte trasera del dorso, justo en la parte superior casi en el borde donde comienza la parte plana donde se coloca la cabeza. Se eligió esta disposición debido a dos principales razones, la primera es que es uno de los lugares mejor protegidos contra los impactos por caída, dado que se encuentra cubierta por una pequeña carcasa que funciona como tapa para las demás tarjetas, la segunda es debido a la altura, ya que, dependiendo de la separación entre el piso y el sensor, este puede captar diferentes cambios tanto en aceleración como en posición ya instalado en el robot (Electronics, 2017).

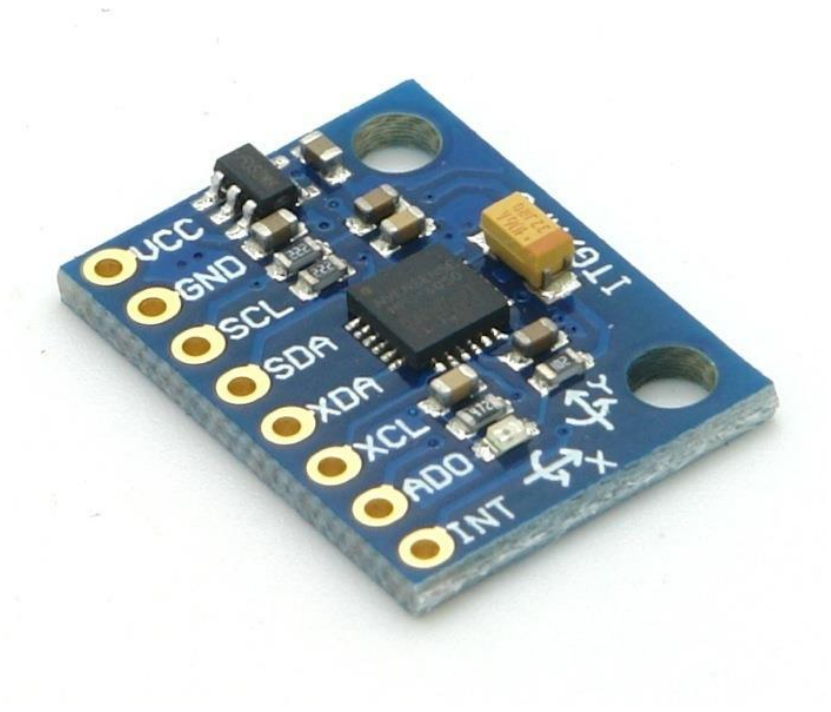


Ilustración 3-4 Giroscopio usado en el humanoide GY-521 (Electronics, 2017)

El uso masivo de giroscopios y acelerómetros ha ido creciendo en la última década con el desarrollo de los dispositivos móviles. Siendo así, a los dispositivos móviles se les han ido agregando diferentes características en las que se destacan aplicaciones para monitorear la actividad física como juegos o

ejercicios y son estas últimas las que requieren información sobre el movimiento o posición del dispositivo en cada momento. Estos nuevos dispositivos tienen poco menos de una década en el mercado y están basados en una tecnología relativamente nueva (a comparación de la mecánica o la metalurgia) llamada MEMS (micro electro mecanical systems) con los cuales es posible integrar en un solo chip, giroscopio, acelerómetro, magnetómetro y un pequeño circuito para procesamiento de posición, un ejemplo de este tipo de dispositivos es el MPU-6050 el cual es un chip de 4x4 mm (Inc. I. , PS-MPU-6000A, 2011).

3.3.1 Acelerómetro y giroscopio GY-521

El dispositivo GY-521 es un sensor de aceleración y giro que es parte de los proyectos arduino de software y hardware abierto, este sensor utiliza el circuito integrado MPU-6050 el cual a su vez tiene un MEMS inmerso en el silicio del circuito (Arduino, Arduino Playground, 2016). La comunicación de este sensor es a través de I2C a una frecuencia preconfigurada de 400 khz y su uso es mediante registros de configuración y medición que se encuentran en el mismo espacio de memoria los cuales son leídos o escritos mediante el mismo I^2C. Como la mayoría de proyectos de arduino este dispositivo lo fabrican varias compañías (Arduino, Arduino Playground, 2016) (Inc. I. , PS-MPU-6000A, 2011).

El dispositivo MPU-6050 cuenta con memoria interna de la cual podemos tener acceso a un direccionamiento de 128 bits (0x7F), en el que se encuentran los registros de configuración para el giroscopio, acelerómetro y magnetómetro (Inc. I. , RM-MPU-6000A, 2012). Los registros a tener en cuenta para este trabajo son solo los que se necesitan para poner a funcionar el dispositivo y los registros donde se encuentran almacenados los datos de cada medición, estos se nombraran con su dirección en hexadecimal.

Registro de energía

- 0x6B o PWR_MGMT_1: Registro que permite configurar el modo de funcionamiento del dispositivo. En el ejemplo para la toma de mediciones se deja el dispositivo haga mediciones continuamente

y use el reloj interno por lo que a este registro se le da el valor de 0x01.

Registros de mediciones: la información de cada uno de los ejes de dirección esta ordenada en 16bits, y debido a que en la memoria solo se puede escribir un byte a la vez (8bits) la información está organizada en pares. Si se recorriera la memoria secuencialmente de menor a mayor nos encontraríamos con el byte de mayor valor seguido del de menor valor.

- Acelerómetro
  - 0x3B y 0x3C: aceleración del eje X
  - 0x3D y 0x3E: aceleración del eje Y
  - 0x3F y 0x40: aceleración del eje Z
- Giroscopio
  - 0x43 y 0x44: giro del eje X
  - 0x45 y 0x46: giro del eje Y
  - 0x47 y 0x48: giro del eje Z

Un ejemplo de lo anterior seria el siguiente:

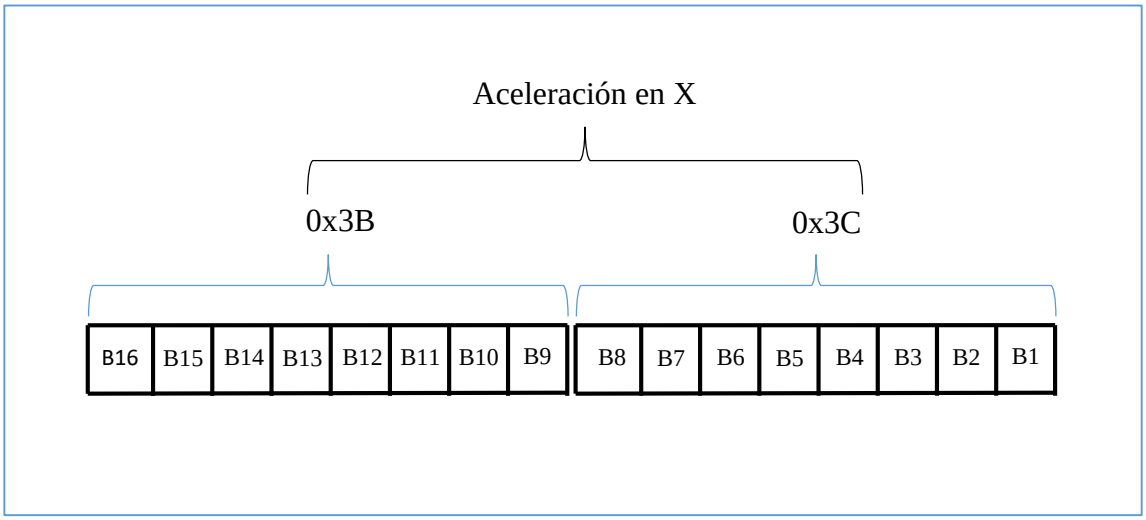


Figura 3-8 Ejemplo de la organización de la información en los registros del MPU-6050

En el cuadro anterior se puede apreciar cómo se encuentra organizada la información de la aceleración del eje X en los registros 0x3B y 0x3C con los valores del byte de mayor valor y el de menor, respectivamente.

Para la implementación para obtener el valor completo de la medición del acelerómetro es necesario leer directamente de los registros la información y una vez obtenida hacer una transformación para juntar ambos bytes y desplegarla en el formato con el cual se deseen hacer las operaciones requeridas.

Cabe destacar que existen más registros que pueden ser utilizados para configurar algunas características como parámetros de la comunicación I^2C, sensibilidad del acelerómetro o giroscopio, habilitar características adicionales de procesamiento, e incluso hacer *Self-Test* (auto pruebas) para saber sobre el correcto funcionamiento del dispositivo.

3.4 Cámara

En las últimas décadas gracias al avance de la electrónica el poder adquirir una cámara digital para tomar fotos y video se ha hecho cada vez fácil para cualquier persona, siendo así actualmente muchos de los dispositivos electrónicos móviles tienen integrada una cámara digital cuyo tamaño, en la mayoría de los casos se aproxima al tamaño de la uña del meñique.

Dado el fácil acceso a estas cámaras que puede ser adquiridas en un supermercado o un establecimiento de venta de dispositivos electrónicos se optó, por usar una cámara web con algunas modificaciones estructurales para poder anclarla al robot.

La cámara cuenta con una conexión USB *plug and play* (conecte y use), la cual se conecta a uno de los dos puertos de la tarjeta PCDuino mediante su propio cable (integrado al dispositivo). Una vez conectada la cámara a la tarjeta, este se le puede hallar dentro de la carpeta "/dev" con el nombre video0 o video1 dependiendo el número de cámara conectadas.

Cabe mencionar que esta cámara no cuenta con ningún tipo de arreglo óptico adicional a lo que viene por defecto. Además de que, sus modificaciones estructurales son únicamente por fines estéticos o mecánicos (evitar golpes).

3.5 Sistema operativo de la PCDuino

Como anteriormente es mencionado, el sistema embebido PCDuino trabaja por defecto con Ubuntu por lo que en el presente apartado se muestra

información de dicho sistema operativo, así como una breve descripción de su vinculación interna con los dispositivos que se le conectaron para el desarrollo de este proyecto.

Un sistema operativo es un conjunto de programas que ayudan a la administración del hardware de un sistema. Se han creado diferentes sistemas operativos para las computadoras modernas como Linux, Mac y Windows, por nombrar a los más usados. Estos sistemas han ido cambiando con el tiempo haciéndose cada vez más simples para el usuario final pero más sofisticado en cuanto a programación.

Linux es un sistema operativo monolítico diseñado aproximadamente en el año 1992 principalmente por Linus Torvals. Este núcleo (*kernel* en inglés) es un conjunto de rutinas que ayudan a la mejor manipulación del hardware (Welsh, 2009).

Entre otras cosas importantes Linux ha sido adoptado por otros diseñadores de sistemas operativos con diferentes propósitos. Desde el usuario final hasta las grandes empresas con grandes demandas de cómputo, pasando por estudiantes; A este núcleo se le han agregado programas y utilidades diversas como el *X Windows system* que es un programa para poder visualizar gráficos y no solo texto en una pantalla. A este conjunto de utilidades específicas para cada junto con el Kernel Linux y las utilidades GNU se les llama distribución (Welsh, 2009).

Cada distribución basada en Linux es diseñada con diferentes objetivos, como podría ser el uso como servidor web, diseño multimedia, centro de entretenimiento, experimentación científica o desarrollo de sistemas embebidos. Entre otras las distribuciones más conocidas son Debian, Ubuntu y sus diferentes versiones como Mandriva, Fedora, Red Hat Linux y Arch Linux de las cuales no todas implementan la filosofía de software libre (Vaduva, 2015) , (Welsh, 2009).

Una de las distribuciones más desarrolladas de Linux es Ubuntu, de la cual existen diferentes versiones orientadas a diferentes usuarios. Entre otras se

encuentran Xubuntu y Lubuntu pero, Ubutu es la distribución principal y es encauzada a usuarios finales, esta tiene un constante cambio en cuanto a las utilidades extras que se le agregan al sistema. Este sistema operativo se divide en dos tipos de versiones: las de soporte prolongado (Long Support) liberada cada dos años y con un soporte promedio de tres años y las normales que son liberadas cada seis meses con un soporte promedio de nueve meses. Ambas versiones son liberadas cada año los meses, cuarto y decimo que corresponden a abril y octubre (Canonical, 2015).

3.5.1 Sistema de archivos de Ubuntu

En Linux todos los dispositivos, carpetas y archivos son tratados como estos últimos lo cual significa que incluso las mismas carpetas y dispositivos tienen usuario, grupo y permisos de lectura, escritura y ejecución. Ubuntu por ser una distribución basada en GNU/Linux tiene exactamente las mismas reglas.

Todos los directorios o carpetas (como se les conoce en Windows) parten, de la raíz que se denota con el símbolo "/", a partir de la raíz surgen las carpetas del sistema, el usuario, dispositivos, carpetas con archivos de configuración, librerías o archivos de arranque.

Cabe destacar algunos directorios que se encuentran en la raíz del sistema, puesto que se trabajará directamente sobre ellos posteriormente y estos son:

1.- /dev: directorio donde se encuentran los periféricos como son el puerto I2C, UART y USB.

2.- /etc: directorio donde se encuentran activos de configuración del sistema los cuales se usan para configurar las conexiones LAN o WLAN.

3.- /home: directorio donde se encuentran los datos del usuario *ubuntu.*

4.- /media: directorio donde se ubican los dispositivos montados como memorias o unidades de disco virtual.

5.- /root: directorio raíz del administrador del sistema operativo o mejor conocido entre los usuarios de Linux como root del sistema.

El directorio donde se trabajaron todos los archivos (códigos y compilaciones) fue el de home únicamente por comodidad más pudo ser cualquier otro (GNU, BASH command line, 2016).

3.5.2 GY-521 en ubuntu

Como se vio anteriormente el giroscopio y acelerómetro usando en este trabajo es el GY-521, este dispositivo utiliza la comunicación electrónica I^2C y al ser conectado a la PCDuino, el sistema operativo genera en automático un puerto de comunicación el cual puede ser accedido desde línea de comandos o por un programa (visto más adelante). En este apartado se describe la relación que se tiene entre dicho dispositivo físico para con el sistema operativo del sistema embebido.

Al igual que con la comunicación UART para la tarjeta controladora de motores, el sistema operativo Linux detecta el puerto de comunicación I^2C el cual puede encontrarse en la misma carpeta donde se encuentran los demás dispositivos, llamada "dev" ubicada en la raíz del sistema. Este dispositivo lo podemos encontrar con el nombre de "i2c-X" donde X representa un número.

A diferencia del puerto UART, donde desde línea de comandos era posible tener comunicación a través del puerto, en el caso de I^2C es necesario tener una utilidad extra la cual permita ver los dispositivos conectados, escribir o leer de los registros. Esta herramienta puede ser incorporada al sistema operativo desde los repositorios oficiales de ubuntu, esta utilidad se llama "i2c-tools" (Frodo Looijard, 2016).

Usando la utilidad anteriormente mencionada es posible hacer un escaneo de dispositivos en el canal de comunicación que reconoce el sistema operativo además de leer y escribir en los registros de cualquier dispositivo encontrados en el mismo.

Además de la utilidad anteriormente mencionada existen librerías en el lenguaje C y C++ que permiten el uso del puerto I^2C sin necesidad de recurrir a la línea de comandos, de tal forma que desde un mismo programa es posible hacer las configuraciones necesarias, como, por ejemplo, leer y escribir sobre los

registros del GY-521. Esta librería se puede instalar desde los repositorios de ubuntu y para agregarla a un programa solo es necesario incluirla en el código.

4 Desarrollo e implementación

En este capítulo se realizan análisis sobre el mejor funcionamiento de los dispositivos, dado su configuración en la conexión con las baterías. Además, se habla sobre algunas consideraciones que se debieron tomar en la programación la cual está orientada al manejo de la información referente al uso de las tarjetas conectadas al sistema embebido (cámara, tarjeta controladora de motores y sensor de aceleración y giro).

Este capítulo a su vez se divide en dos partes, la primera enfocada en la solución encontrada para la distribución de energía proveniente de las baterías, que termina en las tarjetas, y la segunda se centra en la estructura que tienen las librerías para el control de motores y adquisición de mediciones del sensor de orientación.

Para el desarrollo de este proyecto se usó acceso remoto al sistema operativo del sistema embebido mediante Secure Shell (visto más adelante), esto es conveniente tenerlo en cuenta puesto que el tener cables conectados al robot a la hora de hacer pruebas se pueden ver afectados los resultados; como por ejemplo mantener el robot de una posición sin que este caiga a uno de sus costados o hacer mediciones del giroscopio puesto que se pueden filtrar señales mecánicas dada la atención de los cables conectados. Por lo mismo el acceso mediante SSH al PCDuino es una parte clave en el desarrollo de este proyecto dado que, a diferencia de la conexión con el teclado y el mouse, esta puede mantenerse en una red local logrando la movilidad completa del robot (sin impedimentos físicos) y al mismo tiempo monitorear vía remota el comportamiento de todo el sistema operativo en tiempo real.

Secure Shell es una utilidad de internet de software libre que comenzó con la utilidad ssh2 y mucho antes de eso en los comienzos de internet por Telnet diseñado para la red ARPAnet. La idea de SSH es el control remoto de un ordenador a través de una red TCP/IP aplicando algoritmos de encriptación y la

técnica de túnel o *tunneling* para lograr una conexión segura entre el cliente y el servidor (Red Hat, 2016).

4.1 Alimentación eléctrica de circuitos y cálculo de consumo de energía

La alimentación de los circuitos es un tema crucial para los proyectos de robótica debido a que actualmente los sistemas de almacenamiento de energía como las baterías no tienen las características adecuadas para mantener funcionando sin dificultades y por largos periodos de tiempo todos los sistemas necesarios. Así robots como Darwin-OP tienen un tiempo aproximado de operación máximo de 30 minutos (ROBOTIS, TechSupport_ENG, 2010) o ASIMO del corporativo Honda tiene un tiempo operativo cercano a una hora (Inc. A. H., 2016). Por esto el tema de la energía es de suma importancia además del análisis de consumo para los circuitos y actuadores.

4.1.1 Análisis teórico de consumo de energía

En este apartado se hace un análisis sobre el consumo de energía de los componentes proponiendo una forma de conexión optima que asegure el funcionamiento del robot sin poner en riesgo sus componentes ni la programación sobre el sistema embebido.

La cantidad de energía que un robot consume está en función de la suma del consumo de cada uno de sus componentes en cada momento, los cuales tienen variaciones dependiendo de su función, siendo esto así, el mejor análisis que se pudo realizar para saber que todo el sistema se comportará de manera adecuada fue hecho estimando los valores máximos y mínimos de energía.

Para calcular la corriente máxima es posible suponer que el robot se encuentra usando todos los servomotores en su carga máxima, y además las tarjetas, tanto la SSC-32 como la del sistema operativo PCDuino, se encuentran requiriendo la corriente máxima de operación.

Numero de motores	Consumo mínimo total [A]	Corriente máxima total [A]
1	0,2	1,48
21	4,2	31,08

Tabla 4-1Calculos de la Corriente utilizada por los servomotores, comparativa entre uno y los veintiún servomotores del robot (HITEC RCD USA, 2016)

En la tabla 2-4 es posible apreciar los consumos mínimos y máximos de uno y veintiún motores, esta información se puso de manifiesto de tal manera que el cálculo concordara con la cantidad de actuadores que tiene instalado el robot. Así también se tiene la comparativa entre la corriente que podrían llegar a necesitar los servomotores en su estado de reposo y de carga máxima (mínimo y máximo consumo de corriente respectivamente).

Tarjeta	Corriente máxima [mA]
PCDuino	2000,0
SSC-32	31,0
Total	2031,0

Tabla 4-2 Calculo de la corriente para las tarjetas de desarrollo (Lynxmotion, 2015) (DFROBOT, 2016)

De igual manera que en el párrafo anterior, en la tabla 2-5 se puede observar la corriente máxima de operación de la tarjeta PCDuino y de la tarjeta controladora de motores SSC-32 Agregando, además, el cálculo de la suma de estos valores lo que da como referencia la corriente máxima total que requieren las tarjetas para un funcionamiento confiable, esta se pone de manifiesto para su uso en los cálculos consecuentes.

Mencionado lo anterior, se sigue con el cálculo del valor máximo de corriente. Siendo así, el valor máximo de consumo que requerirá el robot será la suma de los veintiún motores con los valores nominales de consumo de las tarjetas (realizado anteriormente):

$$31,08 + 2,031 = 33,111 \text{ [A]}$$

Por el otro lado, para el cálculo del consumo mínimo de corriente con el cual el sistema se encontraría funcionando de manera estable seria, con la suma de la corriente mínima de los veintiún servomotores y de igual manera los valores nominales del consumo de las tarjetas:

$$4{,}2 + 2{,}031 = 6{,}231 \text{ [A]}$$

Cabe aclarar un par de cosas, la primera es que se toma el valor nominal de las tarjetas dado a que estas nunca se deben queda sin energía, de ser así podría haber problemas en la ejecución de las instrucciones en el caso del controlador y perdida de información en el sistema operativo (visto en la experimentación), la segunda es que los dispositivos periféricos como la cámara USB, el giroscopio y los periféricos de entrada para el sistema operativo están contemplados dentro de la corriente máxima de la PCDuino.

En cuanto al funcionamiento en conjunto del robot, un punto importante a destacar es que, de nada sirve tener funcionando la tarjeta SSC-32 si los motores no tienen la energía suficiente para funcionar y esto lo podemos verificar planteando el siguiente escenario.

1- Los servomotores y tarjetas están conectadas a la misma batería
2- La batería está en el punto de carga baja

Es de imaginarse que cada uno de los componentes dependiendo de la energía que requieran empezará a dejar de funcionar, debido a la constante demanda de corriente del sistema en conjunto, siendo los motores los primeros en no responder adecuadamente y seguidos de estos las tarjetas (puesto su alto consumo de energía en comparación con las tarjetas). Por otra parte, si no se tuviera la suficiente corriente para alimentar a la PCDuino, esta puede tener perdida de datos.

Siendo así, con la información y el planteamiento anterior, se concluye que es importante que la tarjeta PCDuino tenga su propia fuente de energía.

Las principales ventajas que se tiene al tener este diseño en las conexiones de alimentación son las siguientes:

1- Si la batería de los servomotores se queda sin energía, la tarjeta del sistema operativo sigue funcionando por lo que no se arriesga la información.
2- Si existe algún problema como, extremidades encontradas durante el movimiento o que algunas de las mismas se llegasen a atorar, es posible desconectar la batería de los servomotores de manera manual para evitar daños en los circuitos o mecanismos.
3- Si la batería de los servomotores se queda sin carga, no se necesita apagar el sistema operativo para cambiar la batería. Esto es sumamente útil en operación.

4.1.2 Selección de baterías

Así como se vio anteriormente, el asegurar el suministro de energía a todos los circuitos del robot es un tema crucial, y es por esto que en este apartado se justifica el uso de baterías de Litio Polímero mediante la comparación entre las tecnologías más comunes a las que se tiene acceso.

Existen muchos tipos de baterías o también llamados acumuladores, con diferentes características como la relación energía-peso que se refiere a la cantidad de energía que una batería puede almacenar dado una cantidad de masa, el tiempo de carga, la constante de carga-descarga, el número de recargas y el porcentaje de descarga cada cierto tiempo.

En la tabla 4-1 se exponen los parámetros de mayor interés para este trabajo, como son la relación energía peso o también conocida como energía específica, el tiempo de carga y el voltaje por celda (referido al voltaje que existe entre las celdas de las cuales está compuesta la batería).

Material Base	Relación energía peso(teórico)[Wh/kg]	Tiempo de carga [h]	Voltaje por celda (circuito abierto)[V]
Plomo-acido	252	8 a 24	2.1
Niquel-cadmio	244	1 a 16	1.35
Litio-polímero	400	3	3 a 4

Ilustración 4-1 Tabla comparativa de baterías por material de construcción (David Linden, 2002)

La unidad watt entre hora ([Wh/kg]) puede desorientar en ocasiones dado que la mayoría de las veces las baterías vienen con una característica

impresa en el empaque que se encuentra en mili ampers hora ([mAh]) para hacer la conversión de unidades solo se debe recordar que:

1 [Ah] * 1 [V] = 1 [VA/h] = 1 [Wh]

En la tabla 4-1, en principio puede notarse una diferencia clara en el tiempo de carga, siendo la tecnología acido-plomo la que más tiempo puede llegar a tomar, en contra parte con la tecnología de níquel-cadmio cuyo tiempo puede ser de tan solo una hora. Esta última tecnología, si bien puede llegar a tener el mejor tiempo de carga, también tiene la menor relación energía peso (o energía especifica) de las 3 tecnologías. Por ultimo las baterías de litio-polímero a pesar de tener un tiempo de carga mayor que las de níquel-cadmio, posen 1,63 veces más energía especifica.

Así entonces las baterías de iones de litio polímero son la mejor solución hasta el momento, para la alimentación de los circuitos del robot.

En la tabla 4-3 se encuentran las características de las baterías que utiliza el robot las cuales son de la marca KingMax, algunos datos a tener en cuenta son la corriente de carga, la constante de carga-descarga, el tiempo de carga y la corriente máxima de descarga puesto que de estos parámetros depende el tiempo que funcionara el robot además de proporcionar información que permite saber si se alimentaran adecuadamente los motores lo cual garantice su correcto funcionamiento.

Batería	Iones de Litio polímero
Voltaje	7.4
Capacidad	2200mAh
Carga/Descarga	30C
Conector de carga	JST-XH
Conector de descarga	Dean's Conector
Peso	206 [g]

Tabla 4-3 Características de las baterías de iones de litio polímero (SparkFun, 2016)

La corriente de carga depende de la capacidad de la batería y el tiempo que se necesita cargar la batería. Para el caso de las baterías de litio el tiempo de carga ronda entre una y cuatro horas (1 a 4). Tomando el tiempo mínimo de carga que es una hora y la capacitada de la batería podemos deducir cual es la corriente máxima de carga de la batería:

Corriente de carga, CC o "charge current"

$$I(carga) = \frac{Capacidad}{1000 * Tiempo} = \frac{2200[\text{mAh}]}{1000 * 1[\text{h}]} = 2{,}2[\text{A}]$$

Esto significa que la batería se tardará una hora en cargar a una corriente constante de 2,2[A], los fabricantes de baterías recomiendan no usar el límite de corriente especificado para carga debido a efectos no deseados que van desde el deterioro en la capacidad de la batería hasta ignición debido a la alta temperatura que alcanza.

Por otra parte, para conocer la corriente máxima de descarga de la batería basta con multiplicar la corriente máxima de carga por la constante de carga-descarga.

Corriente de descarga

$$Imax = C * I(carga) = 30 * 2{,}2[A] = 66[A]\ ^{1}$$

Es importante mencionar que al igual que para la corriente de carga no se recomienda usar el valor máximo de descarga, los fabricantes recomiendan usar de diez a quince (10 a 15) veces menos el valor de la constante de carga-descarga para prolongar la vida de la batería por lo que el valor utilizable para no afectar a la batería es el siguiente:

$$I(preferible) = C * I(carga) = 15 * 2{,}2[A] = 33[A]$$

Haciendo caso a la disposición mencionada en el apartado anterior de separar los motores y la tarjeta ssc32 de la tarjeta PCDuino, es conveniente comparar la corriente que requieren los motores contra la corriente que la

[1]Las ecuaciones pueden ser deducidas de (M.R. Jongerden and B.R. Haverkort, s.f.)

batería puede suministrar sin esforzarse tomando en cuenta el caso extremo donde se requiere la mayor cantidad de corriente.

$$I(SSC\ y\ motores) = 31{,}08[\mathrm{A}] + 31[\mathrm{ma}] = 31{,}111[A]$$

Así podemos deducir que esta batería podrá mantener trabajando el sistema sin caídas de voltaje considerables.

Por último, es importante conocer el tiempo de funcionamiento el cual se puede deducir de la misma manera con la corriente máxima y la capacidad de la batería suponiendo de igual manera que está cargada al máximo.

$$Capacidad = Tiempo * Corriente = 2{,}2[Ah]$$

Suponiendo la peor y la mejor de las condiciones de trabajo con los motores se puede deducir el tiempo mínimo y máximo de trabajo para el robot:

$$Tiempo(Min) = \frac{\text{Capacidad}}{\text{Corriente}} = \frac{2{,}2[Ah]}{31{,}111[A]} = 4{,}24[min]$$

$$Tiempo(Max) = \frac{\text{Capacidad}}{\text{Corriente}} = \frac{2{,}2[\mathrm{Ah}]}{4{,}231[\mathrm{A}]} = 31{,}19[min]$$

Por lo que el rango de tiempo operativo de los motores estará entre 4,24[min] 31,19[min]. Este rango de tiempo es adecuado para realizar pruebas que no sean exhaustivas.

4.1.3 Fuente conmutada tipo Buck

La electrónica de potencia ha dado un salto enorme gracias a la electrónica de estado sólido (transistores entre otros dispositivos a base de cilicio). En los últimos años han cambiado muchos dispositivos a partir de estas tecnologías un par de ejemplos claros es el uso de fuentes conmutadas en lugar de transformadores con rectificadores para cargar dispositivos móviles o el uso de fuentes conmutadas para transformar energía de corriente directa a corriente alterna para el uso de paneles fotovoltaicos (Blanuša, 2015) (Silva, 2015).

Dada la separación propuesta anteriormente de los circuitos de potencia (servomotores y SSC-32) y los de lógica (PCDuino, cámara y sensor de orientación), fue necesario reconectar las diferentes partes del robot con las

baterías correspondientes. Pero en el caso de la tarjeta PCDuino, además, fue necesario realizar un circuito intermediario el cual se encargará de acoplar el voltaje de la batería con valor nominal del sistema embebido.

Las posibles soluciones conocidas para acoplar una batería de 7,4 volts con la tarjeta PCDuino que necesita una alimentación de 5 volts son mediante una resistencia, un regulador o una fuente conmutada. Por parte del acoplamiento de voltajes con una resistencia, este no sería posible dado que la carga que representa la tarjeta PCDuino no es constante y las variaciones de voltaje no permitirían su funcionamiento además de ser una manera poco eficiente del uso de la energía (dado el efecto joule de la resistencia). Con un regulador como el LM7805 no sería posible dado principalmente a dos cuestiones, que su corriente máxima de salida es de 1,5[A] y que los reguladores de potencia se calientan de tal manera que se tendría un problema extra al buscar la solución de la disipación adecuada de la energía (Texas Instruments Incorporated). Por lo que la mejor solución que se encontró para este problema fue una fuente conmutada tipo Buck “Step-Down” capas de alimentar con hasta 3 [A].

Actualmente existen fuentes conmutadas prediseñadas por distintos fabricantes entre ellos Texas Instruments, para voltajes comúnmente usados en la electrónica digital por lo que para acelerar el desarrollo se utilizó la fuente conmutada LM2576 del mismo fabricante Texas Instruments (Instruments, 2015).

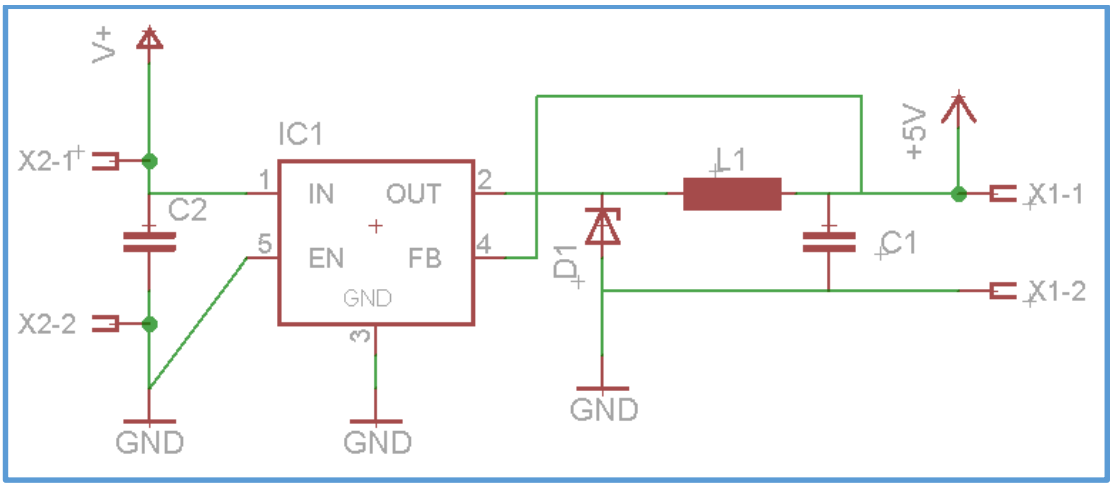


Figura 4-1 Esquemático de la fuente Buck diseñado a partir de (Instruments, 2015, pág. 1) en Eagle PCB

En la figura 2-10 se muestra el esquemático del circuito usado en la fuente Buck que usa el robot, este circuito así como la lista de valores para su construcción se encuentran en un apartado llamado "*Typical Application*" (aplicación típica) en la misma *datasheet* (hoja de datos del fabricante).

Nombre	Componente	Valor
IC1	LM2575	
D1	1N5822	
C1	Cap. Electrolítico	1000 [uF]
C2	Cap. Electrolítico	100 [uF]
L1	Inductor Comercial Modificado	100 [mH]
X1 y X2	Headers tamaño estándar	

Tabla 4-4Lista de los componentes utilizados para la construcción de la fuente Buck

En la figura 2-7 se muestra la lista de componentes usados en la fuente Buck, es notable que en la lista se encuentra un diodo con número de serie 1N5822, este también es especificado en el *datasheet* por el fabricante.

Así teniendo la fuente conmutada solo falta analizar el tiempo de funcionamiento de la tarjeta PCDuino. Para calcular el tiempo idealmente se omitirá la corriente que es usada por la fuente Buck debido a que ronda entre

los 50 [uA] y unos cuantos miliampers y esto no afectan sustancialmente el tiempo de funcionamiento.

$$Tiempo(pcDuino) = \frac{\text{Capacidad}}{\text{Corriente}} = \frac{2{,}2[\text{Ah}]}{2[\text{A}]} = 66[min]$$

Para terminar, si bien el tiempo teórico de operación calculado para la PCDuino es un numero constante, se debe tener en cuenta que existen variaciones en el consumo de corriente de este sistema embebido. Estas variaciones entre otras cosas dependen de los dispositivos conectados, como los sensores y periféricos. Por lo cual, dado que se usó el valor nominal para dicho calculo y que este a su vez representa el máximo posible, en practica, la corriente demandada es menor a la nominal y en consecuencia el tiempo de operación debe ser mayor a los sesenta seis minutos.

4.2 Análisis e implementación del software

Para el desarrollo de la puesta en marcha del robot se programó todo en los lenguajes "C" y C++ debido a que son los lenguajes con los cuales estoy más familiarizado, además de tener mayor documentación disponible. Por otra parte, si bien la eficiencia de un programa no depende totalmente de su lenguaje de programación, este, es un buen factor a considerar por lo que, buscando comparar este lenguaje con otros como java, los códigos escritos en "C" y C++, ofrece un desempeño promedio mayor en velocidad y en ahorro de memoria total ocupada. Estos lenguajes además cuentan con optimización extra directa del compilador (configurable) que, aunque no parece ser mucho, se consideró debido a la cantidad de cálculos que se necesitan para realizar en la visión artificial (wordpress, 2012). Al contrario, como java con C++, el lenguaje de programación C ofrece una mayor optimización para cálculos, más su implementación para tipos de datos complejos es más complicada en la programación. Por lo que respecto a visión o el middleware para comunicación con la tarjeta ssc32 se utilizara el lenguaje C++ (wordpress, 2012).

4.2.1 Análisis de la generación de comandos de posicionamiento de motores

Parte de este trabajo es poner en funcionamiento los motores del robot mediante el uso de un sistema embebido. Si bien, como se vio en el capítulo anterior, es posible mover motores desde línea de comando usando directamente el puesto UART, únicamente interconectando la PCDuino con la tarjeta controladora de motores y su correspondiente alimentación, el manejo simultaneo de todos los actuadores al mismo tiempo no es práctico además de ser inadecuado para hacer pruebas. Por lo que, se propuso automatizar la generación de instrucciones que le llegan a la SSC-32 de tal manera que sea posible su uso en otras aplicaciones además de las pruebas de este trabajo.

Las instrucciones que necesita la tarjeta SSC-32 para mover los motores están compuestas de cuatro variables (visto en el protocolo de comunicación de la tarjeta SSC-32), estas las podemos apreciar junto con una breve descripción en el siguiente listado.

1. Canal: número de motor a mover.
2. Posición: posición en la que se quiere posicionar el motor.
3. Velocidad: velocidad con la que se debe mover el motor.
4. Tiempo: tiempo que durara el movimiento.

Estos cuatro valores se encuentran acotados respecto a las limitaciones físicas y de diseño, por ejemplo; por diseño solo es posible mover 32 motores por lo que el número de canales está limitado a un rango de números enteros comenzando por cero y terminando en treinta y uno. Estos límites pueden ser encontrados en el manual de uso de la tarjeta SSC-32 (Lynxmotion I. , 2005).

Otra característica importante a tener en cuenta son los caracteres que preceden y proceden a los números anteriormente mencionados. Por diseño se notan dos reglas en la forma de las instrucciones, la primera es que el valor a mandar debe tener como carácter principal una letra referente a su utilidad (por ejemplo “P” para posición), la segunda es que se deben dejar caracteres de espacio entre cada una de las partes que componen el mensaje que se mandan (por ejemplo “..P2000 T100…”). Además de lo anterior, dado que la comunicación con la tarjeta SSC-32 utiliza código ASCII, es necesario agregar un

carácter de retorno (Lynxmotion I. , 2005) (Organización Internacional para la Estandarización, 1975).

Finalmente, teniendo en cuenta lo anterior es posible concatenar (juntar cadenas de caracteres) las instrucciones de varios motores para mandarlos secuencialmente desde cualquier dispositivo sin tener mayor margen de error que la propia interconexión física entre las tarjetas.

4.2.2 Librería en el lenguaje C++ para control de motores

Haciendo caso del apartado anterior, se siguió con la programación de la librería para la automatización de las instrucciones. La estructura de esta utilidad tomo en cuenta la forma de las instrucciones a nivel ASCII, la confiabilidad en la acotación de los datos antes de ser mandados, la transición y recepción automatizada de la información y la flexibilidad para su futura modificación, readaptación o reconfiguración (del código).

Toda la librería está escrita en C++, además de las razones mencionadas anteriormente, esto fue debido en gran medida a la ventaja de poder usar listas de cadenas de caracteres. Esto equivaldría en el lenguaje de programación “C” a hacer varios conjuntos de arreglos estáticos los cuales habría que calcularse por medio de su uso en memoria. Además de la ventaja anteriormente mencionada con el lenguaje C++, este, al ser orientado a objetos permite poder tener clases con las cuales es posible manejar información de una manera más ordenada y comprensible para futuros usos.

Además de las características ya mencionadas tomadas en cuenta para el diseño de esta clase, se agregó un método llamado sscLog, el cual solo es necesario para esta misma utilidad, este, se encarga de desplegar errores comunes en el uso de la misma clase. Un ejemplo en donde se puede ver su uso es en el chequeo de los límites de los valores que se quieren insertar en una nueva instrucción.

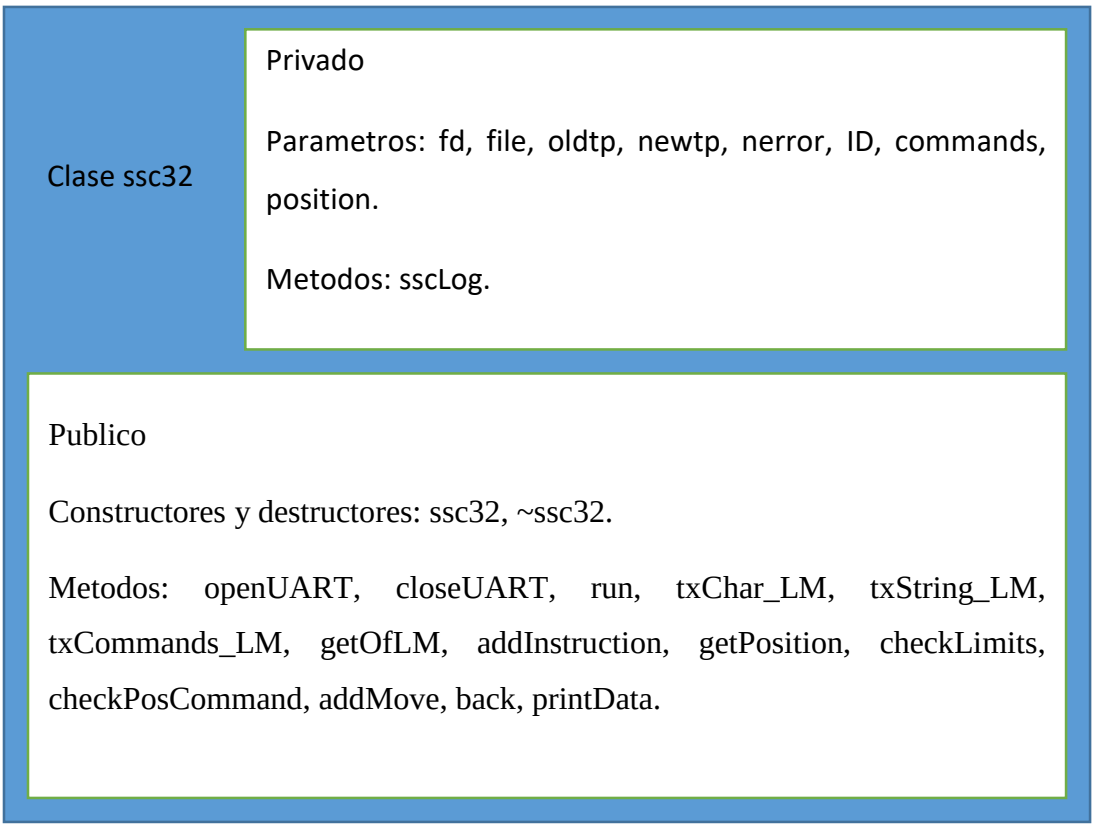


Ilustración 4-2Estructura de la clase ssc32

En la ilustración 4-1 se muestra como está organizada la clase ssc32. Es posible ver que hay dos bloques los cuales representan a las variables y métodos públicos y privados. La diferencia entre estos es que los elementos públicos pueden ser accedidos desde otro programa mientras que los privados solo pueden ser usados por la misma clase, permitiendo así, organizar o aislar la información entre clases.

Todos los parámetros de la clase fueron declarados en la sección de privado utilizado así el concepto de encapsulamiento de la programación orientada a objetos, para impedir la posibilidad de quien use la clase, de acceder a parámetros de configuración directamente. De esta manera se ahorran problemas entre la generación de instrucciones por parte del usuario y la transmisión de los mismos de manera correcta.

Así como las variables se programó un método privado llamado sscLog él cual se encarga de registrar eventos críticos al momento de usar la clase; ejemplos de esto pueden ser el acceso al puerto serial o las instrucciones que

puede pasar de los límites de los servomotores. Se puso este método como privado porque, aunque solo la propia clase tiene la necesidad de reportar los sucesos críticos en el manejo la conexión y las instrucciones.

En cuando a la parte pública de la clase no fue necesario de crear variables debido a que no hay información que se deba manipular externamente a la clase. En cuanto a los métodos, no solo se programó para lograr la comunicación y el manejo de las instrucciones si no también algunas funciones útiles a la hora de depurar en posibles futuros desarrollos.

En los siguientes tres listados se encuentra un breve comentario con el objetivo de conocer los elementos que componen esta clase además de su funcionamiento, el cual será útil en el siguiente capítulo donde será utilizado.

Variables

- fd: variable de tipo entero usada para guardar el retorno de la función que vincula el puerto serial
- file: apuntador de tipo archivo usado a la hora de
- ID: arreglo de caracteres sin signo usado para tener un listado de los motores relacionado con sus posiciones.
- position: arreglo de enteros sin signo que almacenan la posición relativa en forma de pasos en ancho de pulso.
- commands: vector de cadenas el cual se utiliza como pila LIFO para el almacenamiento de instrucciones.
- nerror: variable tipo long sin signo usada para contar el número de puntos críticos alcanzados desde que se construye la clase.

Métodos

void openUART(string)

usado para crear la comunicación entre la clase ssc32 y el puerto serial del sistema, no tiene valor de retorno y recibe como parámetro el nombre del puerto.

void closeUART()

Usado para cerrar el puerto serial. Este método es usado directamente por el destructor, aunque puede ser usado también por el usuario si es necesario.

void run(void)

Usado para transmitir la información almacenada en la pila commands agregando el carácter de ejecución al final y de esta manera ejecutar las instrucciones. Este comando manda llamar primero al método txCommands_LM para la transmisión de las instrucciones ynluego a txChar para agregar el carácter de ejecución.

void txChar_LM(char)

Transmite un carácter a la tarjeta mediante el puerto descrito por el usuario, no tiene valor de retorno y necesita como argumento el carácter a mandar.

void txString_LM(string)

Transmite una cadena de caracteres, no tiene valor de retorno y necesita la cadena a mandar como argumento.

void txCommands_LM(void)

Este método transmite secuencialmente las instrucciones almacenadas en la pila commands, no necesita parámetros ni tiene valor de retorno. La transmisión se realiza comenzando por el último elemento insertado hasta llegar al primero.

char getOfLM(void)

Regresa un punto “.” Si el movimiento mandado en la orden anterior a sido completado y un carácter “+” si sigue en progreso.

void addInstruction(string)

Permite agregar un comando directamente por el usuario. El comando solo puede empezar con el carácter “#” o “Q” y si no es el caso no se agregará a la pila esto es para evitar transmitir comandos no válidos. A pesar de esta limitación existe la posibilidad de mandar comandos inválidos más este método

lo vi necesario a la hora de depurara código o probar la comunicación además de dar la pauta a la transmisión con otras tarjetas que no tengan la misma estructura para los comandos.

unsigned int getPosition(const int &)

Esta función Como argumento pide el número de motor del cual se quiere saber su posición y el valor de retorno es un entero sin signo el cual el ancho de pulso relativo a la posición del motor.

bool checkLimits(const int &,const int &,const int &,const int &)

Método que sirve para comprobar que los parámetros tales como el número de motor, el ancho de pulso, la velocidad o el tiempo que se quiere dar en a cierta instrucción es correcto, comprobando los limites superiores e inferiores de cada variable. Para realizar esto se escribieron los limistes superior e inferior en la cabecera de la clase; a esto se le conoce comúnmente entre programadores como macro.

int checkPosCommand(const int &)

Método que comprueba si se ha insertado anteriormente una instrucción en la pila commands, este método recibe como parámetro el número de motor y como valor de retorno regresa -1 en caso de no encontrarse aún una instrucción o el número de la posición en la pila en caso de que exista una instrucción previamente almacenada.

bool addMove(const int &,const int &,const int &,const int&)

Esta función es de las más importantes pues es la que se encarga de agregar nuevas instrucciones a la pila commands llamando a las funciones anteriores checkPosCommand y checkLimits para asegurarse de que es una nueva instrucción y además está bien escrita. Este método manda a llamar a la función sscLog en el caso en el que no se cumpla con alguna de las condiciones antes mencionadas

String back()

Función que regresa la última instrucción agregada a la pila. Útil para hacer pruebas.

void printData()

Método que muestra toda la información almacenada en la clase a través de la consola.

Constructores

ssc32()

constructor sin argumentos

Inicializa las variables como el contador de errores o el arreglo de los motores con ceros para evitar errores. Agrega la hora de construcción de la clase, así como la dirección del puesto predeterminado al que se conectara al log de la clase.

ssc32(string)

Este constructor hace casi lo mismo que el anterior con la ligera diferencia de que a este se le especifica únicamente la dirección del puerto al cual se quiere conectar.

Un ejemplo de la utilización es el siguiente.

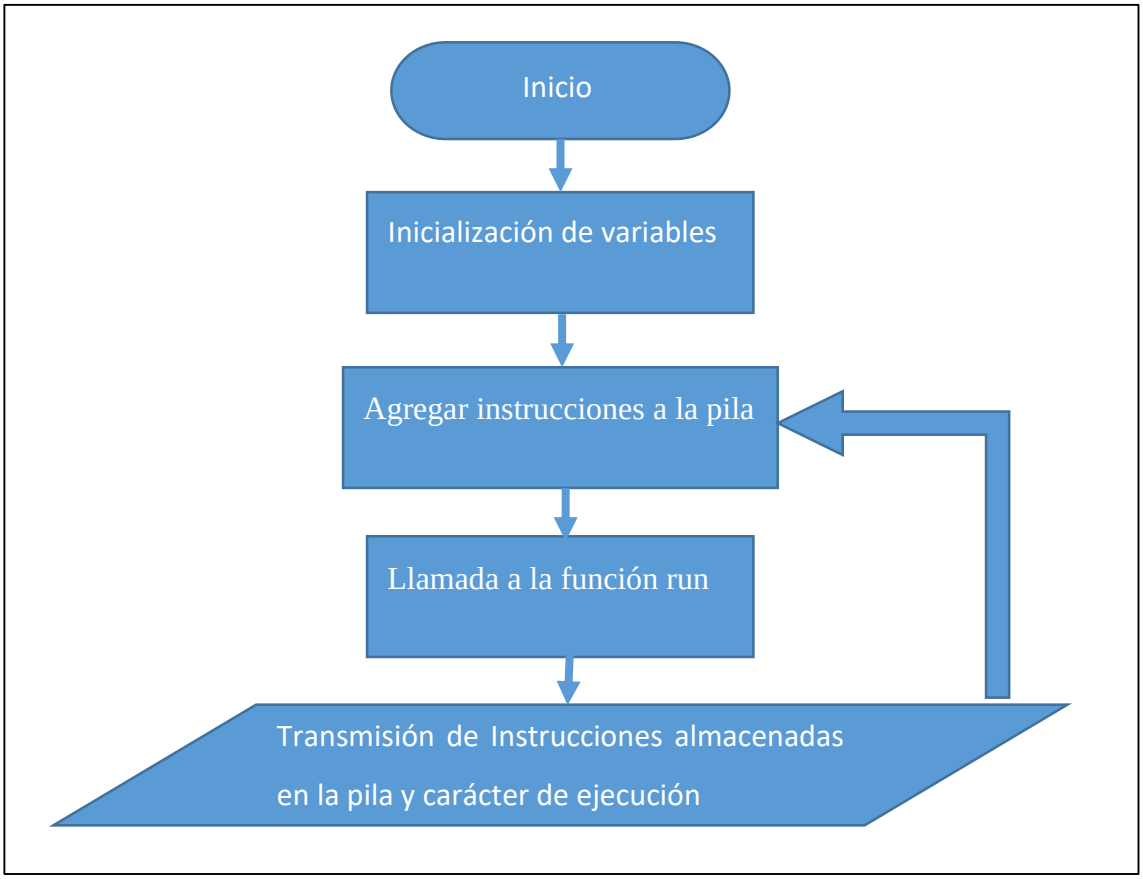


Figura 4-2Diagrama de flujo para la utilización de la clase ssc32

En este ejemplo solo se tiene en cuenta que se necesita mover algún motor o conjunto de motores a cierta posición por lo que solo se agregan instrucciones a una pila y después se mandan con la función run agregando al final, el carácter de ejecución visto en el capítulo de la tarjeta ssc32.

4.2.3 Obtención de valores del sensor de orientación

De una manera parecida que, con la generación de las instrucciones para la tarjeta controladoras de motores, se tiene que para obtener datos del giroscopio de manera iterativa es necesario ejecutar una y otra vez las mismas instrucciones en la línea de comandos, lo cual no es práctico. Por lo que se optó por crear herramientas que realicen estas operaciones en automático.

La mejor opción que se encontró fue programar funciones que terminaron siendo una pequeña librería que permiten el uso del sensor de orientación. Esta programación se hizo con el objetivo de permitir la configuración de manera automática y poder hacer consultas directamente del mapa de memoria.

Siendo así, para la creación de esta librería fue utilizado como guía, un ejemplo que habla del uso del puerto I^2C proporcionado por sparkfun (sparkfun, sparkfun, s.f.). Si bien el código encontrado funciona sin problemas, este estaba programado para funcionar con otro dispositivo, por lo cual, fue necesario hacer una investigación sobre el mapa de memoria dentro del sensor de orientación. Esta información se encuentra en los manuales del MPU-6050 en el apartado que habla sobre los registros (Inc. I. , RM-MPU-6000A, 2012).

En cuanto al puerto de comunicación, fue necesario hacer pruebas previas con las utilidades I2C-tools (mencionadas en la revisión de las comunicaciones electrónicas). Debido a que existen tres dispositivos reconocidos desde el SO, fue necesario probar cada uno de estos hasta encontrar el que se comunicaba con el sensor de orientación. Finalmente fue el numero dos (i2c-2), con el que obtuve buenos resultados en la comunicación.

La programación fue minimalista y escrita en el lenguaje “C” de tal forma que se escribieron los métodos para abrir el puerto, leer, escribir y convertir un par de registros a enteros. Todo lo anterior enfocado únicamente a la adquisición de mediciones del giroscopio y acelerómetro, datos inmersos en el GY-521.

Al igual que con la librería para el control de motores, aquí se muestra el listado de los métodos programados, así como una breve descripción de su uso y/o funcionamiento.

Funciones

- openI2c(char *port, char address)

Encargada en abrir el puerto indicado mediante una dirección absoluta al bus de conexión y la dirección del dispositivo. Un ejemplo de esta función sería: openI2c("/dev/i2c-2", 0x68).

- char readI2c (unsigned char a)

Lee el byte en la posición “a” dentro de la memoria del dispositivo vinculado mediante la función openI2c.

- char writeI2c (char addres_i2c,char value_i2c)

Escribe en la dirección de memoria dada por “addres_i2c” el valor “value_i2c” en el dispositivo vinculado mediante openI2c.

- int twoCharToInt (const unsigned char a, const unsigned char b)

Esta función si bien no tiene que ver directamente con la conexión o la lectura y escritura en el dispositivo considero que puede llegar a ser muy útil para realizar la concatenación de los bytes mayor y menor que contienen la información de aceleración o giro en un eje y que vienen del registro del MPU-6050.

4.2.4 OpenCV (procesamiento de imagen y video)

Para el procesamiento de imagen y video se decidió usar OpenCV. La razón principal por la cual se decidió usar este proyecto fue debido a las gratas experiencias previas que ha tenido el equipo UNAMoids en otros proyectos. Además de esto, la documentación y tutoriales encontrados en internet fue un buen incentivo para su uso en este robot.

OpenCV es un proyecto de software libre con cuenta en github (github, 2016) por lo que cualquier persona interesada en contribuir con el mejoramiento de estas librerías puede hacerlo si necesidad mayor que tener una cuenta del en la misma página github , el nombre completo del proyecto es “Librería de visión computacional de código abierto” (traducción propia de (OpenCV_Developers_Team, 2016)) y es ampliamente utilizado alrededor de mundo por grandes compañías como Google, Yahoo, Microsoft, IBM entre otras. Estas librerías son multiplataforma por lo que pueden ser instaladas y usadas tanto en los sistemas operativos basados en Linux como en Windows, mac o android, el código esta optimizado para ser usado con “C++” o “C” aunque también hay soporte para otros lenguajes como “java” o “Python” entre otros (OpenCV_Developers_Team, 2016).

Para poder hacer uso de las librerías de OpenCV en ubuntu es necesario instalarlas desde los repositorios. Pero en el caso de la versión de Ubuntu disponible para la PCDuino, esta no cuenta con dichas librerías, por lo que fue necesario investigar su instalación de manera manual en este sistema embebido.

El proceso de instalación manual tomo aproximadamente un día, este tiempo no toma en cuenta el proceso de descarga del proyecto de internet ni la investigación previa para su instalación. Este tiempo fue debido a la velocidad de procesamiento en la compilación y acomodación de los archivos dentro del sistema embebido.

Una vez instaladas las librerías en el sistema embebido se propuso probarlas, mediante la ejecución de ejemplos que vienen por defecto en la instalación.

En cuanto al uso de OpenCV en el sistema operativo ubuntu, sea o no en la tarjeta PCDuino, existe un portal donde se tiene toda la documentación necesaria tanto para instalación, configuración e incluso el uso de las funciones en los lenguajes “C” y C++ (opencv-dev-team, 2014).

5 Pruebas y resultados

Entre las principales características que se fijaron como objetivo se destacan, la requisición de datos del giroscopio, la adquisición y procesamiento de video, el control de los servomotores de una manera organizada y la alimentación continua de todos los circuitos. Así, haciendo caso de lo anterior se propuso programar un nuevo conjunto de librerías, las cuales fuesen en lo posible únicamente dependientes de las librerías estándar del lenguaje C++ (instalas por defecto en Ubuntu) las cuales ayudasen en la manipulación del robot, así como a su programación para pruebas experimentales. (ISO, s.f.).

En consecuencia, para poder probar el funcionamiento de lo anterior también fue programado un ejemplo para cada una de las tres librerías, las cuales están dedicadas a; el posicionamiento de los motores en conjunto, la adquisición de datos del giroscopio y el rendimiento en el procesamiento del video mediante OpenCV y la línea de comandos del sistema operativo.

5.1 Prueba para capturas de posiciones

Cabe destacar que este es solo un ejemplo de utilización de la clase ssc32 por lo que en ningún momento se tiene en cuenta la dinámica o movimientos del robot, únicamente se programó para tener una visión real y no solo teórica de las posibilidades que presenta el tener una clase dedicada a manejar la creación de las instrucciones y la transmisión de la misma. Así también existen muchos casos lógicos no considerados en las funciones del programa de ejemplo que pueden desencadenar un mal funcionamiento del programa esto es así porque este programa está orientado únicamente a dar un ejemplo y en ningún caso se piensa en un usuario sin conocimientos de programación.

Anteriormente el robot también tenía un sistema de capturas más sus limitaciones eran bastantes. Por mencionar la más importante: no se podía mover motores específicos en cada captura o lo que es lo mismo se movían todos los motores en cada captura. El mover motores específicos por separado es importante para crear movimientos más fluidos. Podemos apreciarlo en los

movimientos de cualquiera de nuestros miembros, por ejemplo, al intentar atrapar una pelota con la mano se tratar de interceptar la trayectoria del objeto mediante nuestro codo, muñeca y mano que cambian constantemente de posición aproximándose cada vez más a esta.

Pasando al programa, este se encuentra conformado por una subclase llamada cap, un vector de subclases cap, una variable de tipo entero sin signo llamada hoja y cuatro funciones de las cuales se explica el funcionamiento a continuación.

Variables

- Hoja: entero sin signo
- Sheets: vector de hojas
- Cap: subclase con la información necesaria para crear una captura

Funciones

void calibCap(const int &a)

Función que mediante la interacción con el usuario mueve al robot motor por motor con el objetivo de crear una captura. Para su funcionamiento de debe haber creado anteriormente una captura puesto que requiere el número de hoja en la que se está trabajando.

void saveCap(int s)

Necesita como argumento el número de hora que se quiere guardar, este método crea una entrada en el registro con el nombre de la captura a guardar y seguidamente crea un archivo de texto plano iniciando con una cabecera la cual indica el nombre de la captura seguida de las instrucciones línea por línea.

void loadCaps()

Esta función lee un archivo de texto plano llamada reg.txt el cual contiene una lista línea a línea de las capturas que se han realizado anteriormente, el objetivo es cargar todas las capturas hechas anteriormente llamando cada uno de los archivos que contienen la información.

void runCap(int s)

La función runCap recibe el número de la hoja de captura que se quiere ejecutar previamente cargada en memoria con la función loadCaps() para transmitir dicha hora a tarjeta seguidamente de la ejecución mediante la función de run de la clase ssc32.

Tal como en la clase ssc32 en este programa también implemente funciones extra con motivos de depuración del código y pruebas estas son:

void printCap(int s)

Pide como argumento un entero y su función es imprimir lo que se encuentra en la captura con tal número mediante llamadas a funciones directamente en la subclase cap.

void wait()

Útil para hacer pausas en el programa, este método espera a que el usuario oprima la tecla enter para continuar ejecutándose.

El siguiente diagrama de flujo muestra la implementación de las funciones anteriormente mencionadas en el archivo capMoves con el objetivo de mostrar el funcionamiento en conjunto.

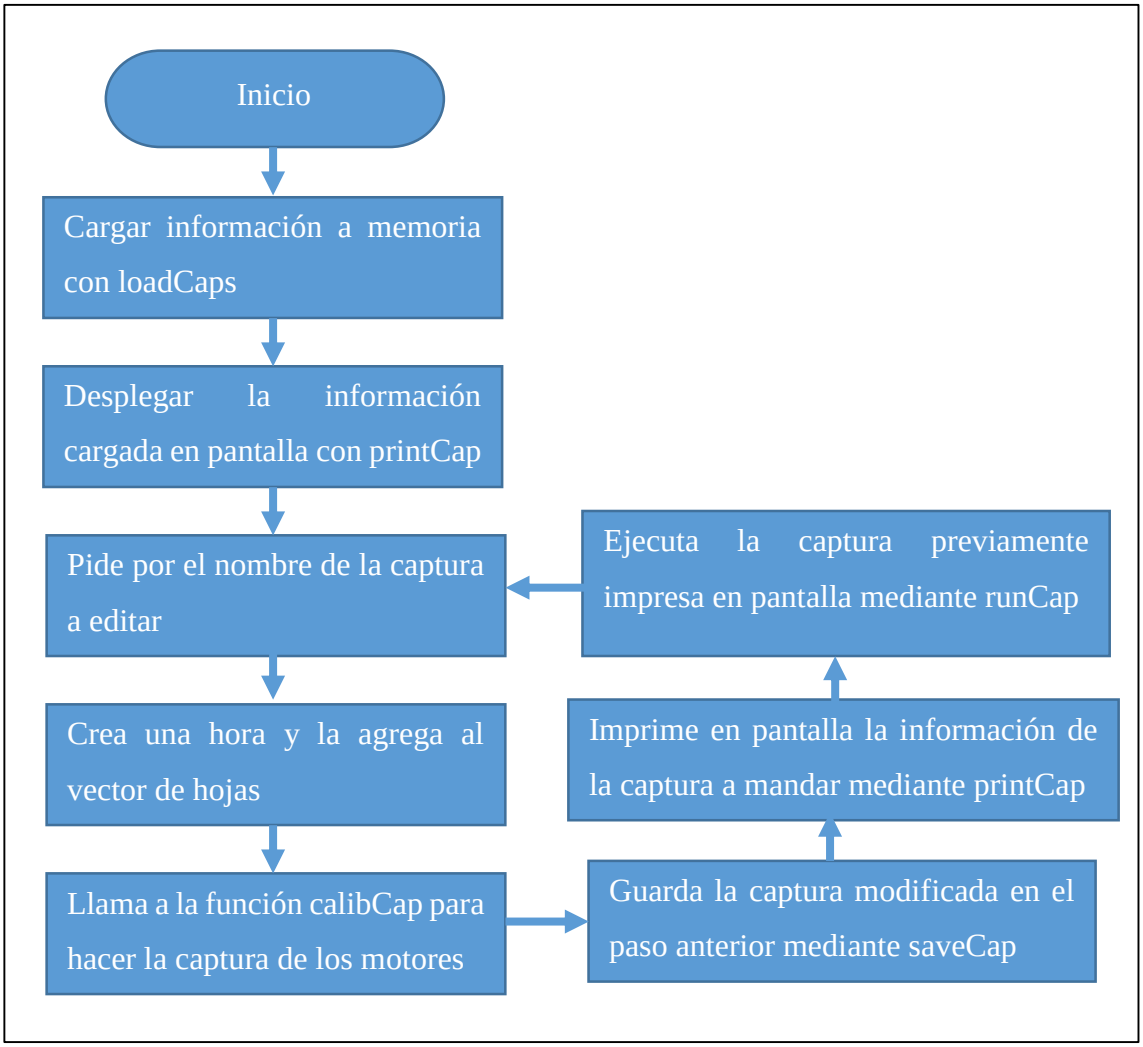


Figura 5-1Diagrama de funcionamiento del programa capMoves

Algo muy importante a tener en cuenta en este programa es que, en las funciones anteriormente mencionadas se está haciendo uso de la clase ssc32 por ejemplo en la función calibCap se crea un objeto el cual sirve para mandar las ordenes a los motores y ejecutarlas de esta manera se crea una realimentación entre las ordenes que manada el usuario para posicionar el motor y la posición que el usuario ve en el motor y quiere corregir, otro ejemplo es runCap que manda toda la información de una captura, imprime la información almacenada en la clase creada por la función y finalmente llama al método run de la misma. Todo lo anteriormente mencionado puede verse en los anexos directamente en el código.

5.1.1 Resultados del uso de la librería para capturas de posición

Con la ayuda de la clase ssc32 y el ejemplo propuesto en el apartado anterior se lograron hacer pruebas en un tiempo mucho menor al que se le dedicaba anteriormente.

Realizar una captura en el laboratorio se redujo a cuestión de minutos, se puede tener una idea de esto si se tiene en mente que posicionar un motor cuesta menos de un minuto. Así también el mover ambos brazos haciendo un par de capturas para abrir y cerrar los brazos del robot me costó apenas escasos 20 minutos.

Aparte de las mejoras que se tuvieron en cuanto a los tiempos; la seguridad para motores es otro punto a destacar pues a ningún motor se le lleva más allá de los límites de movimiento establecidos. Así como para los motores los demás parámetros como el canal al que va dirigida la instrucción, la velocidad y el tiempo de movimiento son respetados pues la clase ssc32 evita mandar estas instrucciones y a su vez hace una pequeña bitácora en la cual reporta aquellas instrucciones erróneas.

5.2 Prueba de mediciones del giroscopio

Con base en las funciones anteriores se realizó un programa básico procurando fuese lo más entendible posible, con el cual se demuestra a la par tanto la funcionalidad de lo descrito anteriormente como del uso del giroscopio, este código inicia configurando el puerto de comunicación dando la dirección “/dev/i2c-2” que corresponde el puerto dos y asignado la dirección del dispositivo en “0x68”, seguidamente lee e imprime el registro correspondiente al modo de funcionamiento, lo rescribe mediante la función ”writeI2c”, espera 4 segundos a que el sistema termine de hacer sus configuraciones correspondientes y finalmente entra en un ciclo infinito donde en cada inicio limpiara la pantalla, leera, concatenara e imprimirá los bytes mayor y menor de cada eje de aceleración y giro esperando un décimo de segundo antes de comenzar de nuevo el ciclo. Así como la explicación anterior anexo el diagrama a bloques para una mejor comprensión.

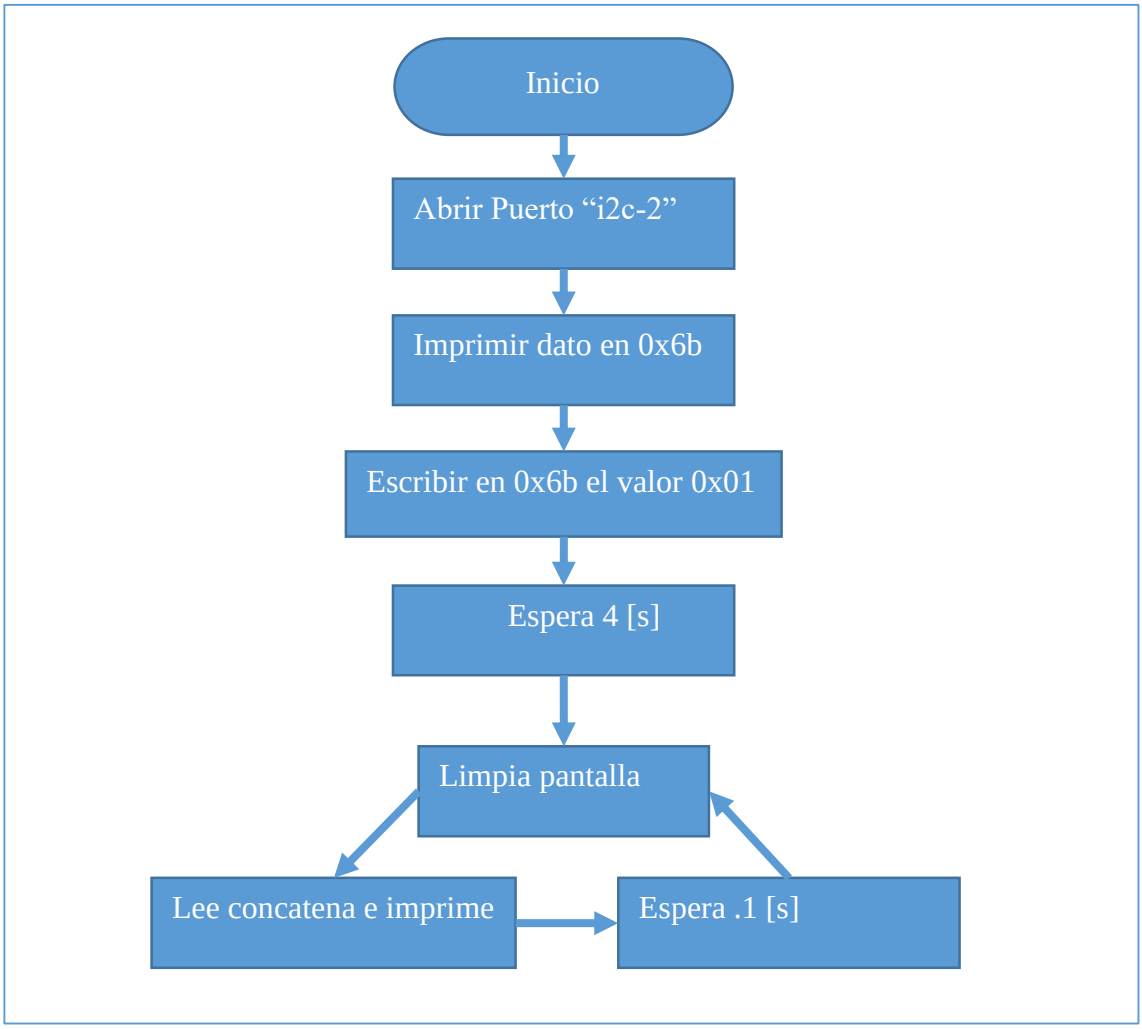


Figura 5-2Diagrama a bloques del código para demostración del giroscopio

Cabe destacar que el código como tal no tiene ninguna forma de llegar al fin de ejecución, a menos que el usuario del sistema operativo termine con dicho proceso ya sea con una señal del sistema como la combinación de teclas “control” mas “C” o desde algún administrador de tareas.

5.2.1 Resultados del uso de la librería para la adquisición de mediciones del giroscopio

Al iniciar el programa del giroscopio se puede apreciar la información proveniente de los registros cambiando constantemente en las cifras menos significativas mientras que las más significativas se mantienen constantes. Así mientras movemos el giroscopio o el robot en conjunto (teniendo en cuenta que el robot ya lo tiene instalado) se puede apreciar en la pantalla desde se tiene acceso al sistema operativo ejecutando el programa, como van cambiando conjuntamente los valores tanto de la aceleración en los tres ejes como de las

lecturas del giroscopio acorde a la posición del robot o dispositivo conforme al campo gravitacional terrestre y la dirección en la que apunta el giroscopio.

Las mediciones que son un tanto erráticas en los tres dígitos menos significativos posiblemente sean debidas en parte a la configuración del dispositivo MPU-6050 y otro tanto, a que en el lugar donde se hicieron las pruebas existían elementos que pudieran interferir con las correctas mediciones del mismo, como imanes o cables energizados, así como una cantidad apreciable de dispositivos diversos para el uso de la experimentación con la electrónica.

A pesar de lo anterior se puede apreciar claramente los cambios que en los valores del sensor que corresponden directamente con el campo gravitacional y posición.

5.3 Prueba de procesamiento de imagen

Para la prueba de video con OpenCV, además de realizar el procesamiento de video se realizaron mediciones de velocidad, tanto en el sistema embebido PCDuino como en una computadora portátil. Esto se realizó de tal forma que se tuviera una noción clara entre las mediciones de cada una. Si bien es cierto que existen diversos factores que pueden influir en estas mediciones, tales como procesos propios del sistema operativo o malas prácticas en el código de otros programas corriendo simultáneamente a los programas de prueba, se trató en lo posible usar herramientas que minimizasen sus efectos.

El código fue inspirado con base a la documentación de OpenCV y puede ser encontrado en internet mediante la búsqueda de “reconocimiento de bordes con OpenCV”. Además de que el código para las pruebas del presente trabajo puede ser consultado directamente en los anexos. El programa se realizó en el lenguaje C++ por dos principales razones, la primera es que existe actualmente mayor información disponible en la red sobre el uso de las librerías de OpenCV en este lenguaje que en otros y la segunda tal vez que sea la más importante, es que el código esta optimizado para ser usado en este mismo lenguaje.

Para la prueba de rendimiento en procesamiento de video se midió el tiempo que toma en procesar un cierto número de cuadros, mediante un código

sencillo capaz de aplicar 3 filtros distintos, (aplicados para el reconocimiento de bordes) extrayendo cada cuadro directamente de la cámara web mediante las librerías de OpenCV escritas en el lenguaje C++ y medir el tiempo de ejecución directamente desde el SO, se realizó de esta forma debido a que mediante el comando "time" en *bash* se muestran los tiempos reales de ejecución y no se necesitan funciones extras para medir el mismo dentro del código.

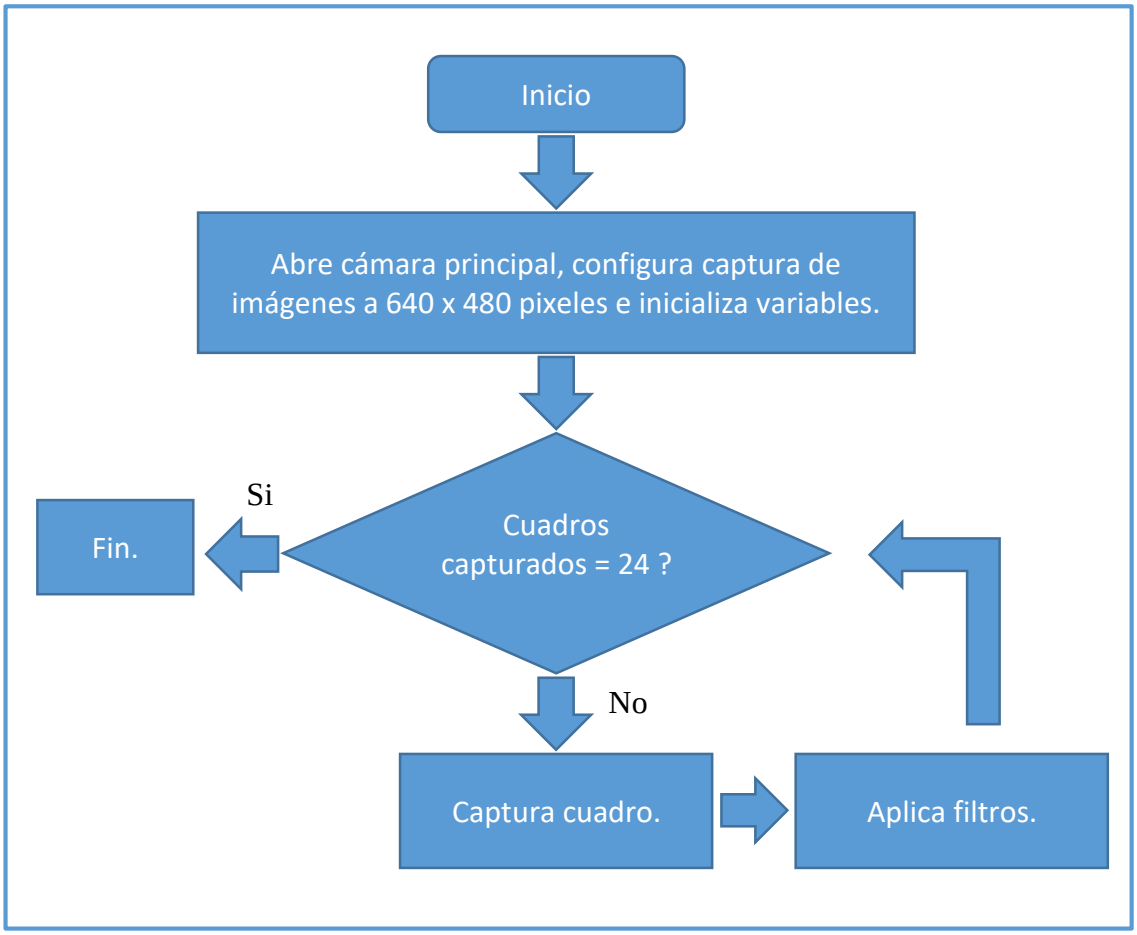


Ilustración 5-1Flujo de programación para ejemplo de procesamiento de video con OpenCV.

En la ilustración 5 -1 se puede apreciar un diagrama que muestra el flujo de programación para la prueba de video, cabe notar que la secuencia se centra directamente en la captura y procesamiento de los veinticuatro cuados propuestos.

Es importante mencionar que si bien se aplicaron estos filtros para el reconocimiento de bordes en ningún caso fue parte del objetivo de este trabajo hacer un procesamiento especifico. Así, esta prueba es hecha para tener saber si funciona o no el procesamiento de video y en dado caso, medir la velocidad de

cuadros por segundo. Por lo que, de igual manera se pudieron haber implementado 5 filtros iguales y obtener las respectivas mediciones. Así entonces la razón específica por la cual se usaron tales filtros, fue para tener un ejemplo muy básico pero aproximado del tipo de procesamiento que debe hacer el robot en tiempo real.

5.3.1 Resultados del procesamiento de video

En esta prueba se procesaron 24 cuadros secuencialmente, con un tamaño de 640x480 pixeles de ancho y alto respectivamente, como si se tratase de video, aplicando a cada cuadro tres filtros, los cuales son un cambio de espacio de color, un filtro gaussiano y un filtro Canny. Se tomó en consideración que únicamente estuviera funcionando lo que el sistema operativo trae instalado por defecto o como se le llama comúnmente en un arranque fresco y se corrieron mínimo 5 pruebas con variaciones del orden de las décimas de segundo las cuales son poco considerables dado los resultados generales. Adicionalmente se hizo uso del comando *time* desde la línea de comandos del sistema operativo, con el objetivo de tener una mejor medición (GNU, time(1), 2015).

Dispositivo	Tiempo Total[s]	Tiempo en el CPU[s]	Cuadro/tiempo(usuario)
PCDuinoV1	7,6	5,4	4,444444444
PC acer D255	2,6	1,8	13,33333333

Tabla 5-1Velocidades en el procesamiento de video

En la tabla anterior podemos ver las mediciones de los tiempos que tomaron cada una de las plataformas, así como su relación cuadro/tiempo que equivaldría a la velocidad de procesamiento. En cuanto a los dos tiempos, el tiempo real o total se refiere al tiempo que tomo en ejecutarse la prueba desde que se da la orden en la línea de comandos hasta que termina la aplicación y el tiempo en el CPU o de usuario es aquel en el que el programa realizo sus operaciones con el uso del mismo CPU (GNU, time(1), 2015). Por lo que, para el cálculo de la velocidad se usó el tiempo de usuario, esto es debido a que el tiempo en abrir la aplicación y mostrarla en la línea de comandos es excluida de esta forma y nos queda el tiempo en el CPU.

6 Conclusiones

En cuanto a lo relacionado con la alimentación eléctrica de este proyecto, cabe destacar que, todas las tarjetas se conectaron adecuadamente y son alimentadas por las baterías, ya sea directamente como en el caso de la ssc32 o mediante la fuente conmutada Buck para la PCDuino. Las pruebas sobre la duración de las baterías fueron parciales, aun así, se logró que el tiempo de funcionamiento fuese el necesario para permitir el desarrollo de todas las demás pruebas con el sistema embebido sin interrupciones. Así también, se notó una reacción no prevista mientras se realizaban pruebas de capturas de posición, esta consistía en la desactivación de todos los motores controlados por la tarjeta ssc32, dicha reacción se iniciaba al poner 2 o más motores contrapuestos o fuera del límite permitido físicamente (visto en el capítulo de Desarrollo e implementación).

El funcionamiento de la tarjeta ssc32 se probó paso a paso con las utilidades de GNU/Linux desde la línea de comandos haciendo las configuraciones previas correspondientes sin tener mayor contratiempo. Después de que la prueba anteriormente descrita fue satisfactoria se comenzó con el desarrollo de la librería para su manejo.

Los resultados de la prueba de las capturas de posición revelan una manera práctica para el manejo de las posiciones de los motores en cada momento gracias al uso de la librería creada en este proyecto, escrita en el lenguaje C++ llamada ssc32. Además, fue posible apreciar la generación automática de instrucciones por parte de la misma clase y en consecuencia dejar de estar pensando en la generación de instrucciones, lo cual, entre otras cuestiones promete el desarrollo de proyectos en menor tiempo y un mejor entendimiento del código.

Por parte del giroscopio, los resultados demuestran que la comunicación entre el sistema embebido y el dispositivo, mediante I^2C es funcional, además de que, la inicialización del sensor y la adquisición de datos relacionados a la

orientación es constante. Finalmente, se encontró que esta información estaba efectivamente relacionada directamente con el campo gravitatorio terrestre y al movimiento de la orientación del robot.

En la sección de pruebas para el procesamiento de video se pone en evidencia que este, es posible realizarlo directamente con el sistema embebido de igual forma que con una laptop. Se encontró la velocidad de procesamiento teniendo las condiciones de 24 muestras con 640x480 pixeles por cuadro, además de poner como comparativa, los resultados de una laptop con la cual se realizó parte del desarrollo.

Viendo el objetivo descrito al principio del presente trabajo, es apreciable observar que se ha logrado desarrollar cada uno de los elementos necesarios para el funcionamiento, así como la generación de este mismo documento que posiblemente sirva como referencia para futuros diseños de robots basados en tarjetas de desarrollo embebido.

Los distintos análisis que se hacen paso a paso a lo largo de este escrito podrían servir como referencia a la hora de tomar decisiones tanto para baterías como para tarjetas de desarrollo e incluso servomotores.

Si bien, los códigos creados para las pruebas de las librerías carecen de un objetivo aplicativo, estos pueden servir como referencia para futuros desarrollos, dada la documentación generada tanto en este trabajo como en los laboratorios donde se trabajó.

Para terminar, tanto el presente trabajo como los códigos para comunicarse con los distintos sensores y actuadores representan una gran ayuda en el tema del desarrollo de la robótica en la actualidad dado que teóricamente, el trabajo que se le dedico a programar y configurar todo en tiempo neto será menor a lo que le tomará a otra persona.

6.1 Perspectiva a futuro

Si bien el robot se encuentra en estado funcional, con todos los sensores y actuadores configurados, una versión final del robot deberá incluir más funciones y programas con los cuales se pueda hacer funcionar al robot de

manera autónoma (sin necesidad de un operador para su posicionamiento), así como tener otra plataforma de desarrollo cuya capacidad de procesamiento sea lo mayor posible (otra SBC) y de ser posible configurar el sistema para trabajar como un sistema operativo en tiempo real. En cuanto a la dinámica del robot que es el tema más importante a considerar, se debe hacer un modelo el cual tome en cuenta la gran cantidad de motores y señales externas, así como su geometría y algunos de los pesos de sus componentes.

Por la parte de los programas que se tienen, estos solo realizan la comunicación y el manejo de instrucciones por lo que se podría considerar que haciendo hipotético el caso en el cual se planease hacer la programación por capas, los programas que se han hecho en este trabajo son la parte más baja de programación, las cuales se encargan de interactuar directamente con el "hardware", en el caso del ejemplo "capMoves" esta sería la capa siguiente en la cual se hace uso de la encargada el uso del hardware y está, se encuentra enfocada en el manejo de cuadros (posiciones del robot) para su uso futuro en la creación de secuencias de movimiento.

Los códigos que se probaron en este trabajo, así como las funciones descritas se encuentran en archivos diferentes, de modo tal que cada conjunto de funciones dedicados a la solución de cada problema se encuentra en un archivo, por lo que falta encontrar una forma de organizarlos de tal forma que la consulta y uso de tales sea de una manera eficiente.

Todos los códigos se encuentran en el anexo del presente trabajo, además de estar en Departamento de Ingeniería de Sistemas Computacionales y Automatización (DISCA) pues se espera que con base en él se pueda crear un "framework" (librerías y utilidades para facilitar la programación del robot) en donde se encuentre todo lo necesario para poner a funcionar el robot en competencias o pruebas que requieran un alto grado de interacción entre los distintos sensores y actuadores de una manera rápida y eficaz.

Bibliografía

Academia Mexicana de la Lengua. (2015). *Academia Mexicana de la Lengua*. Recuperado el 09 de 10 de 2016, de Academia Mexicana de la Lengua: http://www.academia.org.mx/robot

Aliexpress. (2016). *Aliexpress*. Obtenido de Aliexpress: https://es.aliexpress.com/w/wholesale-humanoid-robot.html

Allison, D. K. (17 de 12 de 2015). *Smithsonian Institution*. Obtenido de Gordon Bell: http://americanhistory.si.edu/comphist/bell.htm#Inventing

Arduino. (2016). *Arduino*. Obtenido de Arduino: https://www.arduino.cc/

Arduino. (2016). *Arduino Playground*. Obtenido de MPU-6050 Accelerometer + Gyro: http://playground.arduino.cc/Main/MPU-6050

Barrientos, P. B. (2007). *Fundamentos de Robotica 2da Edicion* . Aravaca (Madrid): McGraw Hill.

Blanuša, B. L. (2015). *Power Electronics Converters and Regulators Third Edition.* Suiza: Springer.

Bovik, A. (2000). *Handbook of image and video processing.* Sand Diego, San Francisco,New York, Boston, London, Sydney, Tokyo: ACADEMIC PRESS.

BSD. (5 de enero de 2016). BSD General Command Manual.

Canonical, U. a. (20 de 12 de 2015). *About ubuntu*. Obtenido de The Ubuntu story: http://www.ubuntu.com/about/about-ubuntu

Cortés, F. R. (2011). *Robótica control de robots manipuladores.* Mexico DF: Alfaomega.

David Linden, T. B. (2002). *Handbook of batteries.* United States: McGraw-Hill.

DAVIES, E. (2012). *Computer and Machine Vision: Theory, Algorithms, Practicalities.* Kidlington, Oxford: ELSEVIER.

DFROBOT. (04 de 02 de 2016). *DFROBOT*. Obtenido de pcDuino: http://www.dfrobot.com/index.php?route=product/product&product_id=891&search=pcDuino&description=true#.VrwdC-b8CUk

Electronics, H. (2017). *Haoyu ELECTRONICS.* Obtenido de GY-521 MPU6050 3-Axis Acceleration Gyroscope 6DOF Module: http://www.hotmcu.com/gy521-mpu6050-3axis-acceleration-gyroscope-6dof-module-p-83.html

Española, R. A. (2 de febrero de 2016). *DLE: autómata - Diccionario de la lengua española edición tricentenario*. Obtenido de autómata: http://dle.rae.es/?id=4TKy9Vs

Frodo Looijard, M. D. (18 de 01 de 2016). i2c-tools man page .

github. (2016). *github*. Obtenido de github: https://github.com/

Gloria Bueno García, O. D. (2015). *Learning Image Processing with OpenCV.* Birmingham: PACKT publishing.

GNU. (21 de 02 de 2015). *time(1)*. Obtenido de Linux man page: http://linux.die.net/man/1/time

GNU. (14 de 01 de 2016). BASH command line.

González, A. G. (2 de 12 de 2016). *panamahitek*. Obtenido de ¿Qué es y cómo funciona un servomotor?: http://panamahitek.com/que-es-y-como-funciona-un-servomotor/

HITEC RCD USA, I. (5 de enero de 2016). *HSR-5498SG*. Obtenido de HSR-5498SG HMI Premium Robot Servo: http://hitecrcd.com/products/servos/discontinued-servos-servo-accessories/hsr-5498sg-hmi-premium-robot-servo/product

Hughes, A. (2006). *Electric Motors and Drives* (Third ed.). Linacre House, Jordan Hill, Oxford OX2 8DP: Newnes & Elsevier.

Inc., A. H. (02 de 02 de 2016). *Asimo Power Source* . Obtenido de Inside Asimo: http://asimo.honda.com/asimo-form/battery-power-supply/

Inc., I. (24 de 10 de 2011). *PS-MPU-6000A.* Obtenido de MPU-6000 and MPU-6050 Product Specification Revision 3.1: http://cdn.sparkfun.com/datasheets/Components/General%20IC/PS-MPU-6000A.pdf

Inc., I. (9 de 03 de 2012). *RM-MPU-6000A.* Obtenido de MPU-6000 and MPU-5050 Register Map and Desciptions Revision 4.0:

http://cdn.sparkfun.com/datasheets/Sensors/Accelerometers/RM-MPU-6000A.pdf

Instruments, T. (04 de 02 de 2015). *LM2576/LM2576HV Series simple switcher.* Obtenido de LM2576/LM2576HV Series simple switcher: http://www.ti.com/lit/ds/symlink/lm2576.pdf

ISO. (s.f.). *ISO/IEC 14882:2014*. Recuperado el 10 de 01 de 2017, de Information technology -- Programming languages -- C++: http://www.iso.org/iso/catalogue_detail.htm?csnumber=64029

jkridner, w. (10 de march de 2016). *beagleboard.org - black*. Obtenido de BeagleBone Black: http://beagleboard.org/black

King, R. (20 de 12 de 2015). *ARM Linux Developer*. Obtenido de ARM Linux Developer: http://www.arm.linux.org.uk/docs/history.php

kubuntu. (14 de 01 de 2016). *Directorios y sistemas de archivos*. Obtenido de Fundamentos de Linux: https://help.ubuntu.com/kubuntu/desktopguide/es/directories-file-systems.html

Lee, J. H. (2006). *General Specification of HSR-5498SG Digital Robot Servo.* http://www.robotshop.com/media/files/pdf/hitec-hsr-5498sg-digital-servo-specsheet.pdf.

linksprite. (2015). *About Us -- linksprite*. Recuperado el 16 de 03 de 2016, de About Us: http://www.linksprite.com.au/about-us/

LinkSprite. (18 de 12 de 2015). *LinkSprite*. Obtenido de Home of pcDuino, The Developer's New Choice: http://www.linksprite.com/linksprite-pcduino1/

Liu, L. Z. (2013). *Introduction to pcDuino.*

Lynxmotion. (4 de noviembre de 2015). *Specifications.* Obtenido de SSC-32 Servo Controller.: http://www.lynxmotion.com/p-395-ssc-32-servo-controller.aspx

Lynxmotion, I. (4 de noviembre de 2005). Obtenido de SSC-32 Ver 2.0 Manual written for firmware version ssc32-1.06XE.: http://www.lynxmotion.com/images/data/ssc-32.pdf

M.R. Jongerden and B.R. Haverkort. (s.f.). Obtenido de Battery Modeling: http://doc.utwente.nl/64556/1/BatteryRep4.pdf

Miquel van Smoorenburg, J. L. (2013). man minicom.

NAKAMURA, M. (1998). *Mechatronic Servo System Control Problems in Industries.* Verlag Berlin Heidelberg NewYork: Springer.

Newark. (12 de 2014). *Electronicdesign uncovered.* Recuperado el 23 de 10 de 2016, de http://www.newark.com/wcsstore/ExtendedSitesCatalogAssetStore/cms/asset/pdf/americas/common/NE14-ElectronicDesignUncovered-Dec14.pdf

Noergaard, T. (2005). *Embedded Systems Architecture "A Comprehensive Guide for Engineers and Programmers".* Oxford: Elsevier.

OpenCV_Developers_Team. (2016). *OpenCV About*. Obtenido de ABOUT: http://opencv.org/about.html

opencv-dev-team. (30 de 12 de 2014). *Installation in Linux -- OpenCV 3.0.0-dev documentation*. Obtenido de Installation in Linux: http://docs.opencv.org/3.0-last-rst/doc/tutorials/introduction/linux_install/linux_install.html#linux-installation

Organización Internacional para la Estandarización. (1975). *El conjunto de caracteres de ISO 646.* Obtenido de Internet Assigned Numbers Authority Registry: web.archive.org/web/20120729193430/http://www.itscj.ipsj.or.jp/ISO-IR/001.pdf

Paul Rubin, D. M. (2010-2016). *GNU coreutils*. Recuperado el 29 de 03 de 2016, de dd - convierte y copia un fichero : http://man.redkaos.net/man1/dd.1

Philips Semiconductors, J.-M. I. (2003). *APPLICATION NOTE, AN10216-01, I2C MANUAL.* California.

Red Hat Enterprise. (27 de 12 de 2016). *Protocolo SSH.* Obtenido de Red Hat Enterprise Linux 4: Manual de referencia: http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

Red Hat, E. (27 de 12 de 2016). *Protocolo SSH.* Obtenido de Red Hat Enterprise Linux 4: Manual de referencia: http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

RoboCup. (2016). *Robocup*. Recuperado el 12 de 05 de 2016, de Robocup: http://www.robocup.org/

RO-BOTICA. (s.f.). *Tienda Robotica*. Recuperado el 13 de 01 de 2017, de ROBOTIS DYNAMIXEL: http://ro-botica.com/tienda/ROBOTIS-DYNAMIXEL

Robotics, I. F. (03 de 02 de 2016). *IFR International Federation of Robotics*. Obtenido de Industrial Robot Statistics: http://www.ifr.org/industrial-robots/statistics/

ROBOTIS. (2010). *TechSupport_ENG*. Recuperado el 29 de 03 de 2016, de ROBOTIS e-Manual v1.27.00: http://support.robotis.com/en/techsupport_eng.htm#product/darwin-op.htm

ROBOTIS. (s.f.). *Overview of Communication*. Recuperado el 02 de 01 de 2017, de ROBOTIS e-Manual v1.29.00: http://support.robotis.com/en/product/actuator/dynamixel/dxl_communication.htm

Saha, S. K. (2010). *Introducción a la robótica.* México, D.F.: Mc Graw Hill.

semiconductors, N. (26 de 01 de 2015). *NXP Semiconductors.* Obtenido de 74HC595; 74HCT595: http://www.nxp.com/documents/data_sheet/74HC_HCT595.pdf

SFUPTOWNMAKER. (s.f.). *pcDuino Hookup Guide*. Recuperado el 01 de 03 de 2017, de Changing the OS: https://learn.sparkfun.com/tutorials/pcduino-hookup-guide/changing-the-os

Silva, E. C. (2015). *Advanced Power Electronics Converters.* New Jersey & Canada: IEEE press.

Sparkfun. (4 de noviembre de 2015). *Sparkfun*. Obtenido de ROBOTIS e-Manual v1.25.00 .: http://support.robotis.com/en/techsupport_eng.htm#product/darwin-op.htm

sparkfun. (18 de 01 de 2016). *I2C - learn.sparkfun.com*. Obtenido de I2C : https://learn.sparkfun.com/tutorials/i2c

Sparkfun. (4 de noviembre de s.f). *Sparkfun.* Obtenido de Programming the pcDuino.: https://learn.sparkfun.com/tutorials/programming-the-pcduino/introduction

sparkfun. (s.f.). *sparkfun*. Recuperado el 09 de 05 de 2016, de Programming the pcDuino: https://learn.sparkfun.com/tutorials/programming-the-pcduino/i2c-communications

SparkFun, E. (04 de 02 de 2016). *Sparkfun*. Obtenido de Polymer Lithium Ion Battery - 1000mAh 7.4v: https://www.sparkfun.com/products/11856

sphero. (2016). *sphero store*. Recuperado el 09 de 10 de sphero, de http://www.sphero.com/starwars/bb8

Texas Instruments Incorporated. (s.f.). *uA7800 series.* Recuperado el 16 de 02 de 2016, de Positive-voltage Regulators: https://www.sparkfun.com/datasheets/Components/LM7805.pdf

The_GNOME_Project. (2015). *apps/Cheese Gnome Wiki*. Obtenido de Cheese: https://wiki.gnome.org/Apps/Cheese

Tomasi, W. (2003). *Sistemas de Comunicaciones Electronicas.* Phoenix, Arizona: Pearson Educació.

Tweed, D. (14 de 01 de 2016). *Circuit Cellar Ink 188*. Obtenido de Circuit Cellar Ink 188: http://www.dtweed.com/circuitcellar/caj00188.htm#3251

UBTECH. (s.f.). *Alpha1 Pro*. Recuperado el 2017, de The new generation with new appearance and joints to bring you better and smoother experience: http://www.ubtrobot.com/product/detail10.html

Vaduva, A. (2015). *Learning Embedded Linux Using the Yocto Project.* Birmingham: Packt Publishing.

Welsh, M. K. (2009). *Running Linux, 5th Edition.* O'Reilly.

wordpress, e. u. (13 de 12 de 2012). *h1Desempeño de C++ vs. Java*. Obtenido de h1Desempeño de C++ vs. Java: todo reside en la calidad del compilador (y del programador también): https://everac99.wordpress.com/2012/12/13/desempeno-de-c-vs-java-todo-reside-en-la-calidad-del-compilador-y-del-programador-tambien/

Ylonen, T. R. (18 de 01 de 2016). scp man page .

ANEXOS

En este apartado se muestran las funciones y los ejemplos creados para las pruebas en el robot, en algunos casos se crearon dos archivos dado que se declararon cabeceras o funciones en un archivo aparte para simplificar el código principal y de esta manera hacerlo más entendible además de ser una buena práctica entre programadores.

Clase de la SSC-32 Lynxmotion

Para la clase ssc32 se tiene una cabecera la cual tiene los macros donde se pueden especificar los límites de los servomotores, si bien vienen limites angulares estos no fueron incrementados en la práctica por razones de experimentación las cuales no es necesario mencionar, pero me parece es pertinente que queden mencionadas en código para su futura implementación la cual deberá ser primero programada en funciones y luego en experimentos con el robot.

ssc32.hpp

```
/*Universidad Nacional Autónoma de México
 *IIMAS
 *Araujo Pino Alan Emir
 *Cabecera de la clase ssc32 para comunicación y manejo de instrucciones
 */

#ifndef SSC32_HPP
#define SSC32_HPP

#define num_motors 32

#define MinAng_width 500
#define Center_width 1500
#define MAxAng_width 2500
#define MinAng_degre -90
#define Center_degre 0
#define MaxAng_degre 90

#define ch_cero 0
#define Max_num_ch 32
#define Min_spd 0
#define Max_spd 2000
#define Min_time_ms 0
#define Max_time_ms 2000
```

```
#define UART_DEFAULT "/dev/ttyS0"

#include <iostream>
#include <string>
#include <vector>
#include <termios.h>
#include <fcntl.h>
#include <unistd.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

using namespace std;

class ssc32{

   public:
   ssc32();
         ssc32(string);
         ~ssc32();
         void openUART(string);
         void closeUART();
         void run(void);
         void txChar_LM(char);
         void txString_LM(string );
         void txCommands_LM(void);
         char getOfLM(void);
         void addInstruction(string);
         unsigned int getPosition(const int &);
         bool checkLimits(const int &,const int &,const int &,const int &);
         int checkPosCommand(const int &);
         bool addMove(const int &,const int &,const int &,const int&);
   string back();

         void printData();
   protected:

   private:
   int fd;
   unsigned char ID[num_motors];
   unsigned int position[num_motors];
         vector<string> commands;
         void sscLog(string);
         unsigned long nerror;

};
```

```
#endif
```

ssc32.cpp

```
/*Universidad Nacional Autónoma de México
 *IIMAS
 *Araujo Pino Alan Emir
 *Prueba de velocidad de captura y procesamiento con 3 filtros
 */
#include <iostream>
#include <vector>
#include <string>
#include <sstream>
#include "ssc32.hpp"

#define BAUDRATE B115200
#define _POSIX_SOURCE 1

using namespace std;

ssc32::ssc32()
{
	nerror =fd= 0;
	time_t now = time(0);
	char* dt = ctime(&now);
	sscLog("Hora de creacion : "+string(dt));
	sscLog("Direccion predeterminada a conectar : " +string(UART_DEFAULT));
	openUART(UART_DEFAULT);
	for(int i = 0;i < num_motors;i++){ID[i] = i;position[i] = 0;}

}
ssc32::ssc32(string a)
{
	nerror = 0;
	time_t now = time(0);
	char* dt = ctime(&now);
	sscLog("Hora de creacion : "+string(dt));
	sscLog( "Direccion a conectar : " + a);
	openUART(a);
	for(int i = 0;i < num_motors;i++){ID[i] = i;position[i] = 0;}
}

ssc32::~ssc32()
{
	closeUART();
}

void ssc32::openUART(string dev)
```

```
{
        fd = open(dev.c_str(), O_RDWR | O_NOCTTY |O_NDELAY );
        cout << "num FILE : "<<fd<<endl;
   if (fd <0){sscLog("Error al abrir el puerto :"+dev);}
        fcntl(fd,F_SETFL,0);
}

void ssc32::closeUART()
{
        close(fd);
}

void ssc32::run(void)
{
        txCommands_LM();
        txChar_LM('\r');
        cout<< "Corriendo : " <<endl;
}

void ssc32::txChar_LM(char a)
{       int send_a = write(fd,&a,1);
        cout << "numero de la escritura : "<<send_a<<endl;
        cout << "transmitido : " << std::hex <<a <<endl;
}
void ssc32::txString_LM(string a)
{
                cout << "transmitido : " << a <<endl;
                int send_a = write(fd,a.c_str(),a.length());
                cout << "numero de la escritura : "<<send_a<<endl;
}
void ssc32::txCommands_LM(void)
{       unsigned int tmp=commands.size();
        for( unsigned int i = 0 ; i < tmp; i++){
                txString_LM(commands.back());
                commands.pop_back();
                }
}

char ssc32::getOfLM(void)
{
        txChar_LM('Q');
        txChar_LM('\r');
        char c;
        read(fd,&c,1);
        return c;
}

void ssc32::addInstruction(string a)
```

```
{       if(a[0] == '#' || a[0]== 'Q'){
                cout << "antes de hacer el add"<<endl;
                commands.push_back(a);
                }else{sscLog(a + "no es un comando valido");}
}
unsigned int ssc32::getPosition(const int &a)
{return position[a];    }

bool ssc32::checkLimits(const int &a,const int &b ,const int &c,const int &d)
{
        if( ch_cero <= a && Max_num_ch > a ){
                if( MinAng_width < b && MAxAng_width > b ){
                        if( Min_spd <= c && Max_spd > c ){
                                if( Min_time_ms <= d && Max_time_ms > d ){
                                        return true;
                                        }else{sscLog("time fuera de limite "); return false;}
                                }else{sscLog("spd fuera de limite ");return false;}
                        }else{sscLog("pw fuera de limite " );return false;}
                }else{sscLog("ch fuera de limite " );return false;}
}

int ssc32::checkPosCommand(const int & ch)
{
        for( unsigned int i = 0 ; i < commands.size(); i++){
                if((char)commands[i][2] == ' '){
                        if((int)(((char)commands[i][1]-(char)'0'))==ch){
                                return i;
                                }else{continue;}
                        }else{
                                if((int)(( 10*( (char)commands[i][1]-(char)'0')) +
                                ((char)commands[i][2]-(char)'0'))==ch){
                                        return i;
                                        }else{continue;}
                                }
                }
return -1;
}

bool ssc32::addMove(const int & ch,const int & pw,const int & spd_us,const int &
time_ms)
{

        string tmp;
        ostringstream ss;
        if(checkLimits(ch,pw,spd_us,time_ms)){
                ss<<ch;
                tmp += "#" + ss.str() + " ";
                ss.str("");
```

```
                ss<<pw;
                tmp += "P" + ss.str() + " ";
                ss.str("");
                if(spd_us != 0){
                        ss<<spd_us;
                        tmp += "S" + ss.str() + " ";
                        ss.str("");
                        }
                if(time_ms != 0){
                        ss<<time_ms;
                        tmp += "T" + ss.str() + " ";
                        }
                int pos =checkPosCommand(ch);
                if((pos==-1)){
                        commands.push_back(tmp);
                        }else{sscLog("Se reescribira el comando : " +commands[pos]+
                        "por " + tmp);
                        position[ch]= (unsigned int)pw;
                        commands.erase(commands.begin()+pos);
                        commands.push_back(tmp);
                        }
                return true;
                }else{
                        ss<<ch;
                        cout << "Funtion addMove ch: " <<ss.str()<<endl;
                        return false;
                        }
}

void ssc32::sscLog(string a)
{
        nerror++;
        fstream f("log.txt", std::fstream::out | std::fstream::app);
        if(f.is_open()){
                f<<"["<<nerror<<"]"<< a << endl;
                }else{
                        cout<<"ERROR AL ABRIR log.txt"<<endl;
                        }
        f.close();
}

string ssc32::back()
{
        return commands.back();
}

void ssc32::printData()
{
```

```
	cout << "-------Datos en la clase ssc32--------" <<endl;
	cout << "numero de instrucciones : " << commands.size() << endl;
	for( unsigned int i = 0 ; i < commands.size(); i++){
		cout <<i<< ") "<<commands[i] << endl;
		}
	cout << "motores" <<endl;
	cout << "# numero de motores : "<< num_motors << endl;
	for (unsigned int i = 0 ; i < num_motors; i++){
		cout <<" motor : "<<i<< "\tID : "<< ID[i]+0x00
		<<"\tposicion :" << position[i]<< endl;
		}
	cout << "-------------------------------------" <<endl;
}
```

Ejemplo de uso de la clase ssc-32

```
/*Universidad Nacional Autónoma de México
 *IIMAS
 *Araujo Pino Alan Emir
 *Ejemplo de uso de la clase ssc-32 en la cual se calibran y guardan posiciones de los
 *motores
 */
#include <iostream>
#include <fstream>
#include <limits>
#include <unistd.h>
#include <dirent.h>
#include "ssc32.cpp"

#define UART_DIR "/dev/ttyS1"

class cap
{
	public:
	cap(){indice =0;}
	cap(string a){name = a;indice=0;}
	string popCommand(void){
		indice--;
		string tmp = commands.back();
		commands.pop_back();return tmp;}
	void addCommand(string a){
		commands.push_back(a);indice++;}
	string getName(void){
		return name;}
	string getElement(int a){
		return commands[a];}
	unsigned int getSize(){
		return indice;}
	void printData(){
```

```
		for(unsigned int i = 0;i<commands.size();i++){
			cout<<commands[i]<<endl;}
			}
	private:
	unsigned int indice;
	string name;
	vector<string> commands;
};

using namespace std;

vector<cap> sheets;
unsigned int hoja;
char cCurrentPath[FILENAME_MAX];

void saveCap(int s)
{
	string tmp =sheets[s].getName() +".txt";
	cout <<"guardando :" +tmp<<endl;
	fstream f;
	f.open(tmp.c_str(), std::fstream::in
	| std::fstream::out | std::fstream::app);
	if(f.is_open()){cout<<"El archivo ya existe desea sobreescribir s/n:";
		string tmp;
		cin >> tmp;
		if(tmp.compare("s")== false){
			f.clear();
			f.seekg(0);
			cout<<endl<< "sobreescribir" << endl;
			f<<"captura_"<<sheets[s].getName()<< endl;
		unsigned int tmpi =sheets[s].getSize();
			for(unsigned int i = 0 ; i < tmpi; i++){
				cout<< sheets[s].getSize();
				f <<sheets[s].getElement(i)<<endl;
				cout << sheets[s].getElement(i)<< endl;
				}
			}
		}
	else{cout<<endl<<"escribiendo nuevo archivo" << endl;
		f<<"captura_"<<sheets[s].getName()<< endl;
		unsigned int tmpi =sheets[s].getSize();
			for(unsigned int i = 0 ; i < tmpi; i++){
				cout<< sheets[s].getSize();
				f <<sheets[s].getElement(i)<<endl;
				cout << sheets[s].getElement(i)<< endl;
				}
		}
```

```
	fstream reg("reg.txt", std::fstream::out
	| std::fstream::out | std::fstream::app);
	reg<< sheets[s].getName()<<endl;
	reg.close();
	f.close();
}

void loadCaps()
{
fstream f("reg.txt", std::fstream::in);
	if(f.is_open()){
		cout<<"Leyendo reg.txt"<<endl;
		while(!f.eof()){
			string name;
			getline(f,name);
			cout << "cargando :" + name<<endl;
			cap tmpCap(name);
			int s = sheets.size();
			sheets.push_back(tmpCap);
			string tmp = name + ".txt";
			fstream fill(tmp.c_str(), std::fstream::in);
			if(fill.is_open()){cout << "si existe el archivo"<<endl;
				while(!fill.eof()){
					tmp.erase();
					getline(fill,tmp);
					sheets[s].addCommand(tmp);
					}
				}else{cout << "no existe el archivo"<<endl;}
			}
		}else{
		cout<<"El archivo \"reg.txt\" no existe, no se reconoceran
capturas"<<endl;
		}
	f.close();
}

void runCap(int s)
{
	cout<< "......Executando cap : "<< sheets[s].getName()<<"......"<<endl;
	ssc32 port(UART_DIR);
	for(unsigned int i = 0; i < sheets[s].getSize();i++){
		port.addInstruction(sheets[s].getElement(i));
		}
	port.printData();
	port.run();
}

void printCap(int s)
```

```
{
	if(sheets.size() != 0){
		string tmp =sheets[s].getName() +".txt";
		fstream f(tmp.c_str(), std::fstream::in);
		if(f.is_open()){
			f.seekg(0, ios::beg);
			cout<< "......Info en cap : "<< sheets[s].getName()<<"......"<<endl;
			string s1;
			sheets[s].printData();

			cout<< "......Info en Archivo......"<<endl;
			while (!f.eof()){
				getline(f,s1);
				cout<< s1<<endl;
				}
			f.close();
			}else{f.close();}
		}else{cout << "aun no hay capturas"<<endl;}
}



void wait()
{
 cout << "Preciona ENTER para continuar...";
 cin.ignore( std::numeric_limits <std::streamsize> ::max(), '\n' );
}

void calibCap(const int &a)
{
	cout << "se calibrara : "<< sheets[a].getName()<<endl;
	int ch=0;
	char op, op1;
	bool flag = true;
	unsigned long pw=2000;
	string tmp;
	ssc32 tar(UART_DIR);
	do{
		cout <<"Que motor se quiere calibrar : "<<endl;
		pw=Center_width;
		while(true){
			cin.clear();
			cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
			getline(cin, tmp);
			stringstream s(tmp);
			if(s >> ch){
				break;}
				cout <<"caracter invalido intentelo otra vez"<<ch<<endl;
```

```
                }
            if(ch_cero < ch && Max_num_ch > ch ){
                do{
                    //cout << "\x1B[2J\x1B[H";
                    cout <<"si desea salir pulse 'q' seguido de enter..."<<endl;
                    cout << "presiona '1'seguido de enter para cambiar de
direccion"<<endl;
                    cout <<"motor : " <<ch<<"\tpulso : "<< std::dec <<pw<<
endl<<" : ";
                    cin.get(op);
                    if(op == '1'){flag ^=true;}
                    if(flag){pw+=10;}else{pw-=10;}
                    tar.addMove(ch,pw,1000,100);
                    tmp = tar.back();
                    tar.run();
                    }while (op != 'q');

                sheets[a].addCommand(tmp);
                cout <<"lista "<<sheets[a].getName()<<endl;
                sheets[a].printData();
                }else{ch =0;cout<<"ch invalido"<<endl;}
            //cout << "\x1B[2J\x1B[H";
            cout <<"si desea salir pulse 'q' seguido de enter..."<<endl
            << "para calibrar otro motor enter....";
            cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
            cin.get(op1);
        }while(op1 != 'q');
}

int main(void)
{
   ssc32 lm(UART_DIR);
   loadCaps();
   cout <<"files ----"<<endl;
        cout << "Numero de Archivos :  " << sheets.size() << endl;
   for(unsigned int i =0; i< sheets.size() ;i++)
   printCap(i);
        runCap(1);
        time_t now = time(0);
        char* dt = ctime(&now);
        cout << now <<endl;
        cout << "La fecha y hora local es: " << dt << endl;
        while(1){
            cout << "nombre de la captura : ";
            string name;
            cin >> name;
            sheets.push_back(cap(name));
            hoja  = sheets.size()-1;
```

```
                cout <<"hoja :"<<hoja<<endl;
                cout <<sheets[hoja].getName()<<endl;
                calibCap(hoja);
                saveCap(hoja);
                printCap(hoja);
                runCap(hoja);
                }
return 0;
}
```

programa para video

```
/*Universidad Nacional Autónoma de México
 *IIMAS
 *Araujo Pino Alan Emir
 *Prueba de velocidad de captura y procesamiento con 3 filtros
 */
#include "opencv2/opencv.hpp"

#define FPS 24

using namespace cv;
using namespace std;

int main(int, char**)
{
   VideoCapture cap(0); // open the default camera
   if(!cap.isOpened())  // check if we succeeded
      return -1;
   cap.set(CV_CAP_PROP_FRAME_WIDTH,640);
   cap.set(CV_CAP_PROP_FRAME_HEIGHT,480);
   Mat edges;
   int i=0;
   Mat frame[FPS];
   //namedWindow("edges",1);
   while(1)
   {
      cap >> frame[i]; // get a new frame from camera

      cvtColor(frame[i], edges, CV_BGR2GRAY);
      GaussianBlur(edges, edges, Size(7,7), 1.5, 1.5);
      Canny(edges, edges, 0, 30, 3);
      //imshow("edges", edges);
      if(waitKey(30) >= 0) break;
      cout << i << endl;
      i++;
      if(i==FPS){break;i=0;}
   }
   // the camera will be deinitialized automatically in VideoCapture destructor
```

```
   return 0;
}
```

Programa para giroscopio

En el caso del giroscopio se necesitó leer y escribir en los registros del MPU-6050 los cuales pueden ser usados de manera más rápida y eficiente mediante macros y es por esto que en el archivo donde programé las funciones de la comunicación mediante I2C decidí poner la mayoría de los registros del dispositivo. Estos pueden ser encontrados en internet con relativa facilidad por lo que esta demás dar referencias exactas.

C_I2C_Funtions.c

```
/*Universidad Nacional Autónoma de México
 *IIMAS
 *Araujo Pino Alan Emir
 *Funciones del giroscopio
 */
#define MPU6050_AUX_VDDIO        0x01   // R/W
#define MPU6050_SMPLRT_DIV       0x19   // R/W
#define MPU6050_CONFIG           0x1A   // R/W
#define MPU6050_GYRO_CONFIG      0x1B   // R/W
#define MPU6050_ACCEL_CONFIG     0x1C   // R/W
#define MPU6050_FF_THR           0x1D   // R/W
#define MPU6050_FF_DUR           0x1E   // R/W
#define MPU6050_MOT_THR          0x1F   // R/W
#define MPU6050_MOT_DUR          0x20   // R/W
#define MPU6050_ZRMOT_THR        0x21   // R/W
#define MPU6050_ZRMOT_DUR        0x22   // R/W
#define MPU6050_FIFO_EN          0x23   // R/W
#define MPU6050_I2C_MST_CTRL     0x24   // R/W
#define MPU6050_I2C_SLV0_ADDR    0x25   // R/W
#define MPU6050_I2C_SLV0_REG     0x26   // R/W
#define MPU6050_I2C_SLV0_CTRL    0x27   // R/W
#define MPU6050_I2C_SLV1_ADDR    0x28   // R/W
#define MPU6050_I2C_SLV1_REG     0x29   // R/W
#define MPU6050_I2C_SLV1_CTRL    0x2A   // R/W
#define MPU6050_I2C_SLV2_ADDR    0x2B   // R/W
#define MPU6050_I2C_SLV2_REG     0x2C   // R/W
#define MPU6050_I2C_SLV2_CTRL    0x2D   // R/W
#define MPU6050_I2C_SLV3_ADDR    0x2E   // R/W
#define MPU6050_I2C_SLV3_REG     0x2F   // R/W
#define MPU6050_I2C_SLV3_CTRL    0x30   // R/W
#define MPU6050_I2C_SLV4_ADDR    0x31   // R/W
#define MPU6050_I2C_SLV4_REG     0x32   // R/W
#define MPU6050_I2C_SLV4_DO      0x33   // R/W
```

```
#define MPU6050_I2C_SLV4_CTRL     0x34   // R/W
#define MPU6050_I2C_SLV4_DI       0x35   // R
#define MPU6050_I2C_MST_STATUS    0x36   // R
#define MPU6050_INT_PIN_CFG       0x37   // R/W
#define MPU6050_INT_ENABLE        0x38   // R/W
#define MPU6050_INT_STATUS        0x3A   // R
#define MPU6050_ACCEL_XOUT_H      0x3B   // R
#define MPU6050_ACCEL_XOUT_L      0x3C   // R
#define MPU6050_ACCEL_YOUT_H      0x3D   // R
#define MPU6050_ACCEL_YOUT_L      0x3E   // R
#define MPU6050_ACCEL_ZOUT_H      0x3F   // R
#define MPU6050_ACCEL_ZOUT_L      0x40   // R
#define MPU6050_TEMP_OUT_H        0x41   // R
#define MPU6050_TEMP_OUT_L        0x42   // R
#define MPU6050_GYRO_XOUT_H       0x43   // R
#define MPU6050_GYRO_XOUT_L       0x44   // R
#define MPU6050_GYRO_YOUT_H       0x45   // R
#define MPU6050_GYRO_YOUT_L       0x46   // R
#define MPU6050_GYRO_ZOUT_H       0x47   // R
#define MPU6050_GYRO_ZOUT_L       0x48   // R
#define MPU6050_EXT_SENS_DATA_00  0x49   // R
#define MPU6050_EXT_SENS_DATA_01  0x4A   // R
#define MPU6050_EXT_SENS_DATA_02  0x4B   // R
#define MPU6050_EXT_SENS_DATA_03  0x4C   // R
#define MPU6050_EXT_SENS_DATA_04  0x4D   // R
#define MPU6050_EXT_SENS_DATA_05  0x4E   // R
#define MPU6050_EXT_SENS_DATA_06  0x4F   // R
#define MPU6050_EXT_SENS_DATA_07  0x50   // R
#define MPU6050_EXT_SENS_DATA_08  0x51   // R
#define MPU6050_EXT_SENS_DATA_09  0x52   // R
#define MPU6050_EXT_SENS_DATA_10  0x53   // R
#define MPU6050_EXT_SENS_DATA_11  0x54   // R
#define MPU6050_EXT_SENS_DATA_12  0x55   // R
#define MPU6050_EXT_SENS_DATA_13  0x56   // R
#define MPU6050_EXT_SENS_DATA_14  0x57   // R
#define MPU6050_EXT_SENS_DATA_15  0x58   // R
#define MPU6050_EXT_SENS_DATA_16  0x59   // R
#define MPU6050_EXT_SENS_DATA_17  0x5A   // R
#define MPU6050_EXT_SENS_DATA_18  0x5B   // R
#define MPU6050_EXT_SENS_DATA_19  0x5C   // R
#define MPU6050_EXT_SENS_DATA_20  0x5D   // R
#define MPU6050_EXT_SENS_DATA_21  0x5E   // R
#define MPU6050_EXT_SENS_DATA_22  0x5F   // R
#define MPU6050_EXT_SENS_DATA_23  0x60   // R
#define MPU6050_MOT_DETECT_STATUS 0x61   // R
#define MPU6050_I2C_SLV0_DO       0x63   // R/W
#define MPU6050_I2C_SLV1_DO       0x64   // R/W
#define MPU6050_I2C_SLV2_DO       0x65   // R/W
```

```
#define MPU6050_I2C_SLV3_DO       0x66   // R/W
#define MPU6050_I2C_MST_DELAY_CTRL 0x67   // R/W
#define MPU6050_SIGNAL_PATH_RESET  0x68   // R/W
#define MPU6050_MOT_DETECT_CTRL    0x69   // R/W
#define MPU6050_USER_CTRL         0x6A   // R/W
#define MPU6050_PWR_MGMT_1        0x6B   // R/W
#define MPU6050_PWR_MGMT_2        0x6C   // R/W
#define MPU6050_FIFO_COUNTH       0x72   // R/W
#define MPU6050_FIFO_COUNTL       0x73   // R/W
#define MPU6050_FIFO_R_W          0x74   // R/W
#define MPU6050_WHO_AM_I          0x75   // R

//#define XlAdres                         0x53
#define TenBitsAddress        0x00

        char rxBuffer[32];  // receive buffer
        char txBuffer[32];  // transmit buffer
        int opResult = 0;   // for error checking of operations
        int i2cHandle;

void openI2c(char *port, char address)
{
        // Create a file descriptor for the I2C bus
        i2cHandle = open(port, O_RDWR);

        // Tell the I2C peripheral that the device address is (or isn't) a 10-bit
        //   value. Most probably won't be.
        opResult = ioctl(i2cHandle, I2C_TENBIT,TenBitsAddress);

        // Tell the I2C peripheral what the address of the device is. We're going to
        //   start out by talking to the gyro.
        opResult = ioctl(i2cHandle, I2C_SLAVE,address);

        // Clear our buffers
        memset(rxBuffer, 0, sizeof(rxBuffer));
        memset(txBuffer, 0, sizeof(txBuffer));
        printf("Puerto on\n");
}

char readI2c(unsigned char a){
        txBuffer[0] = a; // This is the address we want to read from.
        //printf("manda\n");
        opResult = write(i2cHandle, txBuffer, 1);
        //printf("mandado\n");
        if (opResult != 1) {
        printf("No se mando el write\n");}
        opResult = read(i2cHandle, rxBuffer, 1);
        //printf("Part ID: %d\n", (int)rxBuffer[0]); // should print 105
```

```
	return rxBuffer[0];
}
char writeI2c(char addres_i2c,char value_i2c){
	//opResult = ioctl(i2cHandle, I2C_SLAVE, XlAdres);
	txBuffer[0] = addres_i2c;  // This is the address to read from.
	txBuffer[1] = value_i2c;
	opResult = write(i2cHandle, txBuffer, 2);

	if(readI2c(addres_i2c) != value_i2c)
	{	printf("No se escribio %d en %d\n",(int)value_i2c,(int)addres_i2c);
		return 0x01;}

	//printf("Part ID: %d\n", (int)rxBuffer[0]); // should print 229
	return 0x00;
	}

int twoCharToInt(const unsigned char a, const unsigned char b)
{	//Forma ab = c
	unsigned int temp0 = 0, temp1;
	temp0 = a;
	temp1 = temp0 << 8 ;
	temp1 +=(int)b;
	return temp1;
}
```

C_I2C_0.c

```
#include "C_I2C_Funtions.c"
#include <time.h>

/*Universidad Nacional Autónoma de México
 *IIMAS
 *Araujo Pino Alan Emir
 *Prueba de lectura para giroscopio
 */


#define port "/dev/i2c-2"
// define of own program
#define GyroAddress		0x68

int main (void)
{
	openI2c(port ,GyroAddress);
//
	printf("dir 0x6b : %d\n",(int)readI2c(0x6b));
//
	writeI2c(0x6b,0x01);
```

```
        system("sleep 4");
//
        while(1)
        {
        system("clear");
        printf("accel  x :
%d\n",twoCharToInt(readI2c(MPU6050_ACCEL_XOUT_H),readI2c(MPU6050_ACCEL_X
OUT_L)));
        printf("accel  y :
%d\n",twoCharToInt(readI2c(MPU6050_ACCEL_YOUT_H),readI2c(MPU6050_ACCEL_Y
OUT_L)));
        printf("accel  z :
%d\n",twoCharToInt(readI2c(MPU6050_ACCEL_ZOUT_H),readI2c(MPU6050_ACCEL_Z
OUT_L)));

        printf("gyro  x :
%d\n",twoCharToInt(readI2c(MPU6050_GYRO_XOUT_H),readI2c(MPU6050_GYRO_XO
UT_L)));
        printf("gyro  y :
%d\n",twoCharToInt(readI2c(MPU6050_GYRO_YOUT_H),readI2c(MPU6050_GYRO_YO
UT_L)));
        printf("gyro  z :
%d\n",twoCharToInt(readI2c(MPU6050_GYRO_ZOUT_H),readI2c(MPU6050_GYRO_ZO
UT_L)));

        usleep(100000);
        }
return 0;}
```