Instituto Tecnológico de Costa Rica

Área Académica de Ingeniería en Computadores

Taller de Programación
CE-1102

Grupo 2

Documentación
Proyecto 2

20%

Fecha de entrega: viernes 8 de noviembre, 4:30 pm

Profesor: Ing. Milton Villegas Lemus

Estudiantes:

Allan Cheng Liang
2019049482

Luis Pedro Morales Rodríguez
2017089395

II Semestre, 2019

I. Introducción

El presente proyecto se trata de integrar el desarrollo de software y hardware a la hora de construir un carro de juguete, el cual se pueda manejar mediante comandos que se envían desde una interfaz de usuario (Cliente) y los recibe el NodeMCU (Servidor). La parte de Hardware del proyecto sería la construcción del carrito, ya que se integró todos los componentes para la funcionalidad del carrito y la parte del software sería la programación del código para controlar el módulo NodeMCU. Para la parte de hardware, primero se implementó todos los componentes a una protoboard para poder llevar a cabo el primer paso del plan de pruebas, esto se debe a que a protoboard funciona para armar el circuito sin tener la necesidad de soldarlo, esto permite que se pueda hacer las pruebas de conexión hasta estar seguros de que los módulos estén conectados correctamente. Luego, se implementa la protoboard con todos sus componentes conectados al carrito de acrílico para poder comenzar con el plan de pruebas. Al finalizar con la implementación de los módulos a la protoboard, se comienza con la parte del software para que el NodeMCU pueda recibir los comandos y así el carrito realice las acciones. Se utilizó la versión de Arduino 1.8.10 para la creación del servidor (NodeMCU) y la versión 2.0 para el soporte del MPU9250(Sensor formado por acelerómetro, giroscopio y magnetómetro).

La interfaz gráfica utilizada es programada en Python, versión 3.7.3. Se utiliza la biblioteca tkinter para mayor facilidad, ya que la biblioteca trae funciones específicamente para la creación de la interfaz gráfica. En la interfaz el usuario puede agregar los comandos y mandarlos mediante sockets (Enviar mensajes a través de una red) para que el NodeMCU los reciba y se puedan ejecutar.

Se logró completar un 95% del proyecto, hace falta la función del sensor de temperatura.

II. Conclusiones

I. El 74LS164 nos permite controlar varias LED con un solo pin de entrada, y en vez de programarse usando digitalWrite(LedPin, HIGH) se usa shiftOut(), el cual permite hacer el shift de un byte completo en el registro enviando bits individualmente al bit más o significativo(MSB) o bien al menos significativo(LSB).

II. Se deben descargar los archivos necesarios para programar el ESP8266 de la placa NodeMCU y configurar ciertos elementos para que coincidan con las características de la placa en específico utilizada, al igual que se debe descargar las bibliotecas para el control de los módulos MPU9250 y ESP8266, todo esto en la versión 1.8.10 de Arduino.

III. Se concluye que es mejor utilizar cable de calibre 22 AWG para realizar las conexiones entre nodos, siempre y cuando el cable tenga las dimensiones aptas para realizar las conexiones, esto se debe a que al utilizar los "Jumpers" no se logra visualizar bien las conexiones y esto perjudica a la hora de verificar que se hayan hecho correctamente.

IV. En Arduino hay que definir el tipo de la función, eso quiere decir que hay que definir si la función devuelve un "string" (string), un entero (int), un flotante (float) o si no retorna ningún elemento (void). En el caso de las variables también hay que definir el tipo de variable, comprobado en la versión 1.8.10 de Arduino.

V. El operador lógico XOR es un OR exclusivo entre bits, el cual se utilizó para invertir los valores de los bits que controlan las luces LED, ya que permite invertir un valor si este se compara siempre con un 1 lógico y permite que el valor permanezca si se compara con un 0 lógico.

VI. El NodeMCU solo puede entregar señales digitales mediante sus salidas, así que para poder simular una señal analógica se debe ajustar los ciclos de trabajo de la señal cuadrada (pwm). Para poder regular el ancho del pulso de la señal, se utiliza el método analogWrite(pin,valor), el cual recibe como primer parámetro el PIN de salida y como segundo parámetro el valor del pwm (0-1023). El método analogWrite() es una función que ya posee el lenguaje.

VII. Soldar headers o bases para circuitos integrados en lugar de soldar las patillas de los circuitos integrados directamente a la placa preperforada disminuye el riesgo de dañar los circuitos porque se evita que estos sean sobrecalentados con la soldadura. Además, de esta forma se pueden volver a utilizar dichos componentes en otros proyectos sin necesidad de desoldarlos.

VIII. Utilizar el método *millis()*, en lugar del *delay()*, para permitir que una tarea se ejecute durante un determinado lapso, permite que el loop del sistema embebido se siga ejecutando continuamente, lo que favorece la fluidez del funcionamiento del sistema.

IX. Es importante utilizar reguladores de voltaje en el desarrollo de circuitos que pueden ser dañados si no son alimentados con tensiones constante. Estos dispositivos absorben el excedente de tensión y la disipa como calor, previniendo que oscilaciones en los niveles de energía puedan afectar el resto del sistema.

X. Los motores funcionan como generadores si mediante una fuerza externa se les empieza a mover, es decir, pueden convertir la energía mecánica en eléctrica. Por esta razón, si los motores están apagados y no tienen un sistema de detención mecánica, es importante proteger el circuito de las corrientes que estos puedan generar con un arreglo de diodos.

XI. Los registros de corrimiento permiten administrar varias salidas simultáneamente mientras que solo requieren del uso de un pin de la placa del microcontrolador. Este integrado recibe datos de manera serial y permite un acceso a ellos de forma paralela. Una característica muy importante para el manejo de proyectos con microcontroladores porque estos tienen un número finito de pines de salidas, pero con los registros de corrimiento es posible controlar un mucho mayor número de salidas.

XII. La función de un multímetro de medir continuidad es sumamente útil en el proceso tanto de armar el circuito en protoboard como para su soldadura en placa preperforada porque permite evaluar si las conexiones se hicieron correctamente o si existe algún cortocircuito indeseado entre dos puntos del circuito que no deberían de estar conectados.

III. Bitácora

Fecha: miércoles 16 de octubre.

Actividad: se realizó la primera compra de los componentes necesarios para la realización del proyecto. Los siguientes componentes fueron comprados en la tienda MicroJPM: driver para motor DC L298N, fotocelda, carro con 2 llantas traseras impulsadas por motores DC de 3V-6V y una llanta delantera de movimiento libre, regulador de voltaje LM7805, MPU-9250 genérico, resistencias de diversos valores, jumpers. Y los siguientes componentes fueron comprados en la tienda virtual ElectroTEC: registro de corrimiento 74LS164 y Node MCU.

Fecha: miércoles 23 de octubre.

Actividad: se descargaron los archivos necesarios para que el entorno de Arduino pueda programar el ESP8266 de la placa NodeMCU. Para lograr esto se debió de introducir el "link" de la dirección [4] en la que se encuentra dicho paquete en el campo de *Additional Board Manager URLs* ubicado en la ventana de *Preferences* en el IDE de Arduino. Posteriormente, se debió de acceder a la pestaña de Tools, luego en Boards/Boards Manager, buscar el paquete respectivo de la placa NodeMCU e instalar dichos archivos. Finalmente, se configuraron ciertos elementos para que coincidieran con las características de la placa en específico utilizada. Por ejemplo: el baud rate de la comunicación en 115200 badios por segundo, tamaño de la memoria Flash en 4M (1M SPIFFS) y la frecuencia de CPU en 80 MHz.

Seguidamente, para poder compilar el código base proporcionado como parte del enunciado del proyecto, se debieron de descargar un par de librerías necesarias para el control del ESP8266 [5] y del MPU9250 [6] de los repositorios oficiales para el manejo del entorno de Arduino. Estos archivos se debían de colocar dentro de la carpeta de librerías de Arduino. En caso de no realizar esta instalación correctamente, se retornaba un error debido a la ausencia de los archivos referenciados en el código.

Fecha: viernes 25 de octubre.

Actividad: Se consideró que, para seguir avanzando con el proyecto, primero se debería de entender el funcionamiento del software y el hardware y los principios sobre los cuales están estructurados. Para lograr esto, primero se investigó acerca de algunos fundamentos sobre los cuales está basado el funcionamiento del circuito, por ejemplo, la comunicación I2C entre el NodeMCU y el MPU9250 [8]. Además, se buscó información acerca de la implementación de servidores y su comunicación con el cliente mediante sockets. [14]

Además, se analizó la estructura del código base de Arduino. Se investigó acerca de algunos métodos preconstruidos cuyo significado no se conocía; por ejemplo, funciones como el shiftOut() , millis(), map() y analogWrite(). La explicación de cómo implementar dichas funciones se obtuvo del sitio oficial de Arduino. [7]

Fecha: sábado 26 de octubre.

Actividad: Se continuó con la investigación de información relevante a la estructura del proyecto. En esta ocasión, se buscó la familiarización con el funcionamiento de los circuitos integrados y demás componentes de hardware. En primer lugar, se buscó la hoja de datos del L298 [16], el cual corresponde al driver de los motores de corriente directa a utilizar. Fue importante reconocer la funcionalidad de cada uno de los pines del circuito integrado, los cuales se representan en la Figura 1, especialmente reconocer los pines que representan las entradas y los habilitadores. Esto puesto a que dichos pines son los que se conectarán a algunas salidas del NodeMCU y del shift register.

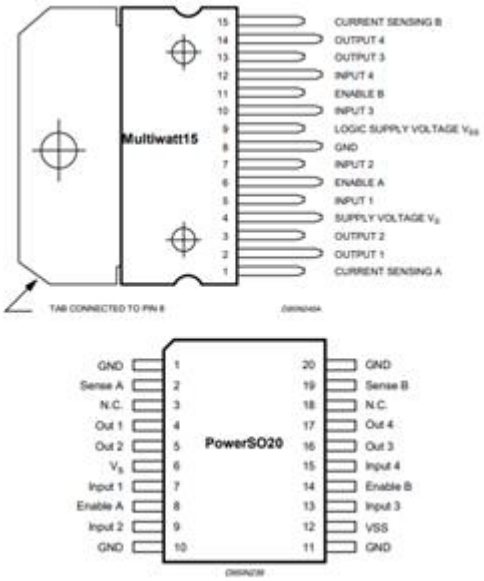


Figura 1. Pines de conexión del L298.

A su vez, se buscaron las datasheets del shift register [1] y el LM7805 [2], para igualmente reconocer cuáles son sus entradas y salidas. En el diagrama especificado en el enunciado del proyecto, se exponen las posiciones de los pines del NodeMCU tal cual son en la tarjeta real, por lo que no fue necesario verificar estas posiciones en su datasheet, sin embargo, esta también se buscó para futuras referencias [3].

Fecha: lunes 28 de octubre.

Actividad: Una vez reconocidos los pines de todos los componentes a implementar, se procedió a probar cada componente por separado para verificar su funcionalidad. En primer lugar, se prueba el driver L298 utilizando un microcontrolador Arduino Uno. Se conectaron los *inputs* del integrado a las salidas digitales del Arduino, los *enables* a salidas digitales con pwm y los *outputs* se conectaron a los dos motores a utilizar.

Se generó un código en Arduino que, a partir de datos enviados por el monitor serial, buscara controlar los motores en todas las direcciones posibles. Para hacer que uno de los motores usados se mueva, se debe tener una diferencia de tensión entre sus terminales de 5V. La polaridad de esta caída de tensión determina la dirección de giro del motor. Para lograr esto, se debe configurar un *input* del *driver* en *HIGH* y el otro *input* en *LOW.* Para que el motor deje de girar, de haber una caída de tensión de 0V entre sus terminales por lo que bastó con colocar ambos *input* en *LOW.*

Además, con las salidas de pwm, se logró regular la velocidad de los motores. Se determinó que un pwm=500 sería el adecuado para utilizar como límite inferior de la velocidad de los motores. Un pwm inferior a este valor no garantiza el movimiento de los mismos sobre una superficie.

Seguidamente, se buscó probar el funcionamiento del *shift register* con el método *shiftOut()* de Arduino. Este método permite hacer el shift de un byte completo en el registro enviando bits individualmente al bit más significativo (MSB) o al menos significativo (LSB) de dicho registro con cada variación del reloj, de manera que con cada invocación del método, habrán 8 *toggles* del reloj para que se reemplace el byte completo. El método recibe cuatro parámetros: el pin de salida de la placa de dónde se envían los bits, el pin del reloj que controla el shift, el orden de los bits (primero el más o el menos significativo) y el byte que se introducirá al registro. Para esto se generó otro código en Arduino que simplemente enviara una serie de bytes al registro, mediante el método anterior, separados por un cierto delay, de manera que hubiera tiempo de medir con un multímetro las salidas del registro y verificar que se estuvieran dando los cambios deseados en cada bit.

Finalmente se probó el regulador de tensión de 5V. Para esto se armó un circuito resistivo simple y se conectó la entrada a una batería de 9V y luego se midió la tensión en una resistencia colocada entre la salida del integrado y la tierra del circuito, para verificar que efectivamente se entregaran 5V.

Problemas:

- Se empleó una protoboard antigua que, probablemente de tanto uso, tenía algunas conexiones internas dañadas por lo que se complicó el proceso de pruebas. En ocasiones el circuito funcionaba y en otras no. Se determinó que el problema era la condición de la *protoboard* ya que, al medir tensión en puntos importantes, si esta no correspondía a la que debería, bastaba con mover un poco el cable de ese nodo para que funcionara correctamente. Se decidió conseguir una *protoboard* diferente, en mejor estado.

Fecha: martes 29 de octubre.

Actividad: Se probó que el código de Arduino, proporcionado como parte del enunciado del proyecto, al ser cargado al NodeMCU generará un servidor y este

pudiera establecer una conexión exitosa con el cliente, para cuya creación se utilizó el código de Python, también proporcionado desde un inicio.

Problemas:

· Al configurar la conexión a la red doméstica que se utilizó, se emplearon valores de IP erróneos. Se colocó la misma IP de la computadora en el código que genera el servidor, por lo que la comunicación con el NodeMCU no iba a ser lograda ya que la computadora intenta hacer una conexión consigo misma. Para que la conexión con el NodeMCU fuera exitosa, se cambió la dirección IP por otra, cuyo valor sea difícilmente usado por otro aparato (como el 200, por ejemplo). De esta forma sí se logró comunicar ambos dispositivos.

Fecha: miércoles 30 de octubre

Actividad: se procedió a armar al circuito en la *protoboard,* tal como se indica en la Figura 2.

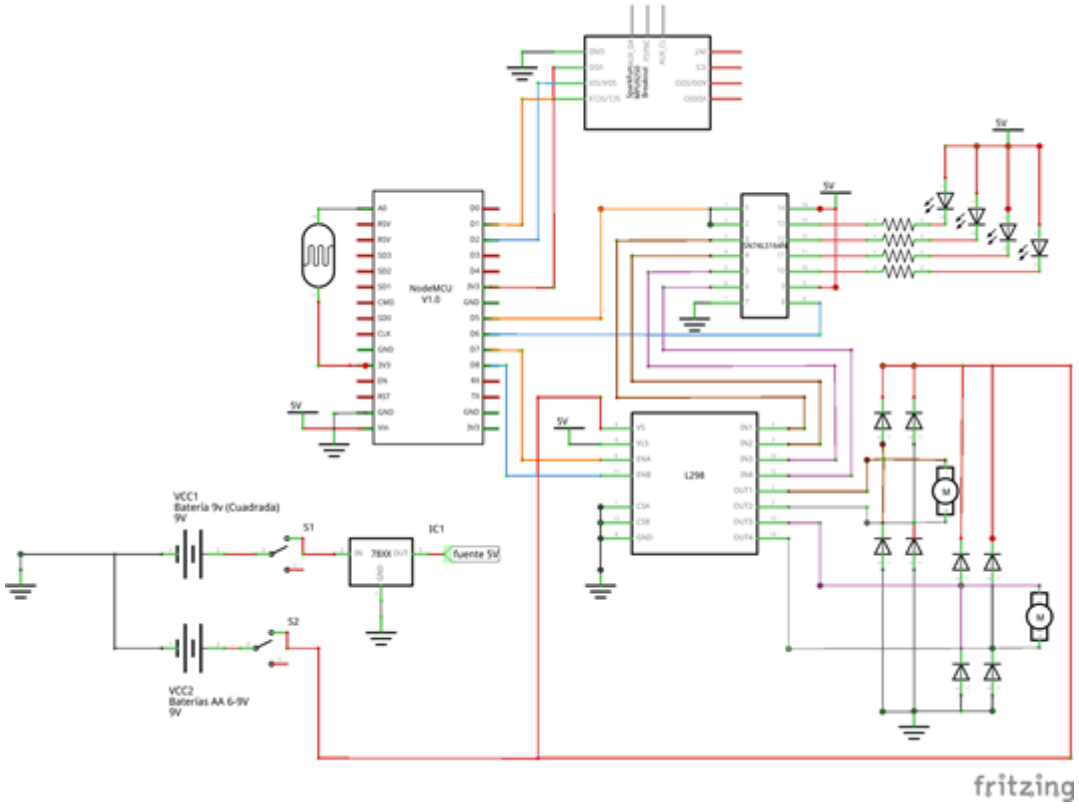


Figura 2. Diagrama del circuito a implementar

Problemas:

- Luego de armar todo el circuito y alimentarlo, se notó que había una caída de tensión importante en el nodo que alimenta a la mayoría de componentes, el

cual teóricamente debería ser de 5V. Se revisó que las conexiones estuvieran correctas y luego se decidió cambiar la batería de 9V que alimenta el regulador de tensión por una nueva, ya que la que se estaba utilizando ya había disminuido un poco su nivel de tensión (a cerca de 6,5 V). Con la nueva batería, la tensión de salida del regulador sí cambió a los 5V deseados.

- El L298N no está diseñado para colocarse en protoboard puesto a que las patillas, por su distribución y estructura, no calzan dentro de orificios que forman los nodos de dicha placa. Se determinó que la mejor solución a este problema sería doblar las patillas, utilizando un alicate con puntas de pinza, con especial cuidado e intención para que pudieran calzar todas las patillas en diferentes nodos (dejando el surco central de la protoboard en medio de las 2 filas de patillas del circuito integrado).
- El NodeMCU tampoco está diseñado para colocarse en protoboard. En este caso, se decidió utilizar cables con terminales macho-hembra para lograr conectar, por un lado, a los pines de la placa con los pines hembra de los cables, y por el otro lado, los pines macho de los cables a los nodos de la protoboard deseados.
- Se utilizaron “jumpers” para realizar las conexiones entre nodos de la protoboard. Una vez que se terminó de construir el circuito del diagrama, se notó que la calidad visual del mismo era pobre; esto en el sentido de que, al estar tan cargado de “jumpers”, era difícil revisar las conexiones y verificar que estas se hayan hecho de forma correcta.
- Se tomó la decisión de rearmar el circuito, pero cortando cable de calibre 22 AWG de las dimensiones aptas para hacer cada una de las conexiones al ras de la *protoboard.*

Fecha: jueves 31 de octubre

Actividad: Se modifica el código del programa que se carga al NodeMCU, para crear las funciones de control de motores en las cuatro direcciones: hacia el frente, hacia atrás, hacia la derecha y hacia la izquierda. Estas funciones utilizan el método *shiftOut()* , probado anteriormente, sin embargo, cabe destacar que el byte de datos que se envía, tomando el bit menos significativo como el primero de izquierda a derecha (posición 0), considera la siguiente distribución para el control del driver L298 y LEDs:

- Bit 0: control de input 1 del driver
- Bit 1: control de input 2 del driver
- Bit 2: control de input 3 del driver
- Bit 3: control de input 4 del driver
- Bit 4: control de LED rojo trasero derecho
- Bit 5: control de LED rojo trasero izquierdo
- Bit 6: control de LED blanco delantero derecho

- Bit 7: control de LED blanco delantero izquierdo

Para las funciones de movimiento hacia adelante y hacia atrás se envía un byte que mueva ambas llantas en la misma dirección, utilizando las siguiente configuraciones: B10101111 para el movimiento hacia adelante y B01011111 hacia atrás. Para los movientos de giro laterales, se configura el byte para que solo se mueva una de las dos llantas: la llanta derecha para el movimiento hacia la izquierda (B10001111) y la llanta izquierda para el movimiento hacia la derecha (B00101111).

Para regular la velocidad de los motores, se debe controlar el nivel de tensión que le llega a los pines de *enable* del L298, el cual debe oscilar en un rango entre 0V y 5V. Sin embargo, como las salidas del NodeMCU sólo pueden entregar señales digitales, una señal analógica se puede simular ajustando los ciclos de trabajo de la señal cuadrada (pwm). Un mayor ciclo de trabajo representaría un nivel de tensión más cercano a los 5V, y viceversa. Entonces, para poder regular el ancho del pulso de la señal, se utiliza el método analogWrite(), el cual recibe como primer parámetro el pin de salida de la placa y como segundo parámetro el valor de pwm (entre 0 y 1023).

De esta forma, se puede cambiar la velocidad del movimiento del carrito utilizando el comando ***pwm:valor;*** con el valor en un rango entre 500 y 1023, puesto a que, como se mencionó anteriormente, se estableció 500 como límite inferior de pwm para que los motores logren hacer que el carrito se mueva sin problemas. Además, si se digita un número negativo en el comando anterior (entre -1023 y -500), las llantas se mueven hacia atrás.

Los comando de dirección establecidos son ***dir:valor;*** con el valor siendo alguno de los siguiente dígitos: -1,0,1. En el caso de que sea 0, se mueve hacia el frente con un pwm de 800, en el caso de -1 gira hacia la izquierda y con 1 gira a la derecha.

Fecha: viernes 1 de noviembre

Actividad: se generaron las funciones que permiten el manejo de las luces. Para lograr esto se utilizan los 4 bits más significativos del byte que se está manejando (4 bits de la derecha). El control de las luces se hace utilizando lógica negativa, es decir, con un 1 lógico se apagan y con un 0 lógico se encienden. Esto puesto a que una terminal de los LEDs siempre está conectada a la alimentación de 5V, por lo que para que estos se prendan, debe haber una caída de tensión que se logra con el 0 lógico. Los comandos para el control de las luces son ***lb:valor;*** y ***lf:valor;*** para las luces traseras y delanteras respectivamente. Los comandos para controlar las direccionales son ***ll:valor;*** y ***lr:valor;*** para el control de las direccionales izquierdas y derechas respectivamente.

Para invertir los valores de los bits que controlan las luces, se utilizó el operador lógico XOR, que permite invertir un valor si se compara siempre con un 1 lógico y permite que un valor permanezca igual si se compara con un 0 lógico.

Problemas:

- En un principio se intentó utilizar *delay()* para el manejo de las funciones de direccionales y permitir que las luces estuvieran prendidas o apagadas por un cierto lapso. Sin embargo, la implementación no fue exitosa. En su lugar, se decidió utilizar la función *millis()* para permitir que cada cierto tiempo se invirtieran los valores de las luces.
- Al finalizar la ejecución de las funciones de direccionales, en algunos casos, la luz no quedaba en el valor inicial; es decir, en veces quedaba encendida cuando inicialmente estaba apagada. Para solucionar el problemas se definieron banderas que indican la condición actual de las luces y analiza si debe invertirlas o dejarlas igual al finalizar la función.
- Se retornaba un error cuando se intentaba enviar un mensaje que incluyera algún caracter que no estuviera dentro de los primeros caracteres imprimibles del código ASCII (del 32 al 127). Se cambió el mensaje por uno compatible con la comunicación.

Fecha: sábado 2 de noviembre

Actividad: se inicia con la soldadura de los componentes a una placa preperforada de 5cm*7cm. Se soldaron las patillas de los headers sobres los cuales se colocará el NodeMCU y la base sobre la que posicionará el shift register. Además, se solda directamente a la placa el regulador de tensión y se añaden algunos headers extra donde se conectaran terminales de jumpers que conecten a otros elementos como los switches, baterías, LEDs o el mismo MPU9250.

Problemas:

- Hacer algunos puntos de soldadura que estén cercanos o adyacentes a otros ya existentes representó todo un reto. Requiere de habilidad y precisión el poder hacer una buena soldadura con tan poco espacio para maniobrar el cautín y evitar generar cortocircuitos indeseados. En algunas ocasiones se tuvo que desoldar puntos de soldadura cuyo estaño terminaba tocando otros puntos.

Fecha: domingo 3 de noviembre

Actividad: se continúa con el proceso de soldadura. Esta vez, se unen todos los puntos que deberían interconectados. En algunas ocasiones, si los pines estaban cercanos entre sí, se unían mediante puentes de estaño. Si las conexiones estaban más alejadas entre sí entonces se conectaban mediante cable de calibre 20 AWG (el mismo utilizado para armar el circuito en la protoboard), cuyas puntas se soldaban a los puntos deseados.

Problemas:

- Esta etapa fue aún más retadora que la anterior. El reducido espacio en la placa preperforada complicó mucho las cosas y se debió de tener extremo cuidado y orden al realizar cada conexión. Cada vez que se conectaban 2 nuevos puntos, se medía continuidad con un multímetro para verificar que efectivamente se hayan conectado.

Fecha: lunes 4 de noviembre

Actividad: se procedió a continuar con la programación de las funciones de movimiento del carrito que no habían sido terminadas. Para esto, primero se conectó el circuito MPU9250 al resto del sistema para verificar su funcionamiento. Se configuraron las funciones de getRoll(), getYaw() y getPitch() que obtienen los valores del nivel de alabeo, de orientación con respecto al norte y de inclinación o elevación, respectivamente. A estos valores se acceden utilizando la biblioteca del MPU empleada, mediante los métodos my IMU.roll, myIMU.yaw y myIMU.pitch.
Seguidamente, se programó la función que registra la máxima aceleración vectorial conseguida mediante la fórmula$a = \sqrt{a_x^2 + a_y^2}$, donde a_xy $a_y$representan la aceleración en x y la aceleración en y, respectivamente. Para acceder a estos valores se utilizó la misma biblioteca con los métodos myIMU.ax y myIMU.ay .
Después, se prosiguió a programar la función getSense() que permite obtener el nivel de iluminación en el ambiente en un rango de 0 a 100. Para acceder al nivel de tensión sensado en el pin A0 del NodeMCU, se utilizó el método analogRead(), que retorna un valor entre 0 y 1023. Finalmente, se utilizó la función map(), para convertir ese valor en un rango de 0-1023 a uno de 0-100.
Problemas:

- En ocasiones, el magnetómetro del MPU9250 se descalibraba y retornaba valores de direcciones completamente distintas a las reales. En estos casos, se procedió a resetear y la placa y reiniciar el sistema de manera que se volviera a calibrar el magnetómetro.
- En veces, el MPU9250 dejaba de funcionar y arrojaba valores de yaw, pitch y roll todos en cero.También se procedió a reiniciar el sistema en estos casos.

Fecha: martes 5 de noviembre

Actividad: se programaron las funciones de telemetría del movimiento especial (el carro hace un recorrido en cuadrado) y la búsqueda del Norte. Para el movimiento especial se configuró que los movimientos hacia el frente y de giro se ejecutaran durante un cierto lapso, de manera que el recorrido se diera y el carro siempre llegue al mismo punto de partida. La búsqueda del Norte emplea la función de getYaw() para obtener la dirección en cada momento y cuando se encuentre en un rango de $\pm$15 grados con respecto al Norte, se detenga. En esta función el carro entra en un ciclo

que ejecuta movimientos de avance, giro y retroceso hasta que encuentre la orientación deseada.

Problemas:

- Nuevamente se tuvieron problemas con los valores de orientación que arrojaba el MPU9250. En ocasiones se descalibraba fuertemente.
- El lenguaje de programación C++ no logra interpretar la sintaxis de desigualdades matemáticas del tipo 0<=x<=10. En su lugar, se deben de utilizar desigualdades simples relacionadas con operadores AND (&&).

Fecha: miércoles 6 de noviembre.
Actividad: se programó la función de TurnTime, que genera un giro del carrito de 360 grados (vuelve a tener la orientación que tenía en el momento inicial) y retorna el tiempo de giro, mediante la implementación del método *millis()*. Seguidamente, se programó la función *Infinite()* que permite el movimiento en un recorrido en forma de ocho invertido o de símbolo de infinito, llegando luego al punto de partida. Para esta función que programaron secuencias de movimiento (giro hacias ambos lados, hacia adelante) que se ejecutan siempre una cantidad de tiempo específica y necesaria para que se llegue al mismo punto del cual se salió.
Finalmente, se programó la función de *Diag(),* la cual permite hacer un diagnóstico de las condiciones del sistema y verificar si se encuentra en buen o mal estado. Para el diagnóstico del MPU9250, se generan lecturas de los datos y se verifican que sean diferentes de 0.0 en diversas posiciones, lo cual indica que el circuito está funcionando. Y para el diagnóstico de los motores, se ejecutan todos los movimientos y se verifica si hay una aceleración del carro posterior.

Fecha: jueves 7 de noviembre.
Actividad: se cambió la configuración de la red que se estaba utilizando, la cual era una doméstica, por los datos de la red generada con un hotspot de celular.

IV. Recomendaciones

I. Se recomienda utilizar herramientas en buen estado, que se utilicen diariamente, ya que esto puede producir un error a la hora de trabajar en el proyecto.

II. Siempre mantener la punta del cautín limpia, debido a que si la punta se llega a oxidar entonces puede contaminar la soldadura. El cautín se puede limpiar utilizando una esponja húmeda o viruta metálica, antes de ser calentado.

III. A la hora de soldar un módulo o componente, siempre tener cuidado de que no haya un mal aspecto en la soldadura. Esto puede producir que la corriente no logre pasar correctamente o que se junten varias conexiones.

IV. Si se van a hacer conexiones muy complejas en una protoboard se recomienda utilizar cable conductor para que las conexiones sean en forma ordenada, sin que ningún cable se suelte y así poder verificar de forma eficaz. En cambio, con los “Jumpers” puede llegar a ser muy tedioso porque es muy fácil que los cables de suelten, no hay orden porque al utilizar “Jumpers” no se pueden poner a ras de la protoboard.

V. Se recomienda utilizar header o bases para integrados en el proceso de soldadura del circuito en lugar de soldar las patillas de los componentes directamente a la placa. Esto porque se disminuyen riesgos de dañar los integrados si se sobrecalientan y además permite retirarlos de la placa y reutilizarlos en otro circuito sin necesidad de retirar puntos de soldadura.

VI. Cuando se hacen las pruebas con la protoboard, siempre es bueno contar con un multímetro para poder saber de cuantos voltios es la tensión de la batería, medir continuidad entre dos puntos, medir resistencias, etc.

VII. Si se utiliza el MPU9250 genérico, como el que se empleó para este proyecto, se recomienda verificar que el magnétometro esté bien calibrado antes de ejecutar alguna función que lo utilice. Esto puesto a que, en ocasiones, puede dar problemas de calibración.

VIII. Se recomienda el uso del método *millis()* en lugar del *delay()* para ejecutar tareas en un cierto intervalo definido dentro de un sistema embebido, porque no interrumpe la ejecución continua del sistema, como sí lo hace el delay(). De esta forma, se puede continuar con la ejecución del *loop* de manera normalizada, y cuando se llegue a un determinado tiempo, detener la tarea en acción o propiciar algún otro cambio.

IX. Se recomienda el uso de operadores bit a bit como el XOR para modificar paquetes de datos en bytes porque permite invertir los bits, si se compara con un 1 lógico, o dejarlo igual si se compara con un 0 lógico, según sea requerido.

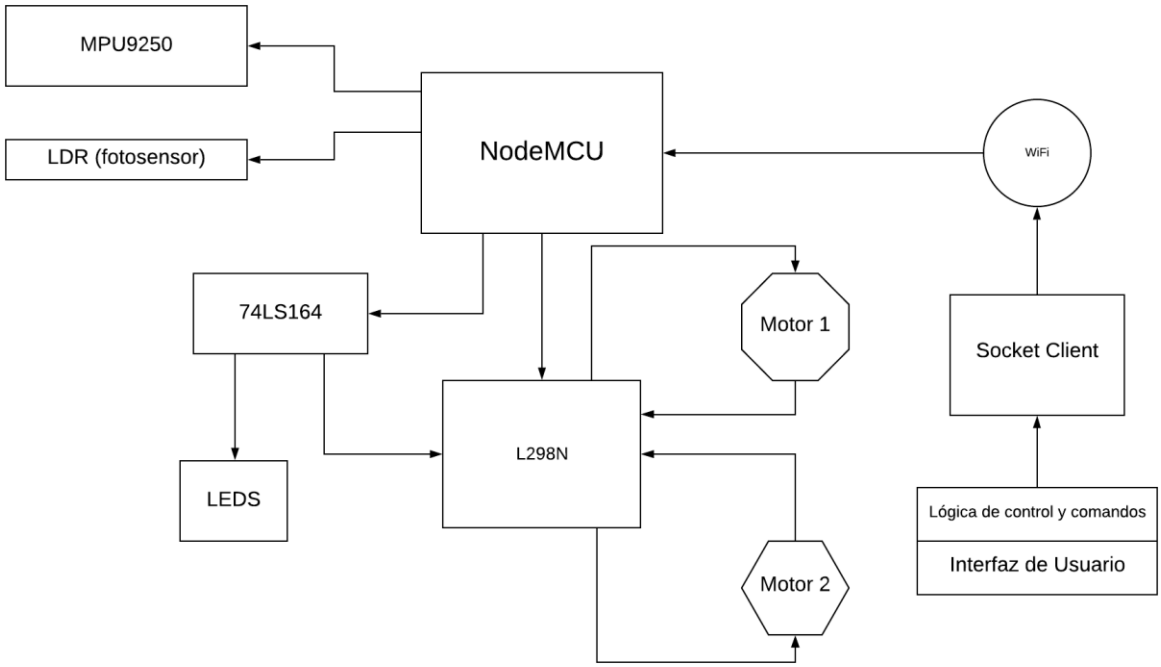
X. Para el desarrollo de proyectos de esta índole, se recomienda utilizar baterías recargables de manera que no sea necesario estar comprando nuevas baterías cada vez que se descargan las que se estén usados.

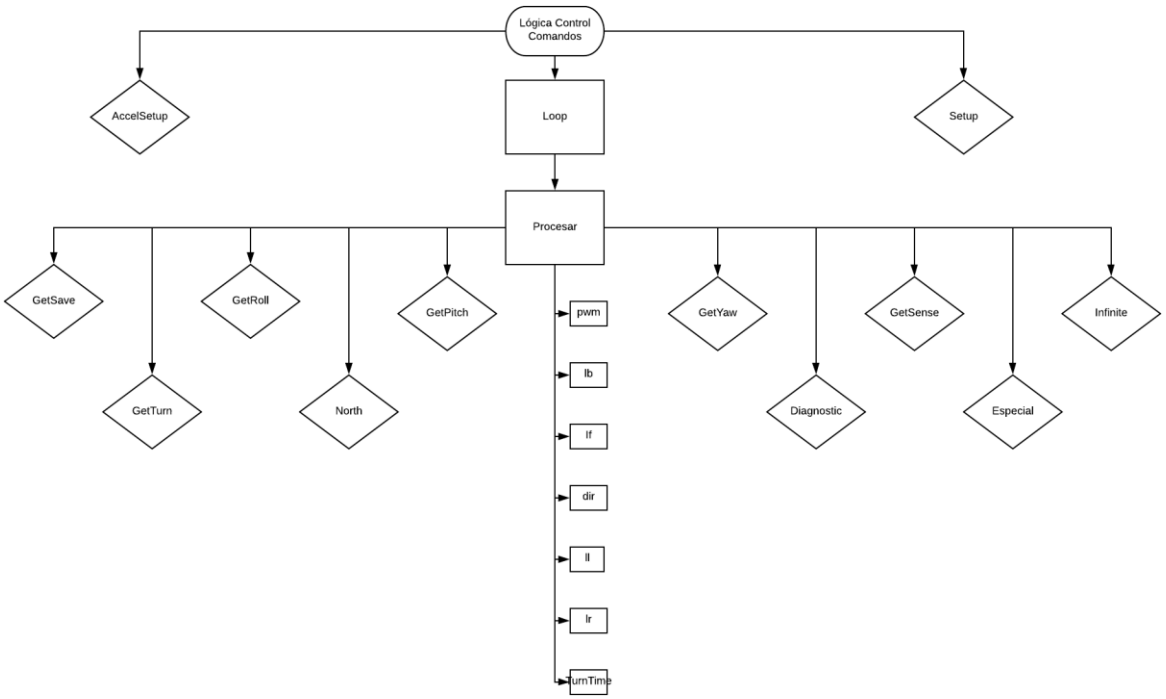
XI. Se recomienda no utilizar caracteres que no estén dentro de los primeros 128 caracteres código ASCII, como las letras tildadas, en la comunicación mediante el socket ya que no son reconocidos.

XII. Se recomienda el uso de registros de corrimiento para aplicaciones con microcontroladores en las que se requiera administrar varias salidas a la vez. El registro de corrimiento permite una entrada de datos de manera serial, por lo que solo se requiere del uso de un pin de la placa, mientras que se pueden acceder a todas las salidas del registro de manera paralela.

V. Análisis de Resultados

I. Diagrama de módulos





II. Plan de pruebas

Funcionamiento del integrado 74LS164: para verificar si el registro de corrimiento funcionaba correctamente, se procedió a construir el circuito de la Figura 3, mediante el cual se enviaban bits mediante el pin 2 del Arduino, mientras que el pin 3 servía de reloj. El programa cargado al microcontrolador, básicamente consistía en el envío de bytes utilizando el método *shiftOut().* Para verificar que los bytes fueran enviados correctamente, se medía la tensión en las 8 salidas del registro con un multímetro y se comparaban los valores con la secuencia de bits que en teoría se enviaba.

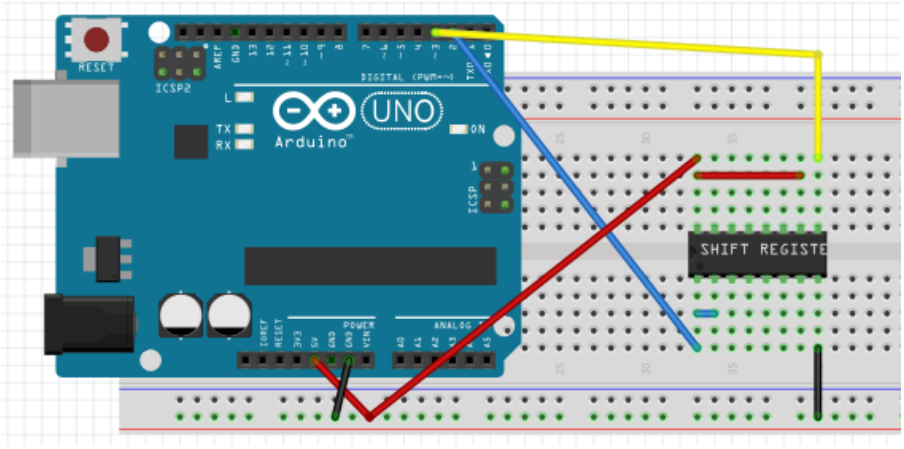
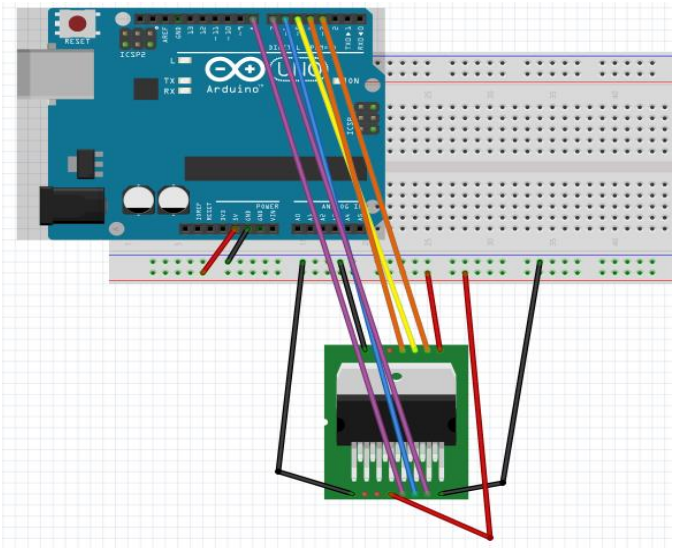


Figura 3. Diagrama del circuito para la prueba del 74LS164

Funcionamiento del integrado L298N: el correcto funcionamiento del driver de los motores se hizo mediante el circuito que se ilustra en la Figura 4. Esta configuración permite que el Arduino envíe señales cuadradas con modulación del ancho de pulso (pwm) de los pines 5 y 6 del Arduino a los pines habilitadores del integrado. Además, envía señales en ALTO o BAJO a los pines de *input.* Mediante el programa en Arduino se configuran los valores de la tensión de los pines de salida del driver (que controlan



la velocidad de los motores), utilizando el pwm, el cual se configura mediante el método analogWrite().

Figura 4. Diagrama del circuito para la prueba del L298N

Funcionamiento del integrado LM7805: para probar que el regulador de tensión funciona correctamente, se construyó el circuito de la Figura 5 y luego se midió la tensión en la resistencia, la cual debería de tener un valor muy cercano a 5V.

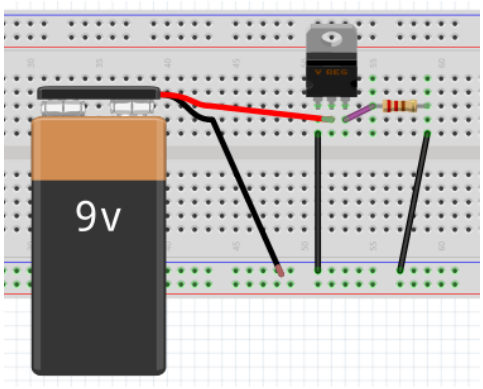


Figura 5. Diagrama del circuito para la prueba del LM7805

Circuito Soldado: una vez terminado de soldar todo el circuito en la placa preperforada, se aseguró que todas las conexiones se realizaron correctamente midiendo la continuidad con un multímetro entre todos los pines que en teoría deberían de estar conectados entre sí.

Funcionamiento del MPU9250: Se probó el módulo utilizando un código en Arduino proporcionado por el asistente del curso que imprimía secuencialmente todos los valores de yaw, pitch y roll. Para probar el correcto funcionamiento, se manipulaba el circuito, moviéndolo y girándolo en diversas direcciones para observar las variaciones de los valores.

Roles:

-Líder: En este Proyecto decidimos que el rol de líder lo manejara Luis Pedro para una mayor organización, esto debido a que las reuniones grupales para la realización del proyecto iban a ser en su apartamento y los avances de proyecto, componentes y herramientas de trabajo se iban a dejar en el apartamento.

El líder organizó las reglas de trabajo para poder realizar de forma correcta el proyecto, con orden y aptitud. Las reglas propuestas son:

1. Ser puntual con las horas establecidas para la realización del proyecto: Se establecieron horarios por días para adelantar el proyecto e intentar aclarar dudas, ya que dos mentes piensan mejor que una.

2. Mostrar interés en el trabajo: Debe de llegar a las reuniones sabiendo que se va a investigar o que se va a hacer, no puede llegar perdido ya que hay que sacar el mejor provecho del tiempo para completar todo el proyecto. También si se tienen dudas, puede contar con las tutorías de Taller para salir de eso de una vez

3. Completa correctamente su parte del trabajo: Hay contenidos del proyecto que se dejaban por partes para poder adelantarlo lo máximo posible, así que la meta era llegar a la siguiente reunión con el trabajo completo

4. Siempre tener presenta la comunicación: No ignorar los mensajes, ya que es importante saber cómo va el progreso del proyecto, si tiene alguna duda, tal vez el otro compañero la pueda resolver.

5. Respetar las horas de clase y estudio del compañero: Es un trabajo complejo, pero al igual que usted, su compañero también cuenta con horas de clase (Pueden tener un horario muy diferente). Entonces no presionar al compañero, para eso están las horas de reunión.

Miembro de desarrollo: El encargado del rol de miembro de desarrollo es Allan.

VI. Evaluación Compañero

Rubro	Porcentaje	Luis Pedro Morales Rodríguez	Li Hao Allan Chen Liang
Puntualidad a la hora de reunirse	**2,5%**	2,5%	2%
Asiste a todas las reuniones programadas en la semana	**2,5%**	2,5%	0%
Completitud de su parte de trabajo	**5%**	5%	0.5%
Muestra interés en el trabajo (Ir a tutorías, hacer horas extra, pulir el trabajo)	**5%**	5%	0.5%
Tener Presente una buena comunicación (Escuchar ideas y siempre estar atento a los mensajes del compañero)	**2.5%**	2,5%	1%
Respetar las horas de clases del compañero y sus horas de estudio (Organización de Tiempo)	**2.5%**	2,5%	2%
	20%	**20%**	**6%**

VII. Bibliografía

[1] Serial-In Parallel-out shift register. Esi. https://www.esi.uclm.es/www/isanchez/apuntes/ci/74164.pdf

[2] LM78XX/LM78XXA 3-Terminal 1A Positive Voltage Regulator. FairChild Semiconductor. http://ee-classes.usc.edu/ee459/library/datasheets/LM7805.pdf

[3] DATASHEET NodeMCU. Esploradores. https://www.esploradores.com/datasheet-nodemcu/

MPU-9250 Product Specification(20 Junio,2016). Invensense. http://www.invensense.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf

[4] https://arduino.esp8266.com/stable/package_esp8266com_index.json

[5] https://github.com/esp8266/Arduino

[6] https://github.com/sparkfun/SparkFun_MPU-9250-DMP_Arduino_Library

[7]Languaje referencee, https://www.arduino.cc/reference/en/

[8] Llamas, Luis.(18 Mayo,2016).El bus I2C en arduino.Luis Llamas.https://www.luisllamas.es/arduino-i2c/

[9] Llamas, Luis.(26 Septiembre,2016).Usar Arduino con los IMU de 9DOF MPU-9150 y MPU-9250.Luis Llamas. https://www.luisllamas.es/usar-arduino-con-los-imu-de-9dof-mpu-9150-y-mpu-9250/

[10] Valle Hernandez, Luis del. NodeMCU tutorial paso a paso desde cero. Programarfacil. https://programarfacil.com/podcast/nodemcu-tutorial-paso-a-paso/ s.f

[11] Martin, German. Como programar NodeMCU con el IDE de arduino. Programarfacil. https://programarfacil.com/esp8266/como-programar-nodemcu-ide-arduino/ s.f

[12] Osgeld. The74HC164 Shift Register and your arduino. Instructables circuits. https://www.instructables.com/id/The-74HC164-Shift-Register-and-your-Arduino/ s.f

[13] Nedelkovski, Dejan.(11 Septiembre,2017). Arduino DC Motor Control tutorial - L298N. How to mechatronics. https://howtomechatronics.com/tutorials/arduino/arduino-dc-motor-control-tutorial-l298n-pwm-h-bridge/

[14] Rueda Barranco, Miguel(Junio,1996). Doctorado en informática, Sockets: Comunicación entre procesos disturbios. http://es.tldp.org/Universitarios/seminario-2-sockets.html

[15] ShifOut(). Arduino Referencee https://www.arduino.cc/reference/tr/language/functions/advanced-io/shiftout/

[16] Dual full-bridge driver. ST. https://www.sparkfun.com/datasheets/Robotics/L298_H_Bridge.pdf