**Non – line sensing notes:**

Non-line sensors check list:

1. Understand what I am provided with to measure the non-line factors; how they work, capabilities and how the microcontroller will get measurements
2. If I plan to use them then how will I implement them
3. If I want an alternative then how will this alternative help me and discuss any ones that I will purchase to beat competitors

Speed sensors:

Purpose:

To be able to control speed to know when instruction to stop, speed up, slow down, move at constant speed is executed

How:

The wheel speed is calculated from the motor shaft speed using gear ratio

Encoders produce closed loop feedback signals by tracking speed and direction

Quadrature:

How incremental encoders measure speed:

A rotary incremental encoder has two output signals, A and B, which issue square waves in quadrature when the encoder shaft rotates. The square wave frequency indicates the speed of shaft rotation, whereas the A-B phase relationship indicates the direction of rotation.

What the index output is in the quadrature:

Some rotary incremental encoders have an additional "index" output (typically labelled Z), which emits a pulse when the shaft passes through a particular angle. Once every rotation, the Z signal is asserted, typically always at the same angle, until the next AB state change.

How to calculate motor shaft speed:

Rotational speed sensors and encoder disk:

An encoder is a disk placed on the motor shaft that has spaces between it like a bicycle

The rotational speed sensor is a phototransistor and led that detects if light passes through is blocked or not and based on that will produce a digital logic 1 or 0 pulse

The shape of the waveform depends on the position of the sensor, the size of slots and the sensor

response time

How speed is detected:

1. The time between rising edges and counting how many rising edges occurred within specific time frame to determine the frequency and hence the linear speed in one second (in this case ill choose a second)

The amount of time the sensor spent blocked or blocked can be calculated and then using the position of the sensor and the amount of blocking we can calculate the RPM (don’t really understand what I mean here)

1. The pulse width of the A or B signal is timed and then the frequency can be known and hence the speed or rotation

The method of amount of counts:

Speed is proportional to the amount of counts per time

the software will sample ie. Take in the total amount of counts at a period in time, at period of time and then will sample again at a later time

the average speed is the gradient of the count- time graph

gradient = count1 – count2/time1 – time2

the electronic circuit of the encoder: An incremental encoder interface is an electronic circuit that receives signals from an incremental encoder, processes the signals to produce absolute position and other information, and makes the resulting information available to external circuitry.

How the quadrature works:

The pulses emitted from an incremental encoder's A and B outputs are quadrature-encoded, meaning that when the encoder is moving at a constant velocity, the duty cycle of each pulse is 50% (i.e., the waveform is a [square wave](https://en.wikipedia.org/wiki/Square_wave)) and there is a 90 degree [phase difference](https://en.wikipedia.org/wiki/Phase_(waves)#Phase_difference) between A and B.[[2]](https://en.wikipedia.org/wiki/Incremental_encoder#cite_note-craig-2) At any particular time, the phase difference will be positive or negative depending on the encoder's direction of movement. In the case of a rotary encoder, the phase difference is +90° for clockwise rotation and -90° for counter-clockwise rotation, or vice versa, depending on the device design.

The software of the encoder:

Regardless of the implementation, the interface must sample the encoder's A and B output signals frequently enough to detect every AB state change before the next state change occurs

Quadrature decoder:

Incremental encoder interfaces commonly use a **quadrature decoder** to convert the A and B signals into the *direction* and *count enable* (clock enable) signals needed for controlling a synchronous, bidirectional (up- and down-counting)

In any two consecutive AB samples, the logic state of A or B may change or both states may remain unchanged, but in normal operation the A and B states will never both change. When neither A nor B changes, the quadrature decoder assumes the encoder has not moved and so it negates its *count enable* output, thereby causing the counts to remain unchanged. When just A or B changes state, the quadrature decoder assumes the encoder has moved one increment of its measurement resolution and, accordingly, it may assert its *count enable* output to allow the counts to change, and assert or negate its *direction* output to cause the counts to increment or decrement (or vice versa).

Quadrature encoders:

Used to know if wheel is going backwards or forwards

Specifications:

Black index lead – sends a pulse when a full rotation happens

A (brown) and B (orange) channels – count 256 ticks per revolution

**Measuring the Encoders using the stm32:**

4 ways of measuring them:

GPIO Interrupt the increments a counter variable

GPIO Interrupt that starts and stops a timer peripheral

Timer peripheral in input capture

Using the built in quadrature timer peripheral and the timer peripherals of the stm32

Steps:

To measure speed, the rate of change of ticks across a specified timeframe needs to be measured

Initialize timer

Clear pulse counts

Read the change of pulse count across specified time frame

Reset timer then repeat 2 to 4 times ???

Higher precision counting:

TIM2 and TIM5 have counters that can count more ticks per revolution than the quadrature

Ended at pg 68

**Bluetooth:**

Bluetooth is the technology where quick wireless connections (but much slower than BT) across short distances at low power consumption

The two ways to communicate data between Bluetooth devices using software:

In software, functions are called from libraries of top layer protocol stacks

Using GATT:

General attribute protocol: used to allow two-way communications between PD and central processors

Using GAP:

Generic access profile:

One way or two-way transmission

Devices roles can be chosen and interactions can be enabled or disabled using this method. ( im assuming interactions are the data transmission)

Device roles with GAP:

* Peripheral devices are (usually) data collectors
* Central devices (usually) data receivers
* Peripheral devices can only connect to one central device
* Central devices can connect to multiple PD

Beacons with GAP:

Beacons are GAP data disabled BLE devices that transmit **advertising data payload to signal where they are and includes some info about device. This data can also be used to measure proximity** or distance to the device by measuring the received signal from the device.

Optional Signal response payload is also a optional signal that is sent to central to supply more info

**To send these signals of advertising they must be sent infrequently to not drain too much battery**

**Popular standards in beacons; iBeacon, Eddystone, ALT Beacon**

**Transferring data using a BLE:**

**Characteristics** are data points analogue sensor value or arrays of xyz axis of a point in 3D. A characteristic value must have declaration attributes before the value that contains its properties??? (I have no idea what’s this). Characteristics may include descriptor attributes too

**Services** are used split data into logic entities and contain specific groups of characteristics ???. **Each service has a unique numeric id** either 16 bit which is adopted for BLE or 128 bits for custom services

NOTE: To transfer data across a BLE link you therefore have to choose a appropriate service or make your own.

**Profiles** are made of services that are needed to make a use case to solve a problem.

Connection of profiles, services and characteristics:

HM-10:

The hm 10 connects to the micro through UART port

HM-10 is the module that will be given to us to connect to STM32 to BLE (our phones)

The HM-10 will be controlled by AT commands which are sent them text commands???

Some terminology about hm – 10:

State pin: when low means no connection and when high there’s a connection

BRK pin: break pin being made high momentarily makes the connection break

How the ble module hm 10 transmits data:

* It sets the value of the custom characteristic to the value of data to be transmitted and then sends a notifying signal to ble device that new data is available
* The custom characteristic can hold 20 characters and to send more they have to be sent separately

Sending commands to and from HM – 10:

From HM – 10:

The HM – 10 when connected to the micro and to phone, the micro can send commands to phone via UART serial monitor and AT commands using a computer

The value of the custom characteristic ie. The word that is sent appears under the characteristic

There are programs to control and set up the BLE module Hm 10

By default, the HM – 10 is in slave/ peripheral mode

To make a connection between two BLEs:

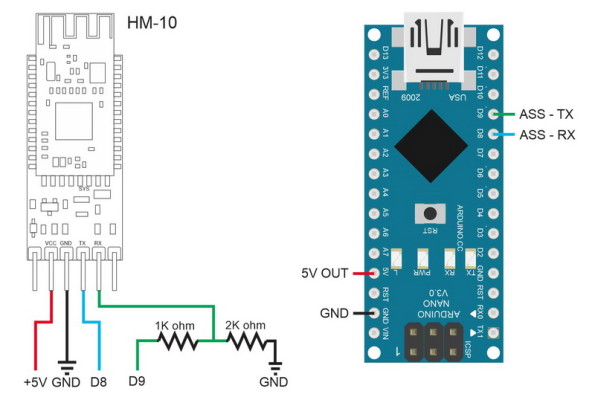
AT+ROLEx: to change the BLE role between master and slave

AT+IMMEx: to change the mode of connection between manual and automatic

AT+ DISC: to find the address of the slave ble

AT+CONAx: to connect to the ble using its address

How to connect the micro to the hm 10:



Digital I/O pins will connect to the HM – 10 Rx/Tx pins but have to check if bit rate is allowed through the digital I/O and how much the current It needs

HM-10 electrical characteristics:

Default Baud rate = 9600 (baud rate is speed of connection) so bit frequency is 9600 bits per second

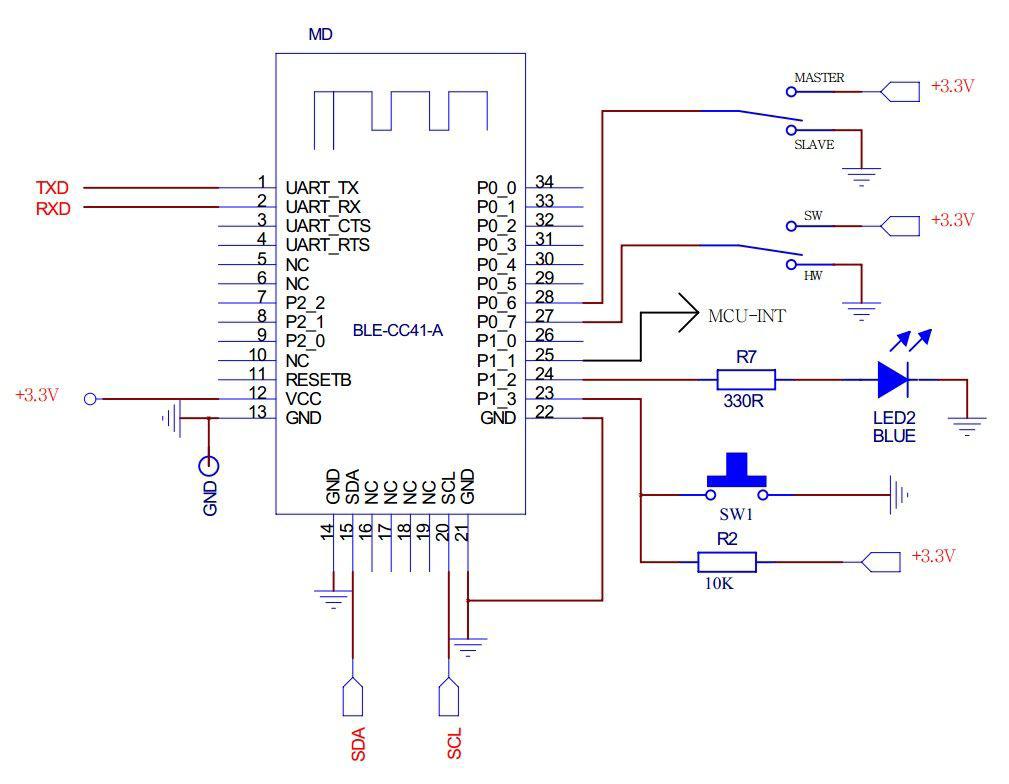
Power: 3.3V 50mA

Range = 100 m

Hm 10 includes a voltage regulator that makes it 5V compatible except for the rx pins

<http://fab.cba.mit.edu/classes/863.15/doc/tutorials/programming/bluetooth.html>

<https://www.instructables.com/id/How-to-Use-Bluetooth-40-HM10/>



AT commands:

They are like low layer commands that are used to create, end or change the parameters of a connection; basically its code to communicate between ble devices

Measuring motor current and battery voltage:

Current sensing:

The current using voltage that is input to the microcontroller and using the analog input and ADC, voltage is calculated.

Steps:

1. The current through the motor flows through a low resistance resistor and then the voltage of across the small resistor is measured by the circuit ( how??). the circuits has a parallel input from the voltage of the resistor and the isolating amplifier (assuming that the isolating amplifier has infinite input resistance. A capacitor is placed in series to store the voltage I think
2. The voltage is inputted into a isolating amplifiers and amplified to give a output between 0 and 5V
3. This is then measured by the microcontroller to give the actual current through the motor

Current sensing calibration:

Using PWM signals to power the motor and an ammeter test the current for multiple different torque loads (for motor to produce different current)

Compare results to measured results by ADC

The results will say the relationship between the voltage ADC and actual current to djust to get real current in code

Non-line sensors I have to research:

Speed sensors:

Bluetooth sensors:

Current sensing:

Acpl 784:

308 mV, in supply

16 mA in supply

16 mA out supply

Positive input (± 200 mV recommended)

100kHz bandwidth

Slew rate at 15kV

Voltage regulator:

Load regulation: change is 40 to 40 mV

Voltage must be higher than 6 to output 5

Max input is 30 V

So my mistake was that I didn’t understand how this schematic works and I saw 0R1 as R1 but its 0.1 ohm. Im guessing that the voltage is inputted from by taking a point before the 0.1-ohm resistor

How long does the ACPL-C78A/C780/C784 take to begin working properly after power-up? Within 1 ms after VDD1 and VDD2 powered the device starts to work. But it takes longer time for output to settle down completely. In case of the offset measurement while both inputs are tied to ground there is initially VOS adjustment (about 60 ms). The output completely settles down in 100 ms after device powering up.

At 5V VDD in and VDD out the gain is 8.029 and the offset input voltage is 0.35 mV.

Current sensing calibration:

Using a pwm generator that connects to pin 3 and pin 6 (V+) to JP5 port GND, the motor can be turned on. Once turned on, an ammeter in series with the motors will measure the current and the pins tp18 and 19 (motor 1) and tp 20 and 21 (motor 2) will measure the voltage across motors 1 and 2 individually. The test points measure the voltage across a 0.1 ohm resistor that is amplified by 8.029. The voltage across the test point’s results will be compared to the current of the ammeter to know how much voltage from ACPL amplifier represents the current through the 0.1-ohm resistor. The voltage at different currents through the motor are measured and plotted on a voltage – current graph to know the relation constant. After we know the relation, we can connect the pins 9, 10, 11 and 12 of port Jp1a of motor drive board to the microcontroller analog inputs for software to monitor the current through the motors.

Based on the control pwm chosen as uni or bipolar we can choose the signal to be tested while current sensing.

Battery sensing:

Using the Dallas Ds2781, the integrated circuit measures the voltage and accumulated current of the battery.

The current is measured by taking the current periodically at every interval of specified time. If Voltage remains constant which it probably will the energy will be the sum of the current and the voltage

Important info about dallas capabilities:

Full scale VDD= 9.99V

Resolution Voltage = 9.76 mV

Current minimum resolution = 1.56 microA

Current resultion = 1.56 microvolts/resistor sensing

Current full scale = 51.2 mA

16 bytes of EEprom non-volatile memory

VDD = supply voltage connected to a decoupling network

VSS = ground

VIN = voltage to be compared to ground

How voltage is measured:

Measured every 440 ms, Voltage is inputted into Vin and compared to Vss.

Voltage is placed in a voltage register in 2’s complement form

Voltage above the max VDD are reported as max number in twos complement and voltage below the Vss are reported at the minimum value

The voltage value is stored in a 16-bit address but only uses 10 bits

Current measurement:

The current flow output of the battery is measured by measuring the voltage drop across low resistance resistor RNS. The voltage sense range is +- 51.2 mV. The input linearly converts peak signal amplitudes up to 102.4mV as long as the continuous signal level (average over the conversion cycle period) does not exceed ±51.2mV.

The current register is updated every 3.515 seconds in two’s complement form. Charge current is positive and discharge current is reported as negative in the register

The current is stored in a 16-bit address that can be read using a software command

Average current:

the average current is taken from 8 measurements and hence updated every 28 seconds. It has a resolution of 1.56 micro volt/ Rsns

current offset bias:

Current offset bias is stored in a byte of data. It can in the range of +198.1 – (-) 199.7 micro volt

the current offset can be written to and then added to all the values of current measurement

Current offset measurement:

The offset current is measured every 1024 values and is used to subtract the error from all previous measurements stored in registers

Accumulated current:

Resolution of current measurement: 6.25μVh/Rsns

Transfer of data and control:

The battery sensor uses 1 wire data bus that acts as power, low speed data and signaling. The data bus is just a data wire and ground wire that uses a capacitor to store power while data is being transferred

STM32:

Powerlines:

Maximum current through each VDD power line

100 mA

Maximum current through all of the VDD power lines:

160mA

output current through the I/O:

output current sunk by the I/O pin: 25mA max

output current sources by I/O pin: -25 mA max

Total I/O current:

Output current sunk by all the I/O pins: 120 mA

Output current sourced by all the I/O pins: -120 mA

Voltage:

Standard operating voltage = 3.6V max and 1.7 V min

Output low voltage: max = 0.4V

Output high voltage: min = 3.2V

Condition at 8 mA

Power:

+5V

+3.3V

GND

In I/O these pins have limitations:

PC13, PC14 and PC15 which can sink or source up to ±3mA. When using the PC13 to PC15 GPIOs in output mode, the speed should not exceed 2 MHz with a maximum load of 30 pF.

ADC characteristics:

Voltage range input: 0 – (1.7 - 3.6)

DC current consumption

Sampling rate: typical 15 – 30 MHz

I/O frequency ???

TIMx frequencies:

Timer frequency: for APB2 its 84MHz and for APB1 its 42 MHz

**total current from micro accounting:**

**supply through power:**

50 mA for the Bluetooth

38 mA for the line sensors

20 mA for the speed sensor

Motor drive board ???

**Voltage and current input:**

Speed sensor input = 4 mA

Line sensors input = 1 mA

Bluetooth input for transmission =

PINS I WILL CONNECT TO:

HM 10:

CN6 3.3 V to VCC

CN6 GND to GND

PA\_2 to TX

PA\_3 to RX

Motor drive board:

Current sensing of motors:

Motor A sensing:

Pin 10 Avago A+ of jp1a to PC\_2 of CN7

Pin 9 Avago A– of jp1a to PC\_3 of CN7

Motor B sensing:

Pin 12 Avago B+ of jp1a to PC\_4 of CN10

Pin 11 Avago B- of jp1a to PC\_5 of CN10

One wire:

Jp1a pin 8 to PB\_8

Sensors:

Sensor 1 resistor voltage value of transistor to PA\_0

Sensor 2 resistor voltage value of transistor to PA\_1

Sensor 3 resistor voltage value of transistor to PA\_4

Sensor 4 resistor voltage value of transistor to PB\_0

Sensor 5 resistor voltage value of transistor to PC\_1

Sensor 6 resistor voltage value of transistor to PC\_0

CN6 5V to VDD

CN6 GND to GND

https://learn.sparkfun.com/tutorials/bluetooth-basics/how-bluetooth-works