Universitat de Girona
Escola Politècnica Superior

Treball final de grau

Estudi: Grau en Enginyeria Electrònica Industrial i Automàtica

Títol: Control de regularitat per a motocicletes

Document: 1. Memòria

Alumne: Miquel Pumarola Garcia

Tutor: Miquel Rustullet Reñe
Departament: Enginyeria Elèctrica, Electrònica i Automàtica
Àrea: ESA

Convocatòria (mes/any): juny/2015

ÍNDEX

1 INTRODUCCIÓ

La regularitat en el món de les proves a motor cada cop és més present, siguent present en proves de circuit, proves per carretera oberta i proves anomenades de regularitat esport amb carretera tallada i mitja superior als 50 Km/h.

A Catalunya, hi han diversos campionats. Podem trobar campionats en circuit com les Clàssics series o també proves amb carretera oberta, com la Challenge Intercomarques o la Copa Catalunya. Apart de les proves realitzades a Catalunya, també trobem diverses proves i altres campionats de regularitat a Espanya i per Europa. Un bon exemple a Europa seria el prestigiós rallye històric de Montecarlo.

S'anomena prova de regularitat aquella que consisteix en realitzar una distancia a una velocitat mitja. La variant la qual parlarem és l'anomenada regularitat clàssica, on la mitja es constant en carretera oberta i inferior a 50 Km/h en tot moment. Una mateixa prova sol tenir varis trams de regularitat o cronometrats en els que s'ha d'intentar complir la mitja imposada per l'organització.

La mitja constant significa que durant tot el tram cronometrat i durant el major temps possible s'ha d'intentar portat aquesta velocitat. És important assumir aquest concepte, ja que es l'essència d'aquest tipus de proves.

L'inici en el tram de regularitat s'inicia en un punt indicat per l'organització, habitualment per el sistema d'auto sortida, a la hora, minut i segon que haurem decidit del carnet de ruta, és a dir, sense cap persona que controli la sortida. A partir d'aquest moment em d'assolir la mitja i mantenir-la durant tot el tram. Teòricament el sistema és senzill, és una aplicació senzilla de la formula espai/temps, en funció del qual s'anirà ajustant la velocitat.

Abans d'inicialitzar la prova s'ha de realitzar la calibració, on haurem d'ajustar el nostre aparell amb la distancia donada per l'organització. Per això l'organització ens facilitarà un tram de calibratge i ens donarà la distancia exacte del tram. D'aquesta manera la distancia donada i la realitzada per nosaltres ha de ser la mateixa.

Com s'ha mencionat anteriorment, el cronometratge de les proves de regularitat esta basat en el pas per un determinat lloc a una hora ideal, realitzant les presses de temps a la dècima de segon. Si entre altres factors, el teu aparell de mesura no funciona

correctament, donarà lloc a un error de mesura, el teu pas ideal no serà el correcte, penalitzant tant per retràs com per avançament. A la següent figura 1 es mostra gràficament el pas ideal d'un cotxe:

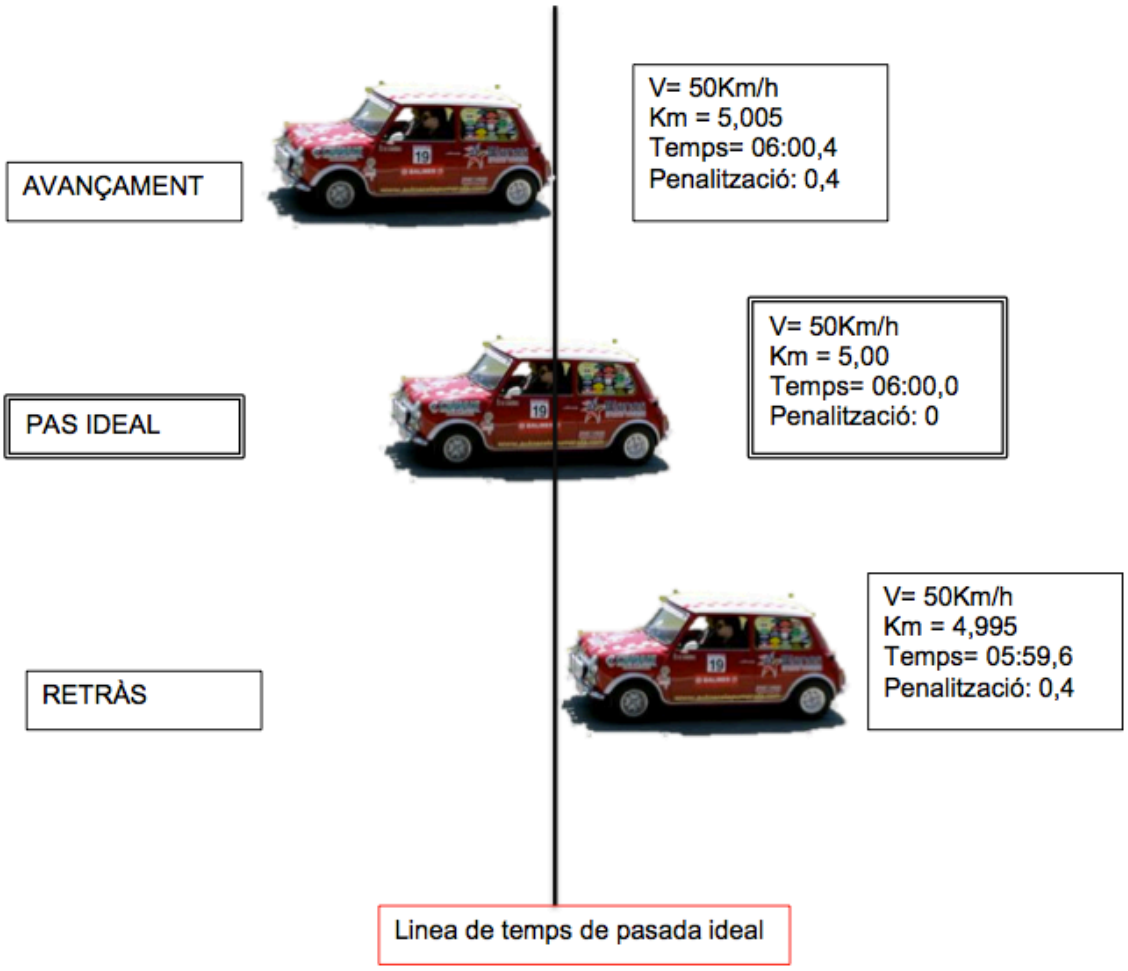

Figura 1. Pas ideal

1.1 Antecedents

Les curses de regularitat, on hi participen cotxes i motocicletes de més de 25 anys, són curses on s'ha de recórrer una distància a una velocitat mitjana. Una mateixa prova sol tenir varis trams de regularitat o cronometrats en els que s'ha d'intentar complir la mitja imposada per l'organització. La mitjana constant significa que durant tot el tram cronometrat i durant el major temps possible s'ha d'intentar portat aquesta velocitat imposada per a una menor penalització.

En el mercat trobem aparells de mesura per a cotxes, però no hi ha cap aparell per a motocicletes. Les variables a mesurar o controlar seran el temps i l'espai (distància).

1.2 Objecte

L'objecte d'aquest projecte és dissenyar i implementar un controlador de regularitat per a motocicletes controlat per un Arduino. La programació dels trams de regularitat és realitzarà mitjançant una aplicació programada per telèfon mòbil o tauleta i transmesa al microcontrolador Arduino, amb la seva configuració per a cada tram de regularitat.

Tot això s'ha de programar abans de la sortida del tram d'una manera fàcil i ràpida realitzada a través de l'aplicació creada per telèfon o tableta.

Per tal que així sigui, volem que el mòdul compleixi les premisses següents: Ha de ser lleuger i robust, versàtil i adaptable a les diverses comprovacions que s'hagin de realitzar. Ha de tenir un muntatge i configuració senzills.

1.3 Especificacions i abast

L'abast d'aquest projecte és crear un controlador de regularitat per a motocicletes. El controlador ha de tenir una senyal de mesura per a poder tenir una calibració de distància. Amb un càlcul matemàtic s'ha de saber en tot moment: la distància recorreguda, el cronometratge del tram i la velocitat imposada per a obtenir la distància recorreguda.

El món de la regularitat per a motocicletes esta limitat al no existir aparells de mesura.

2 HARDWARE

En la següent figura es pot observarla relació dels diferents elements amb el microcontrolador. L'element principal és l'Arduino MEGA, el qual té conectat els següents elements: el mòdul CronoDot RTC, la pantalla TFT, el detector de proximitat inductiu, la barra LED, els polsadors del manillar i el mòdul bluetooth.

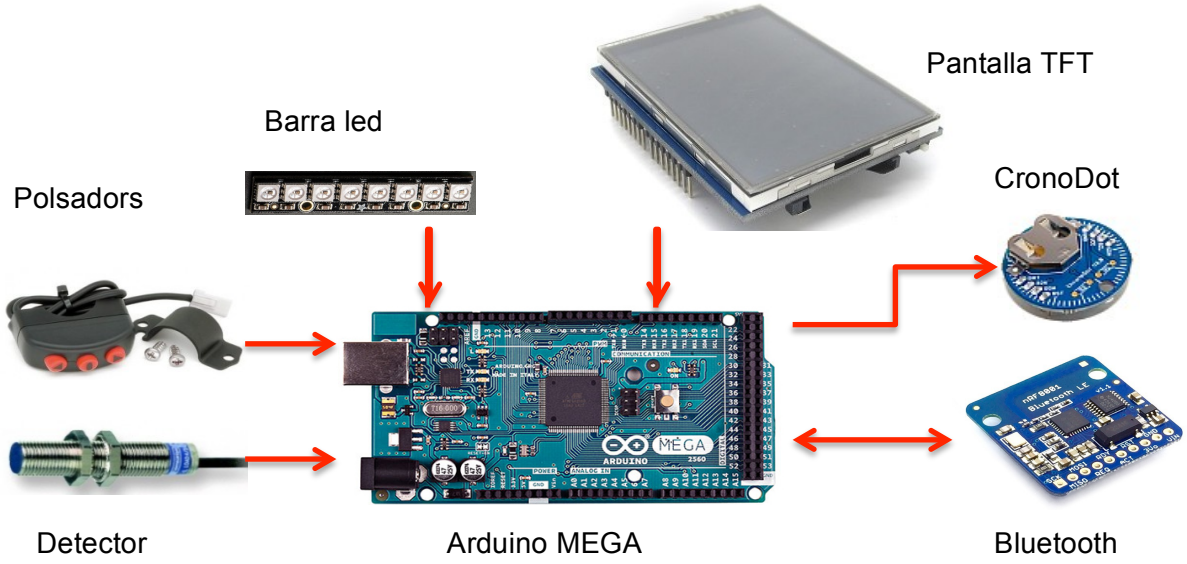

Figura 2. Relació de l'Arduino amb els diferents elements

2.1 Placa Arduino MEGA

Al mercat hi ha milers de microcontroladors, molts d'ells vàlids per les aplicacions que necessitem, però s'ha escollit l'Arduino MEGA 2560. L'Arduino MEGA és un microcontrolador basat en l'ATmega 2560.

Es tracta d'una placa open hardware, on el seu disseny és de lliure distribució i utilització. El programa s'implementarà utilitzant el propi entorn de programació d'ATmel, Atmel Studio i es transferirà utilitzant un cable USB. En la figura 3 es pot veure l'aspecte de la placa.

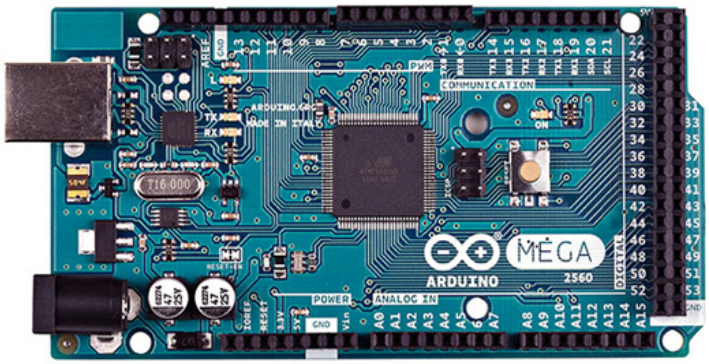

Figura 3. Aspecte placa Arduino MEGA

La placa disposa de 54 pins digitals configurats com entrades o sortides depenen de les necessitats, 14 dels quals són PWM. 4 ports UART`s (per hardware). Disposa d'un oscil·lador de 16MHz. Es disposa també de 16 pins analògics, cada un proporciona 10 bits de resolució. Aquets pins treballen a 5 V cada un i poden proporcionar o rebre un màxim de 40mA.

Seguidament podem veure les seves característiques més importants.

Característiques	**Descripció**
Microcontrolador	ATmega2560
Voltatge d'operació	5 V
Tensió entrada (recomanada)	7-12 V
Tensió d'entrada (límit)	6-20 V
Pins digitals de E/S	54 (dels quals 14 proporcionen sortida PWM)
Pins d'entrada analògica	16
Corrent DC per pins E/S	40 mA
Corrent DC per pin 3.3V	50 mA
Memòria Flash	256 KB (ATmega2560) dels quals 8 KB utilitzat per carregador
SRAM	8 KB (ATmega2560)
EEPROM	4 KB (ATmega2560)
Freqüència de rellotge	16 Mhz

Taula 1. Característiques principals Arduino MEGA

La placa Arduino s'encarregarà de gestionar i interpretar les funcions realitzades per l'usuari, comandades des de l'aplicació de la tablet o smartphone. Serà l'encarregat de comunicar al mòdul bluetooth per a rebre el programa (tram a realitzar). Rebre la senyal de la sonda instal·lada a la roda no motriu de la motocicleta, mostrar l'estat dins del tram mitjançant els led's indicadors i s'encarregarà de mostra en una pantalla TFT els diferents

valors segons en la pantalla pre-seleccionada. Amés a més de la comunicació amb al mòdul de rellotge, per a garantir una hora precisa.

Per a la programació de les diferents comandes del mòdul CronoDot, la pantalla TFT, el detector inductiu i els LED's s'han utilitzat unes llibreries existents. Aquestes llibreries estan introduïdes en el mateix programari Atmel Studio.

La elecció del Arduino MEGA ha sigut considerada ja que aquest té un gran nombre d'entrades digitals. En principi, és podria utilitzar un Arduinio UNO (té 14 d'entrades/ sortides digitals i 6 entrades analògiques) però caldria utilitzar 3 entrades analògiques com entrades digitals amb valors de 0 i 1.

Una altre característica per la qual s'ha realitzat aquesta elecció és per la memòria Flash, medi d'emmagatzematge d'escriptura i lectura volàtil. L'Arduino UNO només té 32KB. El nostre programa no tindria prou capacitat.

Amb aquesta elecció d'Arduino i la implementació de la placa de connexions, podrem realitzar ampliacions de millora per a possibles projectes en el futur.

2.2 CronoDot RTC

En aquet projecte s'utilitzarà un xip CronoDot RTC extremadament precís basat en el DS3231 de la marca Adafruit. Inclou una bateria CR1632 que ha de durar no menys de 8 anys si l'interfice I^2C només s'utilitza quan el dispositiu té 5V disponibles. No és requereix de condensador de cristall o d'ajust extern.

El DS3231 té un mòdul de cristall intern i un banc de condensadors commutats d'afinació. La temperatura de dit cristall és controla contínuament i els condensadors s'ajusten per mantenir una freqüència estable.

El CronoDot pot ser instal·lat sense soldadura en un protoboard estàndard o també disposa d'uns forats per ser instal·lat sobre caixa. A la següent figura 4 es pot veure l'aspecte del rellotge:

Figura 4. Mòdul CronoDot RTC

Seguidament podem veure les seves característiques més importants.

Característiques	Descripció
Senyal de freqüència	0 – 40 MHz
Sortides de senyal	4 sortides. 2 ona sinusoïdals i 2 ona quadrades
DAC SFDR	> 50 dB a 40 MHz de sortida
Sincronització	Paraules de 32 bits
Interface de control	Bits paral·lels o sèrie
Alimentació	3,3V o 5V
Baixa potència	380 mW a 125 MHz a 5V
Baixa potència	155 mW a 110 MHz a 3,3V
Funció	Funció Power-Up

Taula 2. Característiques principals DS3231

Aquet mòdul estarà alimentat a 5V a través de la sortida d'Arduino Pin 5V. Disposa de diferents pins per a realitzar les diferents tipus de senyals. Aquets pins estaran connectats a GND, el SCA i al SCL respectivament de la placa Arduino.

L'assignació dels diferents pins de la placa es poden veure en la taula 3.

PIN	Nom	Funció
1	32 kHz	Sortida de 32kHz. Amb la seva activació, la sortida treballa amb dues fonts d'alimentació.
2	Vcc	Font d'alimentació primària a 5V.
3	SQW	Sortida d'ona quadrada amb 0 i 1 lògics.
4	RST	Estabilització anti-rebot alimentació Vcc.
13	GND	Connexió a massa.
14	VBAT	Copia de seguretat d'entrada dels subministrament d'energia.
15	SDA	Serial d'entrada/ sortida de dades.
16	SCL	Entrada rellotge de sèrie.

Taula 3. Descripció dels pins placa DS3231

Aquet mòdul és utilitzat per a donar una gran fiabilitat en el rellotge intern. Un rellotge que no sigui precís i que al llarg del temps presenti un adelant o un atràs fara que al llarg d'una prova s'acumuli una gran penalització per anar fora d'hora. És a dir, si el rellotge va atrasat

un segon, a cada control secret és penalitzarà aquest segon. Així docns, com la finalitat de les proves de regularitat és anar sempre perfectes amb temps i metres, s'ha escollit el CronoDot RTC per aconseguir anar sempre amb l'hora correcte.

2.3 Pantalla TFT

En aquet projecte s'utilitzarà la pantalla TFT tàctil de 2,8" amb ranura per targetes micro SD inclòs 320x240 pixels de la marca Adafruit. Aquesta pantalla és perfectament compatible amb la majoria de plaques d'Arduino i no requereix de connexió per cable, és plug and play per el cas d'Arduino Uno i només requereix d'unes petites modificacions per adaptarse a un MEGA o un LEONARDO.

És presenta en un format Shield per poder-se connectar directament, encara que no serà el cas, ja que estarà connectada en una placa annexe. Per el seu funcionament utilitza el protocol SPI amb el qual s'aconsegueix l'estalvi de pins per la seva connexió amb l'Arduino.

Posseix una pantalla tàctil resistiva que utilitza el pins analògics d'Arduino per poder recuperar directament la seva posició. El procés de visualització es governat per el potent controlador ILI9341.

L'aspecte de la pantalla es pot veure en la figura 5.

Figura 5. Pantalla TFT

L'assignació dels diferents pins de la pantalla es poden veure en la taula 4.

Nº Pin	Nom Pin	Funció
Reset	Reset	Reset Arduino.
5V	V_{cc}	Entrada alimentació +5V.
GND	GND	Connexió a massa.
3	LLUM	Il·luminació pantalla.
4	SD-CS	Utilitzat per Arduino per dir-li a la microSD que vol enviar/rebre dades des de la microSD
8	TOUCH-CS	Comandament de la pantalla tàctil. Utilitzat per l'Arduino li digui al controlador que vol enviar/rebre dades.
9	TFT-DC	Utilitzat per l'Arduino per controlat la TFT si aquesta desitja enviar comandaments.
10	SS	Utilitzat per Arduino per controlar la TFT, si aquesta vol enviar/rebre dades del TFT.
11	MOSI	És el hardware SPI mestre-esclau d'entrada de dades (IN). S'utilitza aquest pin per la TFT, la microSD i dades de la pantalla tàctil.
12	MISO	És el hardware SPI mestre-esclau de sortida de dades (OUT). S'utilitza aquest pin per la TFT, la microSD i dades de la pantalla tàctil.
13	SCLK	És el hardware SPI que fa la funció de rellotge. S'utilitza aquest pin per la TFT, la microSD i dades de la pantalla tàctil.

Taula 4. Descripció dels pins pantalla TFT

Seguidament podem veure les seves característiques més importants.

Característiques	Descripció
Microcontrolador	ILI9341
Corrent	250 mA
Voltatge	4,5-5,5 V
Resolució pantalla	320x240
Angle de visió	60-120º
Il·luminació de fons	Led
LCD color	65 K
Interficie	SPI
Mides	72,5x54,7x18 mm

Taula 5. Característiques principals

2.4 Detector de proximitat inductiu

En aquets projecte s'utilitzarà un detector de proximitat inductiu de caràcter general. Aquest detectors permeten detectar sense contacte objectes metal·lics a una distancia compresa entre 0 i 60mm.

El seu principi de funcionament consisteix en detectar exclusivament objectes metàl·lics. Esta composat bàsicament d'un oscil·lador els bobinats del qual constitueixen el costat sensible del detector. Davant d'aquest es crea un camp magnètic alternatiu. Quan es col·loca una placa metàl·lica en el camp magnètic del detector les corrents induídes constitueixen una carrega addicional que provoca la parada de les oscil·lacions.

El detector utilitzat és de 2 fils no polaritzats amb sortida NA. Funcionen amb independència del sentit de connexió +/-. Estan protegits contra sobrecarregues i curtcircuits. La seva connexió es sobre cable sobremoldejat, amb bona resistència a mullar-se amb líquids.

En la figura 6 es pot veure l'aspecte del detector de proximitat inductiu.

Figura 6. Detector de proximitat inductiu

Seguidament podem veure les característiques més importants del detector.

Característiques	**Descripció**
Marca i model	Telemecanique XS5 12B1DAL2
Connexió	Per cable; L= 2m
Camp funcionament	0 - 1,6 mm
Grau de protecció	IP 67
Temperatura de funcionament	-25º... +70º
Materials de construcció	Carcassa – llautó niquelat Cable – PvR 2 x 0,11 mm^2
Tensió alimentació	12... 48V (CC)
Corrent conmutació	1,5 – 100 mA
Freqüència conmutació màxima	4000 Hz
Número de cables	2 fils sense polaritzar

Taula 6. Característiques principals

2.5 Barra LED Neopixel

Els Neopixels són originals d'Adafruit i són uns díodes LED del tipus 5050 amb un controlador WS2812 integrat. Són sencills de posar en funcionament en pocs minuts ja que només utilitza un pin.

Els números 5050 fan referencia a les mides dels xips. Així doncs un xip 5050 tindria unes mides de 5mmx5mm.

En la següent figura 7 es pot veure l'aspecte de la barra de 8 LED's:

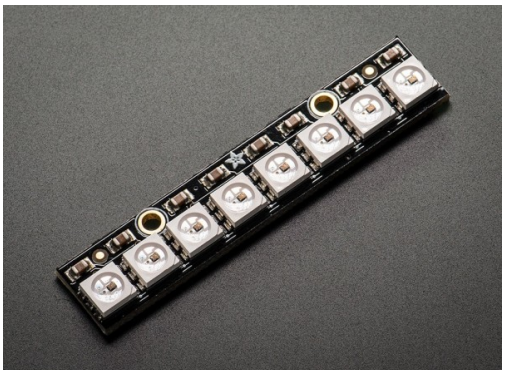

Figura 7. Barra Led's Neopixel

Les barres amb xips 5050 són barres d'alta potencia i consum més alts. Aquestes son conegudes com barres de triple nucli, per les tres diferents àrees que es poden identificar al mirar d'aprop el LED. Són barres apropiades per a treballar en àrees que estiguin exposades a majors nivells d'il·luminació ambiental. Això fa que siguin més cares al ser utilitzades en instal·lacions on hi ha un alt requeriment de lluminositat.

Seguidament podem veure les característiques més importants:

Característiques	**Descripció**
Forma	Rectangular
Número de LED's	8
Dimensions	51x10x3mm
Alimentació	5V
Consum	18 mA per LED

Taula 7. Característiques principals Neopixel

2.6 Polsador de manillar

El grup de polsadors de manillar és fabricat per la casa Vector. El seu muntatge ve preparat en una disposició expressa per anar al manillar de les motocicletes. El grup de polsadors esta composat per 3 polsadors. Els polsadors disposen d'una superfície de pressió gran i tenen un agradable tacte al pressionar-lo.

La següent figura es pot veure l'aspecte dels polsadors:

Figura 8. Polsadors Vector

Seguidament podem veure les característiques més importants:

Característiques	Descripció
Alimentació màx.	50 VDC – 500 mA
Número de maniobres	100.000
Funcionament	ON-OFF (obert en repòs)
Temperatura de treball	-40ºC / +75ºC
Estanqueïtat	IP 54

Taula 8. Característiques principals

Els polsadors aniran connectats al microcontrolador amb una connexió Pull-down. En repòs tindrem una tensió que serà pràcticament 0 i al polsar la corrent arribarà des de Vcc entregant-nos un 1. El valor de les resistències serà de 10KΩ. D'aquesta manera evitarem sorolls elèctrics amb les nostres connexions.

2.7 Bluetooth

Per a la programació de trams per a realitzar les proves de regularitat és realitzarà mitjançant el mòdul Bluetooth que rebrà la programació del tram provinent de la tablet i la transferirà al mòdul Arduino MEGA .

Primerament, necessitem saber quina necessitat tenim per poder realitzar el muntatge per tal que sigui capaç de realitzar els objectius proposats. En aquest apartat es descriu l'estructura i el funcionament del mòdul Bluetooth i la programació des de la tablet.

Aquesta part del projecte quedarà muntada en una caixa de connexions de superfície per poder ser utilitzada instal·lada posteriorment al manillar de la motocicleta. Estarà alimentada a 12V en tot moment.

L'Adafruit Bluetooth nRF8001 permet establir de forma fàcil una comunicació inalàmbrica entre un Arduino i qualsevol dispositiu iOS o Android.

Funciona simulant un dispositiu UART enviant i rebent dades ASCII entre els dispositius connectats, permeten decidir quines dades enviar, que fer amb elles i si és vol finalitzar la connexió.

Al muntar un chip nRF8001 es compatible amb quasi qualsevol microcontrolador que compti amb SPI.

El Bluetooth és una especificació industrial per les xarxes d'àmbit personal sense fils, que bàsicament serveixen per connectar els dispositius que podem portar a sobre o a una distància pròxima.

Amb el Bluetooth, podem obtenir una forma de connectar i intercanviar informació entre dispositius com ordinadors de butxaca, telèfons mòbils, càmeres digitals a través d'una forma segura, de baix cost a través d'ones de ràdio de baixa freqüència.

El Bluetooth nRF8001 permet a aquest dispositiu comunicar-se quan es troba a l'abast, encara que els aparells no estiguin a la mateixa habitació o espai. Aquest mòdul pertany a la Classe 2, la més comuna, i que ens permet una transmissió de fins a 10 metres. A la següent figura 9 es mostra l'aspecte del mòdul bluetooth.

Figura 9. Mòdul Bluetooth

En l'actualitat gairebé tots els ordinadors de butxaca, ordinadors portàtils i telèfons mòbils nous són compatibles amb la tecnologia Bluetooth de sèrie. Això permet a qualsevol persona poder realitzar la programació dels trams de regularitat i transmetre-la al muntatge del Arduino.

L'assignació dels diferents pins de la pantalla es poden veure en la taula 9.

Nº Pin	Nom Pin	Funció
	SCK	Pin SPI del rellotge de dades.
	MISO	Pin SPI de sortida de dades. Les dades s'envien des de el mòdul aquest pin.
	MOSI	Pin SPI d'entrada de dades. Les dades s'envien des de el mòdul aquest pin.
	REQ	Selecció de pin de treball
	RDY	Llest. Pin que informa que les dades estan preparades perquè l'Arduino els pugui llegir.
	ACT	Actiu. Sortida del mòdul. Informa si el nRF8001 esta ocupat.
	RST	Reset.
	3Vo	Sortida del regulador de 3,3V.
	GND	Connexió a massa.
	Vin	Alimentació del mòdul.

Taula 9. Característiques principals Bluetooth

Seguidament podem veure les seves característiques més importants.

Característiques	Descripció
Consum	14,6 mA
Voltatge d'alimentació	1,9 a 3,6 V
Classe	Classe 2 (2,5mW)
Camp de cobertura	10 metres
Temperatura funcionament	-40ºC ÷ 85ºC

Taula 10. Característiques principals mòdul Bluetooth

2.8 Alimentació

L'alimentació del circuit provindrà de la placa Arduino MEGA. L'Arduino s'alimentarà directament de la bateria de la motocicleta a 12V, una potència de 10,8 W i un corrent de sortida màxim de 1A. El consum dels diferents components es pot veure a la taula 5.

Els pins d'entrades i sortides digitals i analògiques de la placa Arduino MEGA consumeixen 40 mA cada un. Mentre que el pin de 3.3V té un consum de 50 mA, però no s'utilitza en cap moment aquest alimentació. En total s'utilitzen 17 pins d'entrades / sortides. Això realitza un consum mig de 680 mA.

L'Arduino MEGA disposa d'un circuit estabilitzador amb un regulador de tensió donant un corrent màxim d'un amper. Aquest regulador és el NCP1117. Esta montat a la placa d'arduino però no dispossa de radiador. Regula la tensió d'entrada entre 6V i 20V i alimenta tot el circuit d'arduino a 5V. En la placa d'entrada que va conectada a l'Arduino, s'ha instal·lat un LM323T, amb el qual s'aconsegueix poder arribar a tenir 3A. S'ha escollit aquest regulador de tensió en format TO-220 per el seva gran capacitat de subministrar un elevat corrent comparat amb la resta de reguladors de tensió com podrien ser la família dels 78XX.

Amb aquesta ampliació de corrent aconseguim tenir un correcte funcionament dels circuits i la seva correcta alimentació.

El mòdul CronoDot RTC s'alimentarà a 5V a través de la placa Arduino MEGA, i el seu consum mig és de 76 mA.

La pantalla TFT s'alimentarà a 5V a través de la placa Arduino MEGA, i el seu consum mig és de 250 mA.

EL detector de proximitat inductiu s'alimentarà a 5V a través de la placa Arduino MEGA, i el seu consum mig és de 100 mA.

La barra de Led's estarà alimentada a través del mòdul d'Arduino MEGA, i el seu consum mig és de 144 mA.

L'Arduino MEGA disposa d'un circuit estabilitzat amb un regulador de tensió d'una amper

Component	Consum mig
Arduino MEGA	680 mA
CronoDot RTC	76 mA
Pantalla TFT	250 mA
Detector	100 mA
Barra Led	144 mA
Total	1250 mA

Taula 11. Consum mig dels components

3 PROGRAMARI

En aquest projecte es programarà en dues parts. Per una banda la programació del Arduino i per altre banda l'aplicació App.

Per crear els diferents controls de senyals, es treballarà amb l'entorn que el mateix fabricant del xip del controlador Arduino. És realitzarà la programació mitjançant el programa Atmel Studio. Aquest programa permet el llenguatge en C# i permet tenir una disposició més practica per a treballar amb programacions relativament llargues.
S'ha escollit aquest microcontrolador per la seva facilitat de programació i la gran quantitat de materials i llibreries que podem trobar gratuïtament per internet. A més, el programa es apte per a les plataformes Windows, Mac i Linux, essent per a tots ells gratuïts a la mateixa pàgina web de la casa d'Atmel.

El programa s'implementarà utilitzant el propi entorn de programació d'Atmel i es transferirà del PC a la placa Arduino mitjançant un cable USB.

El programa, amb la programació de les diferents pantalles, controlarà els senyals d'entrada provinents de la sonda i a través dels diferents polsadors del manillar de la motocicleta. És gestionarà la informació que s'ha rebut del mòdul bluetooth perquè l'Arduino realitzi la funció necessària amb els valors necessaris.

Per altre banda, el programa representarà a la pantalla TFT, depenent de la funció els diferents valors, donant així a l'usuari informació introduïda i per poder veure els canvis realitzats a posteriori.

3.1 Programació App

El programa creat en el projecte està pensat perquè l'usuari pugui modificar els paràmetres a introduir per a les diferents funcions. Per realitzar-lo s'ha buscat la màxima senzillesa a l'hora d'interactuar. D'aquesta manera l'usuari podrà modificar en tot moment els valors necessaris per els càlculs, ja que cada cop que s'envii un tram nou amb el mateix nom incial, l'anterior quedarà borrat.

Figura 10. Relació amb els components

La placa Arduino és l'encarregada de rebre els bytes corresponents als valors dels tram i serà l'encarregat de gestionar cada valor per a la seva necessitat.

L'App té una unica pantalla. Aquesta pantalla està dividida en dos grups. Per una banda l'edició dels trams i per altre banda la connexió i la tramesa dels trams mitjançant el bluetooth a la placa Arduino. Cada grup treballarà de forma independent.

Per la realització de l'aplicació App s'ha utilitzat el programa App Inventor, per aplicacions per dispositius Android. Aquest programa permet el llenguatge en blocs i permet tenir una disposició més pràctica per a treballar amb programacions relativament curtes.

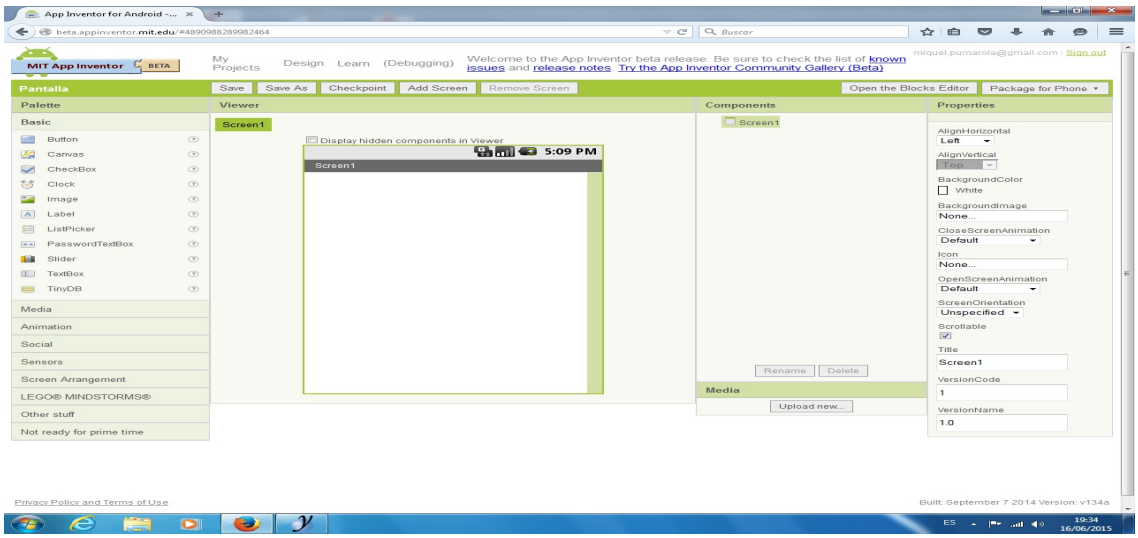

Figura 11. App inventor

La programació esta dividida en dues parts. Per una banda la programació de la pantalla, on és realitzarà el disseny i distribució dels botons, l'introducció dels valors i per altra banda el tractament d'aquestes dades prèviament introduïdes amb la connexió del bluetooth i l'enviament de les dades.

A la següent figura es mostra l'aspecte que té la pantalla principal per l'introducció de les dades:

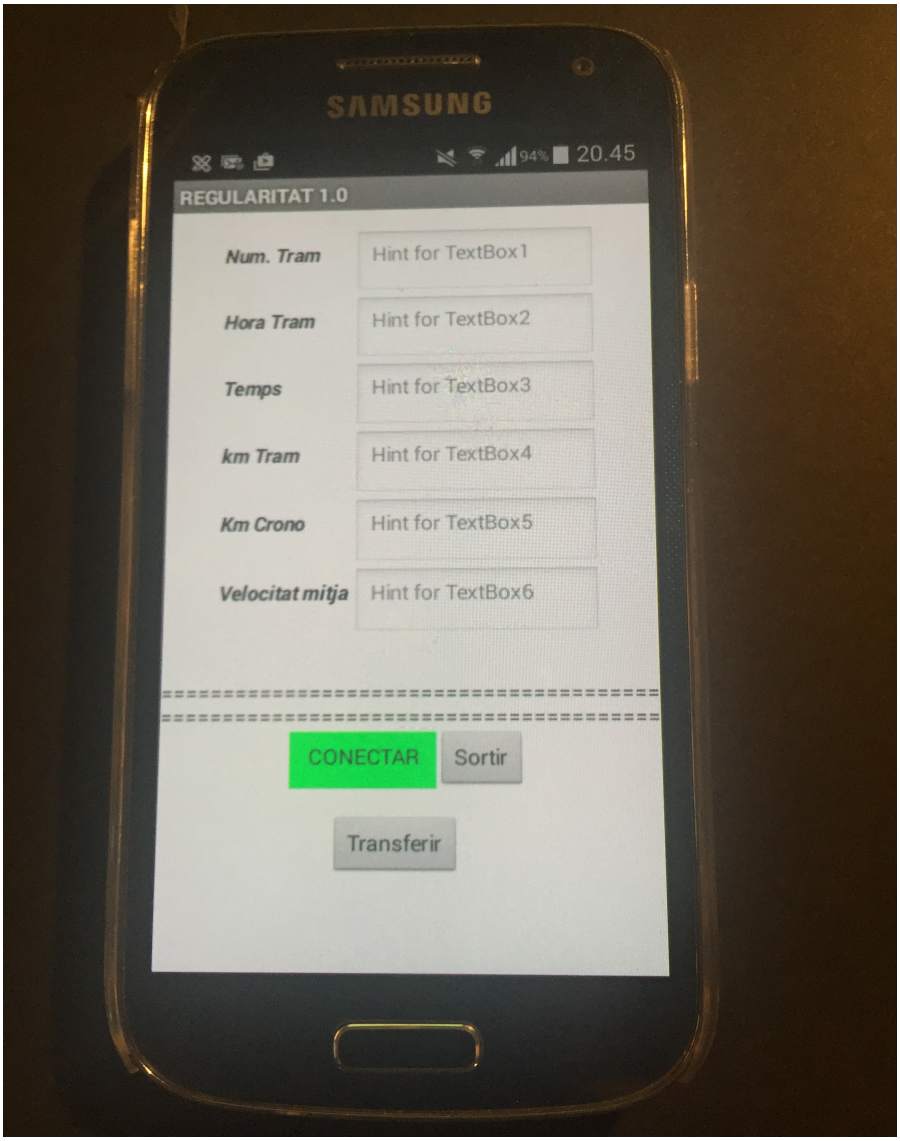

Figura 12. Pantalla App

Com és pot observar a la figura, hi han dues parts. La part inferior és per la connexió bluetooth i l'enviament de les dades introduïdes. Per altre banda a la part superior serà on s'introduiran les dades del tram. Per a ser tractades aquestes dades, cada camp és un número, que serà guardat com un byte. D'aquesta manera cada tram enviat seqüencialment, tindra una trama de 12 bytes.

Aquest valors a introduir són el número de tram, el tram, l'hora de sortida, el temps per a realitzar el tram, els kilòmetres totals del tram, els kilòmetres cronometrats i la velocitat mitjana imposada per l'organització.

El format d'introducció de dades és realitzarà separant cada parell de números per ";", així d'aquesta manera, sempre que l'Arduino rebi un signe ; sap que té un canvi de byte. D'aquest manera la trama d'enviament de dades quedara de la següent manera: 1;12;00;00;00;20;10;50;9;50;45;40;. Aquest valors a introduir són el tram (1;), l'hora de sortida (12;00;00;), el temps per a realitzar el tram (00;20;), els kilòmetres totals del tram (10;50;) , els kilòmetres cronometrats (9;50;) i la velocitat mitjana (45;40;) imposada per l'organització.

Les dades seran enviades en forma de trama de bytes, així d'aquesta manera al ser enviades a l'Arduino amb el seu ordre, aquestes ja seran memoritzades a la tarjeta micro SD a l'arxiu "carrera.txt", per al seu posterior ús en els càlculs. Les dades, només intervindran a l'apartat dels trams. Ja que prèviament s'hauràn realitzat les tasques de calibració i ajust del rellotge.

3.1.1 Edició de tram

La part del programa per a programar els trams i transmetre les dades dels trams a l'Arduino, és la de gestionar els paràmetres necessaris per a la programació dels trams. Les dades introduïdes, les quals seran diferents per a cada tram, seran les següents: número de tram, hora de sortida, velocitat mitja, kilòmetres de tram cronometrat, kilòmetres de tram total i el temps amb el qual s'haurà de recórrer els kilòmetres totals.

Aquest paràmetres són imposats per les organitzacions. Per tant, s'hauran d'introduir un a un manualment un cop l'organització hagi entregat tota la documentació. Al ser introduïts des d'una tablet, el propi teclat d'aquesta ens facilitarà molt la feina i ens ajudarà a realitzar

una programació ràpida. Els trams programats es guardaran un a un, a mesura que la programació per cada tram hagi finalitzat.

Una mala introducció del tram alhora de ser enviats, l'Arduino al rebrel, si la trama de 12 bytes no és correcte apareixerà un error i s'haurà de tornar a enviar el tram. Per altre banda, al introduir dos trams amb el mateix número de tram, l'últim tram enviat borrara el tram existent i quedarà guardat en memòria.

Primerament s'introduiran els valors dels trams manualment a les caixes TextBox amb el propi teclat del smartphone o la tablet. Son 6 blocs que corresponent a una trama de 12 bits. Amb el boto d transferir, a més de pasar les dades, el que fa és la suma de totes les dades introduïdes, formant la trama de bytes per a ser enviades. Al observar la imatge, s'observa la realització de la suma dels diferents apartats i la crida al bluetooth per transferir.

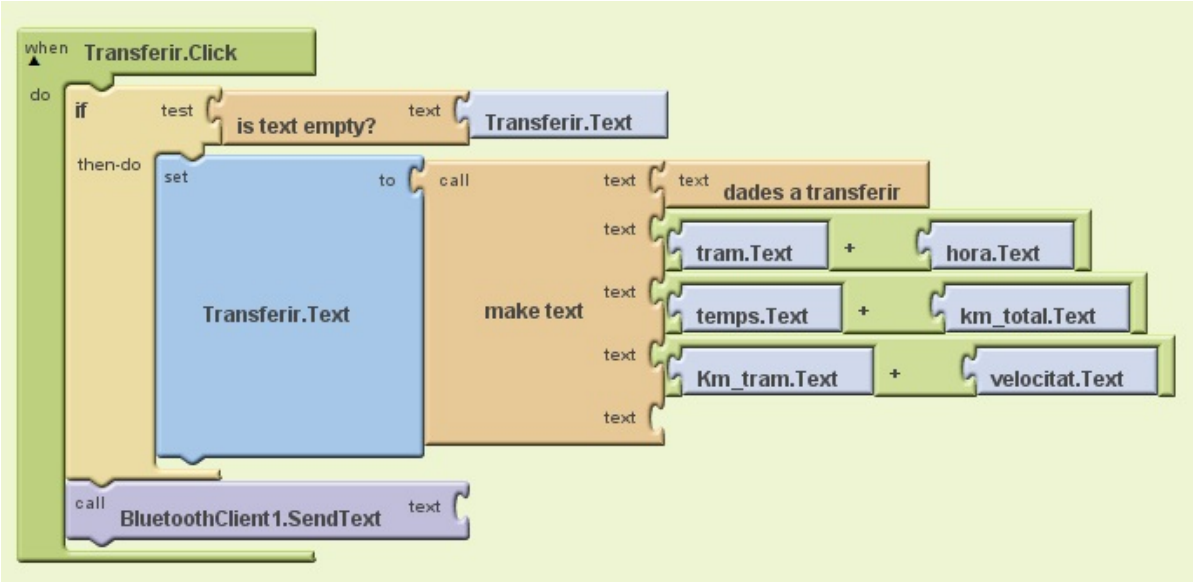

Figura 13. Programació dades trams

Un cop introduïts els trams, s'enviaran a l'Arduino. Aquest seran guardats a la memòria SD. Així és gararntitzarà que en cas de fallada d'alimentació aquestes dades sempre estaran guardades en la memòria i utilitzades en el moment que convingui.

3.1.2 Bluetooth

La part del programa de transmissió de dades dels trams és realitzarà mitjançant Bluetooth, per realitzar la transmissió de dades sense fils. Primerament s'activarà el Bluetooth connectat al Arduino. Un cop connectat és mostrarà visible pels dispositius disponibles per a poder connectar-se i realitzar la tramesa de dades.

Les dades del tram ja seran introduïdes a l'app. Seguidament és conectarà el bluetooth i un cop l'arduino ens informi que estan comunicats, es pasarà a enviar les dades del tram. Aquestes dades s'hauran de passar amb el boto de transferir. Al enviar més d'un tram, únicament s'han d'anar introduint trams sense haver de tornar a conectar-se al bluetooth. L'Arduino és l'encarregat de realitzar una verificació per comprovar que s'han passat les dades correctament. Un cop finalitzada la transferència de dades és desconnectarà manualment el bluetooth.

Amb la programació per blocs, s'ha de defiinir l'adreça del nostre bluetooth, s'ha d'obtenir la seva MAC. Amb aquesta mac és crearà una variable. A la següent imatge és mostra la creació d'aquesta variable.

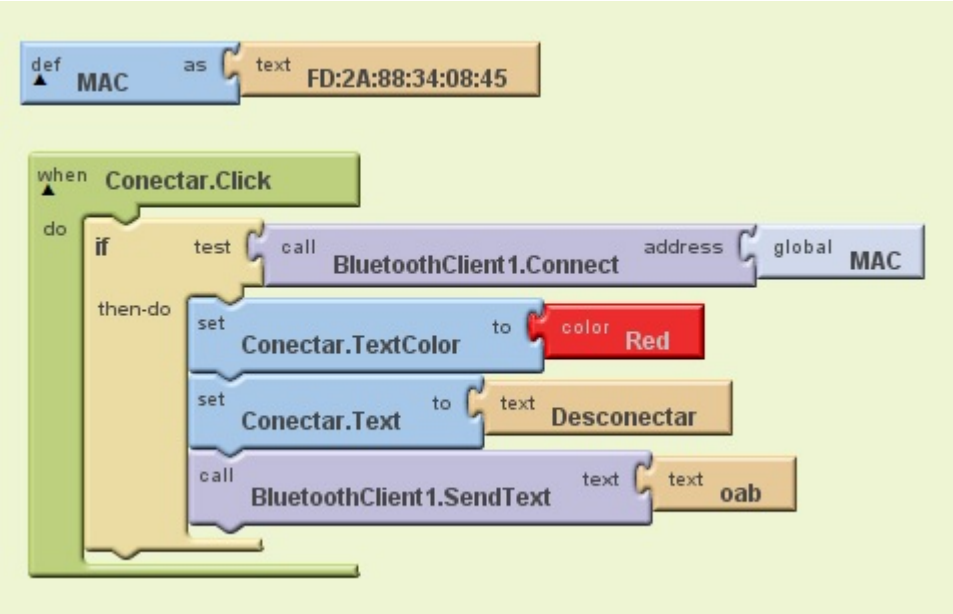

Figura 14. Connexió Bluetooth

Un cop el bluetooth estigui conectat és canviarà el color del boto i el color de la lletra per una millor visualització del seu estat. Per a la desconnexió del bluetooth també és realitzarà un canvi de color en el boto i canvi de text. Aquests passos, és poden veure en les dues figueres, en l'anterior de connexió i en la següent de connexió.

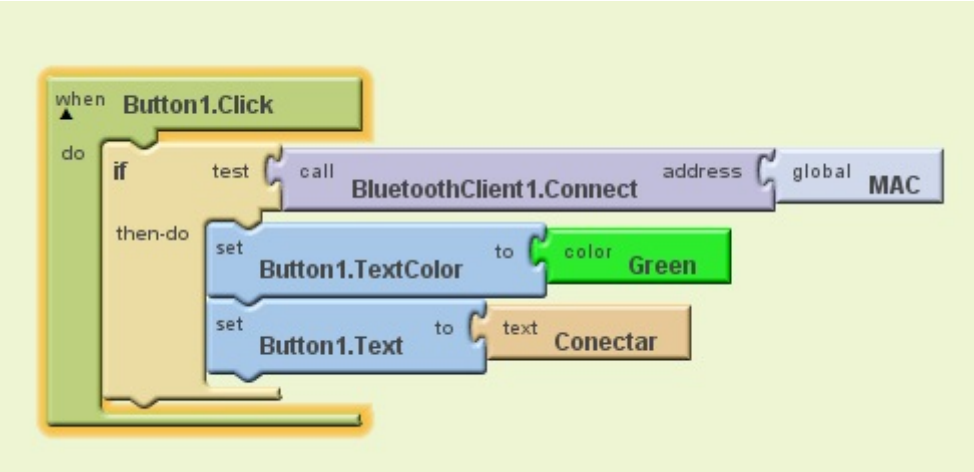

Figura 15. Desconnexió Bluetooth

Per altre banda, un cop finalitzada la introducció de les dades i la desconnexió del bluetooth, amb el boto se sortir, surtirem de l'app tornant a la pantalla de la tablet. En la següent imatge es pot veure la seva accion en el program de blocs.

Figura 16. Sortida aplicació

3.2 Programació Arduino MEGA

El programa creat en el projecte està pensat perquè l'usuari pugui modificar els paràmetres a introduir per a les diferents funcions. Per realitzar-lo s'ha buscat la màxima senzillesa a l'hora d'interactuar. D'aquesta manera l'usuari podrà modificar en tot moment els valors i els tipus de valors necessaris per els càlculs.

El funcionament del programa pel control de la regularitat, és la de gestionar les comandes realitzades per l'usuari, activant les funcions calibratge, funcions d'ajust de rellotge, la transmissió dels trams i la realització dels trams. La placa Arduino és l'encarregada de rebre els senyals d'entrada i mitjançant un menú principal es triarà l'opció que és vol realitzar.

Una vegada posat en marxa l'Arduino, aquest realitza primerament les funcions de declaració de variables, definició dels pins d'entrada i sortida, declaració de variables emmagatzemades a la memòria EEPROM, declaració de variables del bluetooth, declaració de variables de temps. Seguidament inicialitza el port sèrie, inicialitza els perifèrics, verifica si es la primera instal·lació i creació de variables per defecte, carrega variables EEPROM, carrega de dades de la SD, inicialitza el Bluetooth, inicialitza el rellotge i actualització de taules d'entrades.

A continuació es posa en marxa la pantalla i es motra un missatge de comprovació. Al mateix moment verifica el lector SD si té tarjeta i si esta formatejada. Un cop finalitzat aquest passos verifica a la memòria EEProm si existeix la primera variable, sinó la crearà valor per defecte. Llavors carrega els valors que nosaltres haurem introduït com és el cas del diàmetre de la roda. Seguidament és verifica si tenim a la SD el fitxer "carrera.txt", sinó el crearà. Aquest valor dl fitxer "carrera.txt" és guardaran les dades en una matriu per el seu posterior ús en el càlculs.

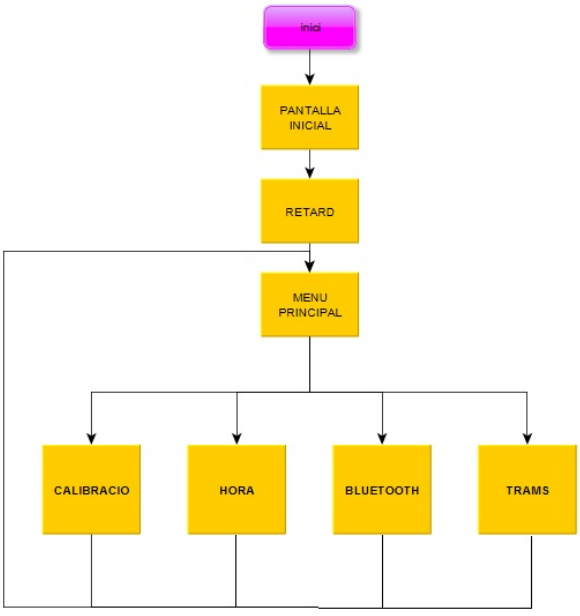

Figura 17. Menus pantalla

És mostrarà una pantalla de benvinguda que restarà activa dos segons i canviara automàticament a la pantalla de menú principal, on tindrem els apartats de calibratge, hora, connexió i tramesa de dades i trams. En la figura 17 s'han mostrat els menús de pantalla.

Tot seguit és mostra la programació del Arduino. Primerament és procedeix a borrar la pantalla anterior, la pantalla de presentació. En pantalla llavors quedarà representats els quatre apartats en que esta dividida la pantalla de menú. El codi utilitzat per la representació és el següent:

```
// TFT. Presenta MENU INICIAL
void pantallaInicio(){
      // borra Pantalla
      FborraPantalla();

      // Pinta Titul
      tft.setCursor(150, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2); tft.print(P("M E
      N U"));
      byte vseleccion = 1;

      tft.setCursor(45, 65); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
      tft.print(P("CALIBRACIO"));
      tft.setCursor(45, 105); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
      tft.print(P("HORA"));
      tft.setCursor(45, 145); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
      tft.print(P("CONNEXIO"));
      tft.setCursor(45, 185); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
      tft.print(P("TRAM"));
```

Aquestes dues primeres pantalles tenen el següent aspecte:

Figura 18. Menus pantalla

Al finalitzar de qualsevol dels quatre menús es retornarà sempre al menú principal, i l'usuari serà l'encarregat de decidir quina acció voldrà realitzar.

3.2.1 Calibratge

La part més important del programa és la introducció del valor de calibració, sense aquesta introducció no és realitzarà cap funció del comptatge de kilòmetres. Per introduir-la s'ha de realitzar prèviament una mesura del diàmetre del gir de la roda i s'obtindrà una mesura prèvia. Aquesta mesura només és realitzarà la primera vegada. Una vegada introduïda, llavors podrem realitzar el tram de calibració amb la mesura donada per l'organització de la prova. D'aqui s'obtindrà el valor del nou diàmetre de la roda, anomenat valor de calibratge. El valor de calibratge obtingut podrà ser canviat en qualsevol altre moment i anar realitzant les proves corresponents.

Per poder realitzar la calibració, s'haurà de realitzar el tram amb la moto en ruta. En tot moment és realitzarà la posada a zero i la posada de final de tram des de el polsadors del manillar. En la figura 19 es mostra el diagrama de la part del menú per a la calibració:

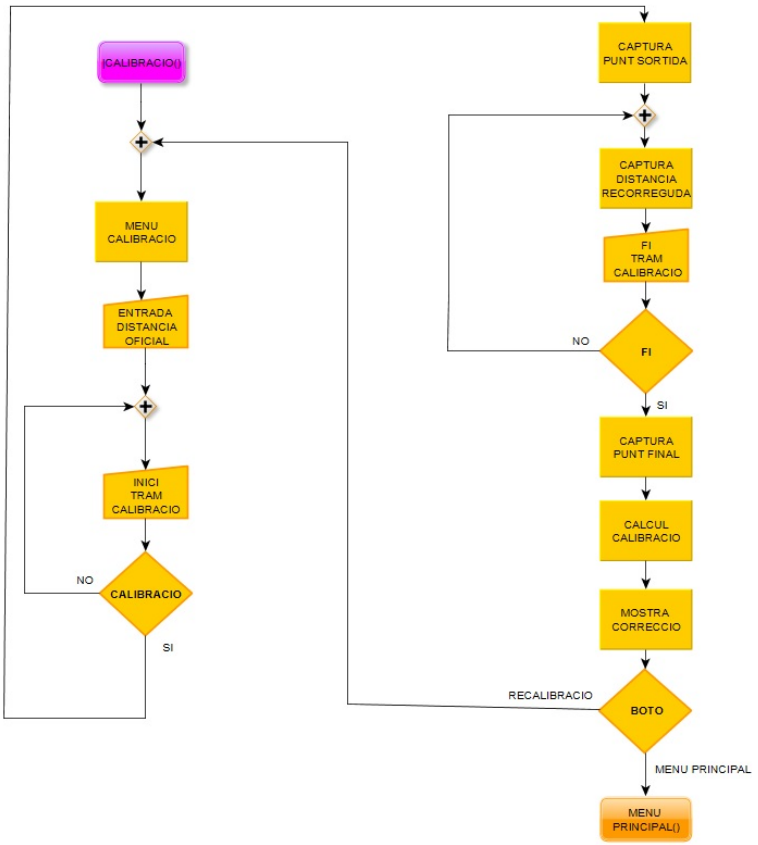

Figura 19. Menú calibració

Aquest menú de calibració esta dividit en quatre pantalles. La primera és el valor del diàmetre de la nostre roda, valor el qual podrem modificar sempre sigui necessari. Aquest valor queda guarda't en memòria EEPROM. A continuació és mostra la part de programa, on podem modificar o no el valor i continuar amb la calibració i accedir a la següent pantalla.

```
//  Nuevo valor diametro rueda
      vDiametroRueda = ((vtcentena*100) + (vtdecena * 10) + vtunidad);
      // Almacenamos el valor en EEPROM
      EEPROM.write(1, (highByte(vDiametroRueda)));              // Diametro rueda (Byte HIGH)
      EEPROM.write(2, (lowByte(vDiametroRueda)));               // diametro rueda (Byte LOW

      // Calculo distancia recorrida por rpm
      vdistrpm = ((2.0 * 3.1416) * (vDiametroRueda/2));
      vdistrpm = (vdistrpm /1000);

      // Fuerza a repintar Horas y min en pantalla
      minold=255;
      horaold=255;
      calibracio2();
```

La calibració és realitza mitjançant un recorregut i un valor de distància donat per l'organització. La finalitat del calibratge és obtenir aquest valor exacte i així l'error en el recorregut serà mínim. Aquest valor serà introduït un cop abans de l'inici d'iniciar el tram de calibració.

Introduïts aquest dos valors i preparats en el lloc de sortida, podrem procedir a relaitzar el tram. Es realitzarà el tram de calibratge fins al final d'aquest. A la pantalla és mostra en tot moment el valor de la distancia que es portarà recorreguda. Al arribar al final del tram i confirmar que s'ha arribat al final, és realitza una comparació entre la distancia donada per l'organització i la realitzada per nosaltres, llavors és realitzarà el càlcul per saber amb quin valor s'hauria d'introduir a la primera pantalla, valor de diàmetre roda, el qual quedarà guardat de nou a la memòria EEPROM.

```
// vcorrecion=vdistOrganizacion / vdistrealTotalKM;
        vcorrecion= vdistrealTotalKM / vDiametroRueda;
        vcorrecion = vdistOrganizacion / vcorrecion;
        tft.setCursor(230, 185); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
      tft.print(vcorrecion,3);
```

Seguidament es mostren les pantalles del menú de la calibració. Aquestes quatre pantalles representen els punts explicats anteriorment.

Figura 20. Menú calibració

3.2.2 Hora

Una altre part, no menys important que la calibració, és la verificació del valor de l'hora. Un mal ajust en l'hora significarà la penalització de tots els controls secrets amb el desfasament amb l'hora de l'organització. Una vegada introduïda la modificació, llavors podrem verificar l'hora, poden canviar-la en qualsevol altre moment i anar realitzant les proves corresponents.

Per poder modificar l'hora, s'hauran d'utilitzar els polsadors del manillar fins a poder tenir l'hora desitjada i poder tenir una hora igual que la facilitada per l'organització. En la figura 21 es mostra el diagrama de la part del menú per l'ajust de l'hora:

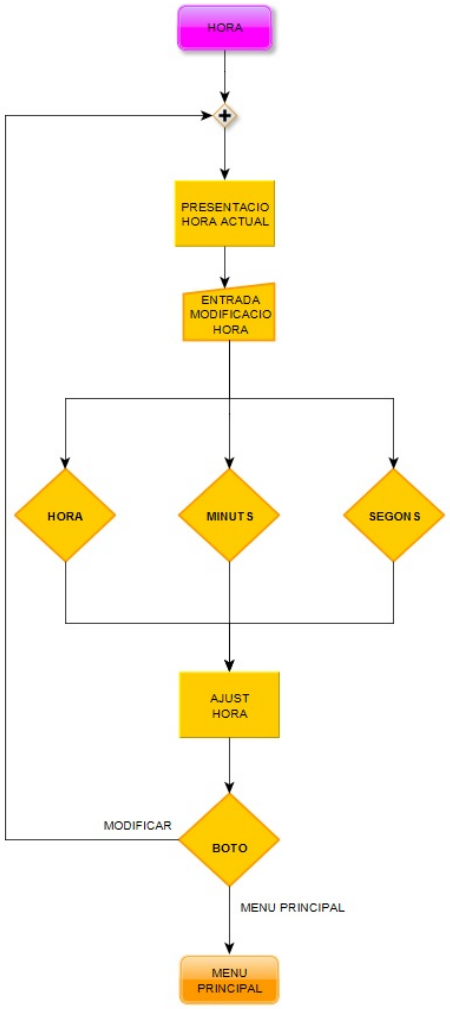

Figura 21. Menú ajust hora

En l'ajust de l'hora primerament s'han creat unes variables per a poder ser modificades per poder diferenciar entre hores, minuts i segons.

```
// Prepara variables
      byte hora, minuto, segundos;
      hora = horatmp;
      minuto = mintmp;
      segundos = segtmp;
```

Llavors mitjançant uns cursos en cada valor, és procedirà a modificar l'hora si escau. Aquest procés el podrem realitzar les vegades que sigui necessària fins a tenir una sincronització perfecta amb l'hora donada per l'organització. Un cop ajustats seran modificats i l'hora de les pantalles i de l'Arduino quedarà canviada. A continuació és mostra la modificació:

```
// Modifica Relog segun valores introducidos
       rtc.stopRTC(); //stop the RTC
       // Ajusta hora, minutos, seg
       rtc.setTime(hora, minuto, segundos);
       // Arranca relog
       rtc.startRTC();
       VESTADO=3;
```

A continuació és mostra la pantalla per a la comprovació de l'hora, la qual podem modificar els valors de les hores, els minuts i els segons. El recuadre en color groc significa aquell valor que tenim activat per modificar.

Figura 22. Pantalla ajust d'hora

3.2.3 Bluetooth

La part del programa de transmissió i recepció de dades dels trams és realitzarà mitjançant Bluetooth, per realitzar la transmissió de dades sense fils. Primerament s'activarà el Bluetooth connectat al Arduino. Un cop connectat és mostraran com a visible per poder ser conectat amb l'app. Un cop conectat ja es podrà realitzar la tramesa de dades. A continuació és pot veure la gestió d'events del mòdul bluetooth desde el programa del Arduino quan esta conectat i visible:

```
// Gestion de Eventos Bluetooth
void aciCallback(aci_evt_opcode_t event)
{
       switch(event)
       {
         case ACI_EVT_DEVICE_STARTED:
         Serial.println(F("VISIBLE"));
         vble=1; break;
         case ACI_EVT_CONNECTED:
         Serial.println(F("CONECTADO"));
         vble=2; break;
```

Serà en aquest punt on l'Arduino restarà en espera per a la connexió des de l'App. Un cop introduides les dades a l'aplicació, tots dos elements estaran en disposició de rebre i enviar dades.

Figura 23. Connexió bluetooth

És realitzarà una verificació per comprovar que s'han passat les dades correctament a l'Arduino, mostrant un missatge d'error en cas de fallada de recepció o mala introducció d'aquestes dades. L'enviament de les dades és realitza seqüencialment, si alguna dades no es correctament introduïda o en falta alguna, apareixerà un error d'enviament i rebuda de la trama de dades. Per altre banda, si s'envien dos trams amb el mateix número, l'ultim tram esborrara el tram introduït anteriorment automàticament, sense confirmació per part del usuari.

Si les dades han arribat correctament, aquestes seran mostrades a la pantalla TFT. Un cop verificada la transferència de dades és tornarà al menú principal. Durant el temps que estiguem enviant dades, es romandra a la mateixa pantalla i s'aniran mostrant les dades dels trams introduïts o els possibles errors.

Figura 24. Transmissió dades

A continuació es mostra la part del programa de l'Arduino per a la recepció de les dades enviades des de l'app.

```
// Recepcion de datos Bluetooth
void rxCallback(uint8_t *buffer, uint8_t len)
{
	Serial.print(F("Received "));
	Serial.print(len);
	Serial.println(F(" bytes: "));


	char chartmp;   // caracter recibidos
	int cifratmp;     // CARACTER recibido en decimal
	int dato=0;
	int digito=0;

	int datotmp[100];
	int multiplicador = 1;

	for(int i=0; (i<(len)); i++){
	  chartmp = ((char)buffer[i]);
	  cifratmp = int(chartmp);
	  cifratmp = (cifratmp - 48);
	  //Serial.println(chartmp);


	  if (chartmp != ';') {
		digito++;
		datotmp[digito] = cifratmp;
		//Serial.print("datotmp");Serial.print(" ");Serial.println(datotmp[digito]);
	  }

	  if (chartmp == ';') {
		for (byte a=0; a<digito; a++){
			dato = dato + ((datotmp[digito-(a)] * multiplicador))  ;
			multiplicador = (multiplicador * 10);
			//Serial.println(dato);
		}

		vCarreratmp[bytetmp] = dato;
		Serial.println(dato);
		bytetmp++;
		digito=0;
```

Les dades rebudes per l'Arduino seran emmagatzemades a la SD, en el fitxer creat al iniciar l'Arduino, el qual s'anomena "carrera.txt" . Les dades són enviades seqüencialment i cada dada és tractada com un byte. D'aquesta manera cada tram constarà de 12 bytes, on el primer byte és el byte de referencia del número de tram que s'està rebent.

Els bytes del 0 al 9 no s'utilitzaràn per les dades dels trams, ja que els considerem com a bytes de dades genèriques. El primer tram començarà a emmagatzemar-se per el byte número 10. Així doncs com cada tram consta de 12 bytes, el segon tram el seu primer byte d'emmagatzematge serà el 22, i així successivament.

D'aquesta manera estarem limitats a 255 bytes, cosa que no implicarà cap problema ja que, es podran introduir 20 trams si escau, cosa que no sol passar en les proves de regularitat, ja que normalment és realitzan grups de trams de com a màxim 10 trams seguits. Amb temps suficient per a poder carregar més trams.

Per poder realitzar la tramesa de dades, s'hauran d'utilitzar els polsadors del manillar. En la figura 25 es mostra el diagrama de la part del programa per la connexió del bluetooth:

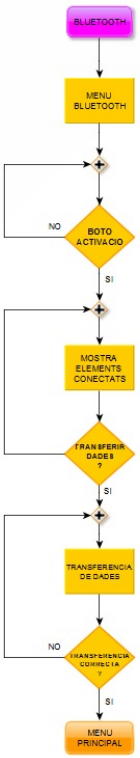

Figura 25. Diagrama del programa

3.2.4 Trams

Els trams estan dividits en dues parts: el tram cronometrat i el tram d'enllaç. El tram cronometrat és aquell on l'organització ens realitzarà el controls secrets. Llavors, entenem com a tram d'enllaç la part del tram des del final del tram cronometrat fins a la sortida del següent tram.

Primerament s'haurà d'escollir el tram que volem realitzar. Només seran mostrats aquells que s'hagin rebut. Un cop triat aquest, és procedirà a marcar l'inici del tram. Seguidament és realitzarà un càlcul entre la hora de sortida i la hora real, així d'aquesta manera sabrem el temps restant per la sortida de tram. Quan sigui la hora de sortida del tram començarà el tram pròpiament dit.

Les dades enviades anteriorment, estaran declarades com a variables i guardades en l'array de treball vCarrera. Serà llavors amb aquesta variable, on les dades enmagatzemades a la SD, en l'arxiu "carrera.txt", d'on procederian a realitzar-se els càlculs en els trams.

Seguidament és mostra la realització del compte enrere amb la diferencia d'hores:

```
// Calcul diferencia horaria "compte enrrera"
void FCuentaatras(byte hora, byte minuto, byte segundo){
      // el resultado queda almacenado en las variables publicas vcaseg, vcamin, vcahora;

      byte segactual, minactual, horaactual;
      long int segundosactuales, segundossalida;

      // Captura hora actual
      minactual = rtc.getMinutes();
      horaactual = rtc.getHours();
      segactual= rtc.getSeconds();

      // Calcul diferencia de segons
      segundosactuales= ((horaactual*3600) + (minactual*60) + segactual);
      segundossalida = ((hora*3600) + (minuto * 60) + segundo);
      vcuentaatras = segundossalida - segundosactuales;
```

Durant el tram cronometrat, la pantalla ens donarà la informació bàsica per a realitzar el tram. S'observarà la velocitat mitja programada, els kilòmetres totals, els kilòmetres parcials i la diferencia de metres calculats entre els metres teòrics i els metres reals. Aquest metres de diferencia seran els metres que per obtenir uns bons resultats hauran de ser en tot el temps possible iguals a cero.

Figura 26. Pantalles de trams

Un cop arribat al final del tram cronometrat, és el moment del tram d'enllaç. Serà llavors quan els valors de la pantalla canviaran. La pantalla mostrarà els kilòmetres totals, els kilòmetres que resten fins al final i el temps restant fins a l'arribada al següent tram.

A la finalització del tram d'enllaç, al confirmar que el tram a finalitzat, és tornarà a la pantalla d'inici d'elecció de trams per poder continuar amb la prova de regularitat.

Per poder realitzar l'elecció dels trams i donar l'acció de finalització d'aquest és realitzarà mitjançant els polsadors del manillar de la motocicleta. En la figura 27 es mostra el diagrama de la part del programa per la realització dels trams:

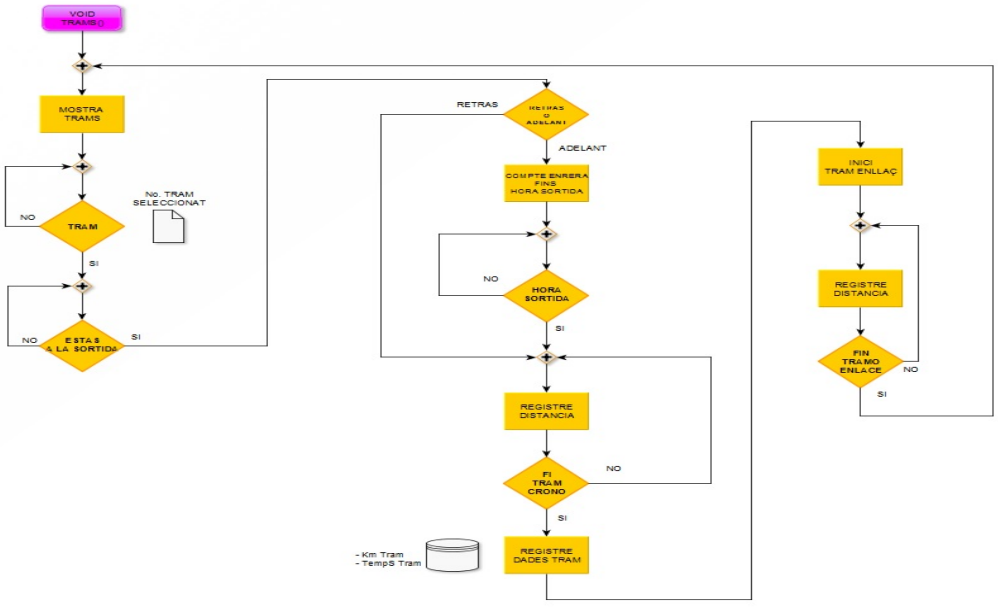

Figura 27. Diagrama del programa

4 RESUM DEL PRESSUPOST

El cost total d'aquest projecte del comprovador d'instrumentació és de mil tres-cents noranta tres euros amb noranta quatre cèntims, sense IVA.

5 CONCLUSIONS

S'han realitzat satisfactòriament els objectius bàsics que es pretenien aconseguir a l'inici del projecte.

S'ha aconseguit el disseny d'un contrololador de regularitat per a motocicletes. Amb aquest controlador s'aconseguirà tenir una bona precisió i una resposta ràpida per poder preparar futures proves de regularitat a través d'una calibració amb una alta precisió, un rellotge precís sense variacions amb dies de proves i una implementació visual clara.

S'aconseguit tenir un sistema lleuger amb un element per instal·lar a la motocicleta i l'utilització del telèfon mòbil, que avui en dia tothom el porta a sobre, estalviant així portar més aparells. A més a més, l'introducció de les daes i el seu tractament és ràpid i alhora en tot moment les comprovacions i els possibles errors són failc de detectar i solucionar.

Aquest projecte, degut al sobredimensionament de la placa Arduino, pot tenir un ampli ventall de possibilitats de millores i ampliacions. Algunes d'aquestes millores serien: el poder realitzar una doble captació de senyals, l'ampliació de la pantalla TFT, l'utilització d'un mòdul giroscopi per encendre uns fars supletoris per la motocicleta.

Degut ha que l'apartat de calibració és important, una millorar per disminuir l'error en la captació del senyal de la roda, seria la instal·lació de més captadors, per aconseguir d'aquesta manera tenir menys recorregut entre senyal i senyal.

Miquel Pumarola Garcia
Graduat en Enginyeria Electrònica Industrial i Automàtica

Blanes, 24 de maig del 2015

6 RELACIÓ DE DOCUMENTS

Aquest projecte es conforme de cinc documents independents. Aquests són la memòria, els plànols, el plec de condicions, l'estat d'amidaments i finalment el pressupost.

7 BIBLIOGRAFIA

ARDUINO. (http://arduino.cc/en/Main/Software, 1 de maig de 2015)

ARDUINO. (http://arduino.cc/en/Main/ArduinoBoardUno, 1 de maig de 2015)

ARDUTEKA. (http://www.arduteka.com/arduino-para-expertos, 4 de maig de 2015)

BRICOGEEK. (http://blog.bricogeek.com/noticias/arduino, 10 d'abril de 2015)

CETRONIC. (http://www.cetronic.es, 10 d'abril de 2015)

EIMODULE. (http://www.eimodule.com, 10 d'abril de 2015)

FARNELL. (http://es.farnell.com/?CMP=ffp-dynF, 10 d'abril de 2015)

MICROSYSTEMS ENGINNERING. (http://www.msebilbao.com/tienda/index.php, 10 d'abril de 2015)

RS COMPONENTS. (http://es.rs-online.com/web, 10 d'abril de 2015)

SPARKFUN ELECTRÒNIC. (https://www.sparkfun.com/categories 10 d'abril de 2015)

8 GLOSSARI

AC: Corrent Alterna (Alternate Current)

ASCII: Codi estàndard americà per a l'intercanvi d'informació. (American Standard Code for Information Interchange)

DC: Corrent Continua (Continuous Current)

GND: Massa. (Common Ground for data and power)

I^2C: Circuits Inter-Integrats. (Inter-Integrated Circuit)

IEC: Comisio electrotècnica internacional. (International Electrotechnical Commision)

iOS: Sistema operatiu mòbil. (Iphone OS)

IP: Index de protecció.

KB: Kilobyte

LED: Díode Emissor de Llum. (Light Emitting Diode)

NA: Normalment obert

PC: Ordinador personal. (Personal Computer)

PWM: Modulació d'ample de pols. (Pulse Width Modulation)

RTC: Rellotge de temps real. (Real Time Clock)

SCA: Serial d'entrada/sortida de dades. (Serial data Input/Ouput)

SCL: Entrada rellotge de sèrie. (Serial Clock Input)

SD: Seguretat digital. (Secure Digital)

SPI: Interfície per a perifèrics sèrie. (Serial Peripheral Interficie)

TFT: Transistor de pel·lícula fina. (Thin Film Transistor)

UART: Transitor/receptor asíncron universal. (Universal Asynchronous Receiver-Transmitter)

USB: Bus sèrie universal. (Universal Serial Bus)

A. PROGRAMA INFORMÀTIC

```
/*Arduino Mega 2560 R3                                                    *
Programa de control de regularitat per a motocicletes                    *
Miquel Pumarola Garcia                                                   *
**********************************************************

 ESTRUCTURA DADES SDCARD
 Los bytes 0-9 contiene datos genericos
 Cada tramo ocupa 10 bytes, empezando desde 10 ej:
      tramo 1  --> byte 10-19
      tramo 2  --> byte 20-29
      --
      tramo 50 --> byte 50-59
 Nombre Fichero: CARRERA
   BYTE     DATOS
      0         Numero de tramos
      10                Hora SALIDA
      11                Minuto SALIDA
      12                Segundo SALIDA
      13                Tiempo TRAMO. Horas
      14                Tiempo TRAMO. Minutos
      15                Distancia Tramo en metros HIGHBYTE
      16                Distancia Tramo en metros LOWBYTE
      17                Distancia CRONO en metros HIGHBYTE
      18                Distancia CRONO en metros LOWBYTE
      19                Velocidad CRONO en Km/h
 */

// Declaracion de librerias ----------------------------------------------------
#include <avr/pgmspace.h>               // libreria para poder guardar variables en FLASH
#include <avr/wdt.h>                    // libreria para poder guardar variables en FLASH
#include <Timer.h>                      // libreria para temporizaciones
#include <Adafruit_GFX.h>               // libreria grafica
#include <SPI.h>                        // librería comunicaciones SPI
#include <Wire.h>                       // libreria comunicaciones GENERICA
#include <Adafruit_ILI9341.h>           // libreria comunicaciones TFT, Touch y SD
#include <Adafruit_STMPE610.h>          // libreria gestion touch
#include <SD.h>                         // libreria gestion SD card
#include <EEPROM.h>                     // Librería gestión EEPROM
#include <Adafruit_BLE_UART.h>          // libreria gestión modulo Bluetooth
#include <swRTC.h>                      // libreria relog en tiempo real

swRTC rtc;                              // Crea una instancia al relog


// Define los pines que utiliza el modulo Bluetoth
#define ADAFRUITBLE_REQ 53
#define ADAFRUITBLE_RDY 19
#define ADAFRUITBLE_RST 49

Adafruit_BLE_UART BTLEserial = Adafruit_BLE_UART(ADAFRUITBLE_REQ, ADAFRUITBLE_RDY, ADAFRUITBLE_RST);

// Declaracion de las instancias para trabajar con la SD Card
Sd2Card card;
SdVolume volume;
SdFile root;
File myFile;
// Permite que los mensajes que se envien por Println se manejen desde Flash
char p_buffer[80];
#define P(str) (strcpy_P(p_buffer, PSTR(str)), p_buffer)


// Definicion de colores personalizados
#define  ILI9341_GRIS 0xffA7ACAD
#define  ILI9341_GRISFUERTE  0xff70686f
#define  ILI9341_VERDEPASTEL 0xffc5866f


// Datos de calibracion del panel tactil
#define TS_MINX 150
```

```
#define TS_MINY 130
#define TS_MAXX 3800
#define TS_MAXY 4000

// Define Pines para Touch
#define STMPE_CS 8
Adafruit_STMPE610 ts = Adafruit_STMPE610(STMPE_CS);

// Define Pines para TFT
#define TFT_CS 10
#define TFT_DC 9
Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC);

// Define Pines para SD Card
#define SD_CS 4

// Definicion de Pines de ENTRADA DIGITALES
int PUL1 = 23;                          // Imagen Pulsador 1
int PUL2 = 24;                          // Imagen Pulsador 2
int PUL3 = 25;                          // Imagen Pulsador 3
int SENSOR = 22;                        // Imagen Señal SENSOR

// Definicion de Pines de SALIDA DIGITALES
int LEDS = 26;                          // Leds

//Definicion de Pines de SALIDA ANALOGICOS
int BACKLIGHT = 3;                      // SALIDA PWM para control BACKLIGHT TFT


// DEFINICION DE VARIABLES TABLA DE ENTRADA
boolean vPUL1;                          // Variable Pulsador 1
boolean vPUL2;                          // Variable Pulsador 2
boolean vPUL3;                          // Variable Pulsador 3
boolean vPUL1old, vPUL2old, vPUL3old;
boolean vSENSOR;                        // Variable SENSOR
boolean vSENSOROLD = false;             // Memoria pulso anterior


// DECLARACION DE VARIABLES TABLA DE SALIDA
int vLEDS;                              // Variable LEDS



// Declaracion de VARIABLES PUBLICAS GLOBALES
byte VESTADO = 0;                       // Estado operativo actual



// Declaracion de variables almacenadas en EEPROM
int vDiametroRueda;                     // DIAMETRO RUEDA

// Declaracion variables CALCULOS DE FECHAS
byte vhora, vmin, vseg, vdia, vmes;
int vyear;

// Declaracion variables RELOG
byte segold, minold, horaold, dayold, mesold, dsemold;
byte segtmp, mintmp, horatmp, daytmp, mestmp,dsemtmp;
int yearold;
int yeartmp;

// Declaracion variables calculo tiempo de ciclo
unsigned long vciclo;                   // tiempo de ciclo
unsigned long vciclo1;                  // tiempo de ciclo temporal para calculos
unsigned long vciclo2;
```

```
// Declaracion variables  Pulsaciones Touch
unsigned int x , y , z;
unsigned int x1, y1, z1;
unsigned int xtmp, ytmp, ztmp;
unsigned int xtmp1, ytmp1,ztmp1;
boolean pulMemo;                        // Memoria de Pulsacion

// Declaracion Variables calibracion
int vcalrueda;                          // Medida diametro rueda
int vdistOrganizacion = 0;              // Distancia organizacion
float vdistrpm;                         // Distancia recorrida real en mm por vuelta de rueda
float vdistrealTotalKM;                 // Distancia recorrida total en Km
float vdistrealtotalKmold;              // Variable temporal auxiliar
float vcorrecion;                       // Factor de correccion sobre la medida real

// Declaracion Variables tramo en curso
float vtspeedReal;
float vtspeedmediaReal;
float vtspeedmediaOrganizacion;
float vtKmparcial;
long int vttiemporealinicial;
long int vttiemporealfinal;

long int vtTiempoTramoSegs;
long int vtTiempoCronoSegs;
long int vtDistTramoMetros;
long int vtdistCronoMetros;


long vnumpulsos;
long vnumpulsosparcial;



// Declaracion Variables Datos tramo
byte vCarreratmp[200];
byte vCarrera[200];
byte vntramos;                          // numero de tramos


// Declaracion Variables RX blu
byte vnumTramo;
byte vHoraSalida;
byte vMinutoSalida;
byte vSegundoSalida;
byte vTiempoHorasTramo;
byte vTiempoMinutosTramo;
byte vDistKmTramo;
byte vDistMtrTramo;
byte vDistKmCrono;
byte vDistMtrCrono;
byte vSpeedKmCrono;
byte vSpeedmCrono;

byte numRXbytes;

// Declaracion Variables Cuenta atras
byte vcaseg, vcamin, vcahora;
signed int segcalc, horacalc, mincalc;
signed long int vcuentaatras;

// Declaracion Variables BLE
byte vble=0;
boolean estabaold =0;
byte bytetmp=1;
```

```
// Declaracion de de variables TIMER
int t4;
boolean t4on=false;                     // Timer antirebote Pul1

int t5;
boolean t5on=false;                     // timer antirebote Pul2

int t6;
boolean t6on=false;                     // timer antirebote Pul3
void setup()
{
      MCUSR = 0;                                // limpia el flag de resets anteriores.

      // Inicializa el puerto serie
      Serial.begin(38400);

            // DECLARACION E/S
      // Declaracion de entradas
      pinMode(PUL1,INPUT);                      //Entrada Pulsador 1
      pinMode(PUL2,INPUT);                      //Entrada Pulsador 2
      pinMode(PUL3,INPUT);                      //Entrada Pulsador 3
      pinMode(SENSOR,INPUT);                    //Entrada Sensor
      pinMode(ADAFRUITBLE_REQ,53);              // Bluetooth REQ
      pinMode(ADAFRUITBLE_RDY,19);              // Bluetooth RDY

      //Declaracion de Salidas
      pinMode(LEDS,OUTPUT);
      pinMode(ADAFRUITBLE_RST,49);              //Bluetooth RST

      //Declaracion de Salidas PWM
      pinMode(3,OUTPUT);                        //BACKLIGHT PWM

      //Declaracion de I/O RESERVA
      pinMode(5,INPUT_PULLUP);
      pinMode(6,INPUT_PULLUP);
      pinMode(14,INPUT_PULLUP);
      pinMode(15,INPUT_PULLUP);
      pinMode(16,INPUT_PULLUP);
      pinMode(18,INPUT_PULLUP);
      pinMode(19,INPUT_PULLUP);
      pinMode(34,INPUT_PULLUP);

      // Inicializa la periferia
      inicHardware();

      // Verifica si es la primera instalacion y crea las variables por defecto
      VerificaEeprom();

      // Carga variables de Eeprom a Memoria
      cargaEeprom();

      // Carga datos de SD
      cargaSD();

      //inicializa Bluetooth
      BTLEserial.setRXcallback(rxCallback);
      BTLEserial.setACIcallback(aciCallback);
      BTLEserial.begin();


      // Inicializa relog
      rtc.stopRTC();                            //stop the RTC
      rtc.setTime(18, 58, 50);          // Ajusta hora, minutos, seg
      rtc.setDate(1, 1, 2015);          // Ajusta Dia, Mes, Año
      rtc.startRTC();


}
```

```
void loop()
{
	// Chequea Bluetooth periodicamente
	BTLEserial.pollACI();

	// Calculos de ciclo
	vciclo1=millis();

		// Chequea timers periodicamente
	t.update();

	///// T E M P O R I Z A C I O N E S /////
	// Actualiza el relog en pantalla
	if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}
	///// F I N    T E M P O R I Z A C I O N E S  /////


	// Actualiza TABLA ENTRADAS
	lecturaIN();

	// Detecta Pulsaciones Touch
	Pulsacion();

	// Carga Pantalla Logo o Menu
	if (VESTADO == 1) {pantalla0();}
	if (VESTADO == 2) {pantalla0();}
	if (VESTADO == 3) {pantallaInicio();}


	// Calculo del ciclo
	vciclo2 = millis();
	vciclo = vciclo2 - vciclo1;
}

// FUNCION Inicializacion Perifericos
void inicHardware(){
	// Inicializa la Pantalla
	Serial.println(P("Comprobacion Pantalla"));
	tft.begin();
	if (!ts.begin()) {
	  Serial.println(P("No se ha INICIALIZADO LA PANTALLA"));
	}
	else {
	  tft.fillScreen(ILI9341_BLACK);
	  tft.setRotation(1);
	  tft.setCursor(0, 0);
	}

	// VERIFICA LECTOR SD
	Serial.println(P("Comprobacion Lector SD"));
	if (!SD.begin(SD_CS)) {
	  Serial.println(P("- SD.........NO SD"));
	  tft.setTextColor(ILI9341_RED); tft.println(P("- SD.........NO SD"));
	  tft.setTextColor(ILI9341_GREEN);
	}
	else{
	  Serial.println(P("- SD.......OK"));
	  tft.println(P("- SD.........OK"));
	}

	// verifica si la tarjeta esta insertada y formateada
	Serial.println(P("Comp.Formato SD CArd"));
	if (!card.init(SPI_HALF_SPEED, SD_CS)) {
	  Serial.println(P("SD CARD Sin Formato"));
	  tft.setTextColor(ILI9341_RED);tft.println(P("              SIN FORMATO"));
	  tft.setTextColor(ILI9341_GREEN);
	}
	else {
	  Serial.println(P("           SD CARD FORMATEADA"));
	  tft.setTextColor(ILI9341_GREEN);tft.println(P("              FORMATO OK"));
	}

	// Indica el tipo de la tarjeta
	switch(card.type()) {
```

```
        case SD_CARD_TYPE_SD1:
        Serial.println(P("           SD1"));
        tft.println(P("              SD1"));
        break;
        case SD_CARD_TYPE_SD2:
        Serial.println(P("           SD2"));
        tft.println(P("              SD2"));
        break;
        case SD_CARD_TYPE_SDHC:
        Serial.println(P("           SD3"));
        tft.println(P("              SDHC"));
        break;
        default:
        Serial.println(P("           Desconocido"));
      }

 // Abrimos el volumen (FAT16 or FAT32). Es necesario para acceder a mas informacion
      if (!volume.init(card)) {
               Serial.println(P("Could not find FAT16/FAT32 partition.\nMake sure you've formatted
        the card"));
      }
      else {
        // Tipo y tamaño del volumen
        uint32_t volumesize;
          Serial.print(P("\nVolume type is FAT")); Serial.println(volume.fatType(), DEC);
        tft.print(P("             FAT")); tft.println(volume.fatType(), DEC);
        //
        volumesize = volume.blocksPerCluster();
        volumesize *= volume.clusterCount();
        volumesize *= 512;
        Serial.print(P("Volume size (Mbytes): "));
        volumesize /= 1024;
        volumesize /= 1000;
        Serial.println(volumesize);
        tft.print(P("              ")); tft.print(volumesize); tft.println(P(" Mb"));
      }

      // Inicializa el touch
      if (!ts.begin()) {

      } else {
        Serial.println("Touchscreen started.");

      }


      // Fin Inicializacion
      tft.println(P(" "));
      Serial.println(P(" "));
      tft.print(P("RAM libre.."));  tft.println(FreeRam());
      Serial.print(P("RAM libre.."));  Serial.println(FreeRam());
}



// FUNCION Carga valores por defecto en EEPROM si es necesario
void VerificaEeprom(){
 // Verifica que exista la primera variable, si no crea en eeprom valores por defecto
      byte valor = EEPROM.read(0);
      if (valor != 13){
        Serial.println(P(".. EEPROM. CORRUPTA"));
        // escribimos CEROS en las primeros 50 bytes
        for (int i = 0; i < 51; i++){EEPROM.write(i, 0);}
        Serial.println(P(".. EEPROM. FORMATEADA"));
        // Ahora creamos la variables por defecto
        EEPROM.write(0, 13);                              // Variable de control
        // Utilizamos el valor de 17" (432mm como medida referencia)
                // Diametro rueda (Byte HIGH)
        EEPROM.write(1, (highByte(432)));
        // diametro rueda (Byte LOW)
        EEPROM.write(2, (lowByte(432)));

        Serial.println(P(".. EEPROM. CREADOS DATOS POR DEFECTO"));
      }
```

```
        else {
          Serial.println(P(".. EEPROM CORRECTA"));
        }
        // SET EL ESTADO
        VESTADO=1;
        consola();
}



//Carga EEPROM a Memoria
void cargaEeprom(){
        byte lowtmp, hightmp;

        hightmp = EEPROM.read(1);                          // Diametro rueda (bytehigh)
        lowtmp = EEPROM.read(2);                                  // Diametro rueda (bytelow)
        vDiametroRueda = word(hightmp,lowtmp);

        // Calculo distancia recorrida por rpm
        vdistrpm = ((2.0 * 3.1416) * (vDiametroRueda/2));
        vdistrpm = (vdistrpm /1000);
        // Diametro rueda (bytehigh)
        hightmp = EEPROM.read(3);
        // Diametro rueda (bytelow)
        lowtmp = EEPROM.read(4);
        vdistOrganizacion = word(hightmp,lowtmp);

        Serial.println(P("- Cargadas Variables de EEPROM a Memoria"));

}



// Verifica SD
void verificaSD(){
        // Fichero Carrera.txt
        if (SD.exists("Carrera.txt")) {
        Serial.println(P("Carrera.txt OK."));}
        else {
          Serial.println("Carrera.txt no existe");
          myFile = SD.open("Carrera.txt", FILE_WRITE);

          if (myFile){
                  for (byte a=1; a<255; a++){
                          myFile.seek(a);
                          myFile.write(byte(0));
                  }
          }
          myFile.close();
          Serial.println(P("Creado Fichero --> Carrera.txt"));
        }
}

// Carga parametros de SD a Memoria
void cargaSD(){
        // Carga datos almacenados en SD a la matriz vCarrera[]
        myFile = SD.open("Carrera.txt", FILE_READ);
        if (myFile){
          myFile.seek(1);
          if (myFile.read() !=0) {
                  for (byte a=10; a<255; a++){
                          myFile.seek(a);
                          vCarrera[a]=myFile.read();
                  }
          }
          Serial.println(P("Tramos Cargados de SD"));
        }
        else {
          for (byte a=0; a<255; a++){
                  vCarrera[a]=0;
          }
          Serial.println(P("MEMORIA TRAMOS INICIALIZADA A 0"));
        }
```

```
      myFile.close();
      Serial.println(P("Datos Carrera almacenados en SD Card"));
      myFile.close();

}

// Rutina de lectura de TABLA DE ENTRADAS
void lecturaIN(){


      // Deteccion de flancos de bajada en Pulsador 1 y antirebote
      if ((!digitalRead(PUL1)) & (vPUL1old==true) & (t4on==false)) {
        vPUL1=true; vPUL1old=false;
        if (!t4on) {t4 = t.after(100, t4antirebote); t4on=true;}
      }
      if ((t4on==false) & (digitalRead(PUL1))) {vPUL1old=true;}

      // Deteccion de flancos de bajada en Pulsador 2 y antirebote
      if ((!digitalRead(PUL2)) & (vPUL2old==true) & (t5on==false)) {
        vPUL2=true; vPUL2old=false;
        if (!t5on) {t5 = t.after(100, t5antirebote); t5on=true;}
      }
      if ((t5on==false) & (digitalRead(PUL2))) {vPUL2old=true;}

      // Deteccion de flancos de bajada en Pulsador 3 y antirebote
      if ((!digitalRead(PUL3)) & (vPUL3old==true) & (t6on==false)) {
        vPUL3=true; vPUL3old=false;
        if (!t6on) {t6 = t.after(100, t6antirebote); t6on=true;}
      }
      if ((t6on==false) & (digitalRead(PUL3))) {vPUL3old=true;}



      vSENSOR = digitalRead(SENSOR);            // Sensor


      if ((vSENSOR==true) && (vSENSOROLD==false)) {
        //vdistrpm = ((2.0 * 3.1416) * (vDiametroRueda/2));
        //vdistrpm = (vdistrpm /1000);
        //vdistrealTotalKM = vdistrealTotalKM + vdistrpm;

        vnumpulsos++;
        vnumpulsosparcial++;

        // Calculo distancia real acumulada
        vdistrealTotalKM = (vnumpulsos*vdistrpm);
        vtKmparcial = (vnumpulsosparcial*vdistrpm);
        vSENSOROLD = true;



      }

      if (vSENSOR==false) {(vSENSOROLD=false);}
      //Serial.print (P("distancia -> ")); Serial.println(vdistrealTotalKM);
}



// Timers antirebote
void t4antirebote(){
      if (t4on) {t.stop(t4); t4on=false;}
}

void t5antirebote(){
      if (t5on) {t.stop(t5); t5on=false;}
}

void t6antirebote(){
      if (t6on) {t.stop(t6); t6on=false;}
}
```

```
// Capturas Touch panel
void Pulsacion(){
       // x1 - y1 se encuentran disponibles cuando se suelta el boton

       TS_Point p= ts.getPoint();
       inicio:
       while (ts.touched()){TS_Point p = ts.getPoint();pulMemo=1 ;}

       if (pulMemo==1) {
         pulMemo=0;
         TS_Point p = ts.getPoint();

         p.x = map(p.x, TS_MINY, TS_MAXY, 0, tft.height());
         // x
         p.y = map(p.y, TS_MINX, TS_MAXX, 0, tft.width());
         // y
         ztmp= p.z;

         ytmp = tft.height() - p.x;
         xtmp = p.y;

         if ((xtmp==xtmp1) && (ytmp==ytmp1) && (ztmp==ztmp1)){goto inicio;}
         xtmp1=xtmp;ytmp1=ytmp;ztmp1=ztmp;

         x=xtmp; y=ytmp;
         x1=x; y1=y;


         Serial.print(P("X > ")); Serial.print(x);Serial.print(P("  y > ")); Serial.println(y);
       }
uint16_t f;

do {
       f = ts.bufferSize();
       TS_Point p=ts.getPoint();

} while (ts.bufferSize()!=0);

}



// TFT. Presenta la Pantalla 0 (por defecto)
void pantalla0(){
       VESTADO=2;
       consola();
       // borra Pantalla sin advertencia presetup mostrada
       tft.fillScreen(ILI9341_BLACK);

       // Dibuja recuadros
       tft.drawRoundRect(2,1,318,33,5,ILI9341_BLUE);
       tft.drawRoundRect(2,35,318,205,5,ILI9341_BLUE);
       // Dibuja Texto
       tft.setCursor(60, 11); tft.setTextColor(ILI9341_YELLOW); tft.setTextSize(2); tft.print(P("@@@@
       WELCOME @@@@"));
       tft.setCursor(65,70); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("Regularitat"));
       tft.setCursor(65,115); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3); tft.print(P("
       Miquel"));
       delay(2000);
       VESTADO=3; consola();

}


// TFT. Presenta MENU INICIAL
void pantallaInicio(){
       // borra Pantalla
       FborraPantalla();

       // Pinta Titul
       tft.setCursor(150, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2); tft.print(P("M E
       N U"));
       byte vseleccion = 1;
```

```
tft.setCursor(45, 65); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CALIBRACIO"));
tft.setCursor(45, 105); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("HORA"));
tft.setCursor(45, 145); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CONNEXIO"));
tft.setCursor(45, 185); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("TRAM"));


minold=0; horaold=0;
do
 {
 // Actualitza el rellotge en pantalla
 t.update();
 if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}
 // Captura entrades
 lecturaIN();

 if (vPUL1==true) {
         vseleccion--;
         if ((vseleccion < 1) | (vseleccion==1)) vseleccion=1;
         vPUL1=false;
 }

 if (vPUL3==true){
         vseleccion++;
         if (vseleccion > 4) vseleccion=4;
         vPUL3=false;
 }

 switch (vseleccion) {
         case 1:
                 tft.setCursor(45, 65); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(P("CALIBRACIO"));
                 tft.setCursor(45, 105); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("HORA"));
                 tft.setCursor(45, 145); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CONNEXIO"));
                 tft.setCursor(45, 185); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("TRAM"));
                 break;

         case 2:
                 tft.setCursor(45, 105); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(P("HORA"));
                 tft.setCursor(45, 65); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CALIBRACIO"));
                 tft.setCursor(45, 145); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CONNEXIO"));
                 tft.setCursor(45, 185); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("TRAM"));

                 break;
         case 3:
                 tft.setCursor(45, 145); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(P("CONNEXIO"));
                 tft.setCursor(45, 65); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CALIBRACIO"));
                 tft.setCursor(45, 105); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("HORA"));
                 tft.setCursor(45, 185); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("TRAM"));
                 break;

         case 4:
                 tft.setCursor(45, 185); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(P("TRAM"));
                 tft.setCursor(45, 65); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("CALIBRACIO"));
                 tft.setCursor(45, 105); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
tft.print(P("HORA"));
```

```
                        tft.setCursor(45, 145); tft.setTextColor(ILI9341_RED); tft.setTextSize(4);
        tft.print(P("CONNEXIO"));
          }
          }

        while (vPUL2==false);
        vPUL2=false;
        switch (vseleccion) {
                        case 1:
                                calibracio1();
                        break;

                        case 2:
                                horaold=0;minold=0;
                                Prelog();

                        break;

                        case 3:
                                horaold=0;minold=0;
                                vPUL2=false;
                                conexio();

                        break;

                        case 4:
                                horaold=0;minold=0;
                                vPUL2=false;
                                trams();
        }
}

// Calibracion Paso 1. Datos Rueda
void calibracio1(){
        /*
        vcalrueda         --> Medida diametro rueda
        vdistOrganizacion --> Distancia organizacion
        vdistreal          --> Distancia recorrida real
        vcorrecion        --> Factor de correccion sobre la medida real
        */

        // Borra Pantalla Menu Principal
        FborraPantalla();
        // fuerza refresco relog
        minold=0;horaold=0;

        tft.setCursor(140, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(P("Calibracio 1/4"));

        // Pinta Datos calibracion rueda
        tft.setCursor(78, 45); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(P("Calibracio roda"));

        // Declaracion variables calculos temporales
        byte vtunidad, vtdecena, vtemporal, vtcentena;

        // Saparacion valor calibracion en digitos
        vtcentena = (vDiametroRueda / 100);
        vtdecena = ((vDiametroRueda - (vtcentena *100)) / 10);
        vtunidad = (vDiametroRueda - ((vtcentena *100)  + (vtdecena *10)));

        // Presentacion digitos
        tft.setCursor(95, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtcentena);
        tft.setCursor(155, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtdecena);
        tft.setCursor(215, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtunidad);

        // Recuadros Digitos
        tft.drawRect(85,130,40,50,ILI9341_RED);   // Centena
        tft.drawRect(145,130,40,50,ILI9341_RED);  // Decena
        tft.drawRect(205,130,40,50,ILI9341_RED);  // Unidad
```

```
// Presentacion Botones

//tft.fillRect(85,80,40,40,ILI9341_RED);
Ftriangulo(0,87,120,6,ILI9341_RED);

//tft.fillRect(145,80,40,40,ILI9341_RED);
Ftriangulo(0,147,120,6,ILI9341_RED);

//tft.fillRect(205,80,40,40,ILI9341_RED);
Ftriangulo(0,207,120,6,ILI9341_RED);

//tft.fillRect(85,190,40,40,ILI9341_RED);
Ftriangulo(2,87,190,6,ILI9341_RED);

//tft.fillRect(145,190,40,40,ILI9341_RED);
Ftriangulo(2,147,190,6,ILI9341_RED);

//tft.fillRect(205,190,40,40,ILI9341_RED);
Ftriangulo(2,207,190,6,ILI9341_RED);


// bucle modificacion valor
byte seleccion = 1;
boolean salir = false;
do
{
 Pulsacion();
 lecturaIN();
 // salida
 if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; goto salto;}

 if (vPUL2==true) {
        seleccion++;
        vPUL2=false;
        if (seleccion > 3) {seleccion=1;}
 }

 if (seleccion==1) {
        tft.drawRect(85,130,40,50,ILI9341_YELLOW);                    // Centena
        tft.drawRect(145,130,40,50,ILI9341_RED);                      // Decena
        tft.drawRect(205,130,40,50,ILI9341_RED);                      // Unidad
 }
 if (seleccion==2) {
        tft.drawRect(85,130,40,50,ILI9341_RED);                       // Centena
        tft.drawRect(145,130,40,50,ILI9341_YELLOW);                   // Decena
        tft.drawRect(205,130,40,50,ILI9341_RED);                      // Unidad
 }
 if (seleccion==3) {
        tft.drawRect(85,130,40,50,ILI9341_RED);                       // Centena
        tft.drawRect(145,130,40,50,ILI9341_RED);                      // Decena
        tft.drawRect(205,130,40,50,ILI9341_YELLOW);                   // Unidad
 }


 // Actualiza el relog en pantalla
 t.update();
 if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

 // Pulsacion  + centenas
 if (((x>83 && x<123) && (y>89 && y< 126)) || ((seleccion==1) && (vPUL1==true))) {
        vPUL1=false;
        x=0;y=0;
        vtcentena++;
        if (vtcentena>9) {vtcentena=0;}
        tft.fillRect(86,131,38,48,ILI9341_BLACK);
        tft.setCursor(95, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtcentena);
 }
 // Pulsacion  + decenas
 if (((x>141 && x<182) && (y>87 && y< 123)) || ((seleccion==2) && (vPUL1==true))) {
        vPUL1=false;
        x=0;y=0;
        vtdecena++;
        if (vtdecena>9) {vtdecena=0;}
```

```
                tft.fillRect(146,131,38,48,ILI9341_BLACK);
                tft.setCursor(155, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtdecena);
          }
          // Pulsacion  + unidades
          if (((x>200 && x<240) && (y>87 && y< 123)) || ((seleccion==3) && (vPUL1==true))) {
                vPUL1=false;
                x=0;y=0;
                vtunidad++;
                if (vtunidad>9) {vtunidad=0;}
                tft.fillRect(206,131,38,48,ILI9341_BLACK);
                tft.setCursor(215, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtunidad);
          }

          // Pulsacion  - centena
          if (((x>80 && x<121) && (y>192 && y< 226)) || ((seleccion==1) && (vPUL3==true))){
                vPUL3=false;
                x=0;y=0;
                vtcentena--;
                if (vtcentena>200) {vtcentena=9;}
                tft.fillRect(86,131,38,48,ILI9341_BLACK);
                tft.setCursor(95, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtcentena);
          }
          // Pulsacion  - decena
          if (((x>138 && x<180) && (y>190 && y< 226)) || ((seleccion==2) && (vPUL3==true))){
                vPUL3=false;
                x=0;y=0;
                vtdecena--;
                if (vtdecena>200) {vtdecena=9;}
                tft.fillRect(146,131,38,48,ILI9341_BLACK);
                tft.setCursor(155, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtdecena);
          }
          // Pulsacion  - unidad
          if (((x>199 && x<239) && (y>192 && y< 226)) || ((seleccion==3) && (vPUL3==true))){
                vPUL3=false;
                x=0;y=0;
                vtunidad--;
                if (vtunidad>200) {vtunidad=9;}
                tft.fillRect(206,131,38,48,ILI9341_BLACK);
                tft.setCursor(215, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtunidad);
          }

        salto:;
        //
        } while (salir==0);
        vPUL1=false;
        vPUL3=false;
        //  Nuevo valor diametro rueda
        vDiametroRueda = ((vtcentena*100) + (vtdecena * 10) + vtunidad);
        // Almacenamos el valor en EEPROM
        EEPROM.write(1, (highByte(vDiametroRueda)));              // Diametro rueda (Byte HIGH)
        EEPROM.write(2, (lowByte(vDiametroRueda)));               // diametro rueda (Byte LOW

        // Calculo distancia recorrida por rpm
        vdistrpm = ((2.0 * 3.1416) * (vDiametroRueda/2));
        vdistrpm = (vdistrpm /1000);

        // Fuerza a repintar Horas y min en pantalla
        minold=255;
        horaold=255;
        calibracio2();
        //
}



// Calibracion Paso 2. Datos Organizacion
void calibracio2(){
        // Borra Pantalla Menu Principal
        FborraPantalla();
```

```
//     // Dibuja recuadros
//     tft.drawRoundRect(2,1,318,33,7,ILI9341_BLUE);
//     tft.drawRoundRect(2,35,318,205,5,ILI9341_BLUE);

       tft.setCursor(140, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
       tft.print(P("Calibracio 2/4"));

       // Pinta Datos calibracion rueda
       tft.setCursor(60, 45); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
       tft.print(P("Distancia RoadBook"));

       // Declaracion variables calculos temporales
       byte vtunidad, vtdecena, vtcentena, vtumillar, vtdmillar;

       // Saparacion valor calibracion en digitos
       vtdmillar = (vdistOrganizacion / 10000);
       vtumillar = ((vdistOrganizacion - (vtdmillar * 10000)) / 1000);
       vtcentena = (((vdistOrganizacion - (vtdmillar*10000)) - (vtumillar*1000)) /100);
       vtdecena = ((((vdistOrganizacion - (vtdmillar*10000)) - (vtumillar*1000)) - (vtcentena*100)) /
       10);
       vtunidad = ((((vdistOrganizacion - (vtdmillar*10000)) - (vtumillar*1000)) - (vtcentena*100)) -
       (vtdecena * 10));



       // Presentacion digitos
       tft.setCursor(124, 150); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
       tft.print(P("."));
       tft.setCursor(35, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
       tft.print(vtdmillar);
       tft.setCursor(95, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
       tft.print(vtumillar);
       tft.setCursor(155, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
       tft.print(vtcentena);
       tft.setCursor(215, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
       tft.print(vtdecena);
       tft.setCursor(275, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
       tft.print(vtunidad);

       // Recuadros Digitos
       tft.drawRect(25,130,40,50,ILI9341_RED);   // Decena de Millar
       tft.drawRect(85,130,40,50,ILI9341_RED);   // Unidad de Millar
       tft.drawRect(145,130,40,50,ILI9341_RED);  // Centena
       tft.drawRect(205,130,40,50,ILI9341_RED);  // Decena
       tft.drawRect(265,130,40,50,ILI9341_RED);  // Unidad

       // Presentacion Botones +
       //tft.fillRect(25,80,40,40,ILI9341_RED);
       Ftriangulo(0,27,119,6,ILI9341_RED);

       //tft.fillRect(85,80,40,40,ILI9341_RED);
       Ftriangulo(0,87,119,6,ILI9341_RED);

       //tft.fillRect(145,80,40,40,ILI9341_RED);
       Ftriangulo(0,147,119,6,ILI9341_RED);

       //tft.fillRect(205,80,40,40,ILI9341_RED);
       Ftriangulo(0,207,119,6,ILI9341_RED);

       //tft.fillRect(265,80,40,40,ILI9341_RED);
       Ftriangulo(0,267,119,6,ILI9341_RED);

       // Presentacion Botones -
       // tft.fillRect(25,190,40,40,ILI9341_RED);
       Ftriangulo(2,27,190,6,ILI9341_RED);

       // tft.fillRect(85,190,40,40,ILI9341_RED);
       Ftriangulo(2,87,190,6,ILI9341_RED);

       // tft.fillRect(145,190,40,40,ILI9341_RED);
       Ftriangulo(2,147,190,6,ILI9341_RED);

       //tft.fillRect(205,190,40,40,ILI9341_RED);
       Ftriangulo(2,207,190,6,ILI9341_RED);
```

```
//tft.fillRect(265,190,40,40,ILI9341_RED);
Ftriangulo(2,267,190,6,ILI9341_RED);

// bucle modificacion valor
byte seleccion=1;
boolean salir = false;
do
{
  Pulsacion();
  lecturaIN();
  // salida
  if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; goto salto;}

  if (vPUL2==true) {
          seleccion++;
          vPUL2=false;
          if (seleccion > 5) {seleccion=1;}
  }

  if (seleccion==1) {
          tft.drawRect(25,130,40,50,ILI9341_YELLOW);   // Decena de Millar
          tft.drawRect(85,130,40,50,ILI9341_RED);   // Unidad de Millar
          tft.drawRect(145,130,40,50,ILI9341_RED);  // Centena
          tft.drawRect(205,130,40,50,ILI9341_RED);  // Decena
          tft.drawRect(265,130,40,50,ILI9341_RED);  // Unidad
  }
  if (seleccion==2) {
          tft.drawRect(25,130,40,50,ILI9341_RED);   // Decena de Millar
          tft.drawRect(85,130,40,50,ILI9341_YELLOW);   // Unidad de Millar
          tft.drawRect(145,130,40,50,ILI9341_RED);  // Centena
          tft.drawRect(205,130,40,50,ILI9341_RED);  // Decena
          tft.drawRect(265,130,40,50,ILI9341_RED);  // Unidad
  }
  if (seleccion==3) {
          tft.drawRect(25,130,40,50,ILI9341_RED);   // Decena de Millar
          tft.drawRect(85,130,40,50,ILI9341_RED);   // Unidad de Millar
          tft.drawRect(145,130,40,50,ILI9341_YELLOW);  // Centena
          tft.drawRect(205,130,40,50,ILI9341_RED);  // Decena
          tft.drawRect(265,130,40,50,ILI9341_RED);  // Unidad
  }
  if (seleccion==4) {
          tft.drawRect(25,130,40,50,ILI9341_RED);   // Decena de Millar
          tft.drawRect(85,130,40,50,ILI9341_RED);   // Unidad de Millar
          tft.drawRect(145,130,40,50,ILI9341_RED);  // Centena
          tft.drawRect(205,130,40,50,ILI9341_YELLOW);  // Decena
          tft.drawRect(265,130,40,50,ILI9341_RED);  // Unidad
  }
  if (seleccion==5) {
          tft.drawRect(25,130,40,50,ILI9341_RED);   // Decena de Millar
          tft.drawRect(85,130,40,50,ILI9341_RED);   // Unidad de Millar
          tft.drawRect(145,130,40,50,ILI9341_RED);  // Centena
          tft.drawRect(205,130,40,50,ILI9341_RED);  // Decena
          tft.drawRect(265,130,40,50,ILI9341_YELLOW);  // Unidad
  }

  // Actualiza el relog en pantalla
  t.update();
  if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

  // Pulsacion  + Decena de Millar
  if (((x>21 && x<63) && (y>88 && y< 126))  || ((seleccion==1) && (vPUL1==true))) {
          vPUL1=false;
          x=0;y=0;
          vtdmillar++;
          if (vtdmillar>9) {vtdmillar=0;}
          tft.fillRect(26,131,38,48,ILI9341_BLACK);
          tft.setCursor(35, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtdmillar);
  }

  // Pulsacion  + Unidad de millar
  if (((x>83 && x<123) && (y>89 && y< 126))  || ((seleccion==2) && (vPUL1==true))) {
          vPUL1=false;
```

```
          x=0;y=0;
          vtumillar++;
          if (vtumillar>9) {vtumillar=0;}
          tft.fillRect(86,131,38,48,ILI9341_BLACK);
          tft.setCursor(95, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtumillar);
  }
  // Pulsacion  + centenas
  if (((x>141 && x<182) && (y>87 && y< 123))  || ((seleccion==3) && (vPUL1==true))) {
          vPUL1=false;
          x=0;y=0;
          vtcentena++;
          if (vtcentena>9) {vtcentena=0;}
          tft.fillRect(146,131,38,48,ILI9341_BLACK);
          tft.setCursor(155, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtcentena);
  }
  // Pulsacion  + decena
  if (((x>200 && x<240) && (y>87 && y< 123))  || ((seleccion==4) && (vPUL1==true))) {
          vPUL1=false;
          x=0;y=0;
          vtdecena++;
          if (vtdecena>9) {vtdecena=0;}
          tft.fillRect(206,131,38,48,ILI9341_BLACK);
          tft.setCursor(215, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtdecena);
  }

  // Pulsacion  + unidad
  if (((x>259 && x<299) && (y>87 && y< 123))  || ((seleccion==5) && (vPUL1==true))) {
          vPUL1=false;
          x=0;y=0;
          vtunidad++;
          if (vtunidad>9) {vtunidad=0;}
          tft.fillRect(266,131,38,48,ILI9341_BLACK);
          tft.setCursor(275, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtunidad);
  }


  // Pulsacion  - decena de millar
  if (((x>28 && x<63) && (y>189 && y< 226))  || ((seleccion==1) && (vPUL3==true))) {
          vPUL3=false;
          x=0;y=0;
          vtdmillar--;
          if (vtdmillar>200) {vtdmillar=9;}
          tft.fillRect(26,131,38,48,ILI9341_BLACK);
          tft.setCursor(35, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtdmillar);
  }

  // Pulsacion  - unidad de millar
  if (((x>80 && x<121) && (y>192 && y< 226))  || ((seleccion==2) && (vPUL3==true)))  {
          vPUL3=false;
          x=0;y=0;
          vtumillar--;
          if (vtumillar>200) {vtumillar=9;}
          tft.fillRect(86,131,38,48,ILI9341_BLACK);
          tft.setCursor(95, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtumillar);
  }
  // Pulsacion  - centena
  if (((x>138 && x<180) && (y>190 && y< 226))  || ((seleccion==3) && (vPUL3==true)))  {
          vPUL3=false;
          x=0;y=0;
          vtcentena--;
          if (vtcentena>200) {vtcentena=9;}
          tft.fillRect(146,131,38,48,ILI9341_BLACK);
          tft.setCursor(155, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
tft.print(vtcentena);
  }
  // Pulsacion  - decena
  if (((x>199 && x<239) && (y>192 && y< 226))  || ((seleccion==4) && (vPUL3==true))) {
          vPUL3=false;
```

```
                x=0;y=0;
                vtdecena--;
                if (vtdecena>200) {vtdecena=9;}
                tft.fillRect(206,131,38,48,ILI9341_BLACK);
                tft.setCursor(215, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtdecena);
          }
          // Pulsacion  - unidad
          if (((x>261 && x<301) && (y>192 && y< 226))  || ((seleccion==5) && (vPUL3==true))) {
                vPUL3=false;
                x=0;y=0;
                vtunidad--;
                if (vtunidad>200) {vtunidad=9;}
                tft.fillRect(266,131,38,48,ILI9341_BLACK);
                tft.setCursor(275, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
        tft.print(vtunidad);
          }
        salto:;

        } while (salir==false);


        vdistOrganizacion= (vtdmillar*10000)+(vtumillar*1000)+(vtcentena*100)+(vtdecena*10)+vtunidad;

        // Almacenamos en eprom
        EEPROM.write(3, (highByte(vdistOrganizacion)));          // Dist. organizacion (Byte HIGH)
        EEPROM.write(4, (lowByte(vdistOrganizacion)));           // Dist. Organizacion (Byte LOW)

        vdistrealTotalKM=0;

        // Fuerza a repintar Horas y min en pantalla
        minold=255;
        horaold=255;

        calibracio3();
        //


}



// Calibracion Paso3.


void calibracio3(){


        String vdistantciatmp;
        int vlongitud, vx, vxold;
        vxold=4;

        // Borra Pantalla Menu Principal
        FborraPantalla();
//      // Dibuja recuadros
//      tft.drawRoundRect(2,1,318,33,7,ILI9341_BLUE);
//      tft.drawRoundRect(2,35,318,205,5,ILI9341_BLUE);

        tft.setCursor(140, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(P("Calibracio 3/4"));

        // Pinta Datos calibracion rueda
        tft.setCursor(100, 45); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(P("DISTANCIA"));

        do
        {
        // Calculos de ciclo
        //vciclo1=millis();

        // Lectura de Pulsos
        lecturaIN();
        // Actualiza el relog en pantalla
```

```
        t.update();
        if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

        // Presenta la Distancia recorrida, solo si cambia

        if (vdistrealtotalKmold!=vdistrealTotalKM){
          // Borra dato anterior
          tft.setCursor(vxold, 125); tft.setTextColor(ILI9341_BLACK); tft.setTextSize(6);
        tft.print(vdistrealtotalKmold,3);

          // rutinilla para centrar el resultado en palabra
          vdistantciatmp = String(vdistrealTotalKM);
          vlongitud = vdistantciatmp.length();
          // Calculo eje X para centrar datos
          vx = (160 - (((vlongitud+1)* 6 * 6)/2));
          tft.setCursor(vx, 125); tft.setTextColor(ILI9341_GREEN); tft.print(vdistrealTotalKM,3);
          vdistrealtotalKmold=vdistrealTotalKM;
          vxold=vx;
        }

        // Calculo del ciclo
//      vciclo2 = millis();
//      vciclo = vciclo2 - vciclo1;
//      Serial.println(vciclo);

        } while (vPUL2==false);
        vPUL2=false;

        // Fuerza a repintar Horas y min en pantalla
        minold=255;
        horaold=255;
        // Paso 4
        calibracio4();

}
// Calibracion Paso 4. Calculo Correccion


void calibracio4(){
        // Borra Pantalla Menu Principal
        FborraPantalla();
//      // Dibuja recuadros
//      tft.drawRoundRect(2,1,318,33,7,ILI9341_BLUE);
//      tft.drawRoundRect(2,35,318,205,5,ILI9341_BLUE);
        do
          {
          // Lectura de Pulsos
          lecturaIN();
          // Actualiza el relog en pantalla
          t.update();
          if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

          tft.setCursor(140, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(P("Calibracio 4/4"));

          // Pinta tITULO
          tft.setCursor(100, 45); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(P("CORRECCION"));

          // Pinta Datos Textuales
          tft.setCursor(5, 85); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);  tft.print(P(" *
        Dist. RoadBook"));
          tft.setCursor(5, 135); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);  tft.print(P(" *
        Dist. Real"));
          tft.setCursor(5, 185); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2); tft.print(P(" *
        F. Correccion"));

          // Pinta Distacias
          tft.setCursor(230, 85); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(vdistOrganizacion);
          tft.setCursor(230, 135); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
        tft.print(vdistrealTotalKM);
```

```
        // vcorrecion=vdistOrganizacion / vdistrealTotalKM;
        vcorrecion= vdistrealTotalKM / vDiametroRueda;
        vcorrecion = vdistOrganizacion / vcorrecion;
        tft.setCursor(230, 185); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
      tft.print(vcorrecion,3);
        } while (vPUL2==false);
        vPUL2=false;
        VESTADO=3;consola();
}

// Bluetooth RECEPCION Y ALMACENAMIENTO DE DATOS EN SDCARD


void conexio(){
      /*
      ESTRUCTURA DATOS SDCARD
      Los bytes 0-9 contiene datos genericos
      Cada tramo ocupa 11 bytes, empezando desde 10 ej:
      tramo 1  --> byte 10-20
      tramo 2  --> byte 22-32
      --
      tramo 50 --> byte 50-59
      Nombre Fichero: CARRERA
      BYTE          DATOS
      0         Numero de tramos
      --
      10                Numero de tramo
      11                Hora SALIDA
      12                Minuto SALIDA
      13                Segundo SALIDA
      14                Tiempo TRAMO. Horas
      15                Tiempo TRAMO. Minutos
      16                Distancia Tramo en KM (ENTERO)
      17                Distancia Tramo en METROS (DECIMAL)
      18                Distancia CRONO en kM (ENTERO)
      19                Distancia CRONO en METROS (DECIMAL)
      20                Velocidad Crono en KM
      21                Velocidad CRONO en m


      */
      // Recepcion de datos
      // Alamacenamiento de datos recibidos en el array de trabajo vCarrera[]



      // Borra Pantalla Menu Principal
      FborraPantalla();
      // Pone titulo
      tft.setCursor(220, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
      tft.print(P("CONEXIO"));
      boolean ON=0;
      boolean linea1mostrada=0;
      boolean linea2mostrada=1;
      boolean linea3mostrada=1;
      boolean linea1estadomostrado = 0;
      boolean linea2estado0mostrado = 0;
      boolean linea2estado1mostrado = 0;
      boolean linea3estado0mostrado=0;
      boolean salir = false;
      horaold=90; minold=90;

      do {
        Pulsacion();
        lecturaIN();
        // Actualiza el relog en pantalla
        t.update();
        if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

        // Salida
        if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; goto salto;}

        // lineas
        if (linea1mostrada == 0){
```

```
          tft.drawRoundRect(15,50,290,46,9,ILI9341_BLUE);
          tft.setCursor(30, 65); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
tft.print(P("BLUETOOTH"));
          tft.fillRoundRect(200,53,95,38,9,ILI9341_RED);
          tft.setCursor(230, 65); tft.setTextColor(ILI9341_BLACK); tft.setTextSize(2);
tft.print(P("OFF"));
          linea1mostrada=1;
  }
  if (linea2mostrada == 0) {
          tft.drawRoundRect(15,105,290,46,9,ILI9341_BLUE);
          tft.setCursor(30, 120); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
tft.print(P("ESTADO BLE"));
          tft.fillRoundRect(200,108,95,38,9,ILI9341_RED);
          tft.setCursor(230, 120); tft.setTextColor(ILI9341_BLACK); tft.setTextSize(2);
tft.print(P("OFF"));
          linea2mostrada=1;
  }
  if (linea3mostrada == 0) {
          tft.drawRoundRect(15,160,290,46,9,ILI9341_BLUE);
          tft.setCursor(30, 175); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
tft.print(P("Rx TRAMOS"));
          tft.drawRoundRect(200,163,95,38,9,ILI9341_RED);
          tft.setCursor(230, 175); tft.setTextColor(ILI9341_BLACK); tft.setTextSize(2);
tft.print(P("0"));
          linea3mostrada=1;
  }

  // detecta pulsacion boton central
  if (vPUL2==true) {ON=!ON; vPUL2=false;}

  // ON / OFF BLUETOOTH
  if (ON==true) {
          if (linea1estadomostrado==0) {
                  tft.fillRoundRect(200,53,95,38,9,ILI9341_GREEN);
                  tft.setCursor(235, 65); tft.setTextColor(ILI9341_RED); tft.setTextSize(2);
tft.print(P("ON"));
                  linea1estadomostrado=1;
                  delay (1000);
                  linea2mostrada=0;
                  if (estabaold) {delay (500); vble=1;}
          }
          // Chequea Bluetooth
          BTLEserial.pollACI();
          switch (vble)
                  {
                  case 0:                                                         //
desconectado
                          break;
                  case 1:
                          if (linea2estado0mostrado == 0) {
                                  tft.fillRoundRect(200,108,95,38,9,ILI9341_YELLOW);
                                  tft.setCursor(207, 120); tft.setTextColor(ILI9341_BLACK);
tft.setTextSize(2); tft.print(P("VISIBLE"));
                                  linea2estado0mostrado =1 ;
                                  if (estabaold) {delay (500); vble=2;}
                          }
                          break;
                  case 2:
                          if (linea2estado1mostrado == 0){
                                  tft.fillRoundRect(200,108,95,38,9,ILI9341_GREEN);
                                  tft.setCursor(205, 120); tft.setTextColor(ILI9341_RED);
tft.setTextSize(2); tft.print(P("CONNECT"));
                                  linea2estado1mostrado = 1;
                                  delay (1000);
                                  linea3mostrada=1;
                          }
                          break;
                  case 3:  // ERROR TAMAÑO TRAMA RECIBIDA
                          if (linea3estado0mostrado == 0){
                                  tft.fillRect(16,161,288,63,ILI9341_BLACK);
                                  tft.drawRoundRect(15,160,290,65,9,ILI9341_BLUE);
                                  tft.setCursor(50, 185); tft.setTextColor(ILI9341_RED);
tft.setTextSize(2); tft.print(P("ERROR DATOS TRAMA"));
                                  linea3estado0mostrado = 1;
```

```
                                        delay (1000);
                                        linea3mostrada=1;
                                }
                                break;
                        case 4:
                                        tft.setCursor(50, 190); tft.setTextColor(ILI9341_RED);
        tft.setTextSize(2); tft.print(P("Tramos Recibidos :  "));
                                        tft.print(vntramos);

                                        break;
                        case 5:
                                        tft.fillRect(20,170,290,70,ILI9341_BLACK);
                                        tft.drawRoundRect(15,160,290,65,9,ILI9341_BLUE);
                                        tft.setCursor(30, 165); tft.setTextColor(ILI9341_RED);
        tft.setTextSize(2); tft.print(P("Datos Recibidos:  "));
                                        tft.setCursor(30, 190); tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(1);

                                        for (byte a=1; a<(numRXbytes+1); a++) {


                                                tft.print(vCarreratmp[a]);tft.print(" ");

                                        }
                                        numRXbytes=0;
                                        // Guarda los datos recibidos ordenadamente en vCarrera y en
        SDCard
                                        byte b ,a;
                                        b = vCarreratmp[1];
                                        b = (10+((b-1)*12));
                                        for (a=0; a<12; a++){
                                                vCarrera[a+b]=vCarreratmp[a+1];
                                                //Serial.print(a+b); Serial.print(P(" t "));
        Serial.println(vCarrera[a+b]);
                                        }
                                        vble=10;
                                        break;
                        default:;
                                break;

                        }

          }

        salto:;
        } while   (salir==false);

        estabaold=1;
        // Alamacenamiento de datos en SD Card
        myFile = SD.open("Carrera.txt", FILE_WRITE);
        byte a;

        if (myFile){
          for (a=1; a<255; a++){
                  myFile.seek(a);
                  myFile.write(vCarrera[a]);
          }
        }
        myFile.close();
        Serial.println(P("Datos Carrera almacenados en SD Card"));



}
```

```
// Gestion de Eventos Bluetooth
void aciCallback(aci_evt_opcode_t event)
{
       switch(event)
       {
         case ACI_EVT_DEVICE_STARTED:
         Serial.println(F("VISIBLE"));
         vble=1;
         break;
         case ACI_EVT_CONNECTED:
         Serial.println(F("CONECTADO"));
         vble=2;
         break;
         case ACI_EVT_DISCONNECTED:
         Serial.println(F("DESCONECTADO"));
         vble=0;
         break;
         default:
         break;
       }
}
```

```
// Recepcion de datos Bluetooth
void rxCallback(uint8_t *buffer, uint8_t len)
{
       Serial.print(F("Received "));
       Serial.print(len);
       Serial.println(F(" bytes: "));


       char chartmp;   // caracter recibidos
       int cifratmp;     // CARACTER recibido en decimal
       int dato=0;
       int digito=0;

       int datotmp[100];
       int multiplicador = 1;

       for(int i=0; (i<(len)); i++){
         chartmp = ((char)buffer[i]);
         cifratmp = int(chartmp);
         cifratmp = (cifratmp - 48);
         //Serial.println(chartmp);


         if (chartmp != ';') {
                 digito++;
                 datotmp[digito] = cifratmp;
                 //Serial.print("datotmp");Serial.print(" ");Serial.println(datotmp[digito]);
         }

         if (chartmp == ';') {
                 for (byte a=0; a<digito; a++){
                         dato = dato + ((datotmp[digito-(a)] * multiplicador))  ;
                         multiplicador = (multiplicador * 10);
                         //Serial.println(dato);
                 }

                 vCarreratmp[bytetmp] = dato;
                 Serial.println(dato);
                 bytetmp++;
                 digito=0;
                 multiplicador = 1;
                 dato=0;
         }
       }
         numRXbytes= bytetmp-1;
```

```
        //Serial.print(P("numrx  ")); Serial.println(numRXbytes);

        if ((numRXbytes%12) != 0) {vble=3;}                          // Error Recepcion
                else {vble = 5; bytetmp=1;}
        // No byte TRAMO
       }


// Funcion. Devuelve el numero de tramos


byte cuentaTramos(){
       vntramos=0;
       for (byte a=10; a<255; a+12){
        if (vCarrera[a]!=0){vntramos++;}
                else {return vntramos;}
       }
}




// TRAMO 1. Seleccion de tramo
void trams(){
       // Borra Pantalla Menu Principal
       FborraPantalla();
       // dibuja Titulo
       tft.setCursor(125, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
       tft.print(P("Seleccion TRAMO"));

       // Actualiza el relog en pantalla
       t.update();
       if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

       // Presenta los tramos disponibles

       minold=0;segold=0;
       byte b=10;
       byte a=1;
       boolean salir=0;
       do
        {

        if (vCarrera[b] == 0) {salir = true; vntramos=(a-1);goto bye;}
        //Serial.print(b);Serial.print("    ");Serial.println(vCarrera[b]);
        tft.setCursor((15+((a/4)*40)),(70+(a*18))) ; tft.setTextColor(ILI9341_WHITE);
       tft.setTextSize(2); tft.print(P("TR "));tft.print(a);
        a=a+1;
        b=b+12;
        bye:;
        } while (salir==false);


       horaold=0; minold=0;
       byte pos=1, posold=0;
       boolean actualiza=0;
       do
       {
        lecturaIN();
        // Actualiza el relog en pantalla
        t.update();
        if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

        // salida
        if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {goto salida;}

        // Deteccion teclas up/down

        Serial.print(P("pos  "));Serial.println(pos);
        Serial.print(P("tramos  "));Serial.println(vntramos);
```

```
  if (vPUL1) {pos=pos+1;vPUL1=false;}
  if (vPUL3) {pos=pos-1;vPUL3=false;}

  if (pos>vntramos) {pos=1;}
  if (pos<1) {pos=vntramos;}

  if (pos != posold) {posold=pos; actualiza=1;}


  if (actualiza) {
          //borra selecciones actuales
          byte b=10;
          byte a=1;
          boolean salir2=false;
          do
                  {
                          if (vCarrera[b] == 0) {salir2 = true; Serial.println(P("sale")); goto
bye2;}
                          tft.setCursor((15+((a/4)*40)),(70+(a*18))) ;
tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("TR "));tft.print(a);
                          b=b+12;
                          a=a+1;
                          bye2:;

                  }while (salir2==false);

          // Cambia seleccion
          if (vntramos!=0) {tft.setCursor((15+((pos/4)*40)),(70+(pos*18))) ;
tft.setTextColor(ILI9341_RED); tft.setTextSize(2); tft.print(P("TR "));tft.print(pos);
                  actualiza=0;
          }
  }

} while (vPUL2==false);

// Seleccion de tramo
  if (vPUL2) {
          vPUL2=false;


          vnumTramo = vCarrera[pos+9];
          vHoraSalida = vCarrera[pos+10];
          vMinutoSalida = vCarrera[pos+11];
          vSegundoSalida = vCarrera[pos+12];
          vTiempoHorasTramo = vCarrera[pos+13];
          vTiempoMinutosTramo = vCarrera[pos+14];
          vDistKmTramo = vCarrera[pos+15];
          vDistMtrTramo  = vCarrera[pos+16];
          vDistKmCrono = vCarrera[pos+17];
          vDistMtrCrono = vCarrera[pos+18];
          vSpeedKmCrono = vCarrera[pos+19];
          vSpeedmCrono = vCarreratmp[pos+20];


          vtDistTramoMetros = ((vDistKmTramo*1000) + (vDistMtrTramo));
          vtdistCronoMetros = ((vDistKmCrono*1000) + (vDistMtrCrono));
          vtspeedmediaOrganizacion = ((vSpeedKmCrono) + (vSpeedmCrono/1000));

          vtTiempoTramoSegs = ((vTiempoHorasTramo*3600) + (vTiempoMinutosTramo*60));
          // vtTiempoCronoSegs =



  }
trams2();
salida:;
vPUL1=0;vPUL2=0;vPUL3=0;
pantallaInicio();
}
```

```
// TRAMO 2. CONFIRMACION
void trams2(){
       // Borra Pantalla Menu Principal
       FborraPantalla();
       // Dibuja Titulo
       tft.setCursor(112, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
       tft.print(P("CONFIRMACIO TRAM"));

       // PRESENTA MENSAJE
       tft.drawRoundRect(68,72,178,135,5,ILI9341_BLUE);
       tft.setCursor(105, 86); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("INICIO"));
       tft.setCursor(113, 116); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("TRAMO"));
       tft.setCursor(148, 150); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(vnumTramo);
       tft.setCursor(105, 190); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(1);
       tft.print(P("Pulsa Boton CENTRAL"));

       segold=0;minold=0;horaold=0;
       boolean salir=0;
       do
        {
        // Actualiza el relog en pantalla
        t.update();
        if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}
        // Detecta Pulsaciones
        lecturaIN();
        // salida
        if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; break;}


        } while (vPUL2==false);
        vPUL2=false;vPUL1=false;vPUL3=false;
        if (salir==1) {trams();}

        trams3();
}
// TRAMS 3. Cuenta atras
void trams3(){
       // Borra Pantalla Menu Principal
       FborraPantalla();
       // Dibuja Titulo
       tft.setCursor(216, 10); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
       tft.print(P("TRAMO"));
       tft.setCursor(292, 7); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(vnumTramo);
       // DIBUJA RECTANGULOS
       tft.drawRoundRect(80,48,177,62,5,ILI9341_BLUE);
       tft.drawRoundRect(80,48,177,20,5,ILI9341_BLUE);

       tft.drawRoundRect(24,133,274,78,5,ILI9341_BLUE);
       tft.drawRoundRect(24,133,274,20,5,ILI9341_BLUE);

       // DIBUJA TEXTO FIJO
       tft.setCursor(136, 55); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(1); tft.print(P("HORA
       SALIDA"));
       tft.setCursor(122, 140); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(1);
       tft.print(P("CUENTA ATRAS"));

       // DIBUJA HORA SALIDA
       tft.setTextSize(2);
       if (vHoraSalida<10)              {tft.setCursor(96, 84);tft.print(0);tft.setCursor(108, 84);
       tft.print(vHoraSalida);}
        else                            {tft.setCursor(96, 84);tft.print(vHoraSalida);}

       if (vMinutoSalida<10)     {tft.setCursor(156,84);tft.print(0);tft.setCursor(168, 84);
       tft.print(vMinutoSalida);}
        else                            {tft.setCursor(156,84);tft.print(vMinutoSalida);}

       if (vSegundoSalida<10)  {tft.setCursor(212, 84);tft.print(0);tft.setCursor(224, 84);
       tft.print(vSegundoSalida);}
        else                            {tft.setCursor(212, 84); tft.print(vSegundoSalida);}
```

```
// Pone ":"
tft.setCursor(136, 84); tft.print(P(":"));
tft.setCursor(192, 84); tft.print(P(":"));

tft.setTextSize(3);
tft.setCursor(108, 168); tft.print(P(":"));
tft.setCursor(200, 168); tft.print(P(":"));

boolean salir=false;
minold=0;horaold=0;
do {
 // Actualiza el relog en pantalla
 t.update();
 if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}
 // Detecta Pulsaciones
 lecturaIN();
 // salida
 if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; break;}

 // Calcula la cuenta atras
 byte vcahoraold, vcaminold, vcasegold;




 FCuentaatras(vHoraSalida,vMinutoSalida,vSegundoSalida);
 horacalc = (vcuentaatras / 3600);
 mincalc = (vcuentaatras - (horacalc * 3600)) / 60;
 segcalc = (vcuentaatras - (mincalc*60));



  Serial.print(vHoraSalida); Serial.print(P("  ")); Serial.print(vMinutoSalida);
Serial.print(P("  "));Serial.println(vSegundoSalida);
  Serial.println(vcuentaatras);
  Serial.print(horacalc);  Serial.print(P("    "));Serial.print(mincalc);Serial.print(P("
"));Serial.println(segcalc);

 // Dibuja cuenta atras
 tft.setTextSize(4);

 tft.setTextColor(ILI9341_GREEN);
 if (horacalc != vcahoraold) {
        tft.fillRect(44,168,60,35,ILI9341_BLACK);
        if (horacalc<10)        {tft.setCursor(44, 168);tft.print(0);tft.setCursor(56, 168);
tft.print(horacalc);}
        else                                    {tft.setCursor(44, 168);tft.print(horacalc);}
 }

 if (mincalc != vcaminold) {
        tft.fillRect(132,168,60,35,ILI9341_BLACK);
        if (mincalc<10)         {tft.setCursor(132,168);tft.print(0);tft.setCursor(144, 168);
tft.print(mincalc);}
        else                            {tft.setCursor(132,168);tft.print(mincalc);}
 }

 if (segcalc != vcasegold) {
        tft.fillRect(228,168,60,35,ILI9341_BLACK);
        if (segcalc<10)         {tft.setCursor(228, 168);tft.print(0);tft.setCursor(240,
168);  tft.print(segcalc);}
        else                            {tft.setCursor(228, 168); tft.print(segcalc);}
 }

 vcahoraold = horacalc; vcaminold = mincalc; vcasegold = segcalc;
 if ((horacalc==0) && (mincalc==0) && (segcalc==0)) {
        // Pone a cero contadores de distancia y tiempo
        vdistrealTotalKM=0;
        vtKmparcial=0;
        vnumpulsos=0;
        vnumpulsosparcial=0;
        vttiemporealinicial=Ftimetoseg();
```

```
                salir = 1;
                break;
          }


       } while (salir==false);
       vPUL2=false;vPUL1=false;vPUL3=false;
       if ((vcahora==0) && (vcamin==0) && (vcaseg==0)) {trams4();}




}
// TRAMS 4. Datos iniciales


void trams4(){
       // Borra Pantalla Menu Principal
       FborraPantalla();
       // Dibuja Titulo
       tft.setCursor(216, 10); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
       tft.print(P("TRAMO"));
       tft.setCursor(292, 7); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(vnumTramo);

       // DIBUJA RECTANGULOS
       tft.drawRoundRect(9,44,304,39,5,ILI9341_BLUE);
       tft.drawRoundRect(9,92,304,39,5,ILI9341_BLUE);
       tft.drawRoundRect(9,140,304,39,5,ILI9341_BLUE);
       tft.drawRoundRect(9,188,304,39,5,ILI9341_BLUE);

       // DIBUJA TEXTO FIJO
       tft.setCursor(16, 56); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("Km
       Total"));
       tft.setCursor(16, 103); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("Km
       Parcial"));
       tft.setCursor(16, 152); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("Vel.
       Media"));
       tft.setCursor(16, 200); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2);
       tft.print(P("Diferencia"));

       // DIBUJA VALORES A CERO
       tft.setCursor(170, 52); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("000.000"));
       tft.setCursor(170, 100); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("000.000"));
       tft.setCursor(170, 148); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("000.000"));
       tft.setCursor(170, 196); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print(P("000.000"));

       horaold=0;minold=0;
       boolean salir=false;
       do{
        // Actualiza el relog en pantalla
        t.update();
        if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}
        // Detecta Pulsaciones
        lecturaIN();
        // salida
        if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; break;}
        if (vPUL3) {vPUL3=false; vnumpulsosparcial=0;}

        // PRESENTACION KM REALES
        tft.fillRect(170, 52,140,23,ILI9341_BLACK);
        tft.setCursor(206, 52); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print((vdistrealTotalKM/1000),3);
        tft.fillRect(170, 100,140,23,ILI9341_BLACK);
        tft.setCursor(206, 100); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
       tft.print((vtKmparcial/1000),3);
        tft.fillRect(170, 148,140,23,ILI9341_BLACK);
```

```
        tft.setCursor(206, 148); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
      tft.print((vtspeedmediaOrganizacion));

        //calculo velocidad media real
        vttiemporealfinal=Ftimetoseg();
        vtspeedmediaReal = vdistrealTotalKM / (vttiemporealinicial-vttiemporealfinal);

        tft.fillRect(170, 196,140,23,ILI9341_BLACK);
        tft.setCursor(206, 196); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
      tft.print((vtspeedmediaOrganizacion-vtspeedmediaReal)/1000);




        } while (vPUL2==false);
        vPUL2=false;vPUL1=false;vPUL3=false;
        if (salir == 1) {trams();}
        //trams5();









      };
void trams5(){


      // Borra Pantalla Menu Principal
      FborraPantalla();
      // Dibuja Titulo
      tft.setCursor(216, 8); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
      tft.print(P("TRAMO"));
      tft.setCursor(292, 8); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(4);
      tft.print(vnumTramo);

      // DIBUJA RECTANGULOS
      tft.drawRoundRect(8,52,304,39,5,ILI9341_BLUE);
      tft.drawRoundRect(8,117,304,39,5,ILI9341_BLUE);
      tft.drawRoundRect(8,183,304,39,5,ILI9341_BLUE);

      // DIBUJA TEXTO FIJO
      tft.setCursor(14, 65); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("Km
      Total"));
      tft.setCursor(14, 129); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("Km
      Parcial"));
      tft.setCursor(14, 195); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(2); tft.print(P("Vel.
      Media"));


      // DIBUJA VALORES A CERO
      tft.setCursor(160, 52); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
      tft.print(P("000.000"));
      tft.setCursor(160, 100); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
      tft.print(P("000.000"));
      tft.setCursor(160, 148); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(3);
      tft.print(P("000.000"));


      do{

      } while (vPUL2==false);
      vPUL2=false;vPUL1=false;vPUL3=false;
      trams5();

}
```

```
// Calcul diferencia horaria "compte enrrera"
void FCuentaatras(byte hora, byte minuto, byte segundo){
      // el resultado queda almacenado en las variables publicas vcaseg, vcamin, vcahora;

      byte segactual, minactual, horaactual;
      long int segundosactuales, segundossalida;

      // Captura hora actual
      minactual = rtc.getMinutes();
      horaactual = rtc.getHours();
      segactual= rtc.getSeconds();

      // Calcul diferencia de segons
      segundosactuales= ((horaactual*3600) + (minactual*60) + segactual);
      segundossalida = ((hora*3600) + (minuto * 60) + segundo);
      vcuentaatras = segundossalida - segundosactuales;


}

// Calcula la hora actual en segundos
int Ftimetoseg(){
      signed int segcalc, horacalc, mincalc;
      byte segactual, minactual, horaactual;
      int fsegundos;
      // Captura hora actual
      minactual = rtc.getMinutes();
      horaactual = rtc.getHours();
      segactual= rtc.getSeconds();

      // Convierte a segundos
      fsegundos=(horaactual*3600);
      fsegundos=fsegundos+(minactual*60);
      fsegundos=fsegundos+(segactual);
      return fsegundos;
}

// Funcion para borrar pantalla "bonito"
void FborraPantalla(){
      tft.fillRoundRect(3,2,316,31,5,ILI9341_BLACK);
      tft.fillRoundRect(3,36,316,203,5,ILI9341_BLACK);


}


// Puesta en Hora Relog


void Prelog(){
      // Rutina setup relog
      // Borra Pantalla Menu Principal
      FborraPantalla();
      // Dibuja recuadros
      tft.drawRoundRect(2,1,318,33,7,ILI9341_BLUE);
      tft.drawRoundRect(2,35,318,205,5,ILI9341_BLUE);

      tft.setCursor(170, 11); tft.setTextColor(ILI9341_GREEN); tft.setTextSize(2);
      tft.print(P("AJUSTE HORA"));

      // Actualiza el relog en pantalla
      t.update();
      if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}

      // Prepara variables
      byte hora, minuto, segundos;
      hora = horatmp;
      minuto = mintmp;
      segundos = segtmp;

      // presenta datos iniciales
      if (hora < 10) {tft.setCursor(58, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
      tft.print(hora);}
```

```
  else {tft.setCursor(43, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
tft.print(hora);}

tft.setCursor(100, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
tft.print(P(":"));

if (minuto < 10) {tft.setCursor(145, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(minuto);}
  else {tft.setCursor(130, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
tft.print(minuto);}

tft.setCursor(190, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
tft.print(P(":"));

if (segundos < 10) {tft.setCursor(235, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(segundos);}
  else {tft.setCursor(220, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
tft.print(segundos);}

// Dibuja controles + -
Ftriangulo(0,52,105,6,ILI9341_RED);
Ftriangulo(0,140,105,6,ILI9341_RED);
Ftriangulo(0,230,105,6,ILI9341_RED);

Ftriangulo(2,52,170,6,ILI9341_RED);
Ftriangulo(2,140,170,6,ILI9341_RED);
Ftriangulo(2,230,170,6,ILI9341_RED);

// Dibuja rectangulos
tft.drawRect(41,113,64,50,ILI9341_RED);
tft.drawRect(124,113,64,50,ILI9341_RED);
tft.drawRect(215,113,64,50,ILI9341_RED);



byte seleccion=1;
boolean salir = false;
do        {
  t.update();
  // Actualiza el relog de la esquina
  if (t2on==false) {t2 = t.after(1000, relog); t2on=true;}
  Pulsacion();
  lecturaIN();

  // salida
  if ((digitalRead(PUL1)) && (digitalRead(PUL3))) {salir=1; goto salto;}

  if (vPUL2==true) {
          seleccion++;
          vPUL2=false;
          if (seleccion > 3) {seleccion=1;}
  }
  if (seleccion==1) {
          tft.drawRect(41,113,64,50,ILI9341_YELLOW);
          tft.drawRect(124,113,64,50,ILI9341_RED);
          tft.drawRect(215,113,64,50,ILI9341_RED);
  }
  if (seleccion==2) {
          tft.drawRect(41,113,64,50,ILI9341_RED);
          tft.drawRect(124,113,64,50,ILI9341_YELLOW);
          tft.drawRect(215,113,64,50,ILI9341_RED);
  }
  if (seleccion==3) {
          tft.drawRect(41,113,64,50,ILI9341_RED);
          tft.drawRect(124,113,64,50,ILI9341_RED);
          tft.drawRect(215,113,64,50,ILI9341_YELLOW);
  }


  // Marcado de zonas elegidas para cambiar
  // Zona hora +
  if (((x1 > 50) && (x1 < 91) && (y1 > 80) && (y1 < 105))  || ((seleccion==1) &&
(vPUL1==true))) {
          vPUL1=false;
```

```
          x=0;y=0; x1=0; y1=0;
          hora ++;
          if (hora > 23) {hora=0;}
          tft.fillRect(43,120,60,40,ILI9341_BLACK);
          if (hora < 10) {tft.setCursor(58, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(hora);}
                  else {tft.setCursor(43, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(hora);}
  }

  // Zona Hora -
  if (((x1 > 48) && (x1 < 92) && (y1 > 171) && (y1 < 213)) || ((seleccion==1) &&
(vPUL3==true))) {
          vPUL3=false;
          x=0;y=0; x1=0; y1=0;
          hora --;
          if (hora > 250) {hora=23;}
          tft.fillRect(43,120,60,40,ILI9341_BLACK);
          if (hora < 10) {tft.setCursor(58, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(hora);}
                  else {tft.setCursor(43, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(hora);}
  }

  // Zona Minuto +
  if (((x1 > 133) && (x1 < 183) && (y1 > 77) && (y1 < 114))  || ((seleccion==2) &&
(vPUL1==true))) {
          vPUL1=false;
          x=0;y=0; x1=0; y1=0;
          minuto ++;
          if (minuto > 59) {minuto=0;}
          tft.fillRect(130,120,60,40,ILI9341_BLACK);
          if (minuto < 10) {tft.setCursor(145, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(minuto);}
                  else {tft.setCursor(130, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(minuto);}
  }

  // Zona Minuto -
  if (((x1 > 123) && (x1 < 183) && (y1 > 172) && (y1 < 212))  || ((seleccion==2) &&
(vPUL3==true))) {
          vPUL3=false;
          x=0;y=0; x1=0; y1=0;
          minuto --;
          if (minuto > 250) {minuto=59;}
          tft.fillRect(130,120,60,40,ILI9341_BLACK);
          if (minuto < 10) {tft.setCursor(145, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(minuto);}
          else {tft.setCursor(130, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
tft.print(minuto);}
  }

  // Zona Segundos +
  if (((x1 > 225) && (x1 < 276) && (y1 > 73) && (y1 < 117))  || ((seleccion==3) &&
(vPUL1==true))) {
          vPUL1=false;
          x=0;y=0; x1=0; y1=0;
          segundos ++;
          if (segundos > 59) {segundos=0;}
          tft.fillRect(220,120,60,40,ILI9341_BLACK);
          if (segundos < 10) {tft.setCursor(235, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(segundos);}
                  else {tft.setCursor(220, 120); tft.setTextColor(ILI9341_WHITE);
tft.setTextSize(5); tft.print(segundos);}
  }
  // Zona Segundos -
  if (((x1 > 230) && (x1 < 281) && (y1 > 170) && (y1 < 212))  || ((seleccion==3) &&
(vPUL3==true))) {
          vPUL3=false;
          x=0;y=0; x1=0; y1=0;
          segundos --;
          if (segundos > 250) {segundos=59;}
          tft.fillRect(220,120,60,40,ILI9341_BLACK);
```

```
                if (segundos < 10) {tft.setCursor(235, 120); tft.setTextColor(ILI9341_WHITE);
        tft.setTextSize(5); tft.print(segundos);}
                else {tft.setCursor(220, 120); tft.setTextColor(ILI9341_WHITE); tft.setTextSize(5);
        tft.print(segundos);}
          }


        salto:;
        } while (salir==false);

        // Modifica Relog segun valores introducidos
        rtc.stopRTC(); //stop the RTC
        // Ajusta hora, minutos, seg
        rtc.setTime(hora, minuto, segundos);
        // Arranca relog
        rtc.startRTC();
        VESTADO=3;
}


//Lee una linea de un archivo de la SD, Contiene un string


String ReadFile(int Linea,char Ruta[]){
        int Lin=0;
          // linea a leer
        stResultado="";
        byte Bin;
        myFile = SD.open(Ruta);
        if (myFile) {
          while (myFile.available()) {                  // Bucle hasta el final del fichero
                Bin = myFile.read();
                stResultado = stResultado + (char(Bin));
                if (Bin==13){Lin++; myFile.read();
                        if (Lin!=Linea){stResultado="";}
                        if (Lin==Linea){break;}
                }
          }
        }
        myFile.close(); return stResultado;
}




//Retorna num de lineas en el fichero
int numLineas(char Ruta[]){
        // Funcion que Retorna el numero de lineas existentes en un fichero
        int lin=0;
        byte lectura;
        myFile = SD.open(Ruta);
        if (myFile) {
          while (myFile.available()) {                                          // Bucle hasta el
        final del fichero
                lectura=myFile.read();
                if (lectura == 13) {lin++; myFile.read();}      // la funcion myfile.read, lee el
        caracter line feed
          }
        }
        myFile.close();return lin;
}
```

```
// Graba un evento en SD con la fecha del sistema
boolean anotaEvento(char Ruta[]){
       myFile = SD.open(Ruta, FILE_WRITE);
       // formatea el string de fecha actual con ceros si es necesario
       stFecha="";
       if (daytmp<10){stFecha = (P("0")); stFecha = stFecha + String(daytmp);}
        else {stFecha = stFecha + String(daytmp);}
       stFecha = stFecha + (P("/"));
       if (mestmp<10){stFecha = stFecha + (P("0")); stFecha = stFecha + String(mestmp);}
       else {stFecha = stFecha + String(mestmp);}
       stFecha = stFecha + (P("/"));
       stFecha = stFecha + String(yeartmp);


       if (myFile){
        myFile.println(stFecha);
        myFile.close();
        return true;}
        else{
                myFile.close();
                return false;
        }
}




// Actualiza el Relog en pantalla CADA SEGUNDO


void relog(){

       tft.setTextColor(ILI9341_YELLOW); tft.setTextSize(2);
       // Pone la hora
       horatmp = rtc.getHours();
       if (horaold != horatmp){
        tft.fillRect(5,11,31,16,ILI9341_BLACK);
        if (horatmp<10) {tft.setCursor(8, 11);tft.print(0);tft.setCursor(20, 11);
       tft.print(horatmp);}
                else            {tft.setCursor(8, 11);tft.print(horatmp);}
        horaold = horatmp;
        tft.setCursor(31, 11);tft.print(P(":"));
       }
       // Pone el minuto
       mintmp = rtc.getMinutes();
       if (minold != mintmp){
        tft.fillRect(41,11,31,16,ILI9341_BLACK);
        if (mintmp<10)  {tft.setCursor(41,11);tft.print(0);tft.setCursor(53, 11); tft.print(mintmp);}
                else            {tft.setCursor(41,11);tft.print(mintmp);}
        minold = mintmp;
        tft.setCursor(63, 11);tft.print(P(":"));
       }
       // Pone los segundos
       segtmp = rtc.getSeconds();
       if (segold!=segtmp){
        tft.fillRect(73,11,31,16,ILI9341_BLACK);
        if (segtmp<10)  {tft.setCursor(73, 11);tft.print(0);tft.setCursor(85, 11);
       tft.print(segtmp);}
                else            {tft.setCursor(73, 11); tft.print(segtmp);}
        segold = segtmp;

       }

       t.stop(t2);
       t2on=false;
}
```

```
// Recupera mensajes de Error de FLASH
char * Fmensaje(int index){
      // Borra Buffer previo
      for (byte i=0; i<20; i++){bufferM[i] = *("");}
      // Carga en buffer el mensaje de error solicitado
      strcpy_P(bufferM, (char*)pgm_read_word(&(MERROR[index])));
      return bufferM;
}



// Funcion Aux para imprimir mensajes centrados en pantalla
int FposX (int q, int campo, int sizetext){
      // x --> Pixel inicio campo en pantalla
      // campo --> Tamaño campo en pantalla en pixels (5+1 en tamaño 1)
      // longtext --> longitud del texto en caracteres
      // sizetext --> Multiplicador de la fuente
      byte numchars = Fcuentachars(bufferM);                                          // cuenta
      los caracteres del mensaje
      int posx = (q + ((campo - ((numchars * 6) * sizetext)) /2));
      return posx;
}



// funcion para dibujar un triangulo
void Ftriangulo (byte up, int x, int y, int size,  uint16_t color) {
      /* byte up -->  0  Flecha arriba
                              1 Flecha derecha
                              2 Flecha abajo
                              3 Flecha izquierda
        int x ---> Coordenada X Base triangulo
        int y ---> Coordenada Y Base triangulo
        int size -> 1  Caracter 8x5
                              2  Caracter 16x10
                              3  Caracter 24x15
      */

if (up==0) {tft.fillTriangle(x,y,(x+(3*size)),(y-(6*size)),(x+6*size),y,color);}
if (up==2) {tft.fillTriangle(x,y,(x+(3*size)),(y+(6*size)),(x+6*size),y,color);}

}
```