Fiuba
Facultad de Ingeniería
Universidad de Buenos Aires

(86.07) Laboratorio de Microprocesadores

Proyecto:
Solar Tracker

Profesor:	**Ing. Guillermo Campiglio**
Cuatrimestre / Año:	**1er Cuatrimestre del 2016**
Turno de clases prácticas:	**Miércoles**
Jefe de Trabajos Prácticos:	**Ricardo Arias**
Docente guía:	

Autores			**Seguimiento del proyecto**								
Nombre	**Apellido**	**Padrón**									
Agustín	**Picard**	**95476**									
Joaquín	**Ulloa**	**93550**									
Mauro	**Giordano**	**94647**									

Observaciones:

Fecha de aprobación		

Firma J.T.P.

COLOQUIO	
Nota final	
Firma Profesor	

Índice

Introducción

El presente trabajo tiene como finalidad brindar autonomía energética a distintos productos donde, en su lugar de aplicación, los recursos energéticos sean escasos o costosos.

Dada la experiencia en el rubro, se observó que la principal falla en los productos del mercado está dada por la forma en que se cargan las baterías de los mismos. Aplicar una tensión constante hace que la batería no cumpla de manera eficiente su ciclo de carga y descarga, dañándola considerablemente y provocando que la tasa de fallo sea temprana.

Además se analizaron estudios, donde se llega a la conclusión que la implementación de paneles solares con rastreadores solares puede aumentar hasta un 25% la energía generada anualmente por el mismo. La mayor parte de esta diferencia se debe a las optimizaciones logradas tanto en invierno, como en otoño. Vale la pena remarcar esto último, ya que en éstas temporadas es donde el sol y la energía más escasean.

Es importante también mencionar el avance en la tecnología de los teléfonos inteligentes y desarrollo de aplicaciones para los mismos. El consumo masivo de ‘smartphones’ y la mentalidad de la sociedad de controlar todo con dicho sistema embebido, ha provocado la necesidad de considerar una interfaz de comunicación adecuada.

Objetivos del proyecto

Para solucionar estas problemáticas, se pretende diseñar e implementar un dispositivo capaz de maximizar la eficiencia en el aprovechamiento de la energía solar captada por un panel fotovoltaico. Dicha energía será almacenada en una batería, la cual es el único suministro energético del sistema de control.

El producto elegido para aplicar dicho dispositivo será un artefacto de iluminación, de modo que, en situaciones donde se carezca de luz natural en el ambiente y la batería esté suficientemente cargada, el dispositivo iluminará.

Sería interesante en algún futuro, (con posteriores conocimientos de materias como 86.24 Electrónica de Potencia y 86.10 Diseño de Circuitos Electrónicos y con ayuda de políticas energéticas adecuadas) poder adicionar un módulo capaz de transformar la energía de la batería en alterna e inyectarla a la red.

Descripción del Proyecto

Concretamente, se pretende medir la intensidad lumínica con sensores LDR, de modo que el panel fotovoltaico se oriente perpendicularmente a los rayos incidentes. Dicho movimiento será provocado por dos motores montados sobre un sistema de ejes de azimut y elevación, cargando la batería mediante un regulador de corriente controlado.

Tener un control de la energía generada y consumida en el artefacto permitirá tomar correctas decisiones en la administración de la misma, para luego, transmitir el estado del sistema por medio de un módulo Bluetooth.

Implementación

Hardware: Módulos, Parte mecánica y diseño y construcción del PCB.

El dispositivo consta de un panel solar capaz de entregar 40 Wh (80 cm x 40 cm). Un controlador de carga para celdas fotovoltaicas o MPPT (del inglés, Maximun Power Point Tracker) capaz de entregar hasta 120 W a 12 V. La lógica de control fue íntegramente programada en lenguaje ensamblador y ejecutada en un microcontrolador ATmega88, de la familia AVR.

Se implementó un dispositivo de censado diferencial de luz, con LDRs, el cual se conecta a los puertos de conversión analógica digital (ADC) del microcontrolador y también con divisores resistivos se midieron las tensiones de la batería y el panel.

Como actuadores se usaron dos motores de corriente continua, controlados por medio de un puente H, con PWM. Es de interés aclarar que los mismos necesitaron reducciones, en particular en el eje de los mismo se usaron engranajes helicoidales, con el objetivo de que para mantener el panel en posición no sea necesario consumir energía. Además el sistema es capaz de controlar una luz de iluminación pública, también con PWM y por medio de una etapa de potencia.

La interfaz de usuario es por comunicación Bluetooth, capaz de conectarse a cualquier dispositivo que cuente con esta tecnología. Por medio de la misma, se puede colectar la información medida por el dispositivo, prender y apagar la luz y orientar el panel.

En la figura a continuación, se muestra una representación esquemática del conexionado:

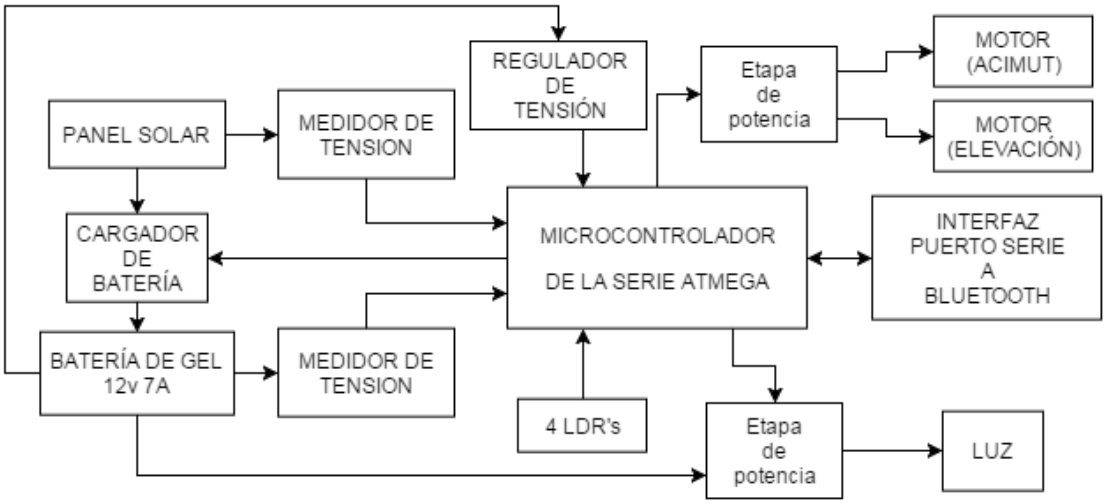

Figura 1: Diagrama en bloque del dispositivo.

La diagramación propuesta en el ante proyecto se respetó en su totalidad, lo que indica que hubo una buena etapa de diseño, investigación y planificación en el proyecto.

A continuación, en la **Figura 2** se puede observar el diseño esquemático del circuito de control:

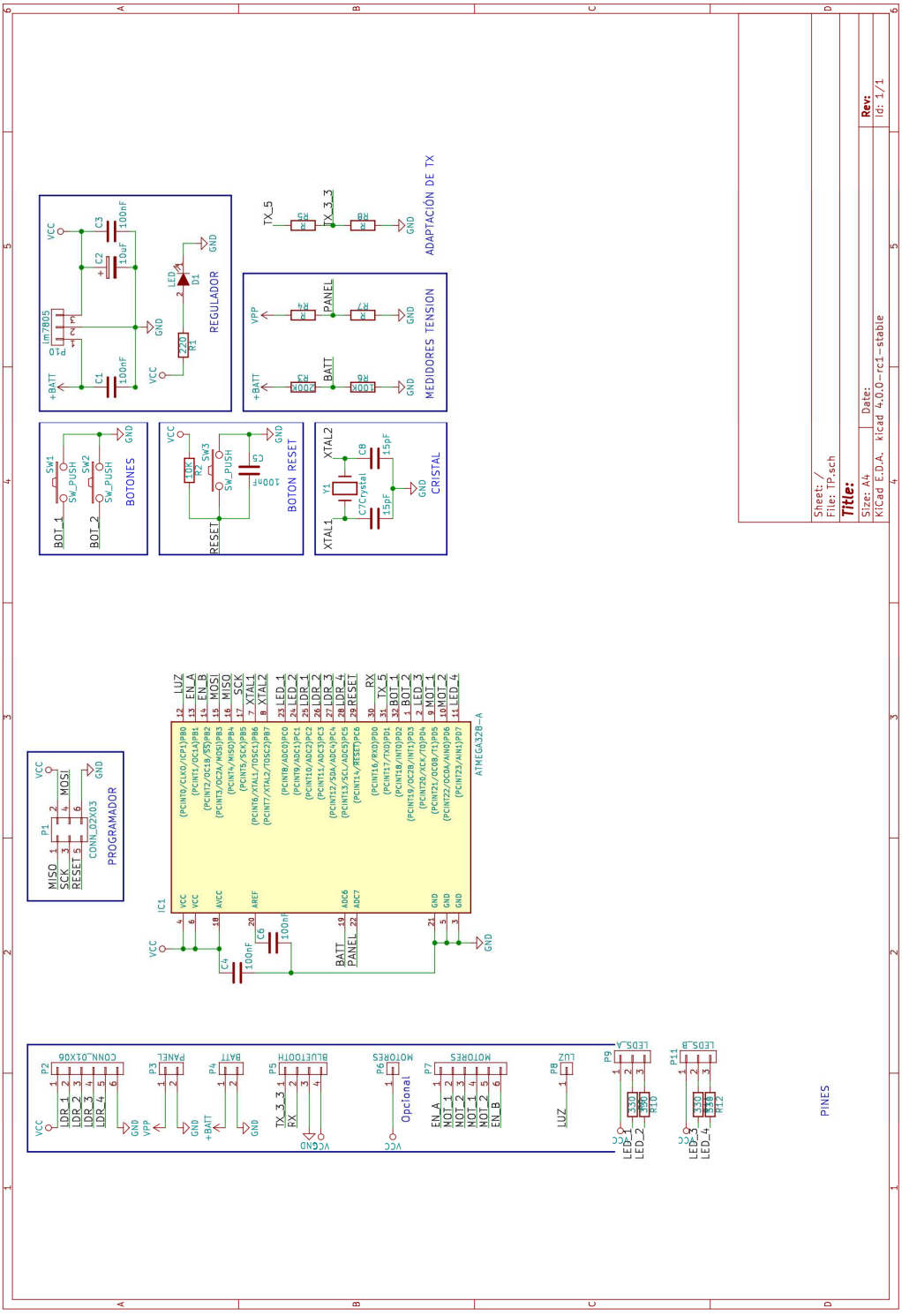

Figura 2: Diagrama esquemático del circuito de control.

El diagrama de conexionado del puente H se presenta en la **Figura 3**:

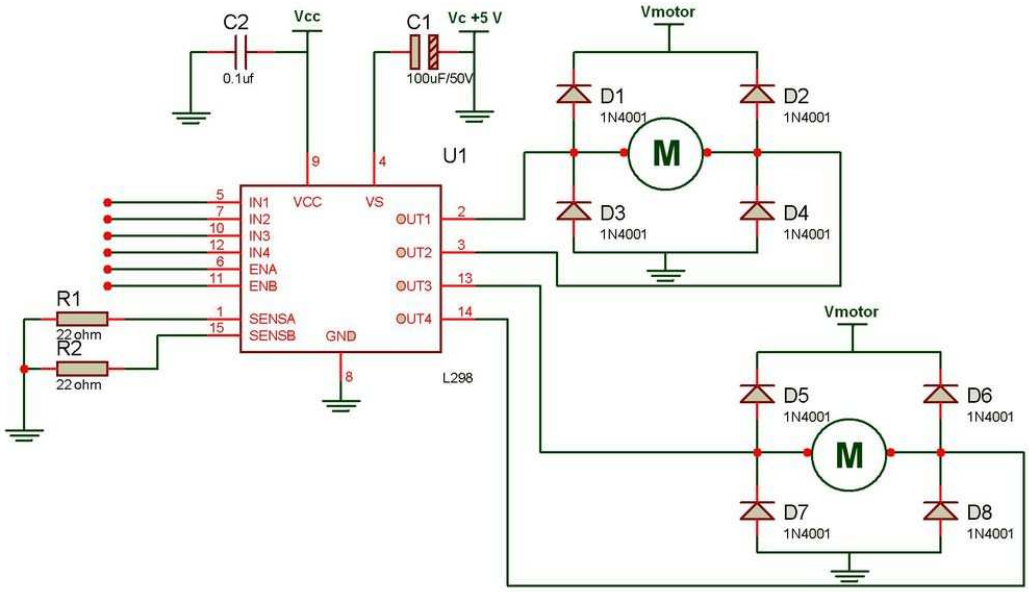

Figura 3: Diagrama esquemático del circuito de potencia para los motores.

Para el dispositivo de censado, se utilizó un divisor resistivo entre los LDR y una resistencia de 10k, midiendo la tensión sobre ésta última para entrar a los pines del ADC.

El módulo de comunicación Bluetooth elegido fue el HC-06, dado que cumplia con nuestros requerimientos de comunicación simple y rápida con cualquier celular ‘Smartphone’.

Por otro lado, para la etapa de potencia para la luz, se utilizó una etapa amplificadora Emisor-Común con un transistor Darlington NPN (TIP122) con una resistencia de 1k en la base. Se escogió dicho transistor ya que tiene una ganancia de aproximadamente 4000 para un consumo de 1.5A (de modo que el consumo de corriente del pin del microcontrolador es inferior 0.5mA).

En cuanto a los circuitos de la **Figura 1**, **Figura 2** y **Figura 3** se optó por construirlos por separado, de manera que cada uno constituya un módulo diferente. Se tomó como decisión de diseño la modularización, ya que se considera que de esta forma es más sencillo encontrar posibles problemas y errores de conexión. Además, permite que el dispositivo sea escalable sin la necesidad de grandes cambios, los cuales podrían ser costosos y llevar mucho tiempo.

Para el módulo del puente H, se compró el usado comúnmente para ‘Arduino’ dado que ya se había usado para otros proyectos anteriores y funcionaba a la perfección. Los motores DC de 24v fueron reciclados de impresoras viejas. El MPPT es un ASC (ver Anexo) y el panel solar es un Solartec KS40TA.

El PCB fue diseño propio y se mandó a fabricar. El mismo cuenta con un regulador LM7805 (con su circuito recomendado en la hoja de datos) para alimentar el microcontrolador ATmega88 SMD. Se eligió este modelo porque cuenta con dos puertos ADC más que su equivalente DIP. A su vez, se incluyó un botón de reset, un divisor resistivo para adaptar comunicación serie con el módulo Bluetooth (3.3 V). Otros dos divisores resistivos se agregaron para adaptar las tensiones del panel y la batería a la de la lógica.

En cuanto a la mecánica, se utilizó un disco rígido obsoleto como ruleman para la base (dada su robustez y su poca fuerza de rozamiento). Sobre el mismo, se montó la estructura de chapa en forma de ‘U’ que sostiene el panel solar desde sus laterales, con varillas roscadas y un par de rulemanes blindados.

Consecuentemente, se eligieron engranajes helicoidales para contrarrestar la fuerza del viento y que no gire el dispositivo de forma involuntaria. Conjuntamente, se agregaron dos etapas de engranajes con ruedas dentadas, dado que la reducción debía ser considerablemente grande.

Software: Lógica y diccionario de comunicación.

En la **Figura 4** se muestra la lógica principal del proyecto:

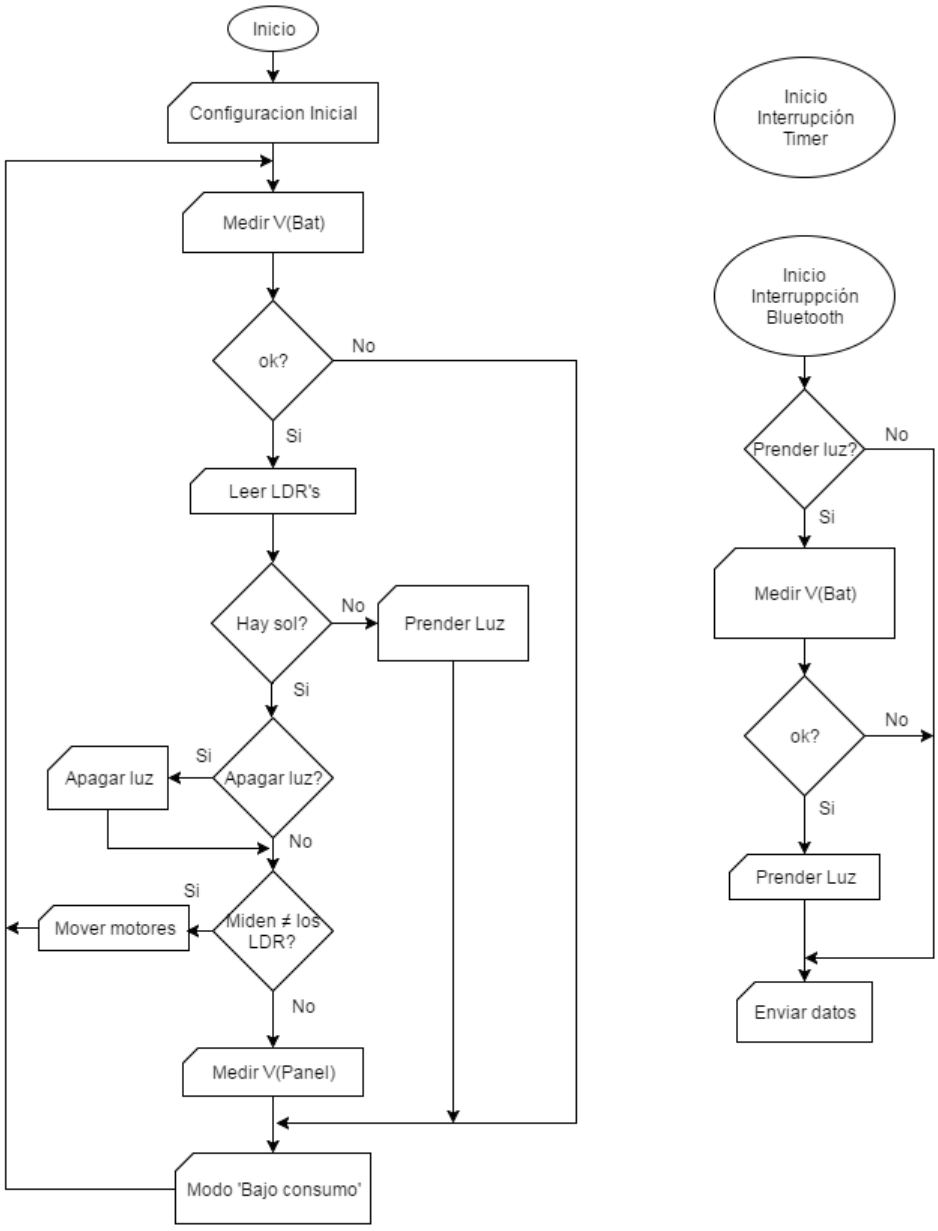

Figura 4: Diagrama de flujo del dispositivo.

A diferencia del diagrama en bloques, es diagrama de flujo si cambió. Se hicieron dos modificaciones a la propuesta inicial. La primera es que para comprobar si hay luz o no se mide la tensión entregada por el panel y no los LDRs. La segunda, es que por medio del Bluetooth se puede controlar la ubicación del panel y para que esto sea posible se bloquea la ubicación automática del mismo.

Para la comunicación Bluetooth, se utilizó un celular con plataforma ‘Android’ mediante la aplicación ‘Bluetooth Terminal’ que se encarga de vincular el Smartphone con el módulo HC-06. En cuanto a la comunicación, se predeterminaron comandos simples basados en la emisión de letras, de modo que sea rápido pedir la información necesaria o enviar alguna acción.

Se extrae un fragmento del código del archivo de configuraciones del proyecto ‘Solar_Tracker.inc’ para mostrar dichos comandos de interacción con el usuario:

```
;SERIE DE COMANDOS DE RECEPCION POR BT
.EQU  BT_COMMAND_PROJECT_NAME          =     'q'
.EQU  BT_COMMAND_V_BAT                 =     'w'
.EQU  BT_COMMAND_V_PANEL               =     'e'
.EQU  BT_COMMAND_LIGHT_TURN_ON         =     'r'
.EQU  BT_COMMAND_LIGHT_TURN_OFF        =     't'
.EQU  BT_COMMAND_MANUAL_LIGHT_OFF      =     'y'
.EQU  BT_COMMAND_ELEVATION_NORTH       =     'u'
.EQU  BT_COMMAND_ELEVATION_SOUTH       =     'i'
.EQU  BT_COMMAND_AZIMUT_EAST           =     'o'
.EQU  BT_COMMAND_AZIMUT_WEST           =     'p'
.EQU  BT_COMMAND_RESET                 =     'z'
.EQU  BT_COMMAND_MANUAL_MOTORS_OFF     =     'l'
```

Código fuente

Solar Tracker.asm

```
/*
 * Solar_Tracker.asm
 *
 *  Created: 08/06/2016 07:45:11 p.m.
 *   Authors: Agustín Picard, Joaquin Ulloa, Mauro Giordano
 */

.include "m88def.inc"      ;Incluye los nombres de los registros
del micro
.include "Solar_Tracker.inc"
.include "avr_macros.inc"  ;Incluye los macros
.listmac                   ;Permite que se expandan las macros
en el listado

.CSEG
.ORG 0x0000
RJMP SETUP

.ORG  INT0addr
RJMP  ISR_INT0

.ORG  URXCaddr         ; USART, Rx Complete
RJMP  ISR_RX_USART_COMPLETA

.ORG  UDREaddr         ; USART Data Register Empty
RJMP  ISR_REG_USART_VACIO

.ORG  OVF1addr
RJMP  ISR_TIMER_1_OV

;------------------------SETUP-------------------------------
.ORG  INT_VECTORS_SIZE
SETUP:
      LDI AUX,LOW(RAMEND)
      OUT SPL,AUX
      LDI   AUX,HIGH(RAMEND)
      OUT SPH,AUX

      RCALL BT_MANUAL_LIGHT_OFF  ;[FLAG=0xFF]: LA LUZ SE MANEJA
MANUALMENTE. [FLAG=0x00]: LA LUZ SE MANEJA AUTOMATICAMENTE.
      RCALL BT_MANUAL_MOTORS_OFF ;[FLAG=0xFF]: LOS MOTORES SE
MANEJAN MANUALMENTE. [FLAG=0x00]: LOS MOTORES SE MANEJAN
AUTOMATICAMENTE.
      RCALL FLAG_AT_NIGHT_OFF
      RCALL BATTERY_INIT
      RCALL SOLAR_PANEL_INIT
      RCALL ADC_INIT                        ;TIENE QUE ESTAR EN
"ADC.inc"

      RCALL LDRS_INIT
      RCALL SERIAL_PORT_INIT                ;TIENE QUE ESTAR EN
"SERIAL_PORT.inc"
```

```
      RCALL PWM_INIT
      RCALL MOTORS_INIT
      SEI

      LDIW  Z,(MSJ_PROJECT_NAME*2)
      RCALL TRANSMITIR_MENSAJE
      RCALL DELAY_50ms

RJMP MAIN

;------------------PROGRAMA_PRINCIPAL--------------------------
MAIN:
;MIDO LA BATERIA
      RCALL READ_V_BATTERY
      RCALL VBATTERY_TO_ASCII
      RCALL CHECK_IF_BATTERY_MINIMUM ;C=1: BAT LOW.C=0: BAT OK
      BRCS  SLEEP_MODE
      RCALL INDICATE_BATTERY_OK
;¿DIA O NOCHE?
      RCALL READ_V_SOLAR_PANEL    ;PARA VER SI ES DE DIA O NOCHE,
MIDO LA TENSION DEL PANEL SOLAR.
      RCALL VPANEL_TO_ASCII
      RCALL CHECK_IF_SOLAR_PANEL_MINIMUM ;C=1:PV LOW.C=0: PV OK
      BRCS  AT_NIGHT
      RCALL INDICATE_SOLAR_PANEL_OK
      RCALL LIGHT_TURN_OFF
      RCALL FLAG_AT_NIGHT_OFF
;SI ESTOY ACA YA TENGO BATERIA SUFICIENTE, ES DE DIA.
      RCALL LDRS_READ  ;LEE LOS LDR'S Y LOS MANDA A RAM.
      RCALL DELAY_100us
      RCALL LDRS_READ  ;LEE LOS LDR'S Y LOS MANDA A RAM.
      RCALL DELAY_100us
      RCALL LDRS_READ  ;LEE LOS LDR'S Y LOS MANDA A RAM.
      RCALL DELAY_100us
      RCALL LDRS_READ  ;LEE LOS LDR'S Y LOS MANDA A RAM.
      RCALL DELAY_100us
      RCALL LDRS_READ  ;LEE LOS LDR'S Y LOS MANDA A RAM.
      RCALL LDRS_MEAN

      RCALL ORIENTATE_SOLAR_PANEL

SLEEP_MODE:
           RCALL SLEEP_TIMER_INIT
           INPUT AUX,SMCR
           ANDI AUX,((~((1<<SM2)|(1<<SM1)|(1<<SM0)|(1<<SE))))
           ORI AUX,((0<<SM2)|(0<<SM1)|(0<<SM0)|(1<<SE))
      ;SETEO EL MODO IDLE.
           OUTPUT SMCR,AUX
           SLEEP
           NOP
           INPUT AUX,SMCR
           ANDI AUX,(~(1<<SE))
           ;CUANDO SALGO DE SLEEP, PONGO SE=0.
           ORI AUX,(0<<SE)
           OUTPUT SMCR,AUX
RJMP MAIN
;--------------------------------------------------------------
```

```
;----------------------MAIN_FUNCTIONS---------------------------
ISR_INT0:
RETI

ISR_TIMER_1_OV:
RETI

AT_NIGHT:
           RCALL LIGHT_TURN_ON
           RCALL INDICATE_SOLAR_PANEL_LOW
;          RCALL RETURN_TO_ORIGIN
      RJMP SLEEP_MODE

FLAG_AT_NIGHT_OFF:
      CLR AUX
      OUTPUT FLAG_AT_NIGHT,AUX
RET

SLEEP_TIMER_INIT:
                LDI   AUX,0xEE        ;Pongo como valor inicial
del timer 34286 para que cuando haga overflow haya contado
(8E6)/256
                OUTPUT TCNT1L,AUX
                LDI AUX,0x85
                OUTPUT TCNT1H,AUX
                LDI AUX,(1<<CS12)|(0<<CS11)|(0<<CS10)
      ;Seteo el prescaler a 256
                OUTPUT TCCR1B,AUX
                LDI AUX,0
                OUTPUT TCCR1A,AUX
                LDI AUX,(1<<TOIE1)
                OUTPUT TIMSK1,AUX
RET

.include "ADC.inc"
.include "PWM.inc"
.include "SERIAL_PORT.inc"
.include "DELAY.inc"
.include "LIGHT.inc"
.include "BATTERY.inc"
.include "SOLAR_PANEL.inc"
.include "LDRS.inc"
.include "MOTORS.inc"
.include "MESSAGES.inc"
```

Solar Tracker.inc

```
.DSEG
.ORG SRAM_START

;--------------------HEADER DE LA LUZ--------------------------
.EQU  LIGHT_PIN       = PD3
;--------------------------------------------------------------
;--------------------HEADER DEL ADC----------------------------
.DEF  ADC_DATA_L      = R19
.DEF  ADC_DATA_H      = R20

.EQU  ADC_BATTERY     = 6 ;LA BATERIA ESTA EN EL PIN 19 [ADC6]
.EQU  ADC_SOLAR_PANEL = 7 ;EL PANEL ESTA EN EL PIN 22 [ADC7]
;--------------------------------------------------------------
;--------------------HEADER DEL PUERTO SERIE-------------------
.EQU  BAUD_RATE       = 103 ; 12  76.8 kbps e=0.2%@8MHz y U2X=1
                            ; 25  38.4 kbps e=0.2%@8MHz y U2X=1
                            ; 51  19.2 kbps e=0.2%@8MHz y U2X=1
                            ; 103 9600 bps  e=0.2%@8MHz y U2X=1

.EQU  BUF_SIZE        = 64  ; bytes del buffer de transmisión

;SERIE DE COMANDOS DE RECEPCION POR BT
.EQU  BT_COMMAND_PROJECT_NAME         =    'q'
.EQU  BT_COMMAND_V_BAT                =    'w'
.EQU  BT_COMMAND_V_PANEL              =    'e'
.EQU  BT_COMMAND_LIGHT_TURN_ON        =    'r'
.EQU  BT_COMMAND_LIGHT_TURN_OFF       =    't'
.EQU  BT_COMMAND_MANUAL_LIGHT_OFF     =    'y'
.EQU  BT_COMMAND_ELEVATION_NORTH      =    'u'
.EQU  BT_COMMAND_ELEVATION_SOUTH      =    'i'
.EQU  BT_COMMAND_AZIMUT_EAST          =    'o'
.EQU  BT_COMMAND_AZIMUT_WEST          =    'p'
.EQU  BT_COMMAND_RESET                =    'z'
.EQU  BT_COMMAND_MANUAL_MOTORS_OFF    =    'l'

TX_BUF:    .BYTE BUF_SIZE   ; buffer de transmisión

.DEF  PTR_TX_L        = R8  ;puntero al buffer de datos a tx.
.DEF  PTR_TX_H        = R9
.DEF  BYTES_A_TX      = R10 ;bytes a transmitir desde el buffer
;--------------------------------------------------------------
;-------------------HEADER DE LA BATERIA-----------------------
.EQU  PIN_BATTERY_LED_OK              =    PD4
.EQU  PIN_BATTERY_LED_LOW             =    PD7

.EQU  MIN_BATTERY_VALUE               =    170
;{0,255} RESULTADO DEL ADC QUE HACE IR A BAJO CONSUMO EL EQUIPO
POR NO TENER SUFICIENTE TENSION.
;YA ESTA AJUSTADO! 10.6v
V_BATTERY_DATA:       .BYTE 8
;--------------------------------------------------------------
```

```
;------------------HEADER DEL PANEL SOLAR-----------------------
.EQU  PIN_SOLAR_PANEL_LED_OK     =     PC1
.EQU  PIN_SOLAR_PANEL_LED_LOW    =     PC0

.EQU  MIN_SOLAR_PANEL_VALUE      =     150
;{0,255} RESULTADO DEL ADC QUE DECIDE SI ES DE DIA O NOCHE.
V_SOLAR_PANEL_DATA: .BYTE 8
;---------------------------------------------------------------
;-------REGISTROS DE USO GENERAL SIN IMPORTAR QUE TENGA---------
.DEF  AUX                        =     R16
.DEF  AUX1                       =     R17
.DEF  AUX2                       =     R18
.DEF  AUX3                       =     R22
.DEF  AUX4                       =     R23
;---------------------------------------------------------------
;------------------------HEADER DE LDRS-------------------------
.EQU CANT_SAMPLES = 8 ;TIENE QUE SER MULTIPLO DE 2

LDR_NO_BUFFER:        .BYTE CANT_SAMPLES
LDR_NE_BUFFER:        .BYTE CANT_SAMPLES
LDR_SE_BUFFER:        .BYTE CANT_SAMPLES
LDR_SO_BUFFER:        .BYTE CANT_SAMPLES
LDR_NO_MEAN:          .BYTE 1
LDR_NE_MEAN:          .BYTE 1
LDR_SE_MEAN:          .BYTE 1
LDR_SO_MEAN:          .BYTE 1
COUNTER:              .BYTE 2
BT_MANUAL_LIGHT:      .BYTE 1 ;FLAG=0xFF:MANUAL.FLAG=0x00:AUTO.
BT_MANUAL_MOTORS:     .BYTE 1 ;FLAG=0xFF:MANUAL.FLAG=0x00:AUTO.
FLAG_AT_NIGHT:        .BYTE 1

;LAS CONSTANTES SON PARA EL MUX EN EL ADC
.EQU LDR_NO           =     3     ;ADC2
.EQU LDR_NE           =     2     ;ADC3
.EQU LDR_SE           =     4     ;ADC4
.EQU LDR_SO           =     5     ;ADC5

.DEF LDR_NO_LOW       =     R2
.DEF LDR_NO_HIGH      =     R3
.DEF LDR_NE_LOW       =     R4
.DEF LDR_NE_HIGH      =     R5
.DEF LDR_SE_LOW       =     R6
.DEF LDR_SE_HIGH      =     R7
.DEF LDR_SO_LOW       =     R11
.DEF LDR_SO_HIGH      =     R12
;---------------------------------------------------------------
;------------------------HEADER DEL PWM-------------------------
.DEF PWM_DATA         =     R21

.EQU PWM_AZIMUT_DEFAULT    = 190 ;{0,255} PARA SETEAR EL PWM
.EQU PWM_ELEVATION_DEFAULT = 190  ;{0,255} PARA SETEAR EL PWM
;200 ES 10V APROX.

;---------------------------------------------------------------
;------------------------HEADER DE MOTORES----------------------
.EQU MOT_1                                  =     PD5
.EQU MOT_2                                  =     PD6
;---------------------------------------------------------------
```

avr_macros.inc

```
;-----------------------------------------------------------------
; colección de macros para microcontroladores AVR
;-----------------------------------------------------------------
; Sintaxis:
;     .macro NOMBRE_MACRO
;          ; cuerpo de la macro
;          ; los parámetros de la macro se referencian como
;          ; @0 (1er parámetro), @1 (2do parámetro), etc.
;     .endm
;-----------------------------------------------------------------

;-----------------------------------------------------------------
; input: resuelve si usa "in"/"lds" según la dirección del
registro
;        de E/S que se lee.
;-----------------------------------------------------------------
.macro     input ; @0= destino {r0, ... , r31}
              ; @1= fuente I/0 ($0000-$FFFF)
.if  @1<0x40
     in    @0,@1 ; si dir del reg de E/S <0x40 uso "in"
.else
     lds   @0,@1 ; sino uso "lds"
.endif
.endm

;-----------------------------------------------------------------
; output: resuelve si usa "out"/"sts" según la dirección del
registro
;         de E/S que se escribe.
;-----------------------------------------------------------------
.macro     output          ; @0= destino I/O ($0000-$FFFF)
                ; @1= fuente, cte o r0..r31
.if  @0<0x40
           out   @0,@1 ; si dir del reg de E/S <0x40 uso "out"
.else
           sts   @0,@1 ; sino uso "sts"
.endif
.endm

.macro     ldiw       ; carga puntero
           ldi   @0L, LOW(@1)
           ldi   @0H, HIGH(@1)
.endm

.macro     movi       ; carga registro con constante
           ldi   AUX,@1
           mov   @0,AUX
.endm

.macro     outi
           ldi    AUX,@1
           output @0,AUX
.endm
```

```
.macro     pushw ;@0   ; Pone el puntero @0 de 16 bits en la pila
           push  @0L
           push  @0H
.endm

.macro     popw  ;@0   ; Saca el puntero @0 de 16 bits de la pila
           pop   @0H
           pop   @0L
.endm

.macro     pushi ;@0   ; Pone en pila un registro de I/O
     in    AUX,@0      ; usa la variable auxiliar t0
                push  AUX
.endm

.macro     popi  ;@0   ; Saca de pila un registro de I/O
           pop   AUX   ; usa la variable auxiliar t0
           out   @0,AUX
.endm

.macro     sti   ;@0,@1 ; Guarda una constante de modo indirecto
           ldi   AUX4,@1 ; Usa: variable aux "t0" y un puntero
           st    @0,AUX4 ;[3 ciclos, 2 words] @0={X, Y, Z}
.endm

.macro     stsi  ;@0,@1 ; Guarda una constante en SRAM
           ldi   AUX,@1 ; Usa: variable auxiliar "t0".
           sts   @0,AUX ;[3 ciclos, 2 words]
.endm

;---------------------MACROS PROPIAS----------------------------
.macro SLDR
;STORE_LDR
;PROTOTYPE: SLDR LDR_XX_LOW,LDR_XX_HIGH,ADC_DATA_H
;RECIBE: ADC_DATA_H EL VALOR DEL LDR
;DEVUELVE: -
     MOV   ZL,@0
     MOV   ZH,@1
     ST    Z+,@2
     MOV   @0,ZL
     MOV   @1,ZH
.endm

.macro LLDR
;LOAD_LDR_TO_POINTER
;PROTOTYPE: LLDR Z,LDR_XX_LOW,LDR_XX_HIGH
     LDI @0L,@1
     LDI @0H,@2
.endm

.macro VECTMEAN ;@0,@1,@2  ;Calcula la media de un vector @0 de
longitud @1 y guarda la media en @2
           PUSH       @1
           LD         AUX3,@0+
           CLR        @2
           DEC        @1
           CLR        AUX2
loop_mean:
```

```
            CLV
            LD          AUX4,@0+
            ADD         AUX3,AUX4
            ADC         @2,AUX2
            DEC         @1
            BRNE        loop_mean
            LDI         AUX2,3
division:
            LSR         @2
            ROR         AUX3
            DEC         AUX2
            BRNE        division
            MOV         @2,AUX3
            POP         @1
.endm

.macro SPWM
;PROTOTYPE SET_PWM: SPWM OCRnx,PWM_DATA
;RECIBE: OCRnx,PWM_DATA
;DEVUELVE: .
      OUTPUT @0,@1
.endm

.macro RPWM
;PROTOTYPE RESET_PWM: RPWM OCRnx
;RECIBE: OCRnx
;DEVUELVE: -
      CLR AUX
      OUTPUT @0,AUX
.endm

.macro ADDI
;PROTOTYPE ADDI: REG,CTE
;RECIBE: REG,CTE
;DEVUELVE: Suma de cte al registro
      LDI AUX4,@1
      ADD @0,AUX4
.endm

.macro ADDP
;PROTOTYPE ADDP: POINTER,REG
;RECIBE: POINTER,REG
;DEVUELVE: Pointer en posición inicial + AUX
      ADD @0L,@1
      BRVC NO_POINTER_OV
      INC @0H
      SUB @1,@0L
      MOV @1,@0L
NO_POINTER_OV: NOP
.endm
;-----------------------------------------------------------------
```

ADC.inc

```
.CSEG
;-----------------------------------------------------------------
;                CONVERSOR ANALOGICO-DIGITAL
;-----------------------------------------------------------------
ADC_INIT:
;ADMUX = REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1 MUX0
;INTERNAL VREF=VCC Y EL DATO AJUSTADO A ¡IZQUIERDA! [ADCH:ADCL].
      LDI
ADC_DATA_L,((0<<REFS1)|(1<<REFS0)|(1<<ADLAR)|(0<<MUX3)|(1<<MUX2)
|(0<<MUX1)|(0<<MUX0))
      OUTPUT ADMUX,ADC_DATA_L
;ADCSRA = ADEN ADSC ADATE ADIF ADIE ADPS2 ADPS1 ADPS0
;SE HABILITA EL ADC, AUTO TRIGGER OFF, FLAG INTERRUPCION EN
CERO, PRESCALER DIV POR 64 [trabaja en aprox 100Khz]
      LDI
ADC_DATA_L,((1<<ADEN)|(0<<ADSC)|(0<<ADATE)|(0<<ADIF)|(0<<ADIE)|(
1<<ADPS2)|(1<<ADPS1)|(0<<ADPS0))
      OUTPUT ADCSRA,ADC_DATA_L

;SETEAR ESTE REGISTRO PARA EL MODO DE AUTO TRIGGER
;     LDI AUX,(0<<ADTS2)|(0<<ADTS1)|(0<<ADTS0)
;     OUTPUT ADCSRB,AUX

;SE DESHABILITA LA PARTE DIGITAL INTERNA DEL PIN A UTILIZAR
;HABILITO SOLO LOS LDRS
      LDI
ADC_DATA_L,((1<<ADC2D)|(1<<ADC3D)|(1<<ADC4D)|(1<<ADC5D))
      OUTPUT DIDR0,ADC_DATA_L
RET
;-----------------------------------------------------------------
ADC_SELECT_INPUT:
;ADMUX = REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1 MUX0
;RECIBE: EL VALOR DEL PIN A SELECCIONAR EN ADC_DATA_L
;DEVUELVE: NADA
      INPUT AUX,ADMUX
      ANDI AUX,(~((1<<MUX3)|(1<<MUX2)|(1<<MUX1)|(1<<MUX0)))
      OR AUX,ADC_DATA_L;NO HAY QUE HACER SHIFT
      OUTPUT ADMUX,AUX
RET
;-----------------------------------------------------------------
ADC_SIMPLE_CONVERSION:
;RECIBE: -
;DEVUELVE: RESULTADO DE LA CONVERSION EN ADC_DATA_H:ADC_DATA_L
      INPUT AUX,ADCSRA
      ORI AUX,((1<<ADEN)|(1<<ADSC))
      OUTPUT ADCSRA,AUX
L2:   INPUT AUX,ADCSRA
      SBRC AUX,ADSC
      RJMP L2
      ANDI AUX,(~(1<<ADEN))
      OUTPUT ADCSRA,AUX

      INPUT ADC_DATA_L,ADCL
      INPUT ADC_DATA_H,ADCH
RET
```

PWM.inc

```
.CSEG
PWM_INIT:
      RCALL PWM_SOLAR_PANEL_INIT
      RCALL PWM_LIGHT_INIT
RET
;-----------------------------------------------------------------
PWM_SOLAR_PANEL_INIT:
;Se inicializan como salida los pines de PWM
      INPUT PWM_DATA,DDRB
      ANDI PWM_DATA,(~((1<<DDB1)|(1<<DDB2)))
      ORI PWM_DATA,((1<<DDB1)|(1<<DDB2))
      OUTPUT DDRB,PWM_DATA
;Se inicializan como Fast PWM y non-inverting mode
      ;Fast PWM: WGM02=0 (por defecto), WGM01=1 y WGM00=1
      ;Non-inverting mode: COM0A1=1 y COM0A0=0
      ;Descripcion de registros en seccion 15.9 (pag 106-112)
      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI
PWM_DATA,((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<WG
M10)|(0<<WGM11));fast PWM, non-inverting
      OUTPUT TCCR1A,PWM_DATA
;Se inicializa el prescaler del PWM
      INPUT PWM_DATA,TCCR1B ;Timer/counter control register B

      ANDI
PWM_DATA,(~((1<<WGM13)|(1<<WGM12)|(1<<CS10)|(1<<CS11)|(1<<CS12))
)
      ORI
PWM_DATA,((0<<WGM13)|(1<<WGM12)|(0<<CS10)|(1<<CS11)|(0<<CS12))
      ;Prescaler = 8
      OUTPUT TCCR1B,PWM_DATA

      CLR   AUX
      OUTPUT     OCR1AL,AUX
      OUTPUT     OCR1BL,AUX
      OUTPUT     OCR1AH,AUX
      OUTPUT     OCR1BH,AUX
RET
;-----------------------------------------------------------------
PWM_LIGHT_INIT:
;Se inicializan como salida los pines de PWM
;Se usa uno solo de los pines (para la luz), el otro es el MOSI,
lo dejamos solo para programar
      INPUT PWM_DATA,DDRD
      ANDI PWM_DATA,(~(1<<DDD3)) ;Mascara para tocar solo D3
      ORI PWM_DATA,((1<<DDD3))
      OUTPUT DDRD,PWM_DATA
;Se inicializan como Fast PWM y non-inverting mode
      ;Fast PWM: WGM22=0 (por defecto), WGM21=1 y WGM20=1
      ;Non-inverting mode: COM2A1=1 y COM2A0=0
      INPUT PWM_DATA,TCCR2A ;Timer/counter control register A
```

```
      ANDI
PWM_DATA,(~((1<<COM2B1)|(1<<COM2B0)|(1<<WGM20)|(1<<WGM21)))
      ORI
PWM_DATA,((1<<COM2B1)|(0<<COM2B0)|(1<<WGM20)|(1<<WGM21))   ;fast
PWM, non-inverting
      OUTPUT TCCR2A,PWM_DATA
;Se inicializa el prescaler del PWM
      INPUT PWM_DATA,TCCR2B ;Timer/counter control register B
      ;hacer mascara de forma tal que los bits 0, 1, 2 no se
toquen
      ANDI PWM_DATA,(~((1<<CS20)|(1<<CS21)|(1<<CS22)))
      ORI PWM_DATA,((0<<CS20)|(1<<CS21)|(0<<CS22))    ;Ver tabla
prescalers al final del archivo
      OUTPUT TCCR2B,PWM_DATA

      CLR   AUX
      OUTPUT     OCR2B,AUX

RET
;-----------------------------------------------------------------
/*
CS22  CS21  CS20  Description
0          0          0          No clock source (timer/counter
stopped)
0          0          1          clkT2S/(no prescaling)
0          1          0          clkT2S/8 (from prescaler)
0          1          1          clkT2S/32 (from prescaler)
1          0          0          clkT2S/64 (from prescaler)
1          0          1          clkT2S/128 (from prescaler)
1          1          0          clkT2S/256 (from prescaler)
1          1          1          clkT2S/1024 (from prescaler)
*/
```

SERIAL_PORT.inc

```
.CSEG

SERIAL_PORT_INIT:
          PUSH  AUX
          PUSH  AUX1
          PUSHW X

          LDI        AUX,HIGH(BAUD_RATE)
          OUTPUT          UBRR0H,AUX ; Velocidad de transmisión
          LDI        AUX,LOW(BAUD_RATE)
          OUTPUT          UBRR0L,AUX

          LDI        AUX,1<<U2X0; Modo asinc., doble velocidad
          OUTPUT          UCSR0A,AUX

     ; Trama: 8 bits de datos, sin paridad y 1 bit de stop
          LDI
     AUX,(0<<UPM01)|(0<<UPM00)|(0<<USBS0)|(1<<UCSZ01)|(1<<UCSZ00
)
          OUTPUT          UCSR0C,AUX

          ; Configura los terminales de TX y RX; y habilita
          ;    únicamente la int. de recepción
          LDI
     AUX,(1<<RXCIE0)|(1<<RXEN0)|(1<<TXEN0)|(0<<UDRIE0)
          OUTPUT          UCSR0B,AUX

          MOVI  PTR_TX_L,LOW(TX_BUF)  ; inicializa puntero al
          MOVI  PTR_TX_H,HIGH(TX_BUF) ; buffer de transmisión.

          LDIW  X,TX_BUF         ; limpia BUF_SIZE posiciones
          LDI        AUX1, BUF_SIZE  ; del buffer de TX
          CLR        AUX
loop_limpia:
          ST         X+,AUX
          DEC        AUX1
          BRNE  loop_limpia

          CLR        BYTES_A_TX ;nada pendiente de transmisión

          POPW  X
          POP        AUX1
          POP        AUX
RET

;-----------------------------------------------------------------
; RECEPCION: Interrumpe cada vez que se recibe un byte x RS232.
;
; Recibe:  UDR (byte de dato)
; Devuelve: nada
;-----------------------------------------------------------------
_LIGHT_TURN_ON:
          CLR AUX
          OUTPUT BT_MANUAL_LIGHT,AUX
```

```
            RCALL LIGHT_TURN_ON
            SER AUX
            OUTPUT BT_MANUAL_LIGHT,AUX
            LDIW  Z,(MSJ_LIGHT_ON*2)
            RCALL TRANSMITIR_MENSAJE
      RJMP SIGO
_LIGHT_TURN_OFF:
            CLR AUX
            OUTPUT BT_MANUAL_LIGHT,AUX
            RCALL LIGHT_TURN_OFF
            SER AUX
            OUTPUT BT_MANUAL_LIGHT,AUX
            LDIW  Z,(MSJ_LIGHT_OFF*2)
            RCALL TRANSMITIR_MENSAJE
            RCALL LIGHT_TURN_OFF
      RJMP SIGO
CALL_PROJECT_NAME:
            LDIW  Z,(MSJ_PROJECT_NAME*2)
            RCALL TRANSMITIR_MENSAJE
      RJMP SIGO
CALL_V_BAT:
            LDIW  Z,(MSJ_V_BAT*2)
            LDIW  Y,V_BATTERY_DATA
            RCALL TRANSMITIR_TENSION
      RJMP SIGO
CALL_V_PANEL:
            LDIW  Z,(MSJ_V_PANEL*2)
            LDIW  Y,V_SOLAR_PANEL_DATA
            RCALL TRANSMITIR_TENSION
      RJMP SIGO
_BT_MANUAL_LIGHT_OFF:
            RCALL BT_MANUAL_LIGHT_OFF
      RJMP SIGO
_BT_COMMAND_MANUAL_MOTORS_OFF:
            RCALL BT_MANUAL_MOTORS_OFF
      RJMP SIGO

ISR_RX_USART_COMPLETA:
            PUSHW Z
            INPUT AUX,UDR0

            CPI AUX,BT_COMMAND_PROJECT_NAME
            BREQ CALL_PROJECT_NAME

            CPI AUX,BT_COMMAND_V_BAT
            BREQ CALL_V_BAT

            CPI AUX,BT_COMMAND_V_PANEL
            BREQ CALL_V_PANEL

            CPI AUX,BT_COMMAND_LIGHT_TURN_ON
            BREQ _LIGHT_TURN_ON

            CPI AUX,BT_COMMAND_LIGHT_TURN_OFF
            BREQ _LIGHT_TURN_OFF

            CPI AUX,BT_COMMAND_MANUAL_LIGHT_OFF
            BREQ _BT_MANUAL_LIGHT_OFF
```

```
            CPI AUX,BT_COMMAND_MANUAL_MOTORS_OFF
            BREQ _BT_COMMAND_MANUAL_MOTORS_OFF

            CPI AUX,BT_COMMAND_AZIMUT_EAST
            BREQ _MOVE_AZIMUT_EAST

            CPI AUX,BT_COMMAND_AZIMUT_WEST
            BREQ _MOVE_AZIMUT_WEST

            CPI AUX,BT_COMMAND_ELEVATION_NORTH
            BREQ _MOVE_ELEVATION_NORTH

            CPI AUX,BT_COMMAND_ELEVATION_SOUTH
            BREQ _MOVE_ELEVATION_SOUTH

            CPI AUX,BT_COMMAND_RESET
            BREQ _RESET

            LDIW  Z,(MSJ_INVALID_COMMAND*2)
            RCALL TRANSMITIR_MENSAJE

            POPW Z
SIGO:       RETI

_RESET:
            POPW Z
      RJMP SETUP

_MOVE_AZIMUT_EAST:
            CLR AUX
            OUTPUT BT_MANUAL_MOTORS,AUX
            LDIW  Z,(MSJ_AZIMUT_EAST*2)
            RCALL TRANSMITIR_MENSAJE
            LDI   AUX3,190
            RCALL MOTOR_AZIMUT_EAST
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL MOTOR_AZIMUT_OFF
            SER AUX
            OUTPUT BT_MANUAL_MOTORS,AUX
      RJMP SIGO
_MOVE_AZIMUT_WEST:
            CLR AUX
            OUTPUT BT_MANUAL_MOTORS,AUX
            LDIW  Z,(MSJ_AZIMUT_WEST*2)
            RCALL TRANSMITIR_MENSAJE
            LDI   AUX3,190
            RCALL MOTOR_AZIMUT_WEST
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL MOTOR_AZIMUT_OFF
            SER AUX
            OUTPUT BT_MANUAL_MOTORS,AUX
      RJMP SIGO
_MOVE_ELEVATION_NORTH:
            CLR AUX
            OUTPUT BT_MANUAL_MOTORS,AUX
```

```
           LDIW  Z,(MSJ_ELEVATION_NORTH*2)
           RCALL TRANSMITIR_MENSAJE
           LDI   AUX3,255
           RCALL MOTOR_ELEVATION_NORTH
           RCALL DELAY_500ms
           RCALL DELAY_500ms
           RCALL MOTOR_ELEVATION_OFF
           SER AUX
           OUTPUT BT_MANUAL_MOTORS,AUX
      RJMP SIGO
_MOVE_ELEVATION_SOUTH:
           CLR AUX
           OUTPUT BT_MANUAL_MOTORS,AUX
           LDIW  Z,(MSJ_ELEVATION_SOUTH*2)
           RCALL TRANSMITIR_MENSAJE
           LDI   AUX3,230
           RCALL MOTOR_ELEVATION_SOUTH
           RCALL DELAY_500ms
           RCALL DELAY_500ms
           RCALL MOTOR_ELEVATION_OFF
           SER AUX
           OUTPUT BT_MANUAL_MOTORS,AUX
      RJMP SIGO

BT_MANUAL_LIGHT_OFF:
           LDIW  Z,(MSJ_MANUAL_LIGHT_OFF*2)
           RCALL TRANSMITIR_MENSAJE
           CLR AUX
           OUTPUT BT_MANUAL_LIGHT,AUX
RET

BT_MANUAL_MOTORS_OFF:
           LDIW  Z,(MSJ_MANUAL_MOTORS_OFF*2)
           RCALL TRANSMITIR_MENSAJE
           CLR AUX
           OUTPUT BT_MANUAL_MOTORS,AUX
RET

;-----------------------------------------------------------------
; TRANSMISION: interrumpe cada vez que puede transmitir un byte.
; Se transmiten "BYTES_A_TX" comenzando desde la posición TX_BUF
del
; buffer. Si "BYTES_A_TX" llega a cero, se deshabilita la
interrupción.
;
; Recibe:  BYTES_A_TX.
; Devuelve: PTR_TX_H:PTR_TX_L, y BYTES_A_TX.
;-----------------------------------------------------------------
ISR_REG_USART_VACIO:       ; UDR está vacío
           PUSH  AUX
           PUSH  AUX1
           PUSHI SREG
           PUSHW X


           TST   BYTES_A_TX ; hay datos pendientes de tx?
           BREQ  FIN_TRANSMISION
```

```
            MOVW  XL,PTR_TX_L; Recupera puntero al próx byte a tx.
            LD    AUX,X+      ; lee byte del buffer y apunta al
            OUTPUT UDR0,AUX ; sgte. dato a tx (en próx int).
            CPI         XL,LOW(TX_BUF+BUF_SIZE)
            BRLO  SALVA_PTR_TX
            CPI         XH,HIGH(TX_BUF+BUF_SIZE)
            BRLO  SALVA_PTR_TX
            LDIW  X,TX_BUF    ; ptr_tx=ptr_tx+1, (módulo BUF_SIZE)

SALVA_PTR_TX:
            MOVW  PTR_TX_L,XL      ; preserva puntero a sgte. dato
            DEC         BYTES_A_TX ; Dec el nro. de bytes a tx.
            BRNE  SIGUE_TX    ; si quedan datos que transmitir
                              ;    vuelve en la próxima int.
;REVISAR ESTE GRUPO DE INSTRUCCIONES
FIN_TRANSMISION:              ; si no hay nada que enviar,
            INPUT AUX,UCSR0B
            CBR         AUX,(1<<UDRIE0)
            OUTPUT      UCSR0B,AUX
            ;se deshabilita la interrupción.

sigue_tx:
            POPW  X
            POPI  SREG
            POP         AUX1
            POP         AUX
            RETI

;-----------------------------------------------------------------
; TRANSMITIR_MENSAJE: transmite el mensaje almacenado en memoria
flash a partir
; de la dirección ¡APUNTADA POR Z! que termina con 0x00 (el 0 no
se transmite).
; Recibe: nada
; Devuelve: PTR_TX_L|H, BYTES_A_TX.
; Habilita la int. de transmisión serie con ISR en
ISR_REG_USART_VACIO().
;-----------------------------------------------------------------
TRANSMITIR_MENSAJE:
            PUSHW Z
            PUSHW X
            PUSH  AUX
            MOVW  XL,PTR_TX_L

LOOP_TRANSMITIR_MENSAJE:
            LPM         AUX,Z+
            TST         AUX
            BREQ  FIN_TRANSMITIR_MENSAJE

            ST          X+,AUX
            INC         BYTES_A_TX

            CPI         XL,LOW(TX_BUF+BUF_SIZE)
            BRLO  LOOP_TRANSMITIR_MENSAJE
            CPI         XH,HIGH(TX_BUF+BUF_SIZE)
            BRLO  LOOP_TRANSMITIR_MENSAJE
            LDIW  X,TX_BUF    ; ptr_tx++ módulo BUF_SIZE
```

```
            RJMP  LOOP_TRANSMITIR_MENSAJE

FIN_TRANSMITIR_MENSAJE:
            INPUT AUX,UCSR0B

            SBR         AUX,(1<<UDRIE0)
            OUTPUT      UCSR0B,AUX

            POP         AUX
            POPW  X
            POPW  Z
RET
;---------------------------------------------------------------
TRANSMITIR_TENSION:
            PUSHW Z
            PUSHW X
            PUSHW Y
            PUSH  AUX
            MOVW  XL,PTR_TX_L

LOOP_TRANSMITIR_TENSION:
            LPM         AUX,Z+
            TST         AUX
            BREQ  LOOP_TRANSMITIR_DATO  ;TERMINO DE MANDAR EL
MENSAJE, AHORA MANDO EL DATO

            ST          X+,AUX
            INC         BYTES_A_TX
            CPI         XL,LOW(TX_BUF+BUF_SIZE)
            BRLO  LOOP_TRANSMITIR_TENSION
            CPI         XH,HIGH(TX_BUF+BUF_SIZE)
            BRLO  LOOP_TRANSMITIR_TENSION
            LDIW  X,TX_BUF   ; ptr_tx++ módulo BUF_SIZE
            RJMP  LOOP_TRANSMITIR_TENSION

LOOP_TRANSMITIR_DATO:
            LD          AUX,Y+
            TST         AUX
            BREQ  FIN_TRANSMITIR_DATO
            ST          X+,AUX
            INC         BYTES_A_TX

            CPI         XL,LOW(TX_BUF+BUF_SIZE)
            BRLO  LOOP_TRANSMITIR_DATO
            CPI         XH,HIGH(TX_BUF+BUF_SIZE)
            BRLO  LOOP_TRANSMITIR_DATO
            LDIW  X,TX_BUF   ; ptr_tx++ módulo BUF_SIZE
            RJMP  LOOP_TRANSMITIR_DATO

FIN_TRANSMITIR_DATO:
            INPUT AUX,UCSR0B
            SBR         AUX,(1<<UDRIE0)
            OUTPUT      UCSR0B,AUX
            POP         AUX
            POPW  Y
            POPW  X
            POPW  Z
RET
```

DELAY.inc

```
;-----------------------------------------------------------------
;          Para crear los delays, no te hagas el crack y
;          usá el bocho de alguien que ya lo pensó:
;          http://www.bretmulvey.com/avrdelay.html
;          TENER EN CUENTA QUE EL MICRO TRABAJA A 8MHZ
;-----------------------------------------------------------------
.CSEG
DELAY_100us:
    ldi  AUX, 2
    ldi  AUX1, 9
L4: dec  AUX1
    brne L4
    dec  AUX
    brne L4
RET

DELAY_50ms:
    ldi  AUX, 3
    ldi  AUX1, 8
    ldi  AUX2, 120
L1: dec  AUX2
    brne L1
    dec  AUX1
    brne L1
    dec  AUX
    brne L1
RET

DELAY_500ms:
    ldi  AUX, 21
    ldi  AUX1, 75
    ldi  AUX2, 191
L3: dec  AUX2
    brne L3
    dec  AUX1
    brne L3
    dec  AUX
    brne L3
    nop
RET
;-----------------------------------------------------------------
```

LIGHT.inc

```
.CSEG
LIGHT_TURN_ON:
      INPUT AUX,BT_MANUAL_LIGHT
      CPI AUX,0xFF               ;[FLAG=0xFF]: SE MANEJA MANUAL.
      BREQ ORDEN_BT_NO_TOCAR_LUZ ;[FLAG=0x00]: SE MANEJA AUTO.

      INPUT PWM_DATA,TCCR2A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM2B1)|(1<<COM2B0)|(1<<WGM20)|(1<<WGM21)))
      ORI
PWM_DATA,((1<<COM2B1)|(0<<COM2B0)|(1<<WGM20)|(1<<WGM21))
;fast PWM, non-inverting
      OUTPUT TCCR2A,PWM_DATA

      RCALL READ_V_BATTERY
      LDI AUX,100
      CPI   ADC_DATA_H,230
      BRSH EXC_CHARGE
      CPI   ADC_DATA_H,170
      BRSH GOOD_CHARGE
      CPI   ADC_DATA_H,150
      BRSH BAD_CHARGE
EXC_CHARGE:
      ADDI AUX,50
GOOD_CHARGE:
      ADDI AUX,50
BAD_CHARGE:
      ADDI AUX,55
      OUTPUT OCR2B,AUX

ORDEN_BT_NO_TOCAR_LUZ:
RET

LIGHT_TURN_OFF:
      INPUT AUX,BT_MANUAL_LIGHT
      CPI AUX,0xFF               ;[FLAG=0xFF]: SE MANEJA MANUAL.
      BREQ ORDEN_BT_NO_TOCAR_LUZ ;[FLAG=0x00]: SE MANEJA AUTO.

      INPUT PWM_DATA,TCCR2A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM2B1)|(1<<COM2B0)|(1<<WGM20)|(1<<WGM21)))
      ORI
PWM_DATA,((0<<COM2B1)|(0<<COM2B0)|(1<<WGM20)|(1<<WGM21))   ;fast
PWM, non-inverting
      OUTPUT TCCR2A,PWM_DATA

      CLR AUX
      OUTPUT OCR2B,AUX
RET
;---------------------------------------------------------------
```

BATTERY.inc

```
.CSEG
BATTERY_INIT:
      INPUT AUX,DDRD
      ANDI AUX,(~((1<<DDD4)|(1<<DDD7
      ORI AUX,((1<<DDD4)|(1<<DDD7))
      OUTPUT DDRD,AUX
      RCALL INDICATE_BATTERY_LOW
;INICIALIZO LA INFORMACION DE LA TENSION PARA Tx POR BT
      LDIW X,V_BATTERY_DATA
      STI   X+,'0'
      STI   X+,'0'
      STI   X+,'.'
      STI   X+,'0'
      STI   X+,'V'
      STI X+,'\r'
      STI X+,'\n'
      STI   X,0
RET

READ_V_BATTERY:
           LDI ADC_DATA_L,ADC_BATTERY ;ELIJO EL PIN DE LA BAT
           RCALL ADC_SELECT_INPUT     ;SELECCIONO LA BATERIA
           RCALL ADC_SIMPLE_CONVERSION ;LLAMO A MEDIR
RET

CHECK_IF_BATTERY_MINIMUM:
      CLC
      CPI        ADC_DATA_H,MIN_BATTERY_VALUE
;COMPARAR PARA VER SI HAY SUFICIENTE BATERIA PARA OPERAR
      BRCS  _INDICATE_BATTERY_LOW
;[CARRY=1]: BATTERY LOW. [CARRY=0]: BATTERY OK
      RCALL INDICATE_BATTERY_OK

RETURN_INDICATE_BATTERY_LOW:
RET

_INDICATE_BATTERY_LOW:
      RCALL INDICATE_BATTERY_LOW
      SEC
      ;[CARRY=1]: BATTERY LOW. [CARRY=0]: BATTERY OK
RJMP RETURN_INDICATE_BATTERY_LOW

INDICATE_BATTERY_LOW:
      INPUT AUX,PORTD
      ANDI
      AUX,(~((1<<PIN_BATTERY_LED_OK)|(1<<PIN_BATTERY_LED_LOW)))
      ORI
      AUX,((1<<PIN_BATTERY_LED_OK)|(0<<PIN_BATTERY_LED_LOW))
            ;PRENDE POR CERO.
      OUTPUT     PORTD,AUX
RET

INDICATE_BATTERY_OK:
      INPUT AUX,PORTD
```

```
      ANDI
      AUX,(~((1<<PIN_BATTERY_LED_OK)|(1<<PIN_BATTERY_LED_LOW)))
      ORI
      AUX,((0<<PIN_BATTERY_LED_OK)|(1<<PIN_BATTERY_LED_LOW))
            ;PRENDE POR CERO.
      OUTPUT     PORTD,AUX
      CLC
      ;[CARRY=1]: BATTERY LOW. [CARRY=0]: BATTERY OK
RET

VBATTERY_TO_ASCII:

                           LDIW X,V_BATTERY_DATA
VBATTERY_DIG:         LDIW Z,(VBATTERY_DIG_TABLE*2)
                           CLR AUX1
                           CLR AUX2
LOOP_DIG_BATTERY:
                           RCALL SET_VBATTERY_DIG
                           BRTS V_BATTERY_DEC
                           INC AUX1
                           CPI AUX1,10
                           BRLO LOWER_THAN_10_DIG_BATTERY
                           CLR AUX1
                           INC AUX2
LOWER_THAN_10_DIG_BATTERY: BRTC LOOP_DIG_BATTERY
                           STI X+,48+1
                           STI X+,48+5

V_BATTERY_DEC:        LDIW Z,(VBATTERY_DIG_TABLE*2)
                           SUBI AUX1,48
                           SUBI AUX2,48
                           LDI AUX3,10
                           MUL AUX2,AUX3
                           MOV AUX2,R0
                           ADD AUX1,AUX2
                           ADDP Z,AUX1
                           LPM AUX1,Z
                           SUB AUX1,ADC_DATA_H

                           STI  X+,'.'
                           ldi  aux2,160
;MULTIPLICAR POR 160/256. DIVIDIR POR 256 ES QUEDARME CON LA
PARTE ALTA
                           MUL  AUX1,AUX2
                           MOV  AUX1,R1
                           SUB AUX3,AUX1

                           ADDI AUX3,48 ;SE TRANSFORMA EN ASCII
                           CPI AUX3,':'
                           BREQ ajuste_9_bat

vuelvo_9_bat:         ST X+,AUX3 ;El ASCII del número en AUX.
                           STI  X+,'V'
                           STI X+,'\r'
                           STI X+,'\n'
                           STI  X,0
                           CLT
RET
```

```
ajuste_9_bat:
           ldi aux3,'9'
           rjmp vuelvo_9_bat

SET_VBATTERY_DIG:
                            LPM AUX,Z+
                            INC AUX
                            CP ADC_DATA_H,AUX
                            BRSH END_SET_VBATTERY_DIG
                            ADDI AUX1,48
                            ADDI AUX2,48
                            ST X+,AUX2
                            ST X+,AUX1
                            SET
END_SET_VBATTERY_DIG: RET
;---------------------------------------------------------------
```

SOLAR_PANEL.inc

```
.CSEG
SOLAR_PANEL_INIT:
      INPUT AUX,DDRC
      ANDI AUX,(~((1<<DDC0)|(1<<DDC1)))     ;Mascara para tocar
los leds del panel solar
      ORI AUX,((1<<DDC0)|(1<<DDC1))
      OUTPUT DDRC,AUX
      RCALL INDICATE_SOLAR_PANEL_LOW
      ;INICIALIZO LA INFORMACION DE LA TENSION PARA TRANSMITIR
POR BLUETOOTH
      LDIW X,V_SOLAR_PANEL_DATA
      STI   X+,'0'
      STI   X+,'0'
      STI   X+,'.'
      STI   X+,'0'
      STI   X+,'V'
      STI X+,'\r'
      STI X+,'\n'
      STI   X,0

RET
;---------------------------------------------------------------
READ_V_SOLAR_PANEL:
;RECIBE: NADA
;DEVUELVE: TENSION DEL PANEL EN ADC_DATA_H
      LDI ADC_DATA_L,ADC_SOLAR_PANEL   ;ELIJO EL PIN DEL PV
      RCALL ADC_SELECT_INPUT           ;SELECCIONAO EL PV
      RCALL ADC_SIMPLE_CONVERSION      ;MIDO EL PV
RET
;---------------------------------------------------------------
CHECK_IF_SOLAR_PANEL_MINIMUM:
      CLC
      CPI        ADC_DATA_H,MIN_SOLAR_PANEL_VALUE;COMPARAR PARA
VER SI HAY SUFICIENTE SOL.
      BRCS  _INDICATE_SOLAR_PANEL_LOW
      ;[CARRY=1]: SOLAR_PANEL LOW. [CARRY=0]: SOLAR_PANEL OK
```

```
      RCALL INDICATE_SOLAR_PANEL_OK

RETURN_INDICATE_SOLAR_PANEL_LOW:RET

_INDICATE_SOLAR_PANEL_LOW:
      RCALL INDICATE_SOLAR_PANEL_LOW
      SEC
      ;[CARRY=1]: SOLAR_PANEL LOW. [CARRY=0]: SOLAR_PANEL OK
RJMP RETURN_INDICATE_SOLAR_PANEL_LOW

INDICATE_SOLAR_PANEL_LOW:
      INPUT AUX,PORTC
      ANDI
      AUX,(~((1<<PIN_SOLAR_PANEL_LED_OK)|(1<<PIN_SOLAR_PANEL_LED_
LOW)))
      ORI
      AUX,((1<<PIN_SOLAR_PANEL_LED_OK)|(0<<PIN_SOLAR_PANEL_LED_LO
W))        ;PRENDE POR CERO.
      OUTPUT     PORTC,AUX
RET

INDICATE_SOLAR_PANEL_OK:
      INPUT AUX,PORTC
      ANDI
      AUX,(~((1<<PIN_SOLAR_PANEL_LED_OK)|(1<<PIN_SOLAR_PANEL_LED_
LOW)))
      ORI
      AUX,((0<<PIN_SOLAR_PANEL_LED_OK)|(1<<PIN_SOLAR_PANEL_LED_LO
W))        ;PRENDE POR CERO.
      OUTPUT     PORTC,AUX
      CLC
      ;[CARRY=1]: SOLAR_PANEL LOW. [CARRY=0]: SOLAR_PANEL OK
RET
;-----------------------------------------------------------------

ORIENTATE_SOLAR_PANEL:
           RCALL PWM_SOLAR_PANEL_INIT
      ;TIENE QUE ESTAR EN "PWM.inc"
;ESTAN LOS PROMEDIOS DE LOS LDR. HAY QUE CP Y MOV EL PV.
;PRIMERO EN ASIMUT, LUEGO EN ELEVACION.
;EL SOL SALE DEL ESTE Y SE PONE EN EL OESTE.
      ;SI AMBOS SON 0xFn NO SE MUEVE
      INPUT AUX,LDR_NO_MEAN
      INPUT AUX1,LDR_NE_MEAN
      ANDI  AUX,0xF0
      ANDI  AUX1,0xF0
      CPI        AUX,0xF0
      BRNE  NO_SON_F_AZIMUT
      SUB        AUX,AUX1
      BREQ  AMBOS_SON_F_AZIMUT
NO_SON_F_AZIMUT:
      ;COMPARAR_NO_NE:
            INPUT AUX,LDR_NO_MEAN
            INPUT AUX1,LDR_NE_MEAN
            ANDI  AUX,0xF0
            ANDI  AUX1,0xF0
            CP AUX,AUX1
            BRLO _MOTOR_AZIMUT_EAST
```

```
      RETURN_MOTOR_AZIMUT_EAST:


      INPUT AUX,LDR_NO_MEAN
            INPUT AUX1,LDR_NE_MEAN
            ANDI  AUX,0xF0
            ANDI  AUX1,0xF0
            CP AUX1,AUX
            BRLO _MOTOR_AZIMUT_WEST
      RETURN_MOTOR_AZIMUT_WEST:
AMBOS_SON_F_AZIMUT:
;-----------ACA NOS MOVEMOS CON NORTE Y SUR
      INPUT AUX,LDR_NO_MEAN
      INPUT AUX1,LDR_SO_MEAN
      ANDI  AUX,0xF0
      ANDI  AUX1,0xF0
      CPI         AUX,0xF0
      BRNE  NO_SON_F_ELEVATION
      SUB         AUX,AUX1
      BREQ  AMBOS_SON_F_ELEVATION
NO_SON_F_ELEVATION:
      ;COMPARAR_NO_SO:
            INPUT AUX,LDR_NO_MEAN
      ;INPUT AUX2,LDR_NE_MEAN
            INPUT AUX1,LDR_SO_MEAN
            ANDI  AUX,0xF0
            ANDI  AUX1,0xF0
            CP AUX,AUX1
            BRLO _MOTOR_ELEVATION_NORTH
      RETURN_MOTOR_ELEVATION_NORTH:

            INPUT AUX,LDR_NO_MEAN
            INPUT AUX1,LDR_SO_MEAN
            ANDI  AUX,0xF0
            ANDI  AUX1,0xF0
            CP AUX1,AUX
            BRLO _MOTOR_ELEVATION_SOUTH
      RETURN_MOTOR_ELEVATION_SOUTH:
AMBOS_SON_F_ELEVATION:
RET
;-----------------------------------------------------------------
_MOTOR_AZIMUT_EAST:
      SUB AUX,AUX1
      ORI   AUX,0xF0;ERA 80
      MOV   AUX3,AUX

      RCALL MOTOR_AZIMUT_EAST
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
      RCALL MOTOR_AZIMUT_OFF
      ;     RCALL DELAY_500ms
RJMP RETURN_MOTOR_AZIMUT_EAST

_MOTOR_AZIMUT_WEST:
      SUB AUX1,AUX
```

```
      ORI   AUX1,0xF0       ;ERA 80
      MOV   AUX3,AUX1

      RCALL MOTOR_AZIMUT_WEST
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
      RCALL MOTOR_AZIMUT_OFF
RJMP RETURN_MOTOR_AZIMUT_WEST


_MOTOR_ELEVATION_SOUTH:
      SUB AUX1,AUX
      ORI   AUX1,0xAF
      MOV   AUX3,AUX1
      LDI AUX3,230
      RCALL MOTOR_ELEVATION_SOUTH
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
      RCALL MOTOR_ELEVATION_OFF
RJMP RETURN_MOTOR_ELEVATION_SOUTH

_MOTOR_ELEVATION_NORTH:
      SUB AUX1,AUX
      ORI   AUX1,0xAF
      MOV   AUX3,AUX1
      LDI AUX3,230
      RCALL MOTOR_ELEVATION_NORTH
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
            RCALL DELAY_50ms
      RCALL MOTOR_ELEVATION_OFF
RJMP RETURN_MOTOR_ELEVATION_NORTH
;---------------------------------------------------------------

VPANEL_TO_ASCII:
                  LDIW X,V_SOLAR_PANEL_DATA
;Apunto X a la sección de memoria donde se guarda la V_PANEL
                  LDIW Z,VPANEL_DIG_TABLE*2
;Apunto Z a la sección de ROM donde se delimita cada unidad
VPANEL_DIG:       CLR AUX1          ;Acá se guardará el x1
                  CLR AUX2          ;Acá se guardará el x10
LOOP_DIG_PANEL:
            RCALL SET_VPANEL_DIG       ;Busco el número
            BRTS V_PANEL_DEC
            INC AUX1 ;para ver si es el sig. en el prox ciclo
            CPI AUX1,10 ;Si superé 10 tengo que poner el x10 en 1
            BRLO LOWER_THAN_10_DIG_PANEL
            CLR AUX1   ;Si es 10, x1 es 0
            INC AUX2   ;Si es 10, x10 es 1
LOWER_THAN_10_DIG_PANEL:   BRTC LOOP_DIG_PANEL
```

```
     STI X+,48+1 ;Si se completaron todos los ciclos, x10 es 1
     STI X+,48+9 ;Si se completaron todos los ciclos, x1 es 9
V_PANEL_DEC:    LDIW Z,VPANEL_DIG_TABLE*2
;Vuelvo a apuntar Z al comienzo de la tabla donde están los
números que delimitan cada unidad
     SUBI AUX1,48    ;Lo vuelvo a convertir en número no ASCII
     SUBI AUX2,48
     LDI AUX3,10     ;Cargo 10 en AUX3 para multiplicar el x10
por 10 y poder sumarselo al x1
     MUL AUX2,AUX3   ;Multiplico el x10 por 10
     MOV AUX2,R0     ;Paso el resultado de R0 a AUX2
     ADD AUX1,AUX2 ;Sumo x1 y x10 para tener el número completo
     ADDP Z,AUX1
;Le sumo al puntero Z la cantidad de posiciones que se tiene que
mover en la tabla para ver el número máximo que puede tener esa
unidad
     LPM AUX1,Z      ;Cargo el número delimitador
     SUB AUX1,ADC_DATA_H   ;Le resto al delimitador lo que medí
     STI  X+,'.'
     ldi  aux2,210
     MUL  AUX1,AUX2
     MOV  AUX1,R1
     SUB AUX3,AUX1
     ADDI AUX3,48-1        ;SE TRANSFORMA EN ASCII

vuelvo_9_panel: ST X+,AUX3 ;El ASCII del número en AUX.
                STI  X+,'V'
                STI X+,'\r'
                STI X+,'\n'
                STI  X,0
                CLT
RET

ajuste_9_panel:
                ldi aux3,'9'
                rjmp vuelvo_9_panel

SET_VPANEL_DIG:
          LPM AUX,Z+ ;Leo de tabla en ROM
          INC AUX    ;Incremento lo leído pP se usará BRSH
          CP ADC_DATA_H,AUX ;Veo si el número es menor al de la
tabla a ver si encontramos el valor
          BRSH END_SET_VPANEL_DIG    ;Si es >= sigo buscandO
          ADDI AUX1,48               ;LO PASO A ASCII
          ADDI AUX2,48
          ST X+,AUX2                 ;Guardo el x10
          ST X+,AUX1                 ;Guardo el x1
          SET
END_SET_VPANEL_DIG:  RET
;---------------------------------------------------------------
```

LDRS.inc

```
.CSEG
LDRS_INIT:
           RCALL LDRS_POINTERS_RESET
LOOP: CLR ADC_DATA_H
           SLDR LDR_NO_LOW,LDR_NO_HIGH,ADC_DATA_H
           SLDR LDR_SO_LOW,LDR_SO_HIGH,ADC_DATA_H
           SLDR LDR_SE_LOW,LDR_SE_HIGH,ADC_DATA_H
           SLDR LDR_NE_LOW,LDR_NE_HIGH,ADC_DATA_H
           INPUT AUX,COUNTER
           INC AUX
           OUTPUT COUNTER,AUX
           CPI AUX,CANT_SAMPLES            ;CHEQUEAR QUE NO ESTE
HACIENDO UNO DE MENOS
           BRLO LOOP
           RCALL LDRS_POINTERS_RESET
RET
;-----------------------------------------------------------------
LDRS_POINTERS_RESET:
      MOVI LDR_NO_LOW,LOW(LDR_NO_BUFFER)
      MOVI LDR_NO_HIGH,HIGH(LDR_NO_BUFFER)

      MOVI LDR_NE_LOW,LOW(LDR_NE_BUFFER)
      MOVI LDR_NE_HIGH,HIGH(LDR_NE_BUFFER)

      MOVI LDR_SE_LOW,LOW(LDR_SE_BUFFER)
      MOVI LDR_SE_HIGH,HIGH(LDR_SE_BUFFER)

      MOVI LDR_SO_LOW,LOW(LDR_SO_BUFFER)
      MOVI LDR_SO_HIGH,HIGH(LDR_SO_BUFFER)

      CLR AUX
      OUTPUT COUNTER,AUX
RET
;-----------------------------------------------------------------
LDRS_READ:
;CHEQUEO QUE SEA MENOR A CANT_SAMPLES. SI ES MAYOR, RESETEO LOS
PUNTEROS. [EL BUFFER ESTA LLENO, SACO LA PRIMER MUESTRA].

      INPUT AUX,COUNTER
      CPI AUX,CANT_SAMPLES
      BREQ _LDRS_POINTERS_RESET
LDRS_POINTERS_RESET_RETURN:
      INPUT AUX,COUNTER
      INC AUX
      OUTPUT COUNTER,AUX

      RCALL READ_LDR_NO
;DEVUELVE EL RESULTADO EN ADC_DATA_H
      SLDR LDR_NO_LOW,LDR_NO_HIGH,ADC_DATA_H
;SLDR: UBICA [LDR_XX_LOW,LDR_XX_HIGH] EN UN PTR Y GUARDA
ADC_DATA_H

      RCALL READ_LDR_NE
      SLDR LDR_NE_LOW,LDR_NE_HIGH,ADC_DATA_H
```

```
      RCALL READ_LDR_SE
      SLDR LDR_SE_LOW,LDR_SE_HIGH,ADC_DATA_H

      RCALL READ_LDR_SO
      SLDR LDR_SO_LOW,LDR_SO_HIGH,ADC_DATA_H
RET
;-----------------------------------------------------------------
_LDRS_POINTERS_RESET:
      RCALL LDRS_POINTERS_RESET
RJMP LDRS_POINTERS_RESET_RETURN

LDRS_MEAN:
;OBS: PARA HACER EL PROMEDIO NO IMPORTA SI NO SE TOMARON
CANT_SAMPLES, EL BUFFER ESTA INICIALIZADO CON CERO.
      LDI   AUX,CANT_SAMPLES ;VECTMEAN NECESITA LA # DE MUESTRAS

      LDIW Z,LDR_NO_BUFFER   ;UBICO EL LDR_NO EN UN PUNTERO.
      VECTMEAN Z,AUX,AUX1   ;CALCULO EL PROMEDIO DE Z DE LARGO
"AUX" DEJANDO EL DATO EN "AUX1"
      OUTPUT LDR_NO_MEAN,AUX1;GUARDAMOS EN RAM EL PROMEDIO.

      LDIW Z,LDR_NE_BUFFER
      VECTMEAN Z,AUX,AUX1
      OUTPUT LDR_NE_MEAN,AUX1

      LDIW Z,LDR_SE_BUFFER
      VECTMEAN Z,AUX,AUX1
      OUTPUT LDR_SE_MEAN,AUX1

      LDIW Z,LDR_SO_BUFFER
      VECTMEAN Z,AUX,AUX1
      OUTPUT LDR_SO_MEAN,AUX1
RET
;-----------------------------------------------------------------
READ_LDR_NO:
      LDI ADC_DATA_L,LDR_NO
      RCALL ADC_SELECT_INPUT
      RCALL ADC_SIMPLE_CONVERSION
RET

READ_LDR_SO:
      LDI ADC_DATA_L,LDR_SO
      RCALL ADC_SELECT_INPUT
      RCALL ADC_SIMPLE_CONVERSION
RET

READ_LDR_SE:
      LDI ADC_DATA_L,LDR_SE
      RCALL ADC_SELECT_INPUT
      RCALL ADC_SIMPLE_CONVERSION
RET

READ_LDR_NE:
      LDI ADC_DATA_L,LDR_NE
      RCALL ADC_SELECT_INPUT
      RCALL ADC_SIMPLE_CONVERSION
RET
;-----------------------------------------------------------------
```

MOTORS.inc

```
.CSEG
MOTORS_INIT:
      INPUT AUX,DDRD
      ANDI AUX,(~((1<<DDD5)|(1<<DDD6)))
      ORI AUX,((1<<DDD5)|(1<<DDD6))
      OUTPUT DDRD,AUX

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      OUTPUT PORTD,AUX

      CLR   AUX
      OUTPUT     OCR1AL,AUX
      OUTPUT     OCR1AH,AUX
      OUTPUT     OCR1BL,AUX
      OUTPUT     OCR1BH,AUX
RET
;---------------------------------------------------------------
MOTOR_AZIMUT_EAST:
;RECIBE EN AUX3 EL VALOR A MOVERSE EN EL PWM
;SETEO MOT_1 Y MOT_2
;SETEO LOS ENABLE
;SETEO EL PW,
      INPUT AUX,BT_MANUAL_MOTORS
      CPI AUX,0Xff ;[FLAG=0xFF]: SE MANEJAN MANUALMENTE.
      BREQ ORDEN_BT_NO_TOCAR_AZIMUT_E ;[FLAG=0x00]:  AUTO.

      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI
PWM_DATA,((1<<COM1A1)|(0<<COM1A0)|(1<<WGM10)|(0<<WGM11));fast
PWM, non-inverting
      OUTPUT TCCR1A,PWM_DATA

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      ORI AUX,((0<<MOT_1)|(1<<MOT_2))
      OUTPUT PORTD,AUX

      CLR   AUX
      OUTPUT     OCR1AL,AUX
      rcall smooth_move_azimut
ORDEN_BT_NO_TOCAR_AZIMUT_E:
RET

MOTOR_AZIMUT_WEST:
;RECIBE EN AUX3 EL VALOR A MOVERSE EN EL PWM
;SETEO MOT_1 Y MOT_2
;SETEO LOS ENABLE
;SETEO EL PWM
      INPUT AUX,BT_MANUAL_MOTORS
      CPI AUX,0xFF ;[FLAG=0xFF]: SE MANEJAN MANUALMENTE.
      BREQ ORDEN_BT_NO_TOCAR_AZIMUT_W ;[FLAG=0x00]: AUTO.
```

```
      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI
PWM_DATA,((1<<COM1A1)|(0<<COM1A0)|(1<<WGM10)|(0<<WGM11))   ;fast
PWM, non-inverting
      OUTPUT TCCR1A,PWM_DATA

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      ORI AUX,((1<<MOT_1)|(0<<MOT_2))
      OUTPUT PORTD,AUX

      CLR  AUX
      OUTPUT     OCR1AL,AUX
      rcall smooth_move_azimut
ORDEN_BT_NO_TOCAR_AZIMUT_W:
RET

smooth_move_azimut:
      clr aux4
_s:   inc aux4
      OUTPUT     OCR1AL,AUX4
      OUTPUT     OCR1BL,AUX4
      RCALL DELAY_100us
      cpi aux4,170
      brsh nxt
      rcall delay_100us
      rcall delay_100us
nxt:
      rcall delay_100us
      rcall delay_100us
      cp aux4,AUX3
      brlo _s
ret

MOTOR_ELEVATION_SOUTH:
;RECIBE EN AUX3 EL VALOR A MOVERSE EN EL PWM
;SETEO MOT_1 Y MOT_2
;SETEO LOS ENABLE
;SETEO EL PWM
      INPUT AUX,BT_MANUAL_MOTORS
      CPI AUX,0xFF ;[FLAG=0xFF]: SE MANEJAN MANUALMENTE.
      BREQ ORDEN_BT_NO_TOCAR_ELEVATION_S ;[FLAG=0x00]: AUTO.
      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI
PWM_DATA,((1<<COM1B1)|(0<<COM1B0)|(1<<WGM10)|(0<<WGM11))   ;fast
PWM, non-inverting
      OUTPUT TCCR1A,PWM_DATA

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      ORI AUX,((1<<MOT_1)|(0<<MOT_2))
```

```
      OUTPUT PORTD,AUX

      CLR   AUX
      OUTPUT     OCR1BL,AUX

      rcall smooth_move_elevation
ORDEN_BT_NO_TOCAR_ELEVATION_S:
RET

MOTOR_ELEVATION_NORTH:
;RECIBE EN AUX3 EL VALOR A MOVERSE EN EL PWM
;SETEO MOT_1 Y MOT_2
;SETEO LOS ENABLE
;SETEO EL PWM
      INPUT AUX,BT_MANUAL_MOTORS
      CPI AUX,0xFF    ;[FLAG=0xFF]: SE MANEJAN MANUALMENTE.
      BREQ ORDEN_BT_NO_TOCAR_ELEVATION_N ;[FLAG=0x00]: AUTO.
      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI
PWM_DATA,((1<<COM1B1)|(0<<COM1B0)|(1<<WGM10)|(0<<WGM11))   ;fast
PWM, non-inverting
      OUTPUT TCCR1A,PWM_DATA

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      ORI AUX,((0<<MOT_1)|(1<<MOT_2))
      OUTPUT PORTD,AUX

      CLR   AUX
      OUTPUT     OCR1BL,AUX

      RCALL smooth_move_elevation
ORDEN_BT_NO_TOCAR_ELEVATION_N:
RET

smooth_move_elevation:
      clr aux4
_sM:  inc aux4
      OUTPUT     OCR1AL,AUX4
      OUTPUT     OCR1BL,AUX4
      RCALL DELAY_100us
      cpi aux4,170
      brsh nxtM
      rcall delay_100us
      rcall delay_100us
nxtM:
      rcall delay_100us
      rcall delay_100us
      cp aux4,AUX3
      brlo _sM
ret

MOTOR_AZIMUT_OFF:
;SETEO MOT_1 Y MOT_2
;SETEO LOS ENABLE
```

```
;SETEO EL PWM
      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI PWM_DATA,((1<<WGM10)|(0<<WGM11))  ;fast PWM, non-
inverting
      OUTPUT TCCR1A,PWM_DATA

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      OUTPUT PORTD,AUX

      INPUT AUX4,OCR1AL

brake_AZIMUT:
      OUTPUT     OCR1AL,AUX4
      cpI aux4,140
      brSH  ASDF
      RCALL DELAY_100us
      RCALL DELAY_100us
ASDF: RCALL DELAY_100us
      RCALL DELAY_100us
      RCALL DELAY_100us
      dec aux4
      brne brake_AZIMUT

      CLR   AUX
      OUTPUT     OCR1AL,AUX
RET

MOTOR_ELEVATION_OFF:
;SETEO MOT_1 Y MOT_2
;SETEO LOS ENABLE
;SETEO EL PWM
      INPUT PWM_DATA,TCCR1A ;Timer/counter control register A
      ANDI
PWM_DATA,(~((1<<COM1A1)|(1<<COM1B1)|(1<<COM1A0)|(1<<COM1B0)|(1<<
WGM10)|(1<<WGM11)))
      ORI PWM_DATA,((1<<WGM10)|(0<<WGM11))  ;fast PWM, non-
inverting
      OUTPUT TCCR1A,PWM_DATA

      INPUT AUX,PORTD
      ANDI AUX,(~((1<<MOT_1)|(1<<MOT_2)))
      OUTPUT PORTD,AUX

      INPUT AUX4,OCR1BL

brake_elevation:
      OUTPUT     OCR1BL,AUX4
      cpI aux4,170
      brSH  ASF
      RCALL DELAY_100us
      RCALL DELAY_100us
ASF:  RCALL DELAY_100us
      RCALL DELAY_100us
      RCALL DELAY_100us
```

```
      dec aux4
      brne brake_elevation

      CLR   AUX
      OUTPUT     OCR1BL,AUX

RET

RETURN_TO_ORIGIN:
;BIEN HARDCODEADO.
;EL SOL SALE DEL ESTE Y SE PONE EN EL OESTE.
      INPUT AUX,FLAG_AT_NIGHT
      CPI AUX,0xFF
;SI ESTA PRENDIDO EL FLAG, QUIERE DECIR QUE YA LO HIZO A LA
NOCHE
      BREQ YA_RETORNO_AL_ORIGEN
      SER AUX
      OUTPUT FLAG_AT_NIGHT,AUX
      LDI AUX3,240
      RCALL MOTOR_AZIMUT_EAST
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
            RCALL DELAY_500ms
      RCALL MOTOR_AZIMUT_OFF
YA_RETORNO_AL_ORIGEN:
RET
;-----------------------------------------------------------------
```

MESSAGES.inc

```
;MENSAJES EN ROM
MSJ_PROJECT_NAME: .DB"BIENVENIDO!",'\r','\n',"GRACIAS POR USAR
EL SOLAR TRACKER!",'\r','\n',0

MSJ_V_BAT:      .DB  "LA TENSION DE LA BATERIA ES: ",0
MSJ_V_PANEL:    .DB "LA TENSION DEL PANEL ES: ",0
MSJ_LIGHT_ON:   .DB "ENCENDISTE LA LUZ.",'\r','\n',0
MSJ_LIGHT_OFF:  .DB "APAGASTE LA LUZ.",'\r','\n',0
MSJ_AZIMUT_EAST:.DB "SE MOVIO EL PANEL AL ESTE.",'\r','\n',0
MSJ_AZIMUT_WEST:.DB  "SE MOVIO EL PANEL AL OESTE.",'\r','\n',0
MSJ_ELEVATION_NORTH: .DB "SE MOVIO EL PANEL AL
NORTE.",'\r','\n',0

MSJ_ELEVATION_SOUTH: .DB "SE MOVIO EL PANEL AL SUR.",'\r','\n',0
MSJ_MANUAL_LIGHT_OFF:  .DB "LA LUZ PASA A CONTROLARSE
AUTOMATICAMENTE.",'\r','\n',0

MSJ_MANUAL_MOTORS_OFF: .DB "LOS MOTORES PASAN A CONTROLARSE
AUTOMATICAMENTE.",'\r','\n',0

MSJ_INVALID_COMMAND:   .DB "EL COMANDO INGRESADO ES
INVALIDO!",'\r','\n',0

;TABLAS EN ROM
VPANEL_DIG_TABLE:     .DB
     12,25,38,51,64,76,89,102,115,127,141,153,166,179,192,205,21
7,230,243,255

VPANEL_DEC_TABLE:     .DB 0,1,2,3,4,5,5,6,7,8,9

VBATTERY_DIG_TABLE:   .DB
16,32,48,65,81,97,114,130,146,163,179,195,212,228,244,255

VBATTERY_DEC_TABLE:   .DB 0,1,1,2,3,3,4,5,5,6,7,7,8,9,9
;---------------------------------------------------------------
```

Anexo: galería de fotografías.

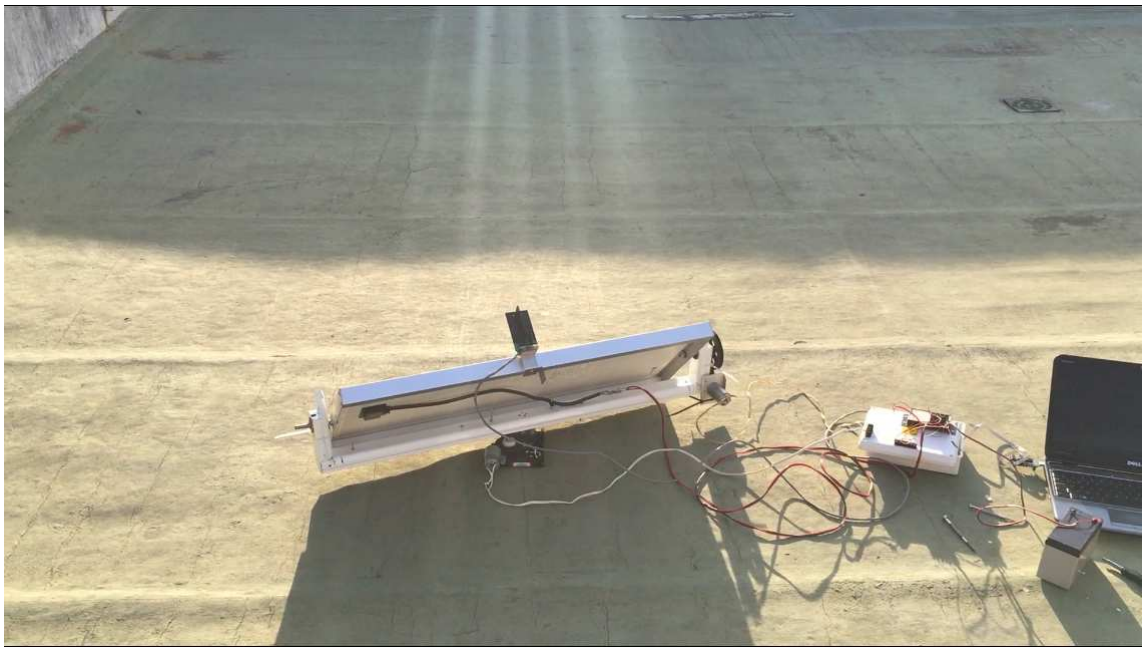

Figura 5: Solar Tracker en funcionamiento en terraza de edificio.

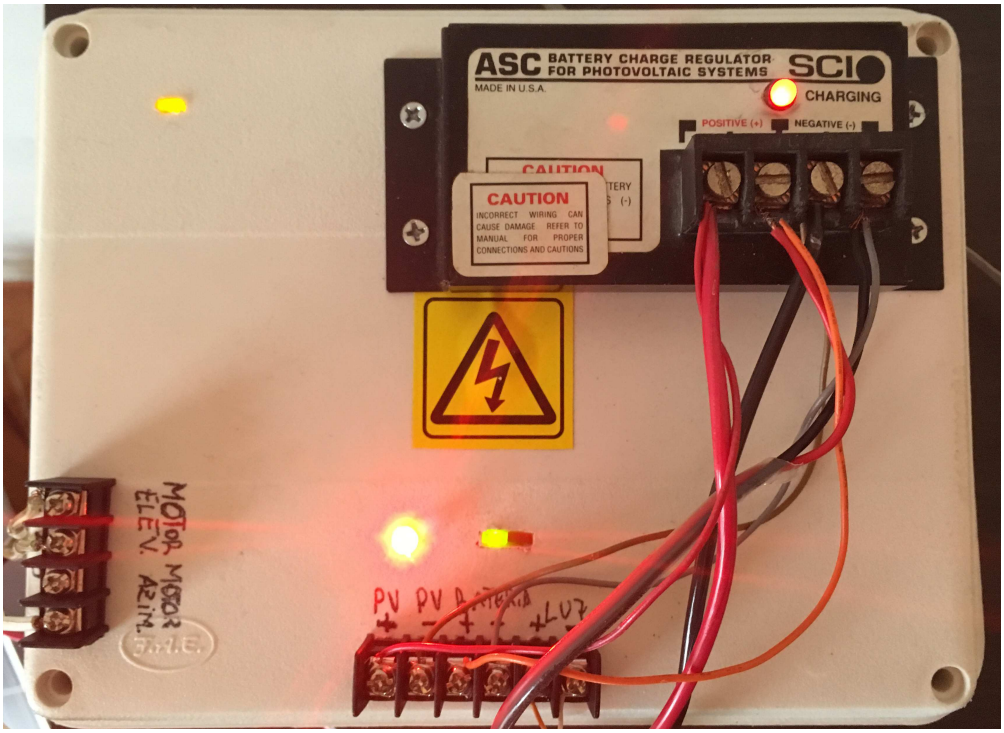

Figura 6: Vista frontal del sistema de control.

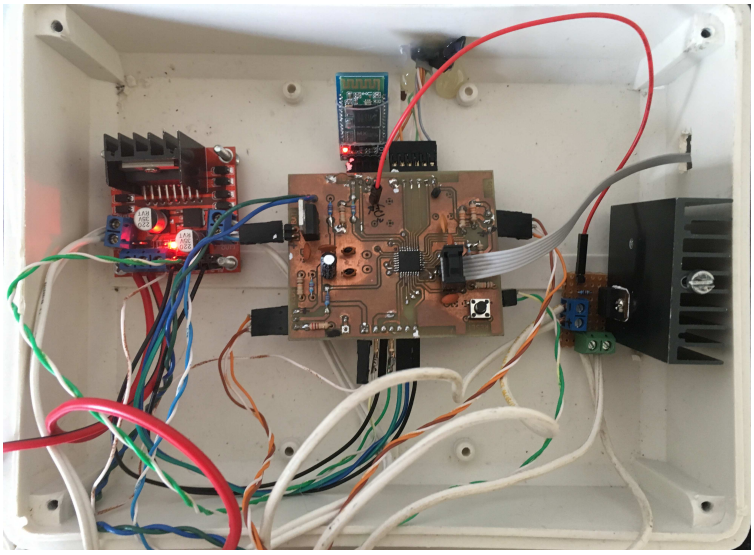

Figura 7: Vista frontal (sin carcaza) del sistema de control.

Figura 8: Engranajes correspondientes al motor de elevación.

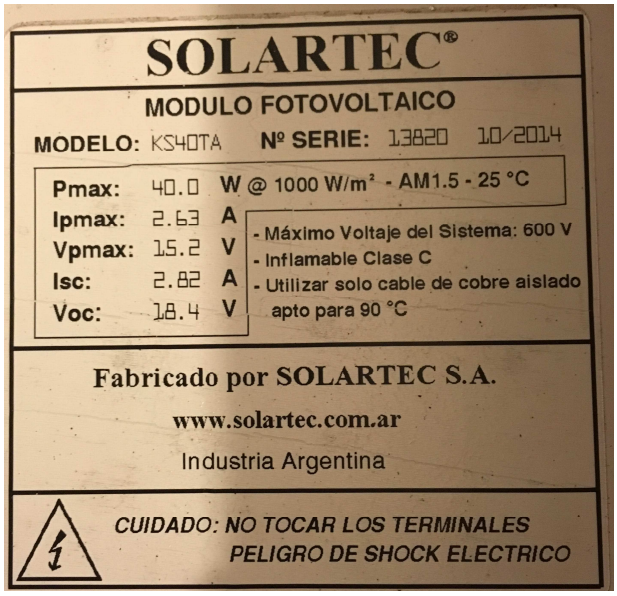

Figura 9: Especificaciones técnicas del panel fotovoltaico.

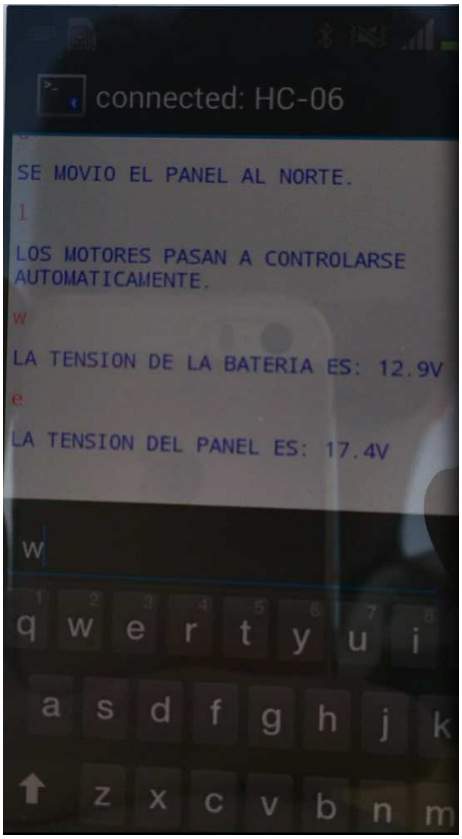

Figura 10: Directivas desde un Smartphone mediante Bluetooth.