

K-JUNIOR OS MANUAL

USAGE AND FUNCTIONS

DECEMBER 2010 EDITION



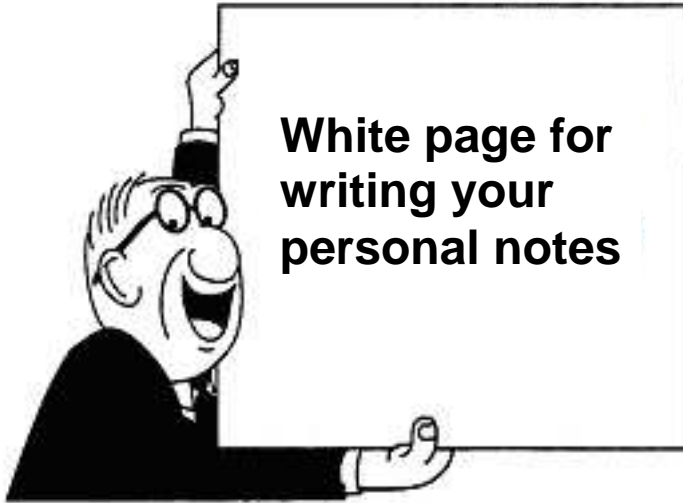
NOTICE

The information included in this manual is subject to change without prior notice.
K-Team cannot be held responsible for any technical or writing omissions, nor any consequent damage that may arise.

In the same way, the product names mentioned in this manual and in the CD-Rom for identification purposes may be trademarks, whether registered or not, belonging to their respective owners.

Table of contents

CHAPTER 1 : INTRODUCTION	Page 5
CHAPTER 2 : THE COMPILER	Page 6
CHAPTER 3 : USING THE C COMPILER (PCW CCS)	Page 7
CHAPTER 4 : USING THE K-JUNIOR UPLOADER	Page 8
4.1 – Safety usage.....	Page 8
4.2 – Reload the « Bootloader »	Page 8
CHAPTER 5 : KJUNIOR OS	Page 9
5.1 – What’s that?.....	Page 9
5.2 – Its structure	Page 9
CHAPTER 6 : FUNCTIONS	Pages 10 - 17
6.1 – Definition	Page 10
6.2 – Configuration functions of the robot.	Page 11
6.3 – « Flags » reading function.....	Page 12
6.4 – Input and Output functions.....	Pages 13 - 14
6.5 – Delay and « Timer » functions.....	Page 15
6.6 – I/O functions of the external connectors.....	Page 16
6.7 – Real time tasks.....	Page 17
6.8 – C compiler specific functions.....	Page 17
CHAPTER 7 : EXAMPLES	Page 18
CHAPTER 8 : MODULES LIBRARIES	Pages 18 - 24
8.1 – Definition	Page 19
8.2 – Functions of the « I/O interface board » module.....	Page 20
8.3 – Functions of the « Ultrasonic detector» module.....	Page 21
8.4 – Functions of the « Linear camera » module.....	Pages 22 - 23
8.5 – Functions of the « Vocal synthesizer» module.....	Page 24
8.6 – Functions of the « LCD screen» module.....	Page 24 - 26
8.7 – Functions of the « Gripper» module.....	Page 27 - 29
APPENDIX	Page 30



1 – INTRODUCTION

K-Junior is provided with a processor from Microchip, the PIC16F887.

This processor is used in many applications; therefore there is already a big community with many users.

Microchip provides a free assembler for the PICs. But this low-level language is not well suitable for a fast development in robotics. It is rather a specific language for the low-level layers.

In this manual we will introduce the particular case of C language to you, which is to our opinion the most suitable.

The whole K-Junior OS (« Operating System », embedded system on the K-Junior) was developed in this language. We distribute the source codes under the LGPL license, which means that you can use it freely and without restraint (except for commercial purpose).

Let's see more in detail the programming part.

This document is suitable for people having already some basic knowledge in C language.

2 – THE COMPILER

Nowadays there are on the market many C compilers for PIC micro-processors on the market, more or less expensive but also more or less reliable.

After having tested many among others, K-Team recommends the « PCW » C compiler from CCS for its quality, support and price ratio and its multi-platforms support. It is one among the best in the market: it comprises many high-level integrated functions along with a friendly user interface.

Following commercial agreements with CCS, you can obtain a specific version of the PWC compiler, including a compiler/debugger (limited to the PIC 16F887 and 16F886) and an editor in full version.

You can obtain this version of the PCW compiler by contacting K-Team.

If you wish, you can obtain an external PIC « ICD_S » programmer to program again the microprocessor.

**Currently we do not assure any support for another C compiler.
If you already have another C compiler for PIC,
you need to adapt the source code of the « K-Junior OS ».**

3 – USAGE OF THE PIC C COMPILER

The following steps are required to install the PIC C compiler from CCS:

- create a directory « PICC » in the directory « *Program Files* » of Windows ;
- In this directory copy the licence « *pcm.crg* » you received;
- Run the setup file « *SetupPcw.exe* » you received or available in the directory « PCW CCS » of the CD-ROM « *Installation Professor* » delivered with the DIDAPACK from K-TEAM.

When the install procedure is finished, you may modify and compile the K-Junior OS as follows :

- Copy on your PC the directory « *K-JuniorOS_Sources* » from the Support CD-ROM or download the last version on the K-Team web site.
- start the PIC C compiler (short-cut on the Windows desktop or in the menu bar « *Start / Programmes / PIC-C* » ;
- Click on « Project » then « Open » and select the file « *KJOs.pjt* » in the directory « *K-JuniorOS / K-JuniorOS_Sources* » copied in your PC;
- Click on « Project » then « Open » and select the file « *KJOs.c* » ;
- Add your code ;
- Click on the object « compile » or press the F9 key of your keyboard to compile.

If there is no error in your code, the compiler will generate a file « *KJOs.hex* », compiled program that can be uploaded with the « *K-Junior Uploader* » into K-Junior (see next CHAPTER).

4 – USAGE OF THE K-JUNIOR UPLOADER

Once your program is compiled, find the file « KJOs.hex » created by the compiler.

Use the « K-Junior Uploader » (Tiny Bootloader) to upload this OS (compiled program) into the « Flash » memory of the microprocessor of the K-Junior robot.

With the "K-Junior Uploader", you can upload a program through the USB serial port into the microprocessor of the K-Junior robot.

Please refer to the K-Junior UserManual to configure the Uploader software (see section 5.1).

The data (program uploading) sent from the "K-Junior Uploader" are processed and saved by a software named "KJ-Bootloader" that K-Team loaded into the Flash memory of the PIC when the K-Junior was assembled.

4.1 Safety usage

A bad manipulation may erase the "Bootloader" program from the memory.

In this case, the upload through the USB serial port will not be possible anymore (usage "K-Junior Uploader").

If you load a program you compiled (C compiler, Assembly, etc.), make sure you respect following rule:

The memory zone from 0x1F00 to 0x1FFF ("Bootloader" reserved zone)
must never be written.

The compiled files (".hex") issued from K-TEAM respect this rule.

If you erase by mistake this memory zone during a bad manipulation and you have an external programmer, reload the "KJ-Bootloader.hex" file included in your K-Junior Support CD-ROM (see next CHAPTER).

Refer to the K-Junior Uploader section in the K-Junior UserManual of the CD-ROM to do this upload.

4.2 Reloading the « BootLoader » by the means of an external programmer

If you have a serial programmer for the PIC16F887 microprocessor (« PicStart © » from Microchip or « ICD-S » from CCS for example) you can reflash the memory of the K-Junior's PIC and reload the "KJ-Bootloader.hex" file included in the K-Junior Support CD-Rom or on the K-team website and recover the serial upload functionality.

5. K-JUNIOR OS

5.1 What's that?

We can say that:

« K-Junior OS is for K-Junior, what Windows or Linux is for your PC ».

The K-Junior OS manages the resources of K-Junior and it enables you to access the embedded hardware.

5.2 It's structure

The source code of the K-Junior OS can be found on the K-junior Support CD-Rom or on the K-Team web site under Support->Download->K-Junior.

This OS is composed of several files :

- « *KJOs.c* », main file containing the « main » ; in this file, you declare your variables, your functions and you write your code;
- « *KJunior.c* », file containing the sources of the control and access function to the K-Junior robot hardware (*);
- « *KJunior.h* », file containing the functions prototypes you use and the K-Junior parameters (quartz frequency, baudrate...) (*);
- « *16f887.h* », file containing the registers addresses of the processor (*);
- « *variables.c* », file containing the internal variables (*);
- « *constantes.h* », file containing the constants definitions (*);
- « *versions.txt* », description of the modifications done at every version.

(*) : these files must not be modified by “ordinary” users programming the behaviours of K-Junior, but “expert” users who would like to modify and improve the functions of « K-Junior OS ».

Other files are available in the directory « *Libs* » for using the extension modules (I/O interface board, linear camera, Gripper, etc.).

6 – FUNCTIONS

6.1 Definition

Here's a small explanation before starting with the list of the available functions for the K-Junior OS.

General notation of the available functions « $a = f(b)$; »

- **a** is the returned value;
- **f** is the function;
- **b** is the parameter;

Variable notations:

- **char** is a 8bits signed integer variable, 1 octet (-128 to 127) ;
- **unsigned char** is a 8 bits unsigned variable, 1 octet (0 to 255) ;
- **int1** is a bit variable (boolean, 0 or 1) ;
- **int** is a 16bits signed integer variable (-32768 to 32767) ;
- **unsigned int** is a 16bits unsigned integer variable (0 to 65535) ;
- **unsigned int16** is a 16bits unsigned integer variable (0 to 65535) ;
- **signed int8** is a 8bits signed integer variable (-128 to 127) ;

6.2 Configuration functions of the robot

void KJunior_init(**void**)

Goal : Initialize KJunior

Parameter : None

Return : Nothing

Example : KJunior_init();

Notice : This function must be at the beginning of every program to initialize the KJunior robot with this default configuration:

- automatic reading of the IR sensor activated
- RS232 management activated
- Automatic reading of the TV receiver activated
- TV remote control activated.

void KJunior_config_auto_refresh_sensors(**int1** Bit)

Goal : Configure the IR sensor reading

Parameter : MANUAL or REFRESH (by default)

Return : Nothing

Example : KJunior_config_auto_refresh_sensors(MANUAL);

Notice : This function configures if the automatic reading of the IR sensors are activated (parameter REFRESH) or not (parameter MANUAL).

void KJunior_config_auto_refresh_tv_remote(**int1** Bit)

Goal : Configure the automatic reading of the TV remote receiver.

Parameter : MANUAL or REFRESH (by default)

Return : Nothing

Example : KJunior_config_auto_refresh_tv_remote(MANUAL);

Notice : This function configures if the TV remote receiver is refreshing automatically (Parameter REFRESH) or not (Parameter MANUAL).

void KJunior_config_rs232_control(**int1** Bit)

Goal : Activate or note the "Serial Remote Control"

Parameter : DISABLE or ENABLE(by default)

Return : Nothing

Example : KJunior_config_rs232_control(DISABLE);

Notice : This function activates (parameter ENABLE) or not (parameter DISABLE) the RS232 "Serial Remote Control".

Warning: if this mode is activated, you can use the "printf" function (i.e. to view some values with a terminal) Goal you can't read anything from the serial port (getc, gets, etc.. are not available), all the data will be read by the "Serial Remote Control" process.

void KJunior_config_tv_remote_control(**int1** Bit)

Goal : Activate or not the "TV Remote Control" mode

Parameter : DISABLE or ENABLE(by default)

Return : Nothing

Example : KJunior_config_tv_remote_control(DISABLE);

Notice : This function activates (parameter ENABLE) or not (parameter DISABLE) the "TV Remote Control" mode (remote control of the motors with a TV remote).

6.3 « Flags » reading function

int1 KJunior_flag_sensors_refreshed(**void**)

Goal : To know if the IR sensors were refreshed

Parameter : None

Return : The bit is set if the sensors just have been refreshed.

Example : `i = KJunior_flag_sensors_refreshed();`

Notice : You must reset the flag after with the « KJunior_flag_sensors_reset » function.

void KJunior_flag_sensors_reset(**void**)

Goal : Reset the IR sensor flag

Parameter : None

Return : Nothing

Example : `KJunior_flag_sensors_reset();`

int1 KJunior_flag_rs232_filtering(**void**)

Goal: To know if the “Serial Remote Control” mode is activated or not

Parameter : None

Return : 1 if the mode is activated and 0 if not.

Example : `i = KJunior_flag_rs232_filtering();`

int1 KJunior_flag_tv_data_refreshed(**void**)

Goal: To know if the TV receiver was refreshed or not.

Parameter : None

Return : 1 if some data was received on the TV receiver.

Example : `i = KJunior_flag_tv_data_refreshed();`

Notice : You must reset the flag after with the “KJunior_flag_tv_data_reset” function.

int1 KJunior_flag_tv_data_emitting(**void**)

Goal: To know if the emitting is complete or not.

Parameter : None

Return : 1 if some data are still emitting or 0 it's done

Example : `i = KJunior_flag_tv_data_emitting();`

void KJunior_flag_tv_data_reset(**void**)

Goal : Reset the TV receiver flag.

Parameter : None

Return : None

Example : `KJunior_flag_tv_data_reset();`

6.4 Input and Output functions

Signed int16 KJunior_get_proximity(char Sensor)

Goal : Get the value of the proximity sensor

Parameter : FRONT, FRONTLEFT, FRONTRIGHT, LEFT, RIGHT, REAR, GROUNDLEFT, GROUNDRIGHT, GROUNDFRONTLEFT, GROUNDFRONTRIGHT

Return : 10bits value (0(nothing) to 1024(an obstacle is very near))

Example : i = KJunior_get_proximity(FRONT); Return the proximity value of the “Front” IR sensor.

signed int16 KJunior_get_brightness(char Sensor)

Goal : Get the brightness value of a IR sensor.

Parameter : FRONT, FRONTLEFT, FRONTRIGHT, LEFT, RIGHT, REAR, GROUNDLEFT, GROUNDRIGHT, GROUNDFRONTLEFT, GROUNDFRONTRIGHT

Return : 10 bits value (0(big luminosity) to 1024(dark))

Example : i = KJunior_get_brightness(REAR); Return the brightness value of the “Rear” sensor.

unsigned char KJunior_get_switch_state()

Goal : Read the state of the dip switch

Parameter : None

Return : 0 (all switch turn Off) to 7 (all switch On).

Example : i = KJunior_get_switch_state(); Return the state of the Dip switch

unsigned char KJunior_get_tv_data(void)

Goal : Get the data received on the TV receiver.

Parameter : None

Return : 6 bits value (0 to 63)

Example : i = KJunior_get_tv_data(); Return the coded value of the pressed TV remote button or from another K-Junior. i.e. the value will be 2 if the button 2 of the numeric pad is pressed.

unsigned char KJunior_get_tv_addr(void)

Goal : Get the address received on the TV receiver.

Parameter : None

Return : 5 bits value (0 to 31)

Example : i = KJunior_get_tv_addr(); Return the address of the TV remote or of another K-Junior. i.e. the value will be 2 if another K-Junior send a command like “KJunior_send_tv_value(2.10)”

void KJunior_send_tv_value(unsigned char addr, unsigned char data)

Goal : send a data and an address value on the IR emitter using the RC5 code

Parameter : addr (0 to 31), data (0 to 63)

Return : Nothing

Example : KJunior_send_tv_value(2,10); Send a message on the IR emitter with an address of 2 and a Data of 10. Every closest K-Junior will received this message.

void KJunior_set_speed(**signed int8** LeftSpeed, **signed int8** RightSpeed)

Goal : Set the speed of the motor left (first value) and right (second value)

Parameter : -20 to 20 (0 = Stop)

Return : None

Example : KJunior_set_speed(5 , -5); set a speed of +5 on the left motor and -5 on the right motor. The robot will turn on itself in this example.

Notice : The PWM command has a fixed frequency of 240Hz and variable duty cycle from 0 to 100% fixed by the consigne, where 20 set a PWM ratio of 100%.

void KJunior_beep(**unsigned char** Freq)

Goal : Make a continuous sound with the « Buzzer » at the selected frequency sound

Parameter : Frequency (0 to 100) where 0 is Off and 100 is 2kHz

Return : None

Example : KJunior_beep(91); Make a sound of 200Hz on the « Buzzer » of the KJunior. Please refer to section 3.2.3 in the K-Junior user manual to calculate the corresponding frequency.

void KJunior_led_left(**int1** State)

Goal : turn ON or OFF the LED “Left”

Parameter : ON (1) or OFF (0)

Return : None

Example : KJunior_led_left(ON); Turn ON the “Left” LED of the robot.

void KJunior_led_frontleft(**int1** State)

Goal : turn ON or OFF the LED “Front Left”

Parameter : ON (1) or OFF (0)

Return : None

Example : KJunior_led_frontleft(ON); Turn ON the “Front Left” LED of the robot.

void KJunior_led_frontright(**int1** State)

Goal : turn ON or OFF the LED “Front Right”

Parameter : ON (1) or OFF (0)

Return : None

Example : KJunior_led_frontright(ON); Turn ON the “Front Right” LED of the robot.

void KJunior_led_right(int1 State)

Goal : turn ON or OFF the LED “Right”

Parameter : ON (1) or OFF (0)

Return : None

Example : KJunior_led_right(ON); Turn ON the “Right” LED of the robot.

void KJunior_led_onoff(int1 State)

Goal : turn ON or OFF the LED « On/Off »

Parameter : ON (1) or OFF (0)

Return : None

Example : KJunior_led_onoff(ON); Turn ON the « On/Off » of the robot.

void KJunior_manual_refresh_sensors()

Goal : Refresh manually the IR sensors.

Parameter : None

Return : None

Example : KJunior_manual_refresh_sensors(); refresh the value of the IR sensors.

Notice : This function is useful only after disabling the automatic refresh sensor mode with the function « KJunior_config_auto_refresh_sensors ».

Then use one of the two functions to read the IR sensors (« KJunior_get_proximity » or « KJunior_get_brightness »).

6.5 Delay and « Timer » functions

void KJunior_delay_s(**unsigned int** Delay)

Goal : Put a delay in second (temporization).

Parameter : Time to wait in seconds (0 to 65535)

Return : None

Example : KJunior_delay_s(120); Wait 2 minutes.

void KJunior_delay_ms(**unsigned int** Delay)

Goal : Put a delay in millisecond (temporization).

Parameter : Time to wait in ms (0 to 65535)

Return : None

Example : KJunior_delay_ms(5000); Wait 5s.

void KJunior_delay_us(**unsigned int** Delay)

Goal : Put a delay in microsecond (temporization).

Parameter : Time to wait in us (0 to 65535)

Return : None

Example : KJunior_delay_us(500); wait 500 us.

A 32 bits timer provides the time (in millisecond) elapsed since the robot is turned on.

unsigned int32 KJunior_get_time(**void**)

Goal : Get the value of the elapsed time in ms.

Parameter : None

Return : Unsigned int32 (0 to 4294967295)

Example : Timer = KJunior_get_time(); Timer = elapsed time in ms.

void KJunior_set_time(**unsigned int32** Time)

Goal : Initialize or force the elapsed time of the timer for the « KJunior_get_time » function.

Parameter : Unsigned int32 (0 to 4294967295)

Return : None

Example : KJunior_set_time(0); Initialize the timer to 0 ms.

6.6 I/O functions of the external connectors

int1 KJunior_ext_read_PINB6(**void**)

Goal : Read the digital Input/Output RB6 of the microcontroller.

Parameter : None

Return : 0 or 1

Example : `i = KJunior_ext_read_PINB6();` Return the state (0 or 1) of RB6 pin.

Notice : This Input/Output is connected to the external connector "Power supply and I²C Bus" (see KJunior schematics).

int1 KJunior_ext_read_PINB7(**void**)

Goal : Read the digital Input/Output RB7 of the microcontroller.

Parameter : None

Return : 0 or 1

Example : `i = KJunior_ext_read_PINB7();` Return the state (0 or 1) of RB7 pin.

Notice : This Input/Output is connected to the external connector "Power supply and I²C Bus" (see KJunior schematics).

void KJunior_ext_write_PINB6(**int1** Bit)

Goal : Set the value of the digital output RB6 of the microcontroller.

Parameter : 0 or 1

Return : None

Example : `i = KJunior_ext_write_PINB6(1);` set the RB6 pin to 1 (+5V)

Notice : This Input/Output is connected to the external connector "Power supply and I²C Bus" (see KJunior schematics).

void KJunior_ext_write_PINB7(**int1** Bit)

Goal : Set the value of the digital output RB7 of the microcontroller.

Parameter : 0 or 1

Return : None

Example : `i = KJunior_ext_write_PINB7(1);` set the RB7 pin to 1 (+5V)

Notice : This Input/Output is connected to the external connector "Power supply and I²C Bus" (see KJunior schematics).

6.8 C compiler specific functions

Like any C compiler, the CCS compiler has a very big set of functions in addition of the « ANSY » functions, like the I²C bus access, random number generation, delay, etc...
You can find the details of these functions in the user manual of the CCS compiler.

7 – EXAMPLES

Many examples with comments are available (free to use in your project) in the support CD-rom.

A RTF file « *KJuniorGlossaireFonctionsC.rtf* » are available too in the « *KJuniorOS/KJuniorOS_Sources* » directory, it allows you to quickly choose the needed function and to copy it to your code (See the C compilation tutorial in the « *programming tools* » Section of the CD-rom.

8 – MODULES LIBRARIES

8.1 Definition

As the Hemisson turrets are compatible with the K-Junior, all the turrets libraries are compatible too. Then all the following functions have the same named as the Hemisson turrets functions.

In your own KJunior OS code, you can use some management and access functions of the turrets (GenIO, LinCam, USSensor, Gripper, etc..) via the I²C bus of the Robot.

These functions are defined in the « **.h* » files available in the « *Libs* » directory of the CD-rom.

These libraries for the management and access functions are located in the different sub-directories with the same summary denomination as the turret. « *GenIO* » e.g. for the Input Output module.

These directories contain the following files:

- « *Hemxx.h* », file having the sources and the define of the turret access function.
- « *Example_xx.c* », file containing an example of turret utilisation and explanation to use it in your project.

To use these functions in your project, place the "Hemxx.h" of the turret in your project directory and add « *#include Hemxx.h* » in your « *KJOs.c* » projectfile.

8.2 Functions of the « I/O interface board » module

void HemGenIO_Init(**void**)

Goal : Initialize the turret

Parameter : None

Return : None

Example : HemGenIO_Init();

Notice : It's the first function to call at the beginning of your Code to initialize the GenIO turret.

char HemGenIO_Read_Version(**void**)

Goal : Read the firmware version (OS) of the GenIO turret connected on the I²C Bus

Parameter : None

Return : Unsigned int8 (Firmware version number)

Example : NumVer = HemGenIO_Read_Version();

Notice : This function can be used to verify the presence of the turret. If 0xFF is returned then no turret is plugged.

int1 HemGenIO_Read_Digital(**char** Input)

Goal : Read the state of a digital Input.

Parameter : 0 to 11 (digital input number)

Return : 0 or 1 (state of the digital input)

Example : EtatD2 = HemGenIO_Read_Digital(2); Return the state of the digital input n^o2.

Notice : This function will automatically configure the selected I/O as a Input.

void HemGenIO_Write_Digital(**char** Output , **int1** State)

Goal : Set the the state of a digital output of the turret.

Parameter1 : 0 to 11 (digital output number)

Parameter2 : 0 or 1 (State of the selected output)

Return : None

Example : HemGenIO_Write_Digital(9, 1); Set the digital ouput n^o9 to +5V.

Notice : This function will automatically configure the selected I/O as an Output

unsigned char HemGenIO_Read_Analog(**char** Input)

Goal : Read the value of an analog Input

Parameter : 0 to 4 (Analog input number)

Return : Unsigned int8 (0 = 0V and 255 = 5V)

Example : ValueA3 = HemGenIO_Read_Analog(3); Return the analog value of the input number 3. (i.e. if ValueA3 = 150, Voltage of A3 = $150 \times 5 / 255 = 2.94V$).

8.3 Functions of the « Ultrasonic detector» module

void HemUltraSon_Init(**void**)

Goal : Initialize the turret

Parameter : None

Return : None

Example : HemUltraSon_Init();

Notice : It's the first function to call at the beginning of your Code to initialize the USsensor turret

char HemUltraSon_Read_Version(**void**)

Goal : Read the firmware version (OS) of the USsensor turret connected on the I²C Bus

Parameter : None

Return : Unsigned int8 (Firmware version number)

Example : NumVer = HemUltraSon_Read_Version();

Notice : This function can be used to verify the presence of the turret. If 0xFF is returned then no turret is plugged.

unsigned char HemUltrason_Read_Brightness(**void**)

Goal : read the value of the brightness sensor

Parameter : None

Return : Unsigned int8.

Example : Lum = HemUltraSon_Read_Brightness(); Return the analog value of the brightness sensor.

void HemUltraSon_Start_Mesure(**void**)

Goal : Start a distance acquisition.

Parameter : None

Return : None

Example : HemUltraSon_Start_Mesure();

Notice : This function includes a delay of 65ms (emitting and receiving time of the Ultrasons).

unsigned int16 HemUltrason_Read_Value(**char** EchoNumber)

Goal : Read the distance value of the selected Echo (unit cm)

Parameter : 0 to 7 (Number of the Echo)

Return : Unsigned int16(0 to 65535)

Example : Dist1 = HemUltraSon_Read_Value(0); Return the centimetre distance of the first echo.

void HemUltraSon_Init_Range_Register(**unsigned char** Value)

Goal : Initialize the maximal range of the sensor.

Parameter : unsigned int16 (0 to 65535)

Return : None

Example : HemUltraSon_Init_Range_Register (24); maximal range distance limited to 1075 mm (see the calculation below)

Notice : The maximum range in mm is defined by:

$$D_{max} = (\text{Parameter} \times 43\text{mm}) + 43 \text{ mm} ;$$

$$1\ 075\text{mm} = (24 \times 43\text{mm}) + 43\text{mm} \text{ for the example.}$$

The echo beyond this value will return 0.

8.4 Functions of the « Linear camera » module

The sensor of the linear camera is composed of 3 detection areas which hold 34 pixels each with 256 grey levels.

The pixels value (grey levels 0 to 255) of each area is defined in different tables of 34 unsigned int8 values as follows:

```
unsigned char HemLinCam_Pixels_Zone1[34] ;
```

```
unsigned char HemLinCam_Pixels_Zone2[34] ;
```

```
unsigned char HemLinCam_Pixels_Zone3[34] ;
```

To use in your code to have access to the pixels value .

```
void HemLinCam_Init( void )
```

Goal : Initialize the turret

Parameter : None

Return : None

Example : HemLinCam_Init();

Notice : It's the first function to call at the beginning of your Code to initialize the LinCam turret.

```
char HemLinCam_Read_Version( void )
```

Goal : Read the firmware version (OS) of the LinCam turret connected on the I²C Bus

Parameter : None

Return : Unsigned int8 (« Firmware » version number)

Example : NumVer = HemLinCam_Read_Version();

Notice : This function can be used to verify the presence of the turret. If 0xFF is returned then no turret is plugged..

```
void HemLinCam_Set_Threshold( unsigned char Value )
```

Goal : Define the threshold value for the pixels reading.

Parameter : Unsigned int8 (0 to 255)

Return : None

Example : HemLinCam_Set_Threshold(100); Define the Threshold to 100

Notice : When the pixels threshold reading is made, the pixels with a grey level >= than 100 will have a value of 255 and the other (< 100) will reset to 0.

```
unsigned char HemLinCam_Read_Threshold( void )
```

Goal : read the setting threshold value.

Parameter : None

Return : Unsigned int8 (0 to 255)

Example : Seuil = HemLinCam_Read_Threshold();

```
void HemLinCam_Set_Exposition_Time( unsigned char Value )
```

Goal : Define the exposure time in ms.

Parameter : 1 to 10

Return : None

Example : HemLinCam_Set_Exposition_Time (2);

Notice : Warning with the too big value, the pixels will be saturate very fast.

unsigned char HemLinCam_Read_Exposition_Time(**void**)

Goal : read the exposure time.

Parameter : None

Return : Unsigned int8 (0 to 255)

Example : ExpoMs = HemLinCam_Read_Exposition_Time (); Return the exposure time in ms.

void HemLinCam_Read_Pixels(**void**)

Goal : Read the value of every pixels.

Parameter : None

Return : None

Example : HemLinCam_Read_Pixels() ;
// Reading of the pixel n°17 of the central area
If (HemLinCam_Pixels_Zone2[17] > 20)
{ ... }

Notice : After the reading, the values are stocked in the different value tables: « HemLinCam_Pixels_Zone1 », « HemLinCam_Pixels_Zone2 » and « HemLinCam_Pixels_Zone3 ». Each table has a size of 34 pixels, which correspond respectively to the left, central and right of the picture.

void HemLinCam_Read_Pixels_Tresholded(**void**)

Goal : Read the value of every pixel after the threshold

Parameter : None

Return : None

Example : HemLinCam_Read_Pixels_Tresholded () ;
// Reading of the thresholded pixel n°17 of the central area
If (HemLinCam_Pixels_Zone2[17] == 255)
{ ... }

Notice : After the reading, the values are stocked in the different value tables: « HemLinCam_Pixels_Zone1 », « HemLinCam_Pixels_Zone2 » and « HemLinCam_Pixels_Zone3 ». Each table has a size of 34 pixels, which correspond respectively to the left, central and right of the picture.

When the pixels threshold reading is made, the pixels with a grey level \geq than the threshold will have a value of 255 and the other ($<$ threshold) will reset to 0.

void HemLinCam_Set_Led_State(**int1** State)

Goal : Turn On or Off the Led

Parameter : ON (1) or OFF (0)

Return : None

Example : HemLinCam_Set_Led_State (ON); Turn On the camera LED.

8.5 Functions of the « Vocal synthesizer» module

void HemTextToSpeech_Init(**void**)

Goal : Initialize the turret

Parameter : None

Return : None

Example : HemTextToSpeech_Init();

Notice : It's the first function to call at the beginning of your Code to initialize the TextToSpeech turret

char HemTextToSpeech_Version(**void**)

Goal : Read the firmware version (OS) of the TextToSpeech turret connected on the I²C Bus

Parameter : None

Return : Unsigned int8 (« Firmware » version number)

Example : NumVer = HemTextToSpeech_Version();

Notice : This function can be used to verify the presence of the turret. If 0xFF is returned then no turret is plugged..

void HemTextToSpeech_Speed(**char** Value)

Goal : Define the speed teechting

Parameter : Unsigned int8 (0 to 255)

Return : None

Example : HemTextToSpeech_Speed(1);

Notice : Set a small value to speak the sentence slowly.

void HemTextToSpeech_Pitch(**char** Value)

Goal : Define the « pitch » value (hue of the pronunciation).

Parameter : Unsigned int8 (0 to 255).

Return : None

Example : HemTextToSpeech_Pitch(2);

Notice : Set a small value to speak the sentence high-pitched.

void HemTextToSpeech_Speak(**char** Phrase[], **char** Taille)

Goal : Play a loaded sentence.

Parameter 1 : Character table which contains the sentence to speak in text format.

Parameter 2 : Size of the sentence to speak, in a unsigned int8 value (0 to 255)

Return : None

Example : **char** Phrase[256];

sprintf(Phrase, "Hello, my name is KJunior");

HemTextToSpeech_Speak(Phrase, sizeof(Phrase));

Notice : The sentence « Hello, my name is KJunior » is speak in function of the speed and the pitch settings

void HemTextToSpeech_Speak_PreDef(**char** Num)

Goal : Speak a sentence which is saved in the turret.

Parameter : Number of the saved sentence to speak (0 to 32)

Return : None

Example : HemTextToSpeech_Speak_PreDef(20);

Notice : Speak the saved sentence number 20 which is loaded with the « SP03.exe » software (see EMP).

8.6 Functions of the « LCD screen» module

The LCD screen turret is made up a screen of 2 lines of 12 characters each and 3 push button. The text for each line is define by two characters table of 12 bytes:

```
char Line1[12] ;
```

```
char Line2[12] ;
```

The state of the push button is stocked in the following variable:

```
int1 SW1, SW2, SW3 ;
```

```
void HemLCD_Init( void )
```

Goal : Initialize the turret

Parameter : None

Return : None

Example : HemLCD_Init();

Notice : It's the first function to call at the beginning of your Code to initialize the LinCam turret

```
char HemLCD_Read_Version( void )
```

Goal : Read the firmware version (OS) of the LCD turret connected on the I²C Bus

Parameter : None

Return : Unsigned int8 (« Firmware » version number)

Example : NumVer = HemLCD_Read_Version();

Notice : This function can be used to verify the presence of the turret. If 0xFF is returned then no turret is plugged..

```
void HemLCD_Set_Backlight( unsigned char Value )
```

Goal : Define the back light intensity of the screen.

Parameter : Unsigned int8 (0 to 255 = 0 to 100%)

Return : None

Example : HemLCD_Set_Backlight(20); define the back light to 20 (= 8%)

```
void HemLCD_Set_Contrast( unsigned char Value )
```

Goal : Define the contrast level of the display.

Parameter : Unsigned int8 (0 to 255 = 0 to 100% of the maximal contrast)

Return : None

Example : HemLCD_Set_Contrast(200); define the contrast to 200 (= 80%)

Notice : below 150, the characters display are no more visible with a too small back light. The contrast value must adapt according to the back light.

```
void HemLCD_Clear_Screen( void )
```

Goal : Clear the screen

Parameter : None

Return : None

Example : HemLCD_Clear_Screen(); clear all the word display on the screen.

void HemLCD_Clear_Line1(void)

Goal : Clear the line 1

Parameter : None

Return : None

Example : HemLCD_Clear_Line1();

void HemLCD_Clear_Line2(void)

Goal : Clear the line 2

Parameter : None

Return : None

Example : HemLCD_Clear_Line2();

void HemLCD_Read_Interruptors(void)

Goal : Read the value of the 3 push-button.

Parameter : None

Return : None

Example : HemLCD_Read_Interruptors ();

Notice : The state of the 3 push-buttons are put in the variables SW1, SW2 and SW3.

void HemLCD_Line1_Left(char* Data)

Goal : Display the line 1 aligned on the left.

Parameter : Table of character, use the table of the Line1

Return : None

Example : strcpy(Line1, "MENU"); define the content of the Line1 table.

HemLCD_Line1_Left(Line1); Display the « MENU » text aligned on the left of the Line 1.

Notice : You must use the predefined characters table of Line1 and Line2.

void HemLCD_Line1_Centered(char* Data)

Goal : Display the Line1 centered

Parameter : Table of character, use the table of the Line1

Return : None

Example : strcpy(Line1, "MENU"); define the content of the Line1 table.

HemLCD_Line1_Centered(Line1); Display the « MENU » text centered of the Line 1.

Notice : You must use the predefined characters table of Line1 and Line2.

void HemLCD_Line1_Right(char* Data)

Goal : Display the line 1 aligned on the right

Parameter : Table of character, use the table of the Line1

Return : None

Example : strcpy(Line1, "MENU"); define the content of the Line1 table.

HemLCD_Line1_Right(Line1); Display the « MENU » text aligned on the right of the Line 1.

Notice : You must use the predefined characters table of Line1 and Line2.

void HemLCD_Line2_Left(**char*** Data)

Goal : Display the line 2 aligned on the left.

Parameter : Table of character, use the table of the Line2

Return : None

Example : strcpy(Line2, "MENU"); define the content of the Line2 table.

HemLCD_Line2_Left(Line2); Display the « MENU » text aligned on the left of the Line 2.

Notice : You must use the predefined characters table of Line1 and Line2.

void HemLCD_Line2_Centered(**char*** Data)

Goal : Display the Line2 centered

Parameter : Table of character, use the table of the Line2

Return : None

Example : strcpy(Line2, "MENU"); define the content of the Line2 table.

HemLCD_Line2_Centered(Line2); Display the « MENU » text centered of the Line 2.

Notice : You must use the predefined characters table of Line1 and Line2.

void HemLCD_Line2_Right(**char*** Data)

Goal : Display the line 2 aligned on the right

Parameter : Table of character, use the table of the Line2

Return : None

Example : strcpy(Line2, "MENU"); define the content of the Line2 table.

HemLCD_Line2_Right(Line2); Display the « MENU » text aligned on the right of the Line 2.

Notice : You must use the predefined characters table of Line1 and Line2.

8.7 Functions of the « Gripper» module

void HemGripper_Init(**void**)

Goal : Initialize the turret

Parameter : None

Return : None

Example : HemGripper_Init();

Notice : It's the first function to call at the beginning of your Code to initialize the Gripper turret

char HemGripper_Read_Version(**void**)

Goal : Read the firmware version (OS) of the Gripper turret connected on the I²C Bus

Parameter : None

Return : Unsigned int8 (« Firmware » version number)

Example : NumVer = HemGripper_Read_Version();

Notice : This function can be used to verify the presence of the turret. If 0xFF is returned then no turret is plugged..

Unsigned int16 HemGripper_Read_Arm_Position(**void**)

Goal : Read the position of the Arm (0-7000)

Parameter : None

Return : Unsigned int16

Example : Position = HemGripper_Read_Arm_Position();

Unsigned int16 HemGripper_Read_Gripper_Position(**void**)

Goal : Read the position of the Gripper (0-5100)

Parameter : None

Return : Unsigned int16

Example : Position = HemGripper_Read_Gripper_Position();

Unsigned int16 HemGripper_Read_Arm_Consign(**void**)

Goal : Read the position consign of the Arm (0-7000)

Parameter : None

Return : Unsigned int16

Example : Consign = HemGripper_Read_Arm_Consign();

Unsigned int16 HemGripper_Read_Gripper_Consign(**void**)

Goal : Read the position consign of the Gripper (0-5100)

Parameter : None

Return : Unsigned int16

Example : Consign = HemGripper_Read_Gripper_Consign();

Unsigned char HemGripper_Read_Arm_Speed(**void**)

Goal : Read the speed of the Arm used for the consign movements (0-255)

Parameter : None

Return : Unsigned char

Example : Speed = HemGripper_Read_Arm_Speed();

Unsigned char HemGripper_Read_Gripper_Speed(void)

Goal : Read the speed of the Gripper used for the consign movements (0-255)

Parameter : None

Return : Unsigned char

Example : Speed = HemGripper_Read_Gripper_Speed();

Unsigned char HemGripper_Read_Arm_Disable(void)

Goal : Read the flag which indicate if the Arm must be turn off or not when a stable position is

reach

Parameter : None

Return : Unsigned char

Example : Flag = HemGripper_Read_Arm_Disable();

Unsigned char HemGripper_Read_Gripper_Disable(void)

Goal : Read the flag which indicate if the Gripper must be turn off or not when the open position is reach

Parameter : None

Return : Unsigned char

Example : Flag = HemGripper_Read_Gripper_Disable();

Unsigned int16 HemGripper_Read_Battery_Voltage(void)

Goal : Read the Voltage level of the battery

Parameter : None

Return : Unsigned int16

Example : Voltage = HemGripper_Read_Battery_Voltage();

Unsigned char HemGripper_Read_Battery_Capacity(void)

Goal : Read the value of the available remaining capacity of the Battery

Parameter : None

Return : Unsigned char

Example : Capacity = HemGripper_Read_Battery_Capacity();

void HemGripper_Set_Arm_Position(unsigned int16 value)

Goal : Set the Arm position to reach without speed control

Parameter : unsigned int16

Return : None

Example : HemGripper_Set_Arm_Position(1500);

void HemGripper_Set_Gripper_Position(unsigned int16 value)

Goal : Set the Gripper position to reach without speed control

Parameter : unsigned int16

Return : None

Example : HemGripper_Set_Gripper_Position(1500);

void HemGripper_Set_Arm_Consign(**unsigned int16** value)

Goal : Set the Arm position to reach using the speed defined with the KJunior_Set_Arm_Speed() function.

Parameter : unsigned int16

Return : None

Example : HemGripper_Set_Arm_Consign(500);

void HemGripper_Set_Gripper_Consign(**unsigned int16** value)

Goal : Set the Gripper position to reach using the speed defined with the KJunior_Set_Gripper_Speed() function.

Parameter : unsigned int16

Return : None

Example : HemGripper_Set_Gripper_Gripper(1500);

void HemGripper_Set_Arm_Speed(**unsigned char** value)

Goal : Configure the speed to use for a Arm movement with speed control

Parameter : unsigned char

Return : None

Example : HemGripper_Set_Arm_Speed(100);

void HemGripper_Set_Gripper_Speed(**unsigned char** value)

Goal : Configure the speed to use for a Gripper movement with speed control

Parameter : unsigned char

Return : None

Example : HemGripper_Set_Gripper_Speed(50);

void HemGripper_Arm_Disable(**unsigned char** value)

Goal : Configure the flag to define if the Arm must turn off its motor when a stable Position is reach.

Parameter : unsigned char

Return : None

Example : HemGripper_Arm_Disable(0);

void HemGripper_Gripper_Disable(**unsigned char** value)

Goal : Configure the flag to define if the Gripper must turn off its motor when a stable Position is reach.

Parameter : unsigned char

Return : None

Example : HemGripper_Gripper_Disable(1);

APPENDIX

All the described functions in this manual are summarised in the « KJuniorCFunctionsGlossary.rtf » file in a RTF format available in the « *KJuniorOS/KJuniorOS_Sources* » directory of the Cd-rom « Professor Installation » which you have copy in your PC.

You can easily choose the needed functions and copy them to your code (See the C compilation tutorial in the « Programming tools » section of the CD-Rom.

If you have purchased the K-Junior Didapack, please use the EMP given with the pack. A tutorial will help you to realize your first C language Software.



Y-Parc - Rue Galilé 9
YVERDON LES BAINS 1400
SWITZERLAND
www.k-team.com

Tel : +41 (24) 423 89 50
Fax : +41 (24) 423 89 60