e-Puck Range & Bearing Board

Álvaro Gutiérrez & Alexandre Campo

November 13, 2009

Contents

1 Introduction

We have designed and built a new open hardware/software board that lets miniaturized robots communicate and at the same time obtain the range and bearing of the source of emission. The open *E-puck Range & Bearing* board improves an existing infrared relative localization/communication software library (*libIrcom*)[1] developed for the *e-puck* robot[2] and based on its on-board infrared sensors. The board allows the robots to have an embodied, decentralized and scalable communication system.

1.1 What you can do with it

With the *E-puck Range & Bearing* board you are able to have local communication between robots. It is based on infrared communication, so a direct vision is needed between two robots communicating. The board offers many different features. For example, a emitting robot transmit a 16 bits data frame. The robots which receive the frame extract the 16 bits data and calculates the distance (range) and orientation (bearing) to the emitter robot. Moreover, the board allows you to modify the range of transmission from 0 cm to 80 cm [3]. Therefore, you can tune the communication range according to your experiment needs. Finally, because of the existence of many different extension modules for the *e-puck* robot, we have implemented two different communication buses (I2C and UART), with which you are able to communicate the board and the robot.

1.2 How to get it work

Getting the *E-puck Range & Bearing* board working is very simple. You need to plug the *E-puck Range & Bearing* board on the robot. You can do it either on the main robot board or any other extension board which has the two main black connectors on the top. Once you have done it, you are ready to work with the *E-puck Range & Bearing* board. We suggest you to take a look at Section 6 were you will find some test examples.

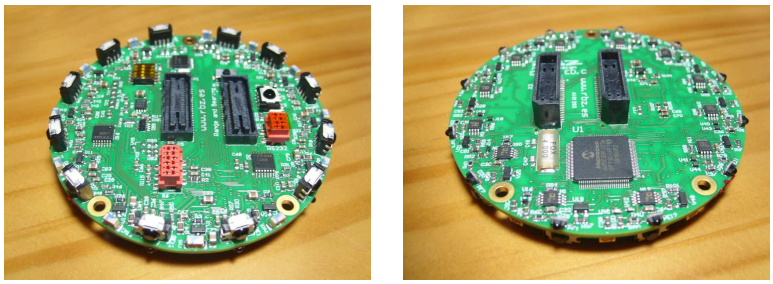
1.3 Where to get it

Actually, RBZ Robot Design is the company who is selling the *E-puck Range & Bearing* board. However, you are able to build the *E-puck Range & Bearing* board by yourself. Go to the official *e-puck* website where you will find all the information about it.

[1] http://www.e-puck.org/index.php?option=com_content&task=view&id=62&Itemid=89
[2] http://www.e-puck.org
[3] These values depend on the light conditions.

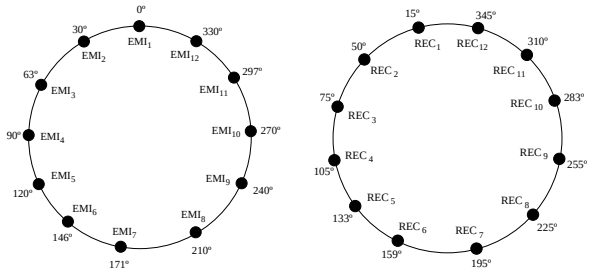
2 Hardware

The designed *E-puck Range & Bearing* board (see Figure 1) is controlled by its own processor. Each board includes 12 sets of IR emission/reception modules. Each of these modules is equipped with one infrared emitting diode, one infrared modulated receiver and one infrared photodiode[4]: The modules, as shown in Figure 2, are nearly uniformly distributed on the perimeter of the board; so, the distance between them is approximately 30°.



(a) (b)

Figure 1: (a) Top and (b) bottom view of the *E-puck Range & Bearing* board.



(a) (b)

Figure 2: (a) Emitters and (b) receivers distribution around the perimeter of the board.

For the correct understanding of the localization and communication system and its replication or modification possibilities, the forthcoming subsections detail the different hardware modules implemented in the board.

2.1 Power Supply Module

The board can be powered from 2.5 V to 6 V. Once the board is switched on, three isolated power lines are created: one for the digital system, one for the

[4]For an exhaustive description of the board see http://www.rbz.es/randb/.

analog and the last one for the emission module. The three power lines are obtained from two different supplies.

The first power supply is in charge of the emission module. This supply is based on a low dropout linear regulator which allows a voltage variation between 0.8 V and 3.46 V. This power variation lets the board modify its emission range. The regulator is connected to a digital SPI potentiometer which varies the load of the ADJ pin modifying the output of the source. Thanks to this digital variable resistor the **emission range and power consumption can be software controlled**.

The second power supply is in charge of the rest of the electronics including the microcontroller. Analog and digital lines, both of 3.3 V, are separated and short circuited just in one point to reduce noise.

The power consumption of the board depends on the settings of the emission power supply.

2.2 Emission Module

The emission module is composed of 12 different emitters. Each sensor set is composed of a narrow beam infrared led and logic gates to create the modulation.

Communication is based on frequency modulation with data at 10 KHz over a carrier of 455 KHz (see Figure 3 for more details).

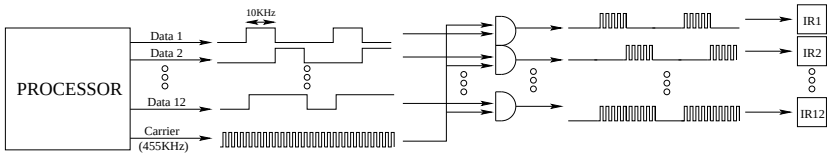


Figure 3: Emission module diagram.

The modification of the V_{emis} power supply changes the current that passes through the emitter modifying the emission range. For a minimum value of 0.8 V, a 0 cm range is achieved while the maximum range is approximately 80 cm for V_{emis}=3.46 V.

2.3 Reception Module

The reception module is divided in two different submodules. A first submodule is in charge of the data reception while the second one takes care of detecting the intensity of the signal. The division in two submodules allows the board to receive data independently of the signal intensity. In the first module, the board is able to work as a simple communication system, where the data are demodulated and received without the extraction of the emitters location. The second submodule measures the intensity of infrared signals during the reception of a

frame. To ensure a proper measure of the signal intensity, intensity and demodulating sensors must have the same orientation and are therefore positioned on top of each other.

The data reception submodule is based on a miniaturized infrared receiver for remote control. The signals are received trough digital inputs in the microcontroller.

The signal intensity submodule is based on a PIN diode and two operational amplifiers. When the photodiode starts receiving infrared signals, the circuit starts charging. The outputs from the peak detector face 12 analog to digital converters in the microcontroller.

2.4 Communication Module

The communication module has been designed to be the slave of a main processor system. Two buses, **I2C** and **RS232**, have been incorporated for facilitating the use of the board. In both communication types, the master has the control of the emission range. The modification of the power supply output can be ordered at anytime and results in an immediate modification of the emission range.

2.4.1 I2C bus

In the I2C communication, the *E-puck Range & Bearing* board acts as slave of the main processor system. The board takes care of the requests of transmission and is continuously checking for incoming frames. The main processor system polls continuously the board to check if any communication has been received.

2.4.2 UART

In the UART communication, interruptions are enabled in both directions. The master board is able to send orders of transmission or range modifications. Once a frame is demodulated by the communication board, it interrupts the master and transmits the demodulated data, the estimated angle and the distance to the emitter.

2.5 Board Configurations

2.5.1 TV Remote Receiver

Because the *E-puck Range & Bearing* was developed with the intention of removing the *e-jumper* board for those who do not want the speaker capabilities, we have replicated the IR TV Remote receiver (see Figure 4). This signal enters the robot main connectors and is attached to the same pin as the *e-jumper* board. Therefore, the receiver works in the same way and it supplants the one on the *e-jumper* board.

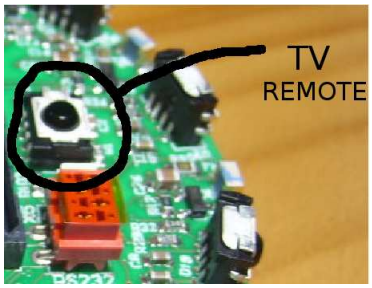
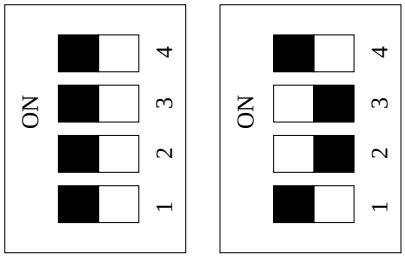


Figure 4: TV remote receiver.

2.5.2 *E-puck Range & Bearing* to *e-puck* UART communication

As aforementioned, there is the possibility of communicate the *E-puck Range & Bearing* with the *e-puck* through the UART. However, since other extension modules do also use the UART, there is a 4 positions microswitch which allows the connection/disconnection of the UART lines. In the S1 microswitch you should put PIN2 and PIN3 OFF if you want to disable this connection (see Figure 5). The connection is enabled by default.



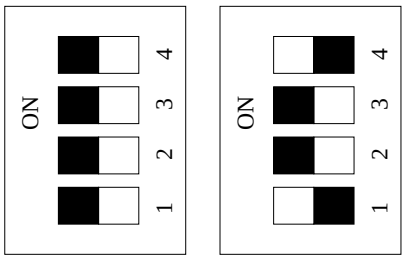
(a) (b)

Figure 5: *E-puck Range & Bearing* to *e-puck* UART communication switch configuration. (a) Communication enabled. (b) Communication disabled.

2.5.3 *E-puck Range & Bearing* to PC UART communication

We have also enable a communication with a computer. In the same way the robot is able to communicate with a PC, the board is able to do it too. However, there are some restrictions. Because of the lack of space on the board, we had used the MAX232 already provided on the robot. Therefore, we are not able to communicate a computer with both the robot and the board at the very same time. By default, the communication with the board is enabled, disabling the one with the robot. If you want to disable this option, you should put PIN1 and PIN4 of the S1 microswitch to OFF (see Figure 6). The communication is enabled by default. (The robot already has a bluetooth to communicate with a

PC). You are able to extract the signals from PIN1 (TX), PIN2 (RX) and PIN3 (GND), from the X5 4 pins micromatch red connector (see Figure 7).



(a) (b)

Figure 6: *E-puck Range & Bearing* to PC UART communication switch configuration. (a) Communication enabled. (b) Communication disabled.

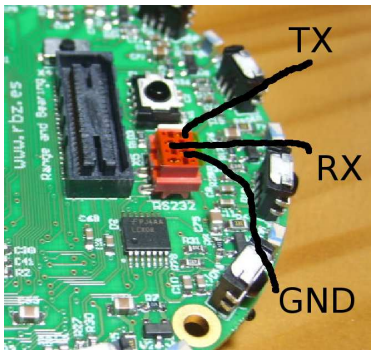


Figure 7: UART connector.

2.6 Different Versions

Actually there are 2 different versions of the board, version C and version D. They perform approximately the same, however there are some differences on the hardware.

- Version C:
  - IR Emitters have their nominal half intensity angle at ±20° (They are small and gold emitters found in the top side of the board).
- Version D
  - IR Emitters have their nominal half intensity angle at ±60° (They are the white emitters found in the top side of the board).

– IR Peak Receivers have been split. Each peak receptor module is made up of 3 photodiodes which information is combined and extracted as one sole sensor. Therefore, there are 12 sets of 3 photodiodes.

3 Installing the *E-puck Range & Bearing* Board

To install the *E-puck Range & Bearing* board, you just need to unscrew the 3 screws which join the *e-jumper* board and the *e-puck* . After it, you must unplugged the *e-jumper* board and plug the *E-puck Range & Bearing* instead. You can plug again the *e-jumper* board on the top of the *E-puck Range & Bearing* board. However, take a look at Section 7 to see the incompatibilities between the two boards.

4 Firmware

Once the board is initialized, a pulse-width modulation (PWM) timer is initialized with a period of $1.09\,\mu s$. This timer creates the carrier of the emission module which will not be stopped until the board is powered down. A Manchester code is implemented to allow any data sent at a certain distance to be received with the same intensity by the receiver. The timer, which takes care of the modulated signal, interrupts each $100\,\mu s$. The implementation of the Manchester code allows a maximum data rate of 5 kbps. Each interruption of this timer takes the buffered data and sends it to the hardware gates for its transmission. Data for transmission is stored in a buffer correctly structured according to the hardware pinout. Three different types of transmission can be asked to the communication board:

- **All the sensors transmit the same data**: One instruction is sent to the board, along with the data to transmit.
- **Only some sensors transmit data**: One instruction for each sensor must be sent to the board. Data and sensor number must also be provided to the board. After all the sensors have been loaded, a "send" instruction must be sent to the board.
- **Different sensors transmit different data:** One instruction for each sensor must be sent to the board. Data and sensor number must also be provided to the board. After all the sensors have been loaded, a "send" instruction must be sent to the board.

Once a transmission order is sent by the master to the board, the communication module is in charge of decomposing the data for the different sensors with a preamble (6 bits), the data (16 bits) and a CRC (4 bits). If the master needs to transmit a flow of data, the communication module buffers all the messages one after the other, in a transparent manner for the transmission timer.

The reception software is continuously checking if a message arrives. Once the preamble of a frame is detected by an infrared modulated receiver, the board continues receiving the data and CRC while it is charging a peak detector through an infrared photodiode. If the frame has correctly arrived (checked by the CRC), the peak detector level is read and stored in a buffer. As the aperture of the receiving sensor is wide, it is likely that several sensors receive the same data at the same time. The information given by the different peak detectors is used to calculate the orientation and distance to the emitter. These two values are then stored in a buffer to be sent to the master board. Figure 8 shows a block diagram of the emission and reception software modules.

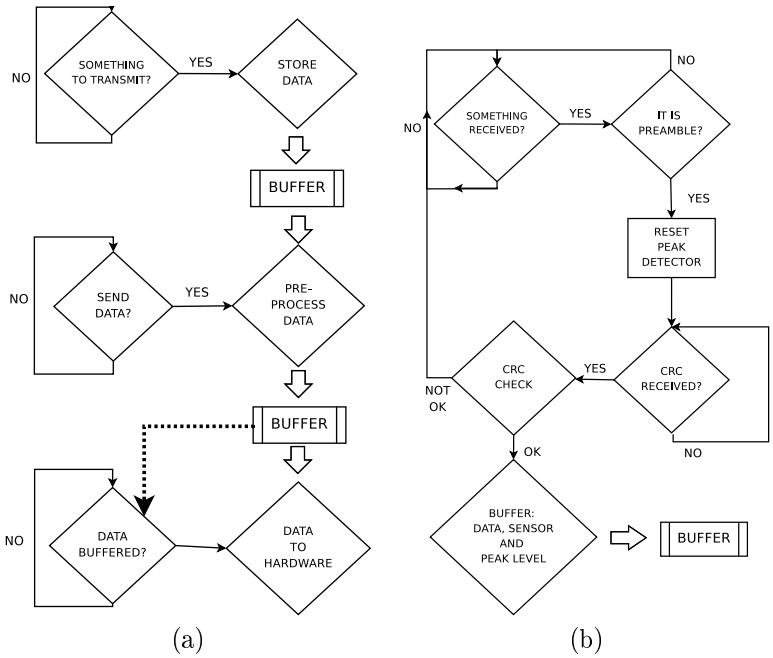


Figure 8: Block diagram of the (a) software emission module and (b) software reception module.

Infrared noise comes mainly from light conditions in the environment. To deal with it, the board measures at the beginning the infrared signal in the environment. Once a frame is correctly received, the board subtracts the environment measure from the peak receptor and returns it as the frame signal intensity.

5 e-RandB Libraries

In this section we describe the libraries created for the e-Puck to communicate with the *E-puck Range & Bearing* board.

Following the actual convention of the code we have placed the libraries in the *src/Epfl* directory. In this directory we find the libraries in *e_randb.h* and *e_randb.c*. We clearly differentiate 3 parts inside the library: The functions related to the I2C, the functions related to the UART and the functions common to both. In what follows we describe all of them.

5.1 Common Functions

5.2 *e_init_randb*

Description	This function initializes the **I2C** or **UART** communications. It is mandatory to initialize the board before using it on the examples.
Prototype	*void e_init_randb (unsigned char mode)*
Arguments	*unsigned char mode*: This defines if the board is communicated through the I2C (0) or the UART (1).
Returns	*void*

5.3 I2C Functions

5.3.1 *e_randb_set_range*

Description	Sets the range of transmission
Prototype	*void e_randb_set_range (unsigned char distance)*
Arguments	*unsigned char distance*: Defines the distance of transmission. 0 –> Full Range (1 m. aprox., depending on light conditions) 255 –> Shortest Range (0 cm aprox., depending on light conditions)
Returns	*void*

5.3.2 *e_randb_set_calculation*

Description	At some point we thought that the board could just take data and leave the calculations for the robot. At the moment, it is better to allow the board to do the calculations.
Prototype	*void e_randb_set_calculation (unsigned char value)*
Arguments	*unsigned char value*: Defines if the calcs are made on the board (0) or the robot (1).
Returns	*void*

5.3.3 *e_randb_store_light_conditions*

Description	Store light conditions to use them as offset for the calculations of the range and bearing.
Prototype	*void e_randb_store_light_conditions (void)*
Arguments	*void*
Returns	*void*

5.3.4 *e_randb_store_data*

Description Tells the board to store some data "data" to be sent later by sensor "channel". We are able to store different data for different sensors.

Prototype *void e_randb_store_data (unsigned char channel , unsigned int data)*

Arguments

- *unsigned char channel*: The sensor which will send the data.
- *unsigned int data*: The data to be sent by the sensor.

Returns *void*

5.3.5 *e_randb_send_data*

Description Tells the board to send all the data stored previously with e_randb_store_data. Because different data could have been stored for different sensors, all the different data can be sent by different sensors at the very same time.

Prototype *void e_randb_send_data (void)*

Arguments *void*

Returns *void*

5.3.6 *e_randb_send_all_data*

Description Tells the board to send the data "data" through all the sensors

Prototype *void e_randb_send_all_data (unsinged int data)*

Arguments *unsigned int data*: The data to be sent by all the sensors

Returns *void*

5.3.7 *e_randb_get_if_received*

Description Checks if a frame has been received on the board.

Prototype *unsigned char e_randb_get_if_received(void)*

Arguments *void*

Returns *unsigned char*: 0 if nothing received or 1 if a frame has been received.

5.3.8 *e_randb_get_data*

Description If a frame has been received returns data.

Prototype *unsigned int e_randb_get_data(void)*

Arguments *void*

Returns *unsigned int*: Data received

5.3.9 *e_randb_get_range*

Description	If a frame has been received returns the range.
Prototype	*unsigned int e_randb_get_range(void)*
Arguments	*void*
Returns	*unsigned int*: Range to emitter.

5.3.10 *e_randb_get_bearing*

Description	If a frame has been received returns bearing.
Prototype	*double e_randb_get_bearing(void)*
Arguments	*void*
Returns	*double*: Bearing to emitter.

5.3.11 *e_randb_get_sensor*

Description	If a frame has been received returns the sensor which has received maximum intensity.
Prototype	*unsigned char e_randb_get_sensor (void)*
Arguments	*void*
Returns	*unsigned char*: The sensor with maximum intensity received.

5.4 UART Functions

5.4.1 *e_randb_set_uart_communication*

Description	Tells the board that communication is through the UART.
Prototype	*void e_randb_set_uart_communication (unsigned char mode)*
Arguments	*unsigned char mode*: Must be 1.
Returns	*void*

5.4.2 *e_randb_uart_set_range*

Description	Sets the range of transmission
Prototype	*void e_randb_uart_set_range (unsigned char distance)*
Arguments	*unsigned char distance*: Defines the distance of transmission. 0 –> Full Range (1 m. aprox., depending on light conditions) 255 –> Shortest Range (0 cm aprox., depending on light conditions)
Returns	*void*

5.4.3 *e_randb_uart_set_calculation*

Description At some point we thought that the board could just take data and leave the calculations for the robot. At the moment, it is better to allow the board to do the calculations.

Prototype *void e_randb_uart_set_calculation (unsigned char value)*

Arguments *unsigned char value*: Defines if the calcs are made on the board (0) or the robot (1).

Returns *void*

5.4.4 *e_randb_uart_store_light_conditions*

Description Store light conditions to use them as offset for the calculations of the range and bearing.

Prototype *void e_randb_uart_store_light_conditions (void)*

Arguments *void*

Returns *void*

5.4.5 *e_randb_uart_store_data*

Description Tells the board to store some data "data" to be sent later by sensor "channel".

Prototype *void e_randb_uart_store_data (unsigned char channel , unsigned int data)*

Arguments

- *unsigned char channel*: The sensor which will send the data.
- *unsigned int data*: The data to be sent by the sensor.

Returns *void*

5.4.6 *e_randb_uart_send_data*

Description Tells the board to send the data stored previously with e_randb_store_data.

Prototype *void e_randb_uart_send_data (void)*

Arguments *void*

Returns *void*

5.4.7 *e_randb_uart_send_all_data*

Description Tells the board to send the data "data" through all the sensors
Prototype *void e_randb_uart_send_all_data (unsigned int data)*
Arguments *unsigned int data*: The data to be sent by all the sensors
Returns *void*

5.4.8 *e_randb_get_data_uart*

Description If data received returns struct with the data, range, bearing, max_peak and max_sensor of the frame received.
Prototype *unsigned char e_randb_get_data_uart (finalDataRegister* data)*
Arguments *finalDataRegister* data*: Pointer to a struct where the information received and calculated will be loaded (data, range. bearing, max_peak and max_sensor). The finalDataRegister is defined as follows:

```
typedef struct {
        unsigned int data;
        float bearing;
        unsigned int range;
        unsigned int max\_peak;
        unsigned char max\_sensor;
} finalDataRegister;
```

Returns *void*

5.4.9 *e_randb_get_uart2*

Description State machine in charged of getting the information from the board through UART interruptions. It is started and managed by the agenda, therefore the controller should not access.
Prototype *void e_randb_get_uart2 (void)*
Arguments *void*
Returns *void*

6 e-Puck Examples

We have developed some examples to test the boards. We hope they are OK for the correct understanding of the board working. We provide the examples code on this manual, however you should download the code and libraries from XXX. We have commented the code for its clarification.

6.1 I2C Emission/Reception

6.1.1 I2Cemitter

This file is the *main.c* of randbEmitter directory. This code should be loaded into a robot which will be the emitter robot. The emitter robot will send frames from 0 to 65535 with non-stop through the emitter sensor number 0. Once the data to be sent reaches 65535 it is reseted to 0, and the robot continues sending frames.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <btcom.h>
#include <e_randb.h>
#include <e_prox.h>
#include <e_uart_char.h>
#include <e_init_port.h>

int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(I2C);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_set_range(0);

        /* At some point we 06 that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        e_randb_set_calculation(ON_BOARD);

        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_store_light\_conditions();
        /* Print on the bluetooth */
        char tmp2[50];
        sprintf(tmp2,"I2C EMITTER\n");
```

```
        btcomSendString(tmp2);

        /* The counter for sending */
        unsigned int data = 0;

        while(1)
        {
                /* Send the data through sensor 0*/
                e_randb_store_data(0,data);
                e_randb_send_data();

                /* Increase data */
                data++;

                /* Data to be sent must be lower than 16 bits */
                if(data==65535) data=0;
        }
        return 0;
}
```

6.1.2 I2Creceiver

This file is the *main.c* of randbReceiver directory. This code should be loaded into a robot which will be the receiver robot. The receiver robot is checking for a frame to be received. Once it is received the robot gets the data, range and bearing and prints it into the bluetooth interface.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <btcom.h>
#include <e_randb.h>
#include <e_prox.h>
#include <e_uart_char.h>
#include <e_init_port.h>

#define M_PI 3.141593
int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(I2C);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
```

```
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_set_range(150);

        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        e_randb_set_calculation(ON_BOARD);

        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_store_light_conditions();

        /* Print on the bluetooth */
        char tmp2[50];
        sprintf(tmp2,"I2C RECEIVER\n");
        btcomSendString(tmp2);

        /* The counter for sending */
        unsigned int data = 0;
        while(1)
        {
                /* If something received */
                if(e_randb_get_if_received() != 0)
                {
                        /* Get data */
                        unsigned int data = e_randb_get_data();
                        /* Get bearing */
                        double bearing = e_randb_get_bearing();
                        /*Get range */
                        unsigned int range = e_randb_get_range();

                        /* Print data received on the bluetooth */
                        sprintf(tmp2,"%u %2f %u", data, (bearing*180/M_PI), range);
                        btcomSendString(tmp2);
                        btcomSendString("\n");
                }
        }
        return 0;
}
```

6.1.3 Testing

You must load I2Cemitter code to one robot and I2Creceiver code to a second robot. As you have seen in the code, the robots are waiting for a command

's' to be received through the bluetooth. You can use the *ircomTest* program provided with the examples code on the *pc* directory, or create your own code to send this command. In any case, you need to have a bluetooth communication where the robots will print the messages.

If you use the *ircomtTest* program provided you need to map your robot MAC address in the */etc/bluetooth/rfcomm.conf*. For example, open */etc/bluetooth/rfcomm.conf* and add

rfcomm19{device 10:00:11:06:E5:34;}

Now type:

./ircomtTest 19.

You must do the same with the second robot. After executing the command, the program is waiting for the return key to be pressed. When pressed the program will send the command 's' and the robot should start working. You should start the receiver robot first. It will print "I2C RECEIVER" and wait for frames to be received. If you start now the second robot, it should print "I2C EMITTER". If once the emitter robot is initialized, you point emitter sensor 0 (the one in the front) of the emitter robot to the receiver robot, the later will print the data, range and bearing on the screen.

6.2 UART Emission/Reception

6.2.1 UARTemitter

This file is the *main.c* of randbEmitterUart directory. This code should be loaded into a robot which will be the emitter robot. The emitter robot will send frames from 0 to 65535 with non-stop through emitter sensor 0. Once the data sent arrives to 65535 it is reseted to 0, and the robot continues sending frames.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <btcom.h>
#include <e_randb.h>
#include <e_agenda.h>
#include <e_prox.h>

#include <e_uart_char.h>
#include <e_init_port.h>

int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
```

```
        /* Init E-RANDB board */
        e_init_randb(UART);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_uart_set_range(0);

        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        e_randb_uart_set_calculation(ON_BOARD);

        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_uart_store_light_conditions();

        /* Print on the bluetooth */
        char tmp2[50];
        sprintf(tmp2,"UART EMITTER\n");
        btcomSendString(tmp2);

        /* The counter for sending */
        unsigned int data = 0;

        while(1)
        {
                /* Send the data through sensor 0*/
                e_randb_uart_store_data(0,data);
                e_randb_uart_send_data();

                /* Increase data */
                data++;

                /* Data to be sent must be lower than 16 bits */
                if(data==65535) data=0;
        }
        return 0;
}
```

6.2.2 UARTreceiver

This file is the *main.c* of randbReceiverUart directory. This code should be loaded into a robot which will be the receiver robot. The receiver robot is

checking for a frame to be received. Once it is received the robot prints the data, range and bearing into the bluetooth interface.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include <e_uart_char.h>
#include <e_init_port.h>
#include <e_led.h>
#include <e_randb.h>
#include <e_agenda.h>
#include <e_prox.h>

#include <btcom.h>

#define M_PI 3.141593
int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(UART);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_set_range(150);

        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        e_randb_set_calculation(ON_BOARD);

        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_store_light_conditions();

        /* Print on the bluetooth */
        char tmp[20];
        sprintf(tmp,"UART RECEIVER\n");
        btcomSendString(tmp);
```

```
        finalDataRegister data;
        while(1)
        {
                if (e_randb_get_data_uart(&data))
                {
                        sprintf(tmp,"%d %d %d\n",data.data,
                                (int) (data.bearing * 180.0 / M_PI), data.range);
                        btcomSendString(tmp);
                }
        }
        return 0;
}
```

6.2.3 Testing

You must load UARTemitter code to one robot and UARTreceiver code to a second robot. As you have seen in the code, the robots are waiting for a command 's' to be received through the bluetooth. You can use the *ircomTest* program provided with the examples code on the *pc* directory, or create your own code to send this command. In any case, you need to have a bluetooth communication where the robots will print the messages.

If you use the *ircomtTest* program provided you need to map your robot MAC address in the */etc/bluetooth/rfcomm.conf*. For example, open */etc/bluetooth/rfcomm.conf* and add

rfcomm19{device 10:00:11:06:E5:34;}

Now type:

./ircomtTest 19.

You must do the same with the second robot. After executing the command, the program is waiting for the return key to be pressed. When pressed the program will send the command 's' and the robot should start working. You should start the receiver robot first. It will print "UART RECEIVER" and wait for frames to be received. If you start now the second robot, it should print "UART EMITTER". If once the emitter robot initialized, you point emitter sensor 0 (the one in the front) to the receiver robot, it will print the data, range and bearing on the screen.

6.3 UART Emission/Reception Through All Sensors

6.3.1 UARTemitter

This file is the *main.c* of randbEmitterUartAll directory. This code should be loaded into a robot which will be the emitter robot. The emitter robot will send frames from 0 to 65535 with non-stop through **all emitter sensors**. Once the

data sent arrives to 65535 it is reseted to 0, and the robot continues sending frames.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <btcom.h>
#include <e_randb.h>
#include <e_agenda.h>

#include <e_uart_char.h>
#include <e_init_port.h>

int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(UART);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_uart_set_range(150);

        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        e_randb_uart_set_calculation(ON_BOARD);

        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_uart_store_light_conditions();

        /* Print on the bluetooth */
        char tmp2[50];
        sprintf(tmp2,"ALL SENSORS EMITTER\n");
        btcomSendString(tmp2);

        /* The counter for sending */
        unsigned int data = 0;
```

```
        while(1)
        {
                /* Send the data through all the sensors */
                e_randb_uart_send_all_data(data);

                /* Increase data */
                data++;

                /* Data to be sent must be lower than 16 bits */
                if(data==65535) data=0;
        }
        return 0;
}
```

6.3.2 Testing

You should use the UARTReceiver or I2CReceiver code as receiver. The program testing is solved as in the previous examples.

6.4 I2C Frame Rate

We have observed the computer takes time printing the information provided by the robot. Therefore, the bluetooth communication delays the reception and the buffer gets full. We have create two more examples to test the frame rate achieved by the *E-puck Range & Bearing* board. In this case, we are testing the I2C frame rate. The robot will receive the frames but will not print them Instead, we have started a timer on the robots which will count the frames received and print the number each 10 seconds. We will observe the frame rate achieved with the I2C communication is something between 30-40 messages per second.

6.4.1 I2C Frame Rate Emitter

This file is the *main.c* of randbEmitterTiming directory. This code should be loaded into a robot which will be the emitter robot. The emitter robot will send frames from 0 to 65535 with non-stop through emitter sensor 0. Once the data sent arrives to 65535 it is reseted to 0, and the robot continues sending frames. The robot prints out the number of frames sent each 10 seconds.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <btcom.h>
#include <e_randb.h>
#include <e_epuck_ports.h>
```

```
#include <e_uart_char.h>
#include <e_init_port.h>

#define SAMPLING_TIME 10 /* milliseconds*/
long int m_uTime;

void initTimer ( void )
{
        T3CON = 0;                      //
        T3CONbits.TCKPS = 3;            // prescsaler
        TMR3 = 0;                       // clear timer 3
        PR3 = (SAMPLING_TIME * MILLISEC)/256.0;
        IFS0bits.T3IF = 0;              // clear interrupt flag
        IEC0bits.T3IE = 1;              // set interrupt enable bit
        IPC1bits.T3IP = 6;              // timer3 priority. Priority set to 6
        T3CONbits.TON = 1;              // start Timer3

        m_uTime = 0;
}

void _ISRFAST _T3Interrupt(void)
{
        /* clear interrupt flag */
        IFS0bits.T3IF = 0;

        m_uTime++;
}

int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(I2C);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_set_range(0);

        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
```

```
        e_randb_set_calculation(ON_BOARD);
        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_store_light_conditions();

        /* Print on the bluetooth */
        char tmp2[50];
        sprintf(tmp2,"I2C EMITTER Timming\n");
        btcomSendString(tmp2);

        /* The counter for sending */
        unsigned int data = 0;

        int counter=0;

        /* Init time to count time */
        initTimer();

        while(1)
        {
                /*if ((m_uTime%100) == 0)*/
                /*{*/
                if((m_uTime%1000)==0)
                {
                        sprintf(tmp2,"%d", counter);
                        btcomSendString(tmp2);
                        btcomSendString("\n");
                        counter = 0;
                }

                /* Send data to one sensor */
                e_randb_store_data(0,data);
                e_randb_send_data();

                /* Increase the data */
                data++;
                counter++;

                /* Data to be sent must be lower than 16 bits */
                if(data==65535) data=0;
        }
        return 0;
}
```

6.4.2 I2C Frame Rate Receiver

This file is the *main.c* of randbReceiverTiming directory. This code should be loaded into a robot which will be the receiver robot. The receiver robot is checking for a frame to be received. Once it is received the robot increases a

counter. After 10 seconds the robot prints the number of frames received.

```
#include <string.h>
#include <math.h>
#include <btcom.h>
#include <e_randb.h>
#include <e_epuck_ports.h>

#include <e_uart_char.h>
#include <e_init_port.h>

#define M_PI 3.141593
#define SAMPLING_TIME 10 /* milliseconds*/

long int m_uTime;

void initTimer ( void )
{
        T3CON = 0;                      //
        T3CONbits.TCKPS = 3;            // prescsaler
        TMR3 = 0;                       // clear timer 3
        PR3 = (SAMPLING_TIME * MILLISEC)/256.0;
        IFS0bits.T3IF = 0;              // clear interrupt flag
        IEC0bits.T3IE = 1;              // set interrupt enable bit
        IPC1bits.T3IP = 6;              // timer3 priority. Priority set to 6
        T3CONbits.TON = 1;              // start Timer3

        m_uTime = 0;
}

void _ISRFAST _T3Interrupt(void)
{
        /* clear interrupt flag */
        IFS0bits.T3IF = 0;

        m_uTime++;
}

int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(I2C);

        /* Wait for a command coming from bluetooth IRCOMTEST on pc directory*/
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
```

```
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_set_range(150);

        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        e_randb_set_calculation(ON_BOARD);
        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_store_light_conditions();

        /* Print on the bluetooth */
        char tmp2[50];
        sprintf(tmp2,"I2C RECEIVER Timing\n");
        btcomSendString(tmp2);

        /* Init timer to count time */
        initTimer();

        int counter=0;
        while(1)
        {
                if((m_uTime%1000)==0)
                {
                        sprintf(tmp2,"%d", counter);
                        btcomSendString(tmp2);
                        btcomSendString("\n");
                        counter = 0;
                }

                /* If something received */
                if(e_randb_get_if_received() != 0)
                {
                        /* Get data */
                        unsigned int data = e_randb_get_data();
                        /* Get bearing */
                        double bearing = e_randb_get_bearing();
                        /*Get range */
                        unsigned int range = e_randb_get_range();

                        counter++;
                }
        }
        return 0;
}
```

6.4.3 Testing

The program testing is solved as in the previous examples.

6.5 UART Frame Rate

This test checks the frame rate achieved when a UART communication is established. We will observe the frame rate achieved with the UART communication is something between 120-130 messages per second. The increase observed with respect to the I2C is because the robot is obtaining the information from the board by means of interruptions and it has not to be asking the board continuously.

6.5.1 UART Frame Rate Receiver

This file is the *main.c* of randbReceiverUartTiming directory. This code should be loaded into a robot which will be the receiver robot. The receiver robot is checking for a frame to be received. Once it is received the robot increases a counter. After 10 seconds the robot prints the number of frames received.

```
#include <p30f6014a.h>
#include <stdio.h>
#include <string.h>
#include <math.h>

#include <e_uart_char.h>
#include <e_init_port.h>
#include <e_led.h>
#include <e_randb.h>
#include <e_agenda.h>
#include <e_epuck_ports.h>

#include <btcom.h>

#define SAMPLING_TIME 10 /* milliseconds*/

long int m_uTime;

void initTimer ( void )
{
        T3CON = 0;                      //
        T3CONbits.TCKPS = 3;            // prescsaler
        TMR3 = 0;                       // clear timer 3
        PR3 = (SAMPLING_TIME * MILLISEC)/256.0;
        IFS0bits.T3IF = 0;              // clear interrupt flag
        IEC0bits.T3IE = 1;              // set interrupt enable bit
        IPC1bits.T3IP = 6;              // timer3 priority. Priority set to 6
        T3CONbits.TON = 1;              // start Timer3
```

```
        m_uTime = 0;
}

void _ISRFAST _T3Interrupt(void)
{
        /* clear interrupt flag */
        IFS0bits.T3IF = 0;

        m_uTime++;
}
int main() {

        /* System Initialization */
        e_init_port();
        /* Init UART1 for bluetooth */
        e_init_uart1();
        /* Init E-RANDB board */
        e_init_randb(UART);

        /* Wait for a command coming from bluetooth */
        /* Important issue when we are transmitting data and don't want
         * the bluetooth stack stuck for programing mode */
        btcomWaitForCommand('s');

        /* Range is tunable by software.
         * 0 -> Full Range (1m. approx depending on light conditions )
         * 255 --> No Range (0cm. approx, depending on light conditions */
        e_randb_set_range(150);
        /* At some point we thought that the board could just take
         * data and leave the calculations for the robot.
         * At the moment, it is better to allow the board to do the calculations */
        /*e_randb_set_calculation(ON_ROBOT);*/
        e_randb_set_calculation(ON_BOARD);

        /* Store light conditions to use them as offset for the calculation
         * of the range and bearing */
        e_randb_store_light_conditions();

        /* Print on the bluetooth */
        char tmp[50];
        sprintf(tmp,"UART RECEIVER Timing\n");
        btcomSendString(tmp);

        finalDataRegister data;

        initTimer();

        int counter=0;

        while(1)
```

```
        {
                if((m_uTime%1000)==0)
                {
                        sprintf(tmp,"%d", counter);
                        btcomSendString(tmp);
                        btcomSendString("\n");
                        counter = 0;
                }

                if (e_randb_get_data_uart(&data))
                {
                        counter++;
                }
        }
        return 0;
}
```

6.5.2 Testing

The program testing is solved as in the previous examples. You can load any emitter code on the emitter robot.

7 Problems Reported

Different problems with the *E-puck Range & Bearing* board have been reported by different groups. Please read carefully the following sections.

7.1 IR Proximity interference

Because the *E-puck Range & Bearing* board communicates through IR there are interferences when the proximity sensors are activated. If you test you examples, you will see that the data of the frames arrive perfectly, because the information is modulated. However, there will be a disruption on the range and bearing calculation. Some people is solving the issue stopping the IR proximity sensors when they are not used.

7.2 e-jumper vs. UART Communication

Some groups are detecting problems when working with the *E-puck Range & Bearing* board and the *e-jumper* board is plugged. The problem is because the *e-jumper* board is shortcircuting some lines which are used by the *E-puck Range & Bearing* board when it is on UART mode. You will be unable to communicate with the *E-puck Range & Bearing* board if you have PIN1 and PIN4 of the four positions switch (S1) ON at the same time than the *e-jumper* board is plugged. If you put them OFF, things will work. Some groups have reported that even with the S1 change, somes boards are not able to communicate through the

UART, while others do. At the time of writing we have not solved this issue, we must check where is the mystery.

8 Contact

The board has been designed by Álvaro Gutiérrez, Alexandre Campo and the RBZ Robot Design company. If you have some problems or want information about how to build or buy the boards, please contact:
Álvaro Gutiérrez: aguti@etsit.upm.es
Alexandre Campo: alexandre.acampo@ulb.ac.be