e-puck2 Library Cheat Sheet

This is a cheat sheet for e-puck2 library which is originally designed by GCtronic. For more details about e-puck2, visit: https://www.gctronic.com/doc/index.php?title=e-puck2

In this cheat sheet, the functions that can be used in your program are given in the following format:

```
return_type function_name(variable_type variable_name);
```

If the `return_type` is *not* `void`, you can store the output of the function to a variable.

If the `variable_type` is *not* `void`, you must provide the values or variables in the specified order between the brackets. If the `variable_type` is `void`, you do not need to type anything between the brackets e.g. `foo()`.

LED

The header file for the LEDs:

```
#include "leds.h"
#include "spi_comm.h"
```

Initiate and turn off all eight ring LEDs (include before the while loop):

```
void clear_leds(void);
void spi_comm_start(void);
```

Set one of the four red LEDs around the ring with `value` 0 for off, 1 for on, or 2 to toggle:

(The `led_name` parameter can be `LED1`, `LED3`, `LED5` or `LED7`)

```
void set_led(led_name_t led_number, unsigned int value);
```

Set one of the four RGB LEDs around the ring to have colour values (up to 10) for red, green and blue:

(The `rgb_led_name` parameter can be `LED2`, `LED4`, `LED6` or `LED8`)

```
void set_rgb_led(rgb_led_name_t led_number, int red_val, int green_val, int
blue_val);
```

Set the main body LED with `value` 0 for off, 1 for on, or 2 to toggle:

```
void set_body_led(unsigned int value);
```

Set the front LED with `value` 0 for off, 1 for on, or 2 to toggle:

```
void set_front_led(unsigned int value);
```

Proximity

The header file for the proximity sensors:

```
#include "sensors/proximity.h"
```

Insert the following lines below includes (outside `main`) to define the inter process communication bus:

```
messagebus_t bus;
MUTEX_DECL(bus_lock);
CONDVAR_DECL(bus_condvar);
```

Insert the following line at the beginning of the main function (but after the three preexisting lines of code) to initiate the inter process communication bus:

```
messagebus_init(&bus, &bus_lock, &bus_condvar);
```

Start the proximity measurement module (before the while loop):

```
void proximity_start(void);
```

Calibrate the proximity sensors (before the while loop):

```
void calibrate_ir(void);
```

Get the proximity reading from `sensor_number` 0-7:

```
int get_prox(unsigned int sensor_number);
```

Get the calibrated proximity reading from `sensor_number` 0-7:

```
int get_calibrated_prox(unsigned int sensor_number);
```

Get the ambient light value from `sensor_number` 0-7:

```
int get_ambient_light(unsigned int sensor_number);
```

Distance

The header file for the distance sensor:

```
#include "sensors/VL53L0X/VL53L0X.h"
```

Initiate the distance sensor module (before the while loop):

```
void VL53L0X_start(void);
```

Return the distance measured in mm:

```
uint16_t VL53L0X_get_dist_mm(void);
```

USB Communication

The header file:

```
#include "chprintf.h"
#include "usbcfg.h"
```

Initiate the USB communication peripheral (before the while loop):

```
void usb_start(void);
```

Send data over USB:

```
int chprintf(BaseSequentialStream *chp, const char *fmt);
```

Example usage for returning the proximity value of sensor 0:
NOTE: Refer to the Proximity section on how to initialise the proximity sensors.

```
// Skip printing if port not opened.
if (SDU1.config->usbp->state == USB_ACTIVE) {
    chprintf((BaseSequentialStream *)&SDU1, "%4d,", prox_values[0]);
}
```

To read data on PC Terminal:

```
sudo cat /dev/ttyACM2
```

UART - Bluetooth Communication

The header files for UART:

```
#include "epuck1x/uart/e_uart_char.h"
#include "stdio.h"
#include "serial_comm.h"
```

Initialise the UART1 channel (before the while loop):

```
void serial_start(void);
```

Send a character buffer array `buff` with length `buff_len` using UART1 channel:

```
void e_send_uart1_char(const char * buff, int buff_len);
```

Example usage for sending `"Hello World"` to the terminal:

```
char str[100];
int str_length;
str_length = sprintf(str, "Hello World\n");
e_send_uart1_char(str, str_length);
```

This code initialises a `char` array called `str` of arbitrary length 100, which will hold the data we send later, and an `int` called `str_length`. The `sprintf` command populates `str` with the string given in quotation marks and fills `str_length` with the length of that string. The `\n` characters are interpreted as a new line. This is sent over Bluetooth in the final line.

To receive data via Bluetooth from the e-puck2, you should do the following.

- Turn the e-puck2 on while holding the esp32 button to enable Bluetooth communication
- Connect to the e-puck2 through the Ubuntu settings panel, as shown Figure 1.
    - Open settings in the top-right of the screen (image 1)
    - Connect to your robot, identified by the number on the top (image 2)
    - Copy the e-puck's MAC address (image 3)
    - *NOTE: Don't worry if the e-puck is shown as "Disconnected". It will show as "Connected" once the e-puck starts sending data to the computer.*
- Use a terminal to bind to the e-puck2 with the command `sudo rfcomm bind /dev/rfcomm0 MAC_Address 2` (don't forget the `2` at the end)
- Connect and receive data with the command `sudo cat /dev/rfcomm0`. This causes the terminal to continually print whatever it receives.

NOTE: If you have issues, try to reset the connection with `sudo rfcomm release /dev/rfcomm0`

In order to print numbers in the terminal, use syntax similar to that shown below:

```
int value = 10;
str_length = sprintf(str, "Printing number %d!\n",value);
e_send_uart1_char(str, str_length);
```

This will print: `Printing number 10!`

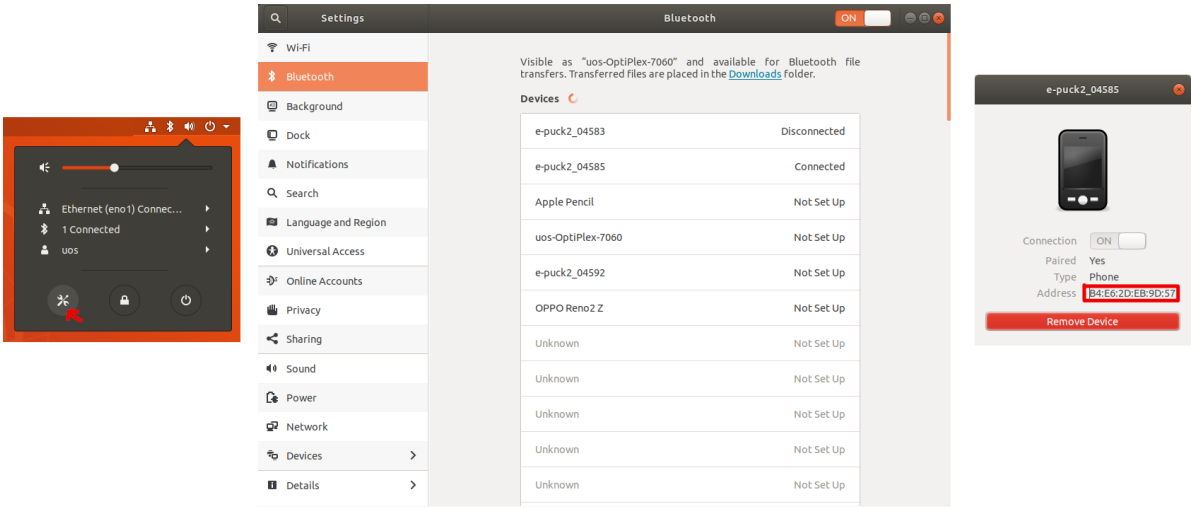

Figure 1: Ubuntu Bluetooth settings.

Motors

The header files for motors:

```
#include "motors.h"
```

The range of the motor speeds you can set is `[-1000,1000]`:

Initialise the motors (before the while loop):

```
void motors_init(void);
```

Set left and right motor speeds:

```
void left_motor_set_speed(int motor_speed);
void right_motor_set_speed(int motor_speed);
```

Get the last set motor speeds:

```
int left_motor_get_desired_speed(void);
int right_motor_get_desired_speed(void);
```

Battery

The header files for the battery sensor:

```
#include "sensors/battery_level.h"
```

Start the battery measurement service (include before the while loop):

```
void battery_level_start(void);
```

Get battery level as a percentage:

```
float get_battery_percentage(void);
```

Selector

The header file for the selector:

```
#include "selector.h"
```

Return the value from the selector knob:

```
int get_selector(void);
```

Useful for programming multiple behaviours, eg.:

```
if(get_selector() == 7) {
    do_something();
} else if(get_selector() == 13) {
    do_something_else();
}
```

Sound

The header files for speakers and audio:

```
#include "audio/audio_thread.h"
#include "audio/play_melody.h"
#include "audio/play_sound_file.h"
```

Initiate the sound peripherals (before the while loop):

```
void dac_start(void);
void playSoundFileStart(void);
```

and/or

```
void dac_start(void);
void playMelodyStart(void);
```

Play sound from a file:

```
void playSoundFile(char* pathToFile, playSoundFileOption_t option, unsigned int
freq);
```

Wait for sound file to play:

```
void waitSoundFileHasFinished(void);
```

- For example, `playSoundFile("example.wav", SF_FORCE_CHANGE, 16000)`.
- There **MUST** be a `waitSoundFileHasFinished()` after the `playSoundFile()` function.
- Note that, you may need to store your file in a micro SD memory card due to the limited memory of the robot itself.

Play melody:

```
void playMelody(song_selection_t choice, play_melody_option_t option, melody_t*
external_melody);
```

- For example, `playMelody(MARIO, ML_SIMPLE_PLAY, NULL)`.
- One can also include any other melodies instead of the given ones. For this, you need to code your melodies in the same way as those in `audio/play_melody.c`.

Wait

One can use the following `Wait` function as a delay. Time in milliseconds so `delay` of 1000 is one second

```
void chThdSleepMilliseconds(int delay);
```

Camera and other capabilities

The `e-puck2_main-processor` folder contains all libraries for the e-puck2 capabilities, with a folder called `src`. If you would like to further expand your code, take a look at the `main.c` files for examples of how to implement the features.