Programming and characterization of a modular lamprey robot

Robotic Practicals Report

Autor:
Xavier BAILLY
Marc URAN
Philipp SPIESS

Instructor:
Alessandro Crespi

27th of July 2019

Contents

1 LEDs **1**

2 Registers **1**

3 Communication with other elements **2**

4 Position control of a module **3**

5 on-board trajectory generation **4**

5.1 Simple on-board trajectory generation 4

5.2 Modulating trajectory parameters 4

Introduction

In this robotic practical we familiarized ourselves with a modular lamprey robot. A number of exercises allowed us to understand different elements of the robot. First we had to code the LEDs, then we had to use registers to do the communication between the robot and the PC. After understanding how to use the registers, we could communicate with the robot and set motor positions that we wish for each of its modules. We implemented the communication of the wished amplitude, frequency and latency of the robot's movement. Finally we used LED tracking to test the results of different parameter settings. This allowed us to assess and understand how each parameter influences the performance of the robot.

1 LEDs

The first program that was put on the robot made the LEDs change colors in a cycle in this order: red, yellow, white and blue. The color changed each 127ms. We modified the code to make the LED blink in green at 1 Hz. We only had to add 4 lines of code in the while loop in the ex1.cc code(we obviously had to remove the original code in the while loop):

```
set_rgb(0,127,0);
pause(HALF\_SEC);
set_rgb(0,0,0);
pause(HALF\_SEC);
```

2 Registers

At first, we had to understand the code beforehand and then verify if it ouputs what we thought it would. To understand clearly what happens with the registers, we had to look at the order and the registers used in the pc code (Table 1).

bytes	register	read/write	value sent
1	6	read	-
1	21	read	-
1	21	read	-
1	6	read	-
1	6	read	-
multi	2	send	buffer
multi	2	read	-
1	2	send	11
1	3	send	22
multi	2	read	-
2	7	send	2121
4	2	read	-
2	7	send	1765
4	2	read	-

Table 1: ex2.cc order of PC commands.

After defining the order of the commands done, we had to understand in the robot code what happens if the registers are read or written. The robot has 4 main variables :

- uint32_t datavar
- uint8_t last_mb_size
- uint8_t counter
- uint_8t mb_buffer[MAX_MB_SIZE]

The register handler is called each time a command (read or write) is sent from the computer. Each time the register handler receives a command, it enters a switch case function. This switch case function switches depending on the function called by the computer(read,write/1,2,4 or multi bytes).

- read register 6 of 1 byte –> gives back counter value and sets it to 0
- read register 21 1 byte –> increment counter value, give back 0x42 which is equal to 66
- read register 2 of 4 bytes –> gives back datavar value
- write in register 2,3 or 4 of 1 byte –> mb_buffer[register-2] = given value written
- write in register 7 of 2 bytes –> datavar = 3 × datavar + written value
- write in register 2 of mulitbyte –> each value + 4 of the written table stored in mb_buffer.

Now that all is set we can understand and predict what the code will output (see Table 2). Note that the following table doesn't include the output of the function called (for example the true output of the first command is : "get_reg_b(6) =0")

ouput	reason for output	variable changed in robot
0	counter =0	counter =0
66	0x42=66	counter ++
66	0x42=66	counter ++
2	counter =2	counter =0
0	counter =0	counter =0
-	-	mb_buffer[i] = buffer [i] + 4 last_mb_size = 8
8 bytes : 104,105,106,107 108,109,110,111	mb_buffer = 104,105, 106,107,108,109,110,111 last_mb_size = 8	-
-	-	mb_buffer[0]=11
-	-	mb_buffer[1]=22
8 bytes : 11,22,106,107 108,109,110,111	mb_buffer = 11,22, 106,107,108,109,110,111 last_mb_size = 8	-
-	-	datavar = 2121
2121	datavar = 2121	-
-	-	datavar = 2121 × 3 + 1765
8128	datavar = 8128	-

Table 2: resulting ouput of exercice 2

3 Communication with other elements

Now that the radio communication and its registers is understood we need to understand how the head of the robot communicates with the rest of the robot. All parts of the robot are connected with a CAN bus. The original ex3.cc program changes the LED colors depending on the position of the motor. We then modified the program to give back the position of all the motors to the pc through radio communication. In the robot program the callback function gave 5 values back with the multi-byte register if the second register of the multi-byte was read. We modified the exercise 2 to return each motor positions in a 5 byte table. At first we create a macro MOTORS and created global tables, motorpos which contains each motor position and MOTOR_ADDR which contains each motors CAN address:

```
#define MOTORS 5
uint8_t motorPos[MOTORS];
const uint8_t MOTOR_ADDR [MOTORS]={4,13,12,14,7};
```

Then after the hardware initiation we added a radio callback function to the robot and a indentation value i for further in the code :

```
radio_add_reg_callback(register_handler);
uint8_t i = 0;
```

Then in the main while loop, we store all motor position in the motorPos table:

```
for(i=0; i < MOTORS; i++)
    motorPos[i] = bus_get(MOTOR_ADDR[i], MREG_POSITION);
```

Finally we added the register_handler function inspired by the exercice 2 callback function:

```
static int8_t register_handler(uint8_t operation, uint8_t address, RadioData* radio_da
{
    uint8_t i;
    switch (operation)
    {
        case ROP_READ_MB:
            if (address == 2) {
                radio_data->multibyte.size = MOTORS;
                for (i = 0; i < MOTORS; i++)
                    radio_data->multibyte.data[i] = motorPos[i];
                return TRUE;
            }
            break;
    }
    return FALSE;
}
```

In the progam code we modified the ex2 read_registers function by removing all the code and adding :

```
cout << "get_reg_mb_(2) = ";
display_multibyte_register(regs,2);
```

The result was successful, we managed to read all the motors angles.

4 Position control of a module

In this section the aim is to communicate a wished position to the robot. At first we set the mode of the robot to 1 by setting the register REG8_MODE to 1 trough radio communication :

```
regs.set_reg_b(REG8_MODE, 1);
```

This pre-programmed mode makes the module move from 21° to −21° with a 0.5 HZ frequency.

We then modified the code to be able to give a sin wave command from the computer to the robot. The sin wave is produced by the computer and it sends positions one by one at each radio communication. At first we get the time of start and initialize the position value l to 0 :

```
double time_start = time_d();
double l = 0;
```

We changed the while loop to make it break as soon as a key is pressed. This allows us to change to mode back to 0 at the end of each test, which then stops the pid so the robot is malleable as soon as we stop the program. The sin wave sent has an amplitude of 40° an a frequency of 1 Herz. We used the function ENCODE_PARAM_8 to be able to send float values to the robot(a calculation is made so that the 8 bit uint8_t value is decode-able by the robot by using DECODE_PARAM_8. The wished value is sent in register address 3:

```
while(!kbhit()){
    l = 40.0*sin((time_d()-timestart)*(2*M_PI));
    regs.set_reg_b(3, ENCODE_PARAM_8(l,-60.0,60.0))
}
regs.set_reg_b(REG8_MODE,0);
```

The robot code needs also to be modified, firstly we create a wished position global variable setPos:

```
volatile uint8_t setPos = 0;
```

Finally to be able to move the robot according to the data sent we had to modify the motor_demo_mode function to set the motor to the wished position. we only had to remove what was in the motor_demo_mode function's while loop and put:

```
bus_set(MOTOR_ADDR, MREG_SETPOINT, DEG_TO_OUPUT(DECODE_PARAM_8(setPos,-60.0,60.0));
```

The result was successful, we were able to move the tail of the robot according to a sin wave.

5 on-board trajectory generation

5.1 Simple on-board trajectory generation

In this section the aim is to make the robot move according to a sin wave produced by the robot itself. first we set the mode of the robot to 2 by setting the register REG8_MODE to 2 trough radio communication from the pc. After pressing a key the mode will switch to 0 (IDLE mode):

```
regs.set_reg_b(REG8_MODE, 2);
ext_key();
regs.set_reg_b(REG8_MODE, 0);
```

This pre-programmed mode makes the module's led switch color according to a sin wave. The goal being to move a motor to this function, we have to remove the varying led part and replace it with a motor position command ($l = 40 \times sin(t \times 2\pi)$):

```
bus_set(7, MREG_SETPOINT, DEG_TO_OUTPUT_BODY(1));
```

The obtained movement looks the same as the one from the section before (Position control of a module). Normally, due to radio communication, the robot could have latency depending of distance or even loss of packets. It is better to do the sin wave from the robot than the pc.

5.2 Modulating trajectory parameters

This section aims to be able to change the parameter values such as amplitude and frequency of the on-board sin-wave generation. This will be useful for for further testing, programming the robot just to change parameters takes much longer than to send the parameters from the pc. To make it easier to change parameters we read the values put in the command line tool when executing the program. In the main we added the retrieval of argc and argv :

```
int main(int argc, char* argv[])
```

If no amplitude and frequency value is given in the command line we set default values 20 and 0,5. We then retrieve the amplitude value from argv[1] and the frequency value from argv[2]. We then show the selected values in the terminal.

```
double amplitude = 20.;
double frequency = 0.5;
    if(argc==3){
    amplitude = atof(argv[1]);
    frequency = atof(argv[2]);
}
cout << "amplitude  =  " << amplitude << endl;
cout << "frequency  =  " << frequency<< endl;
```

To limit the given amplitude to 60° and frequency to 2Hz and give float values through uint8_t variables, the functions ENCODE_PARAM_8 explained before are used. The wished amplitude and frequency are set in register 3 an 2 respectively. To be able to stop the robot as soon a key is pressed (switch back to IDLE mode), ext_key() function is used.

```
regs.set_reg_b(REG8_MODE, 2);
regs.set_reg_b(3, ENCODE_PARAM_8(amplitude,0.0,60.0));
regs.set_reg_b(2, ENCODE_PARAM_8(frequency,0.0,2.0));
ext_key();
regs.set_reg_b(REG8_MODE, 0);
```

In the robot we had to receive and read the given values from the corresponding registers. We added a radio callback function in the main_mode_loop function.

```
radio_add_reg_callback(register_handler);
```

The call back function receives the data trough registers 2 and 3:

```
static int8_t register_handler(uint8_t operation, uint8_t address, RadioData* radio_da
{
    switch (operation)
    {
        case ROP_WRITE_8:
        if (address == 3) {
            amplitude= radio_data->byte;
            return TRUE;
        }else if(address==2){
            frequency= radio_data->byte;
            return TRUE;
        }
        break;
    }
    return FALSE;
}
```

Finally to be able to decode the wished parameters, we use the DECODE_PARAMS_8 function. These values are then used to calculate the sin wave and set the motor positions accordingly :

```
l = DECODE_PARAM_8(amplitude,0.0,60.0)* sin(M_TWOPI * DECODE_PARAM_8(frequency,0.0,2.
bus_set(7, MREG_SETPOINT, DEG_TO_OUTPUT_BODY(l));
```