Lab#4: Mover 4 robotic arm – Simulation mode

Leonardo Fusser (1946995)

Objectives:

- Communicate with the Mover 4 controller thread.
- Create a thread to interface a keyboard.
- Create a thread to display on the screen.
- Visualize Mover 4 robot arm position using an Excel Macro-Enabled file.

Material:

Windows - Visual Studio – Excel.

Pre-requisite: POSIX pthreads and pdcurses in Windows.

To hand in: No report to hand-in. Answers to all questions.

- Indented and commented multi-source file on Git Hub.
- All functions must have a prolog header.
- All files must have their prolog headers: file name, description, date, author and version history.
- Use of symbolic constant is mandatory, all macros in capital letters.
- Answer all questions by filling in the lab sheets using a computer. Must be submitted in paper.

You must follow the following encapsulation and modularization rules:

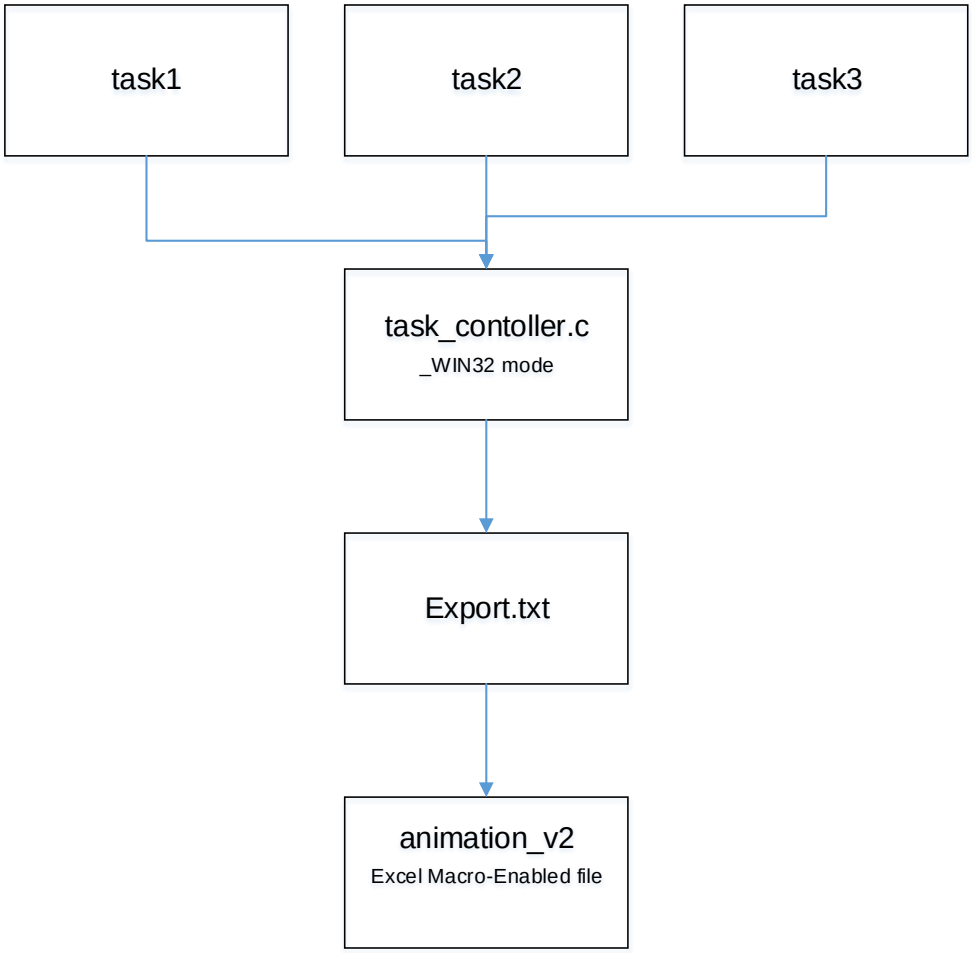
- Abstract all configuration related code by creating functions or macros.

```
#define HUMERUS 146.05
void set_arm( float x, float y, float z, float grip_angle_d,)
```

If the requirements are not fulfilled, the student will be asked to re-submit.

The structure of the lab is as follows:

- The task_controller writes to a text file.
- The text file is read in an Excel Macro-enabled file.
- The Excel Macro animates the mover4 robot arm position.
- All tasks must use APIs to communicate with the task_controller.



File system structure:

The file system organization is as listed in table 1.

Table 1: File organization

Files	Content
Provided `mover4.c`	Partially populated. Contain the main function. Initializes ncurses. Initializes the CAN socket. Starts two controller tasks (see task_contoller.c). Joins two controller task threads (see task_contoller.c).

Provided
`can.c`
`can.h`
	Contains all CAN bus related APIs:
	`open_socket()`
	`write_can_mess()`
	`get_can_mess()`

Provided
`taskController.c`
	Contains two tasks: pTask_Controller and pTask_Rx.
	
	These tasks simulate sending and receiving commands to the servos. There is NO actual robot connection.
	pTask_Controller reads the desired SP values and current PV values, calculates the error (e) and the PID value for each motor.
	It internally simulates all 4 motor movements.
	
	Contains the following public functions:
	
	`double get_sp_angle(int nb);`
	`int set_sp_angle(int nb, double _angle);`
	
	`int set_all_sp_angles (kin_f _angles);`
	`kin_f get_all_sp_angles (void);`
	
	`double get_curr_angle(int nb);`
	`kin_f get_all_curr_angles(void);`
	
	`void startTasksControllerRx(void);`
	`void pthread_joinControllerRx(void);`
	`void set_gripper(int val);`
	`void print_warnings(int v, int h);`
	`void print_errors(int v, int h);`
	
	See annex for API definitions.
	
	SP: set point values.
	PV: values transmitted to the robot (through CAN bus).
	curPV: actual current PV values of the robot.

Provided
`ncurses_init.c`
`ncurses_init.h`
	Provide useful APIs for the ncurses library.

Provided `demo/exec`	Not used for now.

Provide
`excel_sim/animation_v2`
`excel_sim /export`
	Excel Macro-Enabled file that simulates the robot. The export file is the interface between the executable and the Excel file.

To create `TaskKeybd`	Provides a user interface. Reads keys q w e r to jog up the angles. Reads keys a s d f to jog down the angles. Reads key t to open the grip. Reads key g to close the grip. Reads key x to exit the program (calls endwin() and exit(0)). To set the SP angles, you must use `set_sp_angle()` API. To set the gripper, you must use `set_gripper()` API. Note: It must stop increasing its value as soon as the jog key is released.
To create `TaskDisplay`	Must read the curPV and SP angles and display them. See `get_curr_angle()` and `get_sp_angle()`
To create `public.h`	Contains all public prototypes, shared macros and new types.
`build`	Script file used only with BBB. NOT used in this current lab.

See the following hierarchy. You must populate the files in **orange.**

- **multithread_mover4_v6\pdcurses_test\mover4_v6**
  - can.c
  - task_controller.c
  - log
  - state
  - ncurses_init.c
  - adc.c
  - **sol_mover4**
    - build
    - **mover4.c**
    - **taskKeybd.c**
    - **taskDisplay.c**
  - **header**
    - can.h
    - adc.h
    - **public.h**
    - task_controller.h
    - ncurses_init.h
  - **demo**
    - exec
  - **excel_sim**
    - animation_v2
    - export

Important:

1. When you create files in Visual Studio, they MUST be c file. They cannot be CPP files otherwise the code cannot be ported to a Linux c project.

2. Make sure all newly created files are located as specified above (e.g., h files inside the header folder).

Requirements:

You must write a multithread program to control the mover4 robot.

The following threads/files must be programmed:

```
taskDisplay.c
taskKeybd.c
```

See Table 1 and table 2 for details.

`taskKeybd` thread
This thread must read the keyboard and increase the angles of the robots' joints using specific keys. Every time a key is pressed the angle must jog one step. Step size must be set experimentally.

`taskDisplay` thread
This thread must display:

- a menu.
- the SP and curPV values in real time.
- all errors and warnings using `print_warnings() and print_errors()` APIs.

The CPU utilization must be less than 1% (see Task Manager).

See annex to find out how to get and set various angles.
See table 2 for details.

Table 2: Display menu.

Main menu display area **Line 0 to Line 9**

```
*****************************************************************************
* Use the following keys to jog the joints or grip                          *
*                                                                           *
*    q         w        e       r       t                                   *
*  Joint1   Joint2   Joint3   Joint4  Gripper                                *
*    a         s        d       f       g                                   *
*                                                                           *
*  Use also the following key:                                             *
*  Exit: x                                                                  *
*****************************************************************************
```

PV and SP display area **Line 11 to Line 14**

```
*****************************************************************************
Angles: SP      Bas:-19.95deg.  Shd:46.91deg.   Elb:96.82deg.   Wst:-57.97deg.
        curPV   Bas:-19.95deg.  Shd:46.91deg.   Elb:96.82deg.   Wst:-57.97deg.
*****************************************************************************
```

Warning message area **Line 16** – see `print_warnings()` API.

```
z or r or angle limits exceeded
*****************************************************************************
```

Error message area **Line 18 -** see `print_errors()` API.

```
The system cannot restart - Call the teacher to clear the errors -Ctrl C to quit
 ****************************************************************************
```

SP: set point values.
curPV: actual current values of the robot.

Lab Work

GitHub:

Accept the following invitation:

https://classroom.github.com/a/DDcWwtRp

You will get a repo for the next few labs:

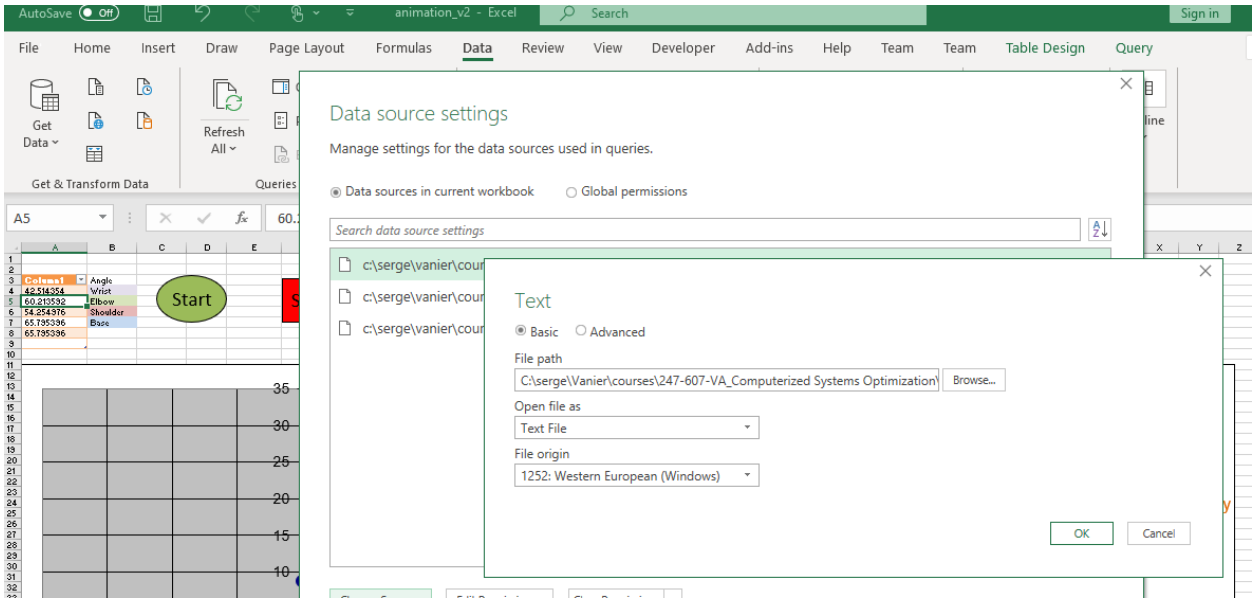
https://github.com/Embedded-OS-Vanier/mover4_simulation_mode-yourName

You must complete, modify, or create the tasks as mentioned in the above tables.

Export file path

When you clone your project to a new location, the export file path gets corrupted.
To change export path:

- On the **Data** tab, in the **Get Data** group, click **Data Source Setting…**
- Click Change Source.
- Specify the export file location on your hard drive.



Also, In the animation_v2 excel file, change the link lengths of the robot arm. They must be set to Mover4 link length.

Programming

In this part, you are going to write the code without the actual robot connected to BBB. An Excel macro will simulate the robot's position.

Open the Macro Excel file: `mover4_v6\excel_sim\animation_v2`

Complete the threads according to the requirements.

Upgrading to your private repository

Now it is time to update your repo.

But before, erase the following folder to save space:
C:\Users\sh\source\repos\mover4_sim1\.vs\pdcurses_test\v16

Approval before leaving!

Annex – APIs and structures definitions

Important structure types are included in `task_controller.h` :

```
typedef struct{
	double data[4]; //Angle.
}kin_f;
```

API definitions

```
/* Sets the gripper:
	GRIP_CLOSE
	GRIP_OPEN
*/
void set_gripper(int);
```

```
/* Returns the SP angle of the specified joint
   A value from 0 to 3 must be provided.
*/
double get_sp_angle(int);
```

```
/* Sets angles for joints 0 to 3.
   Base (0) -150 to +150 degrees.
   Shoulder(1)-50 to +65 degrees.
   Elbow (2) -110 to +140 degrees.
   Wrist (3) -140 to +140 degrees.
*/
void set_sp_angle(int i, double _angle);
```

```
/* Returns all four SP angles.
   The return values are inside a kin_f structure.
*/
kin_f get_all_sp_angles(void)
```

```
/* Sets SP angles for joints 0 to 3.
   All angles must be stored in a kin_f strucure
   before the call.
   Base (0) -150 to +150 degrees.
   Shoulder(1)-50 to +65 degrees.
   Elbow (2) -110 to +140 degrees.
   Wrist (3) -140 to +140 degrees.
*/
void set_all_sp_angles(kin_f _angles);
```

```
/* Gets all curPV angles for joints 0 to 3.
   The return values are inside a kin_f structure.
*/
kin_f get_all_curr_angles(void)

/* Returns the curPV angle of the specified joint.
   A value from 0 to 3 must be provided.
*/
double get_curr_angle(int nb);



/* Prints all current warnings to the line and column specified. */
void print_warnings(int y, int x);


/* Prints all current errors to the line and column specified. */
void print_errors(int y, int x);


/* Sets a blocking delay in mS. */
void delay_ms(int d);
```