Embedded Control Laboratory

Ball on inclined plane

MECHATRONICS MASTERS

Report of lab exercise 3

WS 2019/20

Group 08

1522099 Kurapati Chakradhar Reddy

1536256 K Harish

1536227 Surya Vara Prasad Alla

26.01.2020

Table of Contents

1. Introduction ……3
   1.1 ProgrammingLanguage:Embedded C ……...3
   1.2 Operating system: FreeRTOS ……...3
   1.3 DevelopmentEnvironment:AtmelStudio……3
   1.4 Microcontroller:ATmega128…….4
   1.5 Pulse Width Modulation……4
2. Objective ……....5
3. Model Calibration……..……5
4. PROGRAMMIN…….……9
   4.1 ServoSensor.c……9
   4.2 ModifyingtheWelcome Message……...9
5. PID CONTROL ALGORITHM…….10
   5.1 Implementation……..11
6. QUEUES AND TASKS…….13
7. CONCLUSION ……..14
8. CITATION AND REFERENCES…….14

1. Introduction

In this exercise, the objective is to control the position of ball on an inclined plane using an ATmega128 microcontroller. In an attempt to achieve the desired position of ball on an inclined plane, we make use of 16MHz ATmega128 microcontroller taking into consideration results from previous tasks. In the implementation, Embedded C is used as the programming language and FreeRTOS as the operating system. For developing and debugging Embedded C application code, we have used Atmel Studio 7 Integrated Development Environment (IDE).

1.1. Embedded C Programming

Embedded C Programming is the soul of the processor functioning inside each and every embedded system. This C Programming is used for limited resources like RAM,ROM and I/O peripherals on embedded controller. Two important features of Embedded Programming are code speed and code size. Code speed is governed by the processing power while code size is governed by program memory. By using embedded system programming, we get maximum features in minimum space and minimum time[1].

1.2. Operating System: FreeRTOS

FreeRTOS is a class of RTOS that is designed to be small enough to run on a microcontroller. It has a small memory footprint which results in minimal RAM, ROM, and processing overhead. FreeRTOS provides the core real time scheduling functionality, inter-task communication, timing and synchronisation primitives only. This means it is more accurately described as a real time kernel, or real time executive. Additional functionality, such as a command console interface, or networking stacks, can then be included with add-on components[2].

1.3. Development Environment: Atmel Studio 7

Atmel Studio 7 is the integrated development platform (IDP) for developing and debugging all AVR® and SAM microcontroller applications. The Atmel Studio 7 IDE gives you a seamless and easy-to-use environment to write, build, and debug your embedded-C and assembler application code[3].

Key features:

- Support for 500+ AVR and SAM devices
- Integrated C/C++ compiler
- Integrated editor with visual assist
- Advanced debugging features

1.4. Microcontroller: ATmega128

To realize our BOIP system, we use the ATmega128 microcontroller, which is a high performance, low power 8 bit microcontroller. It is based on advanced RISC architecture[4].

Key features:

- High-performance, Low-power
- Write/Erase cycles: 10,000 Flash/100,000 EEPROM
- High Endurance, Non-volatile memory
- Data retention: 20 years at 85°C/100 years at 25°C
- 128Kbytes of In-System Self-programmable Flash program memory
- Software Selectable Clock Frequency

1.5. Pulse Width Modulation

One simple and easy way to control the speed of a motor is to regulate the amount of voltage across its terminals and this can be achieved using “Pulse Width Modulation” or PWM. The power applied to the motor can be controlled by varying the width of these applied pulses and thereby varying the average DC voltage applied to the motors terminals. By changing or modulating the timing of these pulses the speed of the motor can be controlled, ie, the longer the pulse is “ON”, the faster the motor will rotate and likewise, the shorter the pulse is “ON” the slower the motor will rotate.The key reason that PWM circuits are so efficient is that they don’t try to partially restrict the flow of current using resistance. They turn the current fully on and fully off. They just vary the amount of time that it is on instead[5].

2. OBJECTIVES

- Calculate correct value for the middle-position of the inclined plane.

- Find the correct value of the variable “Delta”, which represents the gain necessary to move the arm to the right or left angle.
- Modify the welcome message of the LCD.
- Find the correct PID parameters according to the BoiP behaviour.
- Identify and explain tasks, queues and exchanged data between different tasks and queues.

3. MODEL CALIBRATION

Position of the ball on an inclined plane(BoIP) is a function of the angle of the plane alpha and servo motor angle beta and the position is obtained by programming PID Controller (Proportional-Integral-Derivative) and controlling the angle of the inclined plane. The relation between position and angle of the plane or $\alpha = f(\beta)$ was established in the first exercise and we have carried out the simulation using continuous and discrete PID controller in the second exercise. To achieve the desired position of ball on an inclined plane, we make use of 16MHz ATmega128 microcontroller taking into consideration results from previous tasks. We have used Embedded C as the programming language and FreeRTOS as the operating system in microcontroller. For developing and debugging the Embedded C application code, we have used Atmel AVR Studio 7 Integrated Development Platform (IDP).

In this exercise, We use a servo motor to move the plane depending on the position of the ball. A servomotor is a rotary actuator that is capable of providing precise control of angular position, velocity and acceleration. Output from the microcontroller are voltages in the form of Pulse Width Modulation (PWM) which controls the motion of the servo motor and the servo motor interprets these voltages in fixed pulse width to rotate at a desired angle.The wider the pulse width, the more average voltage applied to the motor terminals, the stronger the magnetic flux inside the armature windings and the faster the motor will rotate The pulse width determines the direction and angle of rotation of the servo motor and it depends on the clock frequency of microcontroller and on the value of pre-scalar. Pre-scalar is used to reduce the high frequency electrical signal to low frequency electrical signal.

In this task, servomotor position is controlled by positive pulses and pulses are repeated every 18 ms. Different pulse widths are used to control the servo motor position as given below.

Servomotor positions:

Left (L) position is achieved at 1 ms.

Middle (M) position is achieved at 1.5 ms.

Right (R) position is achieved at 2 ms.

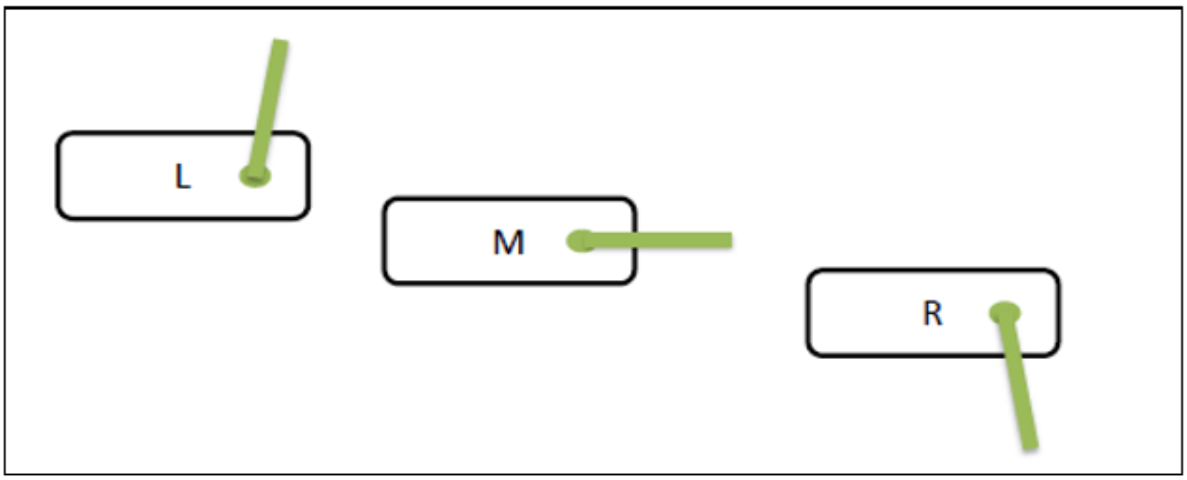

Figure 3.1: Position of Servo Motor Arm

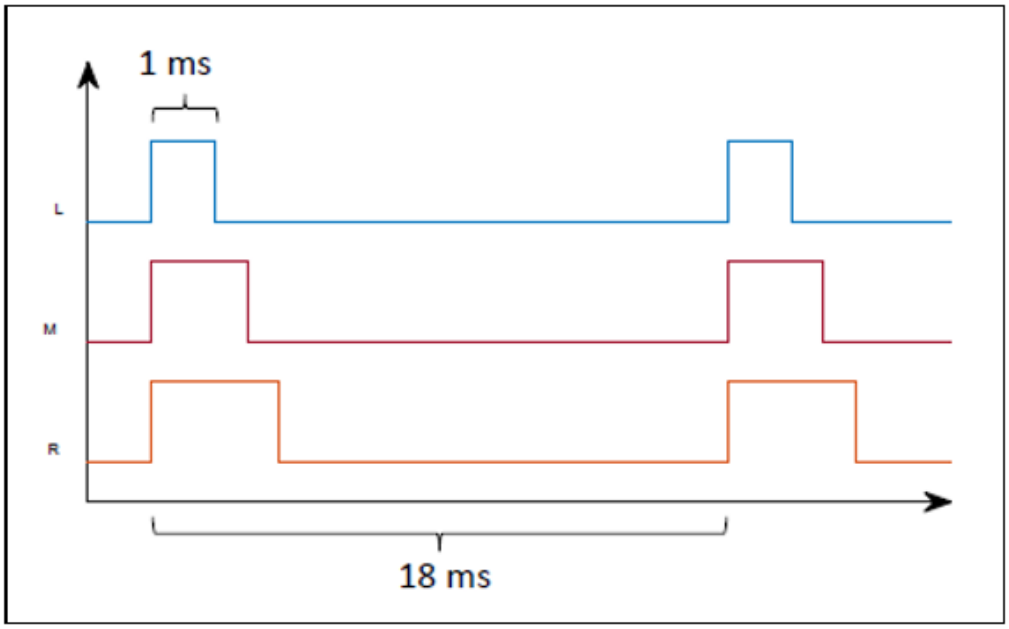

Figure 3.2: Pulse Width Modulation

The servo motor is first programmed to rotate to its mid position or MIDPOS. In the program, following Timers/Counters are used to determine the mode of operation and prescaler settings,

TCNT3 – TCNT3 is a 16 bit Counter/Timer which can count 0 to 65535 values. This counter is used to count the pulses.

OCR3A – The Output Compare Registers contain a 16-bit value that is continuously compared with the counter value (TCNT3). A match can be used to generate an output compare interrupt, or to generate a waveform output on the OCR3 pin.

TCCR3B – This Timer/Counter3 Control Register B gives the prescaler value of the input frequency. The bit configuration of TCCR3B is set to 00000011 in the program.

Timer/Counter3 Control Register B – TCCR3B

Bit	7	6	5	4	3	2	1	0	
	ICNC3	ICES3	–	WGM33	WGM32	CS32	CS31	CS30	TCCR3B
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 3.3: TCCR3B Register

As per the datasheet, we get the Prescaler value of frequency by the table given in figure 4. According to ATmega128 microcontroller's datasheet, the clock frequency of the microcontroller is 16MHz. The clock frequency of the

microcontroller will be divided by 8 in order to allow the timer to be clocked at desired frequency.

Clock Frequency = Maximum Clock Frequency/8

= 16 MHz / 8

= 2 MHz

We are reducing the pulse speed from 16 MHz to 2 MHz

Clock Timing = 1/ 2*e6 s

= 0.5*e-6 s

= 0.5 µs

The time taken by a pulse to reach the timer is 2 µs

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figure 3.4: Clock selection bit description

As mentioned above, the servo motor turns to its mid position at pulse timer 1.5ms.

Therefore, Mid Position = 1.5e-3 / 0.5e-6 = 3000

The pulse width values for the servo motor to turn left and right most positions are 1.0 and 2.0 ms respectively.

Left Position = 1.0e-3 / 0.5e-6 = 2000

Right Position = 2e-3 / 0.5e-6 = 4000

Delta is the amplification factor. The OCR3A value depends upon the value of Delta and this value works as the gain for the value of Beta. We get the value of

Delta as 1000 (Mid Position – Left Position) which is used as gain to move the servo motor to the left and right with respect to mid position.

4. PROGRAMMING

4.1. ServoSensor.c

The ServoSensor code file computes the position of the ball based on the LED values which are then stored in the registers. The file contains the code that sends pulses to the servo motor to rotate it to the required angle and value depending on the PID output. The MIDPOS and the gain DELTA are mentioned in servosensor.c file. The mid position of the plane as per theoretical calculation is 3000 but due to mechanical constraints, the actual mid position of the plane varies. We determined the actual value of MIDPOS to be at 2965 based on trial and error method. The value of MIDPOS and DELTA are initialised in the file as shown below.

```
//-----------------------------------------------------------
void  vServo ( void * pvParameters)
{ unsigned int  MIDPOS = 2965;
  float  DELTA = 1000, beta;
  for (;;)  // Super-Loop
  { // Read sensor data from queue
    xQueueReceive (QueueServo, &beta, portMAX_DELAY);
        // Check the limit of the value
        if (beta > +1.0)  beta = +1.0;
        if (beta < -1.0)  beta = -1.0;
        // Give the value to global var
        angle = beta;
        // Generate timer compare value of servo signal
        //OCR3A = MIDPOS; // check horizontal position of BoiP
        OCR3A = (unsigned int) (MIDPOS - (int)(beta * DELTA));
  } // end for
} // end of task servo
//-----------------------------------------------------------
```

Figure 4.1: MIDPOS and DELTA value in the code

4.2. Modifying the Welcome Message

We have changed the title in the LCD Display to Group 8 Ex #3 2020 by modifying the text in the HMI.c file as shown below

```
/*-------------------------------------------------------------*/
// Init of HMI menus
const char  Main_Menu [80] PROGMEM =
 "Group08 Ex 3 2020 >Change Parameter   >Show Ball Position >Start PID Control ";
//01234567890123456789012345678901234567890123456789012345678901234567890123456789012345678
const char  PI_Parameter_Menu [80] PROGMEM =
 "Parameter Values PI >Proportional=+0.123>    Integral=+0.123>Next             ";
const char  DRT_Parameter_Menu [80] PROGMEM =
 "Parameter Values DR >  Derivative=+0.123>   Reference=+0.123>Back             ";
const char  PID_Menu [80] PROGMEM =
 "PID is running ...  -   PID-Input=-0.123-  PID-Output=-0.123>Back and Stop PID ";
const char  SBP_Menu [80] PROGMEM =
 "Show Ball Position  -SlowPosition=-0.123-L0000000  FT1111111>Back             ";
const char err_status [21] PROGMEM =          "status out of range ";
const char err_Cursor_Y [21] PROGMEM =        "Cursor_Y out of rang";
const char err_Cursor_X [21] PROGMEM =        "Cursor_X out of rang";
// Enum for HMI state machine
enum {Main, Para1, Para2, PIDr, SBP};
//-------------------------------------------------------------------
void  vHMI ( void * pvParameters)
{ unsigned char  lauf, para = NADA , status = Main;
  // Stop sensor task
  vTaskSuspend (vSensor_Handle);
  servo_power = 0;
```

Figure 4.2: Modified Welcome Message

5. PID CONTROL ALGORITHM

A simple algorithm is implemented on the PID controller to get the desired output. The previous output is compared with the desired output and a correction is performed on the error generated till the desired output is produced. The microcontroller is fed with the values of Proportional (P), Integral (I) and Derivative (D) to achieve a stable system. The set value is denoted as reference parameter and actual position is denoted by 'x'. Thus the error 'e' can be given as:

$$e = \text{reference} - x$$

The proportional action works as per magnitude of the error. It means control variable should be adjusted proportionally to the amount of error in the system.When we increase the proportional gain (Kp), the response of the control system will increase. However, if the proportional gain is too large, the process variable will begin to oscillate and the system will become unstable and may even oscillate out of control[6].

$$\text{Proportional term} = \text{Proportional gain}.\text{error}$$

The integral component takes into account the sum of the errors over the system operation. As a result, even a small error term will increase the integral component. Unless the error is zero, the integral response will continually increase over time, so the effect is to drive the Steady-State error, caused by the proportional component of the PID controller to zero. The integral component of the control algorithm can remove any steady-state error in the system because it accumulates that error over time and compensates for it, rather than just looking at an instantaneous snapshot of the error at one moment in time.

$$\text{Integral Term} = \text{Integral Gain}.(I+(e.td))$$

The derivative response is proportional to the rate of change of the process variable. Derivative action predicts the future by projecting the current rate of change, thus improving the controller action. This means that it is not using the current measured value, but a future measured value[6].

$$\text{Derivative term} = \text{Derivative gain}.(\text{error} - \text{previous error})$$

Thus, the output is the sum of all PID responses together and is given by:

$$Y=\text{Proportional term} + \text{Derivative term} + \text{Integral term}$$

5.1. Implementation

We implement final equations given above for a Discrete PID control task to test our code on the real model. These equations are added to the PID.c file and the file is represented as shown below:

```
void  vPID ( void * pvParameters)
{ float x = 0.0, y = 0.0, error=0.0, previous_error=0.0, dt=0.018, p=0.0,i=0.0,d=0.0;
    // Place here your local parameters

  // Super loop of task
  for (;;)
  { // Get position value from sensor queue
    xQueueReceive (QueueSensor, &x, portMAX_DELAY);


      /*  Compute here your digital PID formula with the
            help of following global variables:
            reference, proportional, integral, derivative
            and please remember the sampling time
      */

      // For example a simple P control startup formula
      error = reference - x;
      p=proportional*error;
      i=i+(error*dt);
      d=(error-previous_error)/dt;
      previous_error=error;
      y=p+i*integral+d*derivative;
```

Figure 7: PID Control Algorithm

The values of proportional, derivative and integral gains are declared in HMI.c file and these are tuned to find good working parameters of PID controller. This is done by manually tuning the values by trial and error method till the system is stabilised.

We used the following steps to tune the PID controller

a) Set all three gains to zero.

b) The proportional gain is increased until the oscillations are stable.

c) Then increase derivative gain in order to minimise the oscillations.

d) Repeat steps b & c to get the minimum oscillation values of proportional gain and derivative gain.

e) Finally, increase the integral gain till the minimum time is achieved.

After testing on the real model, we obtained the following PID Parameters for optimum performance. The setup takes around 8 seconds to stabilize the ball at mid position

P = 0.9, I = 0.022, D = 0.4

Time(sec)	PID Values
7	P = 0.9, I = 0.022, D = 0.4
4.5	P = 0.9, I = 0.022, D = 0.4
8.9	P = 0.9, I = 0.022, D = 0.4
4.2	P = 0.9, I = 0.022, D = 0.4
7.3	P = 0.9, I = 0.022, D = 0.4
8.2	P = 0.9, I = 0.022, D = 0.4
7.4	P = 0.9, I = 0.022, D = 0.4
5.3	P = 0.9, I = 0.022, D = 0.4
6.4	P = 0.9, I = 0.022, D = 0.4
4.2	P = 0.9, I = 0.022, D = 0.4

Table1 : PID OUTPUT vs Iteration time

Ts =18ms

The sampling time denotes the end of one cycle and start of a new cycle after the sampling time, because it is needed to compute the derivative and integral values in a loop. A sampling time of 18ms is specified for this exercise.

6. QUEUES AND TASKS

Citation: Atmel studio 7 program

Every action to be performed is sequentially programmed in to Tasks. The order of execution is decided by the task scheduler. Queues are used to communicate between two tasks. Declarations of the tasks and Queues are done under main program. At the end of each task, the output of the task is sent to a Queue using xQueueSend command with a wait period of

some ticks. Then the RTOS will wake xQueueRecieve and corresponding task will be initialised.

The following queues and tasks are used in the above program.
Queues: QueueTaster; QueueSensor; QueueServo
Tasks: vSensor; vTaster; vHMI; vServo; vPID

Data exchange:

"vTaskStartScheduler()" starts all the tasks at the beginning of the program. "vSensor" task is first called and it determines the position of the ball with the help of photo sensors which are connected to the PORTC of the Microcontroller board. The position of the ball is obtained and set to the parameter "smooth_pos". This value is sent to "QueueSensor".

That sent parameter is received by the task "vPID".Here the PID implementation is done and the output is sent to "QueueServo".The output is received by "vServo". In this the output signal is compared with the Midpos value and generates OCR3A signal. This moves the Servo motor arm. This process is continued till the ball is reached to desired position.

In addition to these there are other tasks in collaboration with LCD display. "vTaster" is used to record the event done by the user ,ie., pressing up/down/side buttons. The event is recorded and sent to "QueueTaster" and is received by "vHMI".This task analyse the event and gives information to respective pins .The parameters will be updated to the microcontroller, changes display on LCD and sends updated parameter values to "vPID". And the cycle continues.

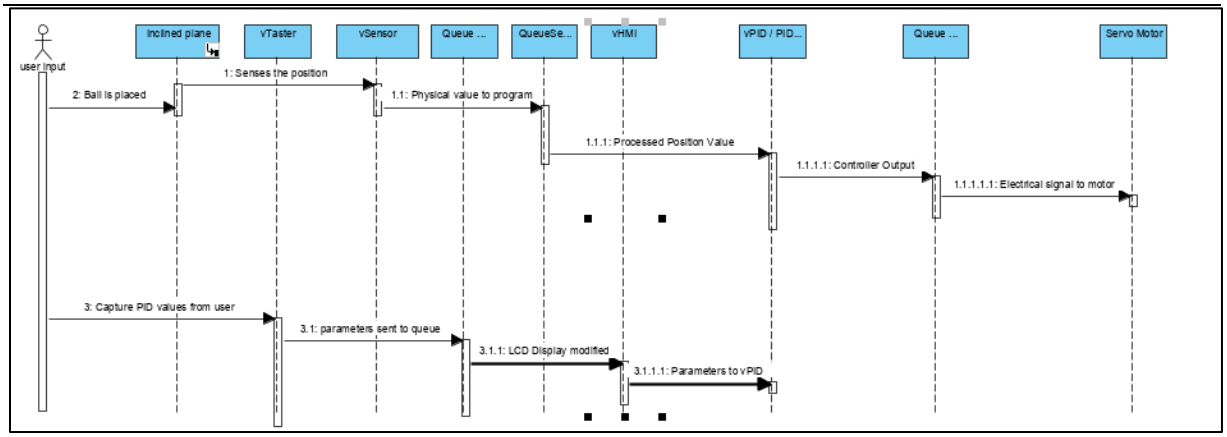

Figure 8: UML Diagram for BoIP System

7. CONCLUSION

When conducted with different times of iterations the ball on the inclined plane got stabled within 10 secs but very few times it got a PID alue of +-0.050 in the form of offset error.

8. Citations and References

1. [https://www.elprocus.com/ieee-projects-on-embedded-systems/]
2. [https://www.freertos.org/about-RTOS.html]
3. [https://www.microchip.com/mplab/avr-support/atmel-studio-7]
4. [ATmega128 datasheet] &[http://www7.informatik.uni-wuerzburg.de/fileadmin/10030700/user_upload/Cansat/atmega128.pdf]
5. .[https://www.electronics-tutorials.ws/blog/pulse-width-modulation.html]
6. [https://spin.atomicobject.com/2016/06/28/intro-pid-control/]