Colorado School of Mines
Department of Mechanical Engineering
MEGN 540: Mechatronics

Laboratory 4: Motors and System ID

Deliverable: Specify PWM output to motors and send back control and encoder values. Use Matlab for Sys ID, 2nd order model for plant with dead-bands identified.

Outcomes:

- Implement PWM control of motor
- Experience how different PWM duty-cycles affect performance
- Conduct system identification/calibration using Matlab
- Implement event-based safety to prevent running motors when the voltage is too low

---

Equipment:

· Lab Kit Mouse Keyboard Monitor Internet Connection

Background:

The Zumo car has two geared motors (75:1) that drive the treads on either side of the car. These motors are controlled using an H-Bridge circuit provided by TI's DVR8838 integrated circuit. This provides both directionality control through a digital IO pin and speed control through a pulse-width-modulation (PWM) output from the atmega32u4. Both PWM signals (left and right), are driven by Timer 1 in the atmega32U4, as timer 1 has multiple compare outputs. Your task in this lab is to setup the Timer 1 PWM and send effectively different voltages to the car's motors. Further comments and hints are in the provided MotorPWM.h/c files and associated functions. I suggest you begin by reading the atmega32u4 documentation and PWM information for Timer 1 and figuring out which pins you'll be interfacing with, given the Zumo's wiring schematic and pin-out description.

Word to the wise: The motors won't spin if the battery switch on the Zumo is not ON. You can tell as two blue LED's will show on the back of the car when in the ON position. I also suggest you modify your program to check the battery voltage and send a warning (POWER OFF) to the serial-monitor, to prevent the embarrassingly long time it takes to realize your PWM code is fine, you simply forgot to switch on the motor's power. Also, take care not to run the motors if the motor-drive voltage drops below 4.75V, as this can damage the Ni-MH rechargeable batteries, rendering them useless. On fresh batteries, this gets you 10-20 minutes of drive time.

Primary Functions to add: (note the file can be found in MEGN540/c_lib):

void Motor_PWM_Init(uint16_t MAX_PWM);

MotorPWM.h/c

Function MotorPWM_Init initializes the motor PWM on Timer 1 for PWM based voltage control of the motors. The Motor PWM system shall initialize in the disabled state for safety reasons. You should specifically enable Motor PWM outputs only as necessary.

void Motor_PWM_Enable(bool enable);

MotorPWM.h/c

Function MotorPWM_Enable enables or disables the motor PWM outputs.

bool Is_Motor_PWM_Enabled();

MotorPWM.h/c

Function Is_Motor_PWM_Enabled returns if the motor PWM is enabled for output.

Motor_PWM_Left(int16_t pwm);

MotorPWM.h/c

Function Motor_PWM_Left sets the PWM duty cycle for the left motor.

Motor_PWM_Right(int16_t pwm);

MotorPWM.h/c

Function Motor_PWM_Right sets the PWM duty cycle for the right motor.

int16_t Get_Motor_PWM_Left();

MotorPWM.h/c

Function Get_Motor_PWM_Right returns the current PWM duty cycle for the right motor. If disabled it returns what the PWM duty cycle would be.

int16_t Get_Motor_PWM_Right();

MotorPWM.h/c

Function Get_Motor_PWM_Right returns the current PWM duty cycle for the right motor. If disabled it returns what the PWM duty cycle would be.

uint16_t Get_MAX_Motor_PWM();

MotorPWM.h/c

Function Get_MAX_Motor_PWM() returns the PWM count that corresponds to 100 percent duty cycle (all on), this is the same as the value written into ICR1 as (TOP).

void Set_MAX_Motor_PWM(uint16_t MAX_PWM);

MotorPWM.h/c

Function Set_MAX_Motor_PWM sets the maximum pwm count. This function sets the timer counts to zero because the ICR1 can cause undesired behaviors if change dynamically below the current counts, see page 128 of the atmega32U4 datasheat.

Assignment:

You will be implementing PWM for this assignment.

1. Review the provided functions, understand what they are doing, and how their called. Enable the functionality defined for the new functions.
2. Read the Timer 1documentation by atmel32u4. Look up which pins/interrupts are connected to which motor H-Bridge controls. Begin working on one motor, once that's working, move to the other.
3. Enable the message handling for the new 'p', 'P', 'q' and 'Q' messages to send desired PWM values and report back on the status.
    a. Plot the results using the provided plotting capability. Is everything working as desired?
4. Once it is all working, design a trajectory for doing system-identification. I suggest you leverage the 'send CSV commands' button in the provided serial-monitor GUI to send

multiple commands automatically. Record the output from a ‘Q’ command to the car between 10 and 2ms update periods.

5. Using this data, leverage the System Identification toolbox from Matlab to determine a first-order (velocity) model of the left and right motor for use in controller design. I suggest numerically differentiating the encoder counts on the Matlab-side to put it in velocity space and reduce the number of degrees of freedom for the optimization, as you know the relationship between velocity and position with no ambiguity.

Deliverables:

1. Demo: Make a 1-2-Minute Video that talks through your code and how you completed the lab. Make sure to show me how you handled the functions added to interface with the encoders and the battery voltage. Also show me how you are handling messaging and sending the battery warning.
2. Demo: Upload a video of you interacting with the car demoing each operation as well as error states.
    a. Make sure you show the car moving along your system-id trajectory.
    b. Make sure to demo at least one command from each of Labs 1, 2, and 3.
3. Lab Report: Prepare a summary of what you have completed for this assignment. Follow a typical lab report format. Make sure that you:
    a. Include the information that you found important and critical while completing this lab, and would be useful if you wanted to replicate it in the future.
    b. Indicate which pins on the electrical schematic were used in this project, and how you can tell which registers you needed to initialize and set.
    c. Indicate what PWM frequency you chose and why.
    d. Provide a section on you system-identification and resulting models for the left and right side of the car. Are they the same? Different? Comment on the effects (if any) you expect between left and right, do they need to be considered or can we ignore them?

Host -> Device Communication Messing Definition					
	CMD char	CMD Hex	# Bytes	Format	Resulting Action
Lab 1	~	0x7e	1	c	Reset the program (return 0)
	*	0x2a	9	cff	Multiply the two floats and return the product.
	/	0x2f	9	cff	Divide the first float by the second and return the result.
	+	0x2b	9	cff	Sum the floats and return the result.
	-	0x2d	9	cff	Subtract the second from the first and return the result.
	UNDEF				If the command char is unrecognized, flush the input buffer, and respond with a '?' (0x3f) character followed by the unrecognized command.
Lab 2	t	0x54	2	cc	Return the time it requested followed by the time to complete the action specified by the second input char. 0x00 -> Time Now 0x01 -> Time to complete a full loop iteration Others as you define
	T	0x74	6	ccf	Return the time it requested followed by the time to complete the action specified by the second input char and returns the time every X milliseconds. If the time is zero or negative it cancels the request without response.
Lab 3	e	0x65	1	c	Return the encoder counts for the left and right motors
	E	0X45	5	cf	Return the left and right encoder values every X milliseconds specified by float sent. If the float sent is less-than-or-equal-to zero, this cancels the send request.
	b	0x62	1	c	Return the current battery voltage level
	B	0x42	5	cf	Returns the current battery voltage level every X seconds as specified by the float sent. If the float is less-than-or-equal-to zero 0, the request is canceled.
Lab 4	p	0x70	5	chh	Set the PWM command for the left (first) and right (second) side with the sign indicating direction, if power is in acceptable range.
	P	0x50	9	chhf	Set the PWM command for the left (first) and right (second) side with the sign indicating direction, if power is in acceptable range. The following float provides the duration in ms to have the PWM at the specified value, return to 0 PWM (stopped) once that time duration is reached.
	s	0x73	1	c	Stop PWM and disable motor system
	S	0x54	1	c	Stop PWM and disable motor system
	q	0x71	1	c	Send system identification data back to host. (total msg length is 16 bytes to fit one endpoint) time(s) [float] PWM_L [int16_t] PWM_R [int16_t] Encoder_L [int16_t] Encoder_R [int16_t]
	Q	0x51	5	cf	Send the system identification information (above) back to the host every X milliseconds (as specified in the second float). If this float is zero or negative, then the repeat send request is canceled.

Device -> Host Communication Messaging Definition

[MSG Length] [Format C-Str][Host Initiating CMD Char][DATA]

Response Format	Function Call
0x8 'c' 'f' 0x0 '*' [float]	*usb_send_msg*("cf", '*', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 '/' [float]	*usb_send_msg*("cf", '/', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 '+' [float]	*usb_send_msg*("cf", '+', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 '-' [float]	*usb_send_msg*("cf", '-', &data, *sizeof*(data))
0x5 'c' 'c' 0x0 '?' [char]	*usb_send_msg*("cc", '?', &data, *sizeof*(data))
0xA 'c' 'H' 'f' 0x0 't' [uint8] [float]	*usb_send_msg*("cHf", 't', &data, *sizeof*(data))
0xA 'c' 'H' 'f' 0x0 'T' [uint8] [float]	*usb_send_msg*("cHf", 'T', &data, *sizeof*(data))
0xD 'c' 'f' 'f' 0x0 'e' [float] [float]	*usb_send_msg*("cff", 'e', &data, *sizeof*(data))
0xD 'c' 'f' 'f' 0x0 'E' [float] [float]	*usb_send_msg*("cff", 'E', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 'b' [float]	*usb_send_msg*("cff", 'e', &data, *sizeof*(data))
0x8 'c' 'f' 0x0 'B' [float]	*usb_send_msg*("cff", 'B', &data, *sizeof*(data))

0xE 'c' '9' 's' 0x0 '!' 'P' 'O' 'W' 'E' 'R' ' ' 'O' 'F' 'F'
 	
0xC 'c' '7' 's' 0x0 '!' 'B' 'A' 'T' ' ' 'L' 'O' 'W'
	If Power Switch Off:
	struct {char let[9]; } data = { .let = {'P','O','W','E','R', ' ', 'O', 'F', 'F'} };
	usb_send_msg("c9s", '!', &data, sizeof(data));
	
	If Battery Low:
	struct {char let[7]; float volts; } data =
	{ .let = {'B','A','T',' ','L','O', 'W'}, .volt = [float] };
	usb_send_msg("c7sf", '!', &data, sizeof(data));

0xE 'c' '9' 's' 0x0 '!' 'P' 'O' 'W' 'E' 'R' ' ' 'O' 'F' 'F'
 	
0xC 'c' '7' 's' 0x0 '!' 'B' 'A' 'T' ' ' 'L' 'O' 'W'
	If Power Switch Off:
	struct {char let[9]; } data = { .let = {'P','O','W','E','R', ' ', 'O', 'F', 'F'} };
	usb_send_msg("c9s", '!', &data, sizeof(data));
	
	If Battery Low:
	struct {char let[7]; float volts; } data =
	{ .let = {'B','A','T',' ','L','O', 'W'}, .volt = [float] };
	usb_send_msg("c7sf", '!', &data, sizeof(data));

0x12 'c' 'f' '4' 'h' 0x0 'q' [float] [int16] [int16] [int16] [int16]	*usb_send_msg*("cf4h", 'q', &data, *sizeof*(data))
0x12 'c' 'f' '4' 'h' 0x0 'Q' [float] [int16] [int16] [int16] [int16]	*usb_send_msg*("cf4h", 'Q', &data, *sizeof*(data))

Every Second (if battery power is low), the board should send:

0xC 'c' '7' 's' 0x0 '!' 'B' 'A' 'T' ' ' 'L' 'O' 'W'	struct {char let[7]; float volts; } data = { .let = {'B','A','T',' ','L','O', 'W'}, .volt = [float] }; usb_send_msg("c7sf", '!', &data, sizeof(data));