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

Laboratory 5: Control

Deliverables: Control the car at the velocity level from USB commands with closed loop (PD/PID) control on car velocity.

Outcomes:

- Control the Car from the USB communications.
- Design and Implement a Z-Space closed-loop PI control for velocity.
- Convert application space (linear/angular velocities) to control space (encoder-ticks and PWM).
- Explore iterative gain tuning and effects of choices on performance and dynamic response.
- Implement trajectory-based control for position targets.

Equipment:

· Lab Kit Mouse Keyboard Monitor Internet Connection

Background:

The Zumo-Car is a skid-steer type robot where the treads can be independently controlled on the left and right side of the car. Thus, the car can achieve a desired forward speed and turning rate. However, difference in floor roughness, battery charge, minimum breakout torque, etc can affect the overall motion of the car, so closed loop control of motor speed is necessary for good performance. This labs focus is to implement closed loop control to so the car's motion can be more precisely and intuitively controlled.

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

void Controller_Init(Controller_t* p_cont, float kp, float* num, float* den, uint8_t order, float update_period);

Controller.h/c

Function Initialize_Controller sets up the z-transform based controller for the system.

void Controller_Set_Target_Velocity(Controller_t* p_cont, float vel);

Controller.h/c

Function Controller_Set_Target_Velocity sets the target velocity for the controller.

void Controller_Set_Target_Position(Controller_t* p_cont, float vel);

Controller.h/c

Function Controller_Set_Target_Position sets the target position for the controller, this also sets the target velocity to 0.

float Controller_Update(Controller_t* p_cont, float measurement, float dt);

Controller.h/c

Function Controller_Update takes in a new measurement and returns the new control value.

float Controller_Last(Controller_t* p_cont);

Controller.h/c

Function Controller_Last returns the last control command.

void Controller_SetTo(Controller_t* p_cont, float measurement);

Controller.h/c

Function Controller_SettTo sets the Filter's input and output lists to match the measurement so it starts with zero error.

void Controller_ShiftBy(Controller_t* p_cont, float measurement);

Controller.h/c

Function Controller_ShiftBy shifts the Filter's input and output lists by the desired amount. This is helpful when dealing with wrapping.

Pseudo Code:

Prepare pseudo need only address the following:

1. Controller_Init function
2. Controller_Update function
3. Converting from forward/rotation displacement/speed to encoder position/rate targets.

Assignment:

You will be implementing closed loop velocity and position control 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. Note that they are intended to control the left and right tracks separately.
2. Work out the math to convert linear and angular car velocity to track rotational speed. This will require knowing the separation distances and track-wheel diameters.
3. Implement a task in your main loop to update the left and right controllers according to the target speeds/positions. Note that, just as with PWM commands, the car should not move if the battery needs to be charged.
4. Update your response to the s and S commands to have them cancel motion controls.
5. Enable the message handling for the new ‘v’, ‘V’, ‘d’ and ‘D’ messages to send desired motions and report back on the status.
6. Using the results from lab 4, design a z-space filter/controller to perform closed loop control. Implement and choose an appropriate update rate. Make sure your loop-completion time can support that update rate!
7. Perform testing to verify d and D move the distances desired. Some calibration of your math from 2 may be required.
8. *(Optional)* Drive the car around using an X-Box or PS controller plugged into the USB port on the PI. How does it do? (use the serial_monitor_joy.py script)
9. *(Extra Credit 3pts)* Implement the task-scheduler to control operations instead of message flags. Discuss in your report the pros and cons of both approaches.

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 defined distances and rotation amounts (pi for example).
    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. Include the math and measurements for the forward/angular speed to wheel speed calculations.
    c. Discuss how you choose and tuned your controller gains.
    d. Discuss the performance in position mode vs velocity mode. Did you need to do anything to have position mode work well?

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) PWM_L PWM_R Encoder_L Encoder_R [float] [int16_t] [int16_t] [int16_t] [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.
Lab 5	d	0x64	9	cff	Specifies the distance to drive (linear followed by angular).
	D	0x44	13	cfff	Specifies the distance to drive (linear followed by angular), terminates after X milliseconds as specified by the third float. If the third float is negative, the car shall stop.
	v	0x76	1	c	Specifies the speed to drive (linear followed by angular).
	V	0x56	5	cf	Specifies the speed to drive (linear followed by angular), terminates after X milliseconds as specified by the third float. If the third float is negative, the car shall stop.

Device -> Host Communication Messaging Definition

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

	CMD char	Response Format	Function Call
Lab 1	~		
	*	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))
	UNDEF	0x5 'c' 'c' 0x0 '?' [char]	*usb_send_msg* ("cc", '?', &data, *sizeof* (data))
Lab 2	t	0xA 'c' 'H' 'f' 0x0 't' [uint8] [float]	*usb_send_msg* ("cHf", 't', &data, *sizeof* (data))
	T	0xA 'c' 'H' 'f' 0x0 'T' [uint8] [float]	*usb_send_msg* ("cHf", 'T', &data, *sizeof* (data))
Lab 3	e	0xD 'c' 'f' 'f' 0x0 'e' [float] [float]	*usb_send_msg* ("cff", 'e', &data, *sizeof* (data))
	E	0xD 'c' 'f' 'f' 0x0 'E' [float] [float]	*usb_send_msg* ("cff", 'E', &data, *sizeof* (data))
	b	0x8 'c' 'f' 0x0 'b' [float]	*usb_send_msg* ("cff", 'e', &data, *sizeof* (data))
	B	0x8 'c' 'f' 0x0 'B' [float]	*usb_send_msg* ("cff", 'B', &data, *sizeof* (data))

Lab 4	p	x0 '!' 'P' 'O' 'W' 'E' 'R' ' ' 'O' 'F' 'F'0xC 'c' '7' 's' 0x0 '!' 'B' 'A' 'T' ' '	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));

	P	x0 '!' 'P' 'O' 'W' 'E' 'R' ' ' 'O' 'F' 'F'0xC 'c' '7' 's' 0x0 '!' 'B' 'A' 'T' ' '	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));

	s		
	S		
	q	0x12 'c' 'f' '4' 'h' 0x0 'q' [float] [int16] [int16] [int16] [int16]	*usb_send_msg* ("cf4h", 'q', &data, *sizeof* (data))
	Q	0x12 'c' 'f' '4' 'h' 0x0 'Q' [float] [int16] [int16] [int16] [int16]	*usb_send_msg* ("cf4h", 'Q', &data, *sizeof* (data))
Lab 5	d	Same as p/P	Same as p/P
	D	Same as p/P	Same as p/P
	v	Same as p/P	Same as p/P
	V	Same as p/P	Same as p/P

Every Second (only if battery power is low), the board should send the low battery warning.