TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

Texas Instruments

Module 17

Lab 17: Control Systems

Lab: Control Systems

17.0 Objectives

The purpose of this lab is to develop a control system. In this module,

1. You will combine input capture measurements from Timer A3 and PWM outputs with Timer A0.
2. You will develop a system to control the speed of the two motors.
3. You will evaluate the performance of the control system.

Good to Know: Control systems are a rich and complex field within engineering spanning: electrical engineering, aerospace engineering, mechanical engineering, and computer engineering. This module provides a brief introduction.

17.1 Getting Started

17.1.1 Software Starter Projects

In addition to your solutions to Labs 13 and 15, look at this project:
Lab17_Control (starter project for this lab)

Note: Similar to Lab 14, you will find noise is a major problem for control systems. Continue to monitor the stability and accuracy of the tachometer measurements during this lab. Jittery measurements will cause even the most robust control system to fail.
The second issue with control systems is delay. Consider the closed loop between motor power -> motor speed -> tachometer measurement -> controller execution -> new duty cycle output. Delays within this loop (e.g., low pass filtering, slow controller execution rate) can cause the system to be unstable. Unstable systems produce oscillations.

17.1.2 Student Resources (in datasheets directory-Links)

MSP432P4xx Technical Reference Manual, Timer_A (SLAU356)
MSP432P401R Datasheet, msp432p401m.pdf (SLAS826)
Data sheets for TI-RSLK chassis board and sensors

17.1.3 Reading Materials

Chapter 17, “Embedded Systems: Introduction to Robotics"

17.1.4 Components needed for this lab

All the components you need in the lab are provided in the TI-RSLK MAX kit (TIRSLK-EVM). A portion of the lab will require adding Sharp distance IR sensor kit. Batteries will be needed to power your robot.

Quantity	Description	Manufacturer	Mfg P/N
1	TI-RSLK MAX kit	TI	TIRSLK-EVM
1	Sharp Distance sensor kit	Pololu	#3677

17.1.5 Lab equipment needed

Oscilloscope (one or two channels at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling

17.2 System Design Requirements

The goal of this lab is implement a control system to independently set the speed of the two motors. Let X^* be the desired speed (the units of X^* should match the units of the speed measurements obtained in Lab 16). Let $X'(t)$ be the estimated speed as implemented in Lab 16. We define the **controller error**, $e(t)$, to be the difference between the desired and estimated speed:

$$e(t) = X^* - X'(t)$$

The minimum desired speed should be larger than minimum speed measurable with your input capture system. The maximum desired speed should be the speed on the ground when the robot is moving with a duty cycle of 90%. The controller should be **stable,** meaning the robot moves with approximately constant speed. An **unstable** controller exhibits widely varying speeds oscillating between fast and slow.

The **accuracy** of the controller will be limited by the accuracy of the tachometer measurements. You will be required to measure accuracy, which we define as the average steady state error, but there is no requirement for this lab, that the accuracy be less than a specific value.

The **stability** of the controller will be determined by the stability of the tachometer measurements and by the parameters of the controller. You will be required to measure stability, which we define as the standard deviation of the error, but there is no requirement for this lab that the stability be less than a specific value.

The **time constant** of the controller is defined as the time it takes to reach $(1\text{-}e^{-1})$ = 0.63 of the final speed given a change in desired speed. For example, if the current and desired speeds are 100 RPM and the desired speed is changed to 200 RPM, then the time constant is the time it takes to reach 163 RPM. You are required to measure time constant, but there is no requirement for this lab that the time constant be less than a specific value.

17.3 Experiment set-up

The construction of the robot has been performed in lab 5, software to adjust power to the motor was developed in Lab 13, and software to measure motor speed was developed in Lab 16. The following tables list pin connections for the tachometer and motor drivers. More details are provided in the user guides and datasheets.

Warning: Please ensure the +5V jumper on the MSP432 LaunchPad is disconnected or removed. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

LaunchPad	TI-RSLK chassis board	DRV8838	Description
P5.5	DIRR	PH	Right Motor Direction
P3.6	nSLPR	nSLEEP	Right Motor Sleep
P2.6	PWMR	EN	Right Motor PWM
P5.4	DIRL	PH	Left Motor Direction
P3.7	nSLPL	nSLEEP	Left Motor Sleep
P2.7	PWML	EN	Left Motor PWM

LaunchPad	TI-RSLK chassis board	Encoder	Description
P10.5/TA3CCP1	ELA	OUT A	Left Encoder A
P5.2/GPIO	ELB	OUT B	Left Encoder B
P10.4/TA3CCP0	ERA	OUT A	Right Encoder A
P5.0/GPIO	ERB	OUT B	Right Encoder B

17.4 System Development Plan

17.4.1 Selection of the controller period

You will run the controller at a fixed rate using a periodic interrupt. Similar to sampling, running the controller at a regular rate allows you to implement digital signal processing. Let Δt be the period of the interrupt. For example, the integral equation

$$u(T) = \int_0^T a * e(t)dt$$

can be approximated as

$$u(T) = \sum_{n=1}^{T/\Delta t} a * e(n\Delta t)\Delta t$$

and implemented more simply as

$$u = u + a * e * \Delta t$$

There are multiple factors to consider when choosing a controller rate:

- The rate does not need be faster than the rate at which new speed data are obtained.
- Running the controller faster than the input rate is a waste of processor time because the controller equations will be executed multiple times with the same input data.
- The controller rate must be faster than the response rate of the motors. One rule of thumb is to choose the time interval for running the digital controller about 10 times slower than the time constant of the motor.

- Running the controller slower than the response time of the motor leads to instability.
- Running the controller faster will consume more processor time; running the controller slower allows for low pass filtering of the input data.

Note: Write your control software so it is easy to adjust the controller rate. This way you can experimentally test which rates work well for your robot.

17.4.2 Integral Controller

Write the two integral controllers that will run periodically within an Interrupt Service Routine (ISR). Use global variables to pass data into the controller. The two inputs to the left motor controller are **XstarL** (the desired speed) and **XprimeL** (the estimated speed). The output of the controller is the PWM duty cycle **UL** (e.g., 2 to 14998). For the left motor perform steps 1 – 5:

1. Read desired left motor speed: **XstarL**
2. Collect estimated left motor speed: **XprimeL**
3. Calculate error: **ErrorL** = **XstarL**- **XprimeL**
4. Calculate integral: **UL** = **UL** + (**A***__ErrorL__)/1024
5. Antireset windup: make sure 2 ≤ **UL** ≤ 14998

where **A** is a constant that defines the behavior of the integral controller. Perform similar steps for the right motor. Use signed 32-bit integer math.

After running the controller for each motor send outputs to the motor driver

Motor_Forward(UL,UR);

Compare the theoretical integral to the software implementation. The theoretical to software

$$u = u + a * e * \Delta t \leftrightarrow \textbf{UL} = \textbf{UL} + (\textbf{A}*\textbf{ErrorL})/1024$$

From this comparison you can see the software constant **A** is equivalent to *a*Δt*/1024.

Note: Consider the complete loop (motor power -> motor speed -> tachometer measurement -> controller execution -> new duty cycle output). Some delays are unavoidable, like the response time of the motor.

17.4.3 Tune the controller

Perform your initial tuning with the robot on blocks so the wheels do not touch the ground. For the initial value of **A**, take a large error value of 100 RPM and match it to a large change in duty cycle 10% (1500/15000). For example

A = 1024*1500/100 = 15,360

Start with a desired speed that you estimate to require a duty cycle of 50%. The first tuning will be for stability. Run the controller, and if the speed eventually stabilizes to more or less a constant then define it as stable. We define stability as the standard deviation of the error once it has reached steady state. It is unstable if

- The motor stops (0% duty cycle)
- The motor runs full speed (100% duty cycle)
- The motor oscillates fast and slow.

Saturated responses (stopped or full) are probably a result of a software bug or the sign that **A** is incorrect. Oscillations are probably a result of the **A** being two large. When initially searching for the best value of **A**, double or half the values of **A**, so you can quickly cover a wide range of values.

Once you have found a range of values that are stable, next you will tune for accuracy (average steady state error) and time constant (how quickly it stabilizes). Again, run this motor test on blocks so the wheels do not touch the ground. We define the **time constant**, τ, of the motor as the time it takes to achieve $(1\text{-}e^{-1})$ = 0.63 of the final speed, given a step change in desired speed to the motor. Read a switch on the LaunchPad and use this operator input to change the desired speed from typical (requiring about 50% duty cycle) to fast (requiring about 75% duty cycle). Use the debugger to observe error while running. If you performed Lab 11, you could plot speed versus time on the LCD.

Experiment to find the minimum and maximum speeds at which the controller is still stable and accurate. For this test run the robot on the ground. The goal is to run as straight as possible at more or less a constant speed.

17.4.4 Performance Evaluation

Write a test program that periodically collects motor speeds each time the controllers are run. Include the bumper driver from Lab 10 or Lab 14 so the robot

stops on a collision. Dump desired speed, power (duty cycle) and speed data into buffers similar to Lab 10. For very long tests, you can dump into flash ROM. For shorter tests, you can dump into RAM. In the main program, perform these steps running the robot for 10 seconds.

1. Run forward at medium speed for 3 seconds
2. Run forward at fast speed for 4 seconds
3. Run forward at medium speed for 3 seconds
4. Stop the motors and stop the recording

Run this motor test on blocks and on a flat surface. We define the **time constant**, τ, of the motor as the time it takes to achieve $(1\text{-}e^{-1}) = 0.63$ of the final speed, given a step change in desired speed. Fit the speed versus time data to an exponential to estimate the time-constant of your controller.

$$y(t) = S_0 + \Delta S\, e^{-t/\tau}$$

where S_0, ΔS, and τ are least squares fit of the y(t) speed data verses time. Initial time is defined at the point the desired speed was changed.

17.5 Troubleshooting

Controller will not stabilize:

- Check the sign of **A** (too slow means increase duty cycle)
- Check the stability of the speed measurements given a constant duty cycle. If the inputs are noisy, the controller cannot function.
- Try an incremental controller. Let **K**=10 be a constant. Add **K** if too slow, subtract **K** if too fast.
- Check for overflow at multiply (**A*****ErrorL**). If this exceeds $\pm 2^{31}$, then reduce A and 1024.
- Check for underflow at divide by 1024. You will get zero if **A*****ErrorL**<1024. To solve, increase A.

Motors are very different:

- A little difference (±25%) is normal. A big difference may be friction or a bad motor.

17.6 Things to think about

In this section, we list thought questions to consider after completing this lab. These questions are meant to test your understanding of the concepts in this lab. The goal of this module is for you to understand Timer_A and its use for measuring period.

- In this lab we tuned the controller empirically. Why did we not use a mathematical model for the motor, and solve the optimal control parameters theoretically?
- There are three performance measures (accuracy, stability, and time constant) and only two adjustable tuning parameters (controller rate and A). From an engineering perspective what are the consequences of having so few parameters? Think about advantages and disadvantages of having only two parameters.
- What happens to your controller if the motor spins too slowly, e.g., less than 30 RPM?
- What happens to your controller if the motor stops, e.g., does not spin at all?
- How do we debug this system if the robot is moving along the ground?
- Why do performance measures (accuracy, stability, and time constant) differ if the robot is on blocks versus on the ground?

17.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. For example,

- If your robot has IR distance sensors, then you could design a robot that travels down the middle of the road. Assume the left and right IR sensors are measuring distance to the left and right walls, see Figure 1. For this controller we define error as the difference between distance to left and right. **Error** = $\mathbf{D_l}$-$\mathbf{D_r}$. Set the duty cycle of one wheel to a constant, and have the output of the controller determine the duty cycle of the other wheel.

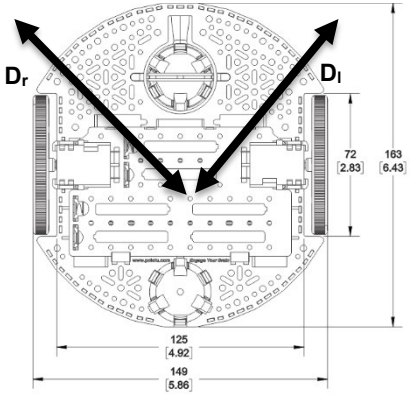

Figure 1. Define distance measured from a central point on the robot.

- If you completed Lab 11, add LCD outputs for each of the test functions. Remember to perform LCD output only in the main program and not during an ISR.
- If your robot has a line sensor, then you could design a robot that follows the line. Recall the output parameter for the **reflectance.c** driver in labs 6 and 10 was a number, where 0 meant on the line, positive numbers mean off center in one direction and negative numbers mean off center in the other direction. For this controller we define error simply as this reflectance measurement. Set the duty cycle of one wheel to a constant, and have the output of the controller determine the duty cycle of the other wheel.
- To improve time constant without affecting accuracy or stability, you could add a proportional term, implementing a PI controller.

1. Read desired left motor speed: **XstarL**
2. Collect estimated left motor speed: **XprimeL**
3. Calculate error: **ErrorL** = **XstarL**- **XprimeL**
4. Calculate integral: **UIL** = **UIL** + (**A***__ErrorL__)/1024
5. Antireset windup: make sure 2 ≤ **UIL** ≤ 14998
6. Calculate proportional: **UPL** = (**B***__ErrorL__)/1024
7. Combine: **UL** = **UIL**+**UPL**
8. Constrain: make sure 2 ≤ **UL** ≤ 14998

17.8 Which modules are next?

After this module, you are ready to solve any of the robot design challenges. If you wish to extend your robot to include wireless communication you have two options:
Modules 18 and 19) Add Bluetooth functionality.
Module 20) Add Wifi functionality.

17.9 Things you should have learned

In this section, we review the important concepts you should have learned in this module:

- Understand how the controller allows you to manage the uncertainties of friction.
- Know how to tune a digital controller empirically.
- Know how to use interrupts to build complex real-time systems. From a systems standpoint, your robot now has many components: bumper switches, line sensor, LCD, IR distance sensor, tachometer, and digital controller (PWM). You used a single main program for the non-real-time tasks like the LCD and operator buttons, but used interrupts for the real-time tasks.

ti.com/rslk

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265