TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

TEXAS INSTRUMENTS

Module 7

Introduction: Finite State Machine

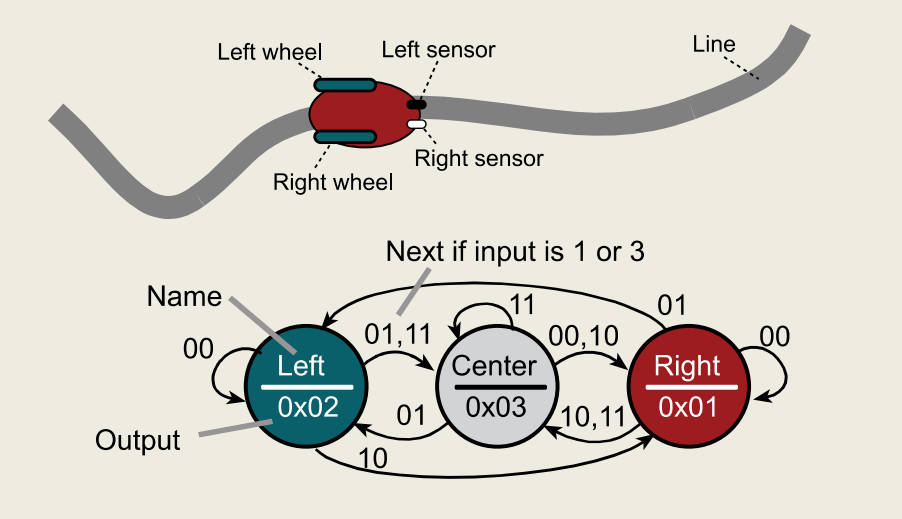
Introduction: Finite State Machine

Educational Objectives:

REVIEW C programming
UNDERSTAND variables, numbers, pointers, structures, arrays
DEVELOP debugging techniques
LEARN how to solve problems with finite state machines
DESIGN, BUILD & TEST A SYSTEM
Controller for a line tracking robot

Prerequisites (Modules 1, 4, and 6)
- Running code on the LaunchPad using CCS (Module 1)
- Basic C programming (Module 4)
- GPIO (Module 6)

Recommended reading materials for students:
- Chapter 7, **Embedded Systems: Introduction to Robotics**, Jonathan W. Valvano, ISBN: 9781074544300, copyright © 2019



Software abstraction allows us to define a complex problem with a set of basic abstract principles. We can then construct a system solution using these abstract building blocks. Using the abstraction gives us a better understanding of both the problem and its solution. This is because we can separate what the system does (policies) from the details of how the system works (mechanisms). This separation simplifies the design process by first describing what the system does, and then we can translate the description into a system that implements that description. Abstraction provides for a proof of correct function and simplifies both extensions and customization. The abstraction presented in this section is the Finite State Machine (FSM). The abstract principles of FSM development are the inputs, outputs, states, and state transitions. The FSM state transition graph (STG) defines the time-dependent relationship between its inputs and outputs. If we can take a complex problem and map it into a FSM model, then we can solve it with simple FSM software tools. Our FSM software implementation will be easy to understand, debug, and modify.

The problem is mapped into a well-defined model with a set of abstract yet powerful rules. Then, the software solution is a matter of implementing the rules of the model. In our case, once we prove our software correctly solves one FSM, then we can make changes to the state transition graph and be confident that our software solution correctly implements the new FSM.

Embedded systems are often deployed in safety critical systems. In these situations we must certify the solution operates exactly as intended. An abstract approach like a finite state machine (FSM) allows us to separate what it does from how it works. The complexity of a FSM is in the state transition graph, while the controller should be trivially simple. Once we certify the low-level controller is operational, we can verify the system at a high or abstract level.

In the lab associated with this module, we will use a finite state machine to create a controller for a simple line following robot. Inputs will come from two switches (simulating two line sensors) and outputs will go to two LEDs (simulating two motors on a differential drive robot). The goal of the controller is to follow the line. The purpose of this lab is to provide another lab on C programming, and serve as an introduction to robot control. In a previous module (**6. GPIO**), you interfaced an actual line sensor. Other labs will provide additional sensors for the robot controller. In **10. Debugging** you will add bumper switches. In **15. ADC** you will add IR distance sensors. In **17. Tachometer** you will add tachometers to measure wheel speed. These sensor measurements could be used as inputs to a FSM controller. In **12. DC Motors** and **13. Timers** you will interface the robot motors, which will be the outputs of the real robot controller.

The basic approach to system development is to create components and then piece the components together to create the system. In this module, you will learn how to use FSMs as a central controller for the system.

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