EE 472 Lab 5

Winter 2015

Final Project

University of Washington - Department of Electrical Engineering

Mitch Ishihara, Joshua Fromm, Tien Lee

Lab Objectives:

In this lab you will tie together all the previous work done in the class to finalize the RoboTank system. A functional system will require two physically separate sub-systems that work together, a remote controller subsystem and the tank subsystem, that communicate over Bluetooth.

The demo for this lab will be the demonstration of a working RoboTank system, that your team can drive around the lab. You are allowed near total creative control of how your team implements the RoboTank sytem, however you **must** meet the specifications provided in Lab 4 as well as the additional requirements in this lab.

Due to cost restraints, there is only one tank for every two groups. It is recommended you team up with one other group to share a tank. You can develop the entirety of your system **before** downloading it to the tank and verify it works through the debugger. This means the tank should only be used for your final testing. Try not to hog it from the other group.

Recommended Reading:

- Bluetooth modem datasheet and user manual
- Motor driver datasheet
- LM3S8962 UART section of the datasheet

Specifications:

All the requirements for this lab have already been enumerated in detail in lab 4. This week we will simply be adding to the functionality of the overall RoboTank system. The report for this lab may be an expansion of your group's lab 4 report, with the addition of fully commented code. Note that there will be two distinct IAR projects, one for the remote controller subsystem and one for the tank subsystem; both shall use FreeRTOS. The respective project will be downloaded onto one of the two boards that make up the finished RoboTank system.

If you partitioned your design well into functional sub-systems in your previous labs (kept things very well separated and modular), it should be quite easy to reuse and integrate the right parts for each of the two physically separate subsystems. You should document each of the two subsystems fully; this means including an amended Functional Design Diagram and UML System Data and Control Flow Diagram for each. However, each of the physically separate subsystems may be a version of lab 4; you may make a simply design decision to snip out parts of that diagram to repartition what you need for each physically separate subsystem or you may decide to do something more sophisticated.

The control version shall monitor user input on the buttons or keypad and send strings to the Bluetooth when they are pressed. It shall also display information about the current state of the tank. The tank side shall receive Bluetooth strings, control the motors, and monitor the distance sensors. Note that the controller side shall not have any code for the distance sensors or motors and the tank side may (or may not) check the keypad or display information. Each of these two systems should have full documentation similar to that in Lab 4. How you

implement these systems is up to you so long as they meet the basic specifications given in Lab 4.

Goals:

- Assemble the hardware for your RoboTank
- Configure and incorporate a Bluetooth module into your existing code
- Split the code into remote side and tank side subsystems
- Write a simple Bluetooth communication interface between the two systems
- Download your code and drive a tank

Tank Hardware:

Each tank consists of the following parts

- A chassis with treads and wheels
- 2x Bluetooth modems
- 1x dual channel motor driver
- 1x AA to USB power supply
- Lots of batteries

You are in charge of constructing your tank, although all the needed parts will be provided. Feel free to incorporate any other hardware you like for extra credit. To assemble the tank you should start by attaching the wheels and treads to the chassis. The motor driver is used to interface between your Stellaris board (which doesn't have enough power to drive a motor on its own) and the tank's electric motors. You should have already coded the control logic for the driver in lab 4, so hooking it up is just a matter of wiring. The AA to USB power supply allows a Stellaris board to be conveniently powered while in the tank and is very easy to connect. Finally, the Bluetooth modem provides a communication channel between the two systems and should be hooked up to the board's UART. Note that soldering may be useful to help keep things in the tank together. It is recommended that you don't assemble the tank until you are fairly certain your code is finished, since taking it apart for significant changes will become annoying quickly.

Datasheets:

The datasheets for the important hardware, the Bluetooth modem and dual channel motor driver is provided in the homework folder. It is recommended you familiarize yourself with both of these documents.

Bluetooth Configuration:

The first thing that needs to be done for this lab is getting comfortable with your Bluetooth module. Each group will be provided a single module, but you can simulate having another by using a smartphone or Bluetooth dongle to send and receive messages from your module. If a member of your group has Android, download Bluetooth SPP pro. There is probably some equivalent iOS software. This app allows you to send raw characters over Bluetooth and will be extremely useful for testing. After providing power to your module, open the app and confirm that a Bluetooth device is detected (this might be kind of tricky if everyone in the lab is doing it at the same time). Pair with the device and you can begin sending strings. As noted in the datasheet, sending the string '$$$' should cause the Bluetooth to enter command mode and report the message 'OK'. You can then issue the configuration commands in the datasheet. When debugging the controller side of the board, you can monitor the app to confirm that messages are sent as you expect. Similarly, when debugging the tank side of the board, you can send messages your controller would. This allows you to fully test your code with only a single board.

Bluetooth Code Development:

As noted in the Bluetooth datasheet, all interface with the device is performed over UART, which is a standard serial communication bus. The LM3S8962 has a built in UART module that requires only minimal configuration to operate. You can do this configuration using either direct register access or the provided IAR functions. All transactions with the Bluetooth modem will use strings, so the supporting code can be quite simple. You need to write two sets of UART software. The controller version needs to have functions that are called when a specific command is issued that send a control string to the modem for transfer. The motor version needs to monitor for available Bluetooth data and read the modem when it is available. The controller version of the software will be extremely simple since writing data requires very little overhead. The motor version is a little trickier since you need to figure out when data is available. You can do this either with polling in a task or by implementing a UART data available interrupt. Additionally, the motor version must have some sort of simple parsing code to correlate received strings to actions. Test both versions of your code thoroughly before continuing.

Autonomous Mode:

Although it is not required, implementing autonomous driving of the tank will be worth a substantial amount of extra credit and also be really cool. The depth of your autonomous mode is up to you, but it could be as simple as turning before hitting something.

Demo:

You must demonstrate your working tank to a TA. You should be able to easily drive your tank around and explain the interface. Remember that this is a finished product, and using it should be intuitive and fun.

Write Up:

It's been said before and it'll be said again: you must provide full documentation of your system for this final project. This includes a completed spec sheet as provided in Lab 4, fully documented code, and a comprehensive user manual. Your grade will be heavily dependent on this documentation so do your best to make it good.

In addition, a response to the following from your group is required to be included and may be supplied as an appendix in your lab write up.

Implementing 'safe' inter-task (or inter-process, inter-thread) communication and synchronization is an important concept in multitasking (or multiprocessing, multithreading) systems. Discuss 'safe' inter-task (or inter-process, inter-thread) communication and synchronization in some detail, explain why it is important, and provide some examples.