LAB 4

COMMUNICATION USING THE UART AND SENSOR DATA TRANSFER

INTRODUCTION

This week in lab you will collecting sensor measurement data from the CyBot using a precompiled scan function and sending this data to the PC for analysis. This functionality will be very useful for the lab project. Data are sent between the CyBot and PC using UART serial communication. You will also use some precompiled code for the UART, using functions for sending and receiving data and for some of the initialization of the UART. You will write code for part of the initialization of the UART – the part that configures the GPIO port for the UART alternate function. So the focus of this lab remains on configuring GPIO port registers to use port pins for input/output. In Lab 5, you will fully configure the UART and write your own functions, rather than use precompiled functions. In Labs 6 and 7, you will learn more about the sensors, the I/O peripherals of the microcontroller that are connected to them, and how to configure and use the I/O interfaces to read the sensors. This lab will help you get started understanding the sensor data and how the data can be analyzed for object detection.

REFERENCE FILES

The following reference files will be used in this lab:

- lab4_template.c, contains a main function template that you will implement for this lab
- cyBot_uart.h, header file for pre-compiled CyBot-PC UART communication library
- libcybotUART.lib: pre-compiled library for CyBot-PC UART communication (note: must change extension of file from .txt to .lib after copying)
- cyBot_Scan.h, header file for pre-compiled library for CyBot sensor scanning
- libcybotScan.lib: pre-compiled library for CyBot sensor scanning (note: must change extension of file from .txt to .lib after copying)
- lcd.c, program file containing various LCD functions
- lcd.h, header file for lcd.c
- timer.c, program file containing various wait commands
- timer.h, header file for timer.c
- open_interface.c, API functions for basic Open Interface functions
- open_interface.h, header file for open_interface.c
- TI Tiva TM4C123G Microcontroller Datasheet
- TI TM4C123G Register Definitions C header file: REF_tm4c123gh6pm.h
- Cybot baseboard and LCD schematics: Cybot-Baseboard-LCD-Schematic.pdf
- GPIO Reading Guide: reading-guide-GPIO.pdf
- UART Reading Guide: reading-guide-UART.pdf

NOTE: Be sure to use the cybotUART files provided for Lab 4. This is a new library with new functions compared to the library provided for the UART part in Lab 3. Notice the functions in the header file. In Lab 4, use these new cybotUART header and library files.

The code files are available to download from a Google Drive folder:
https://drive.google.com/drive/folders/1LPneqbbOkbS_1rPpEeJsEhUi4gdsLAk5?usp=sharing

PRELAB

See the prelab assignment in Canvas and submit it prior to the start of lab.

STRUCTURED PAIRING

You are expected to continue to use structured pairing in this lab and in future labs. It was introduced in Lab 2.

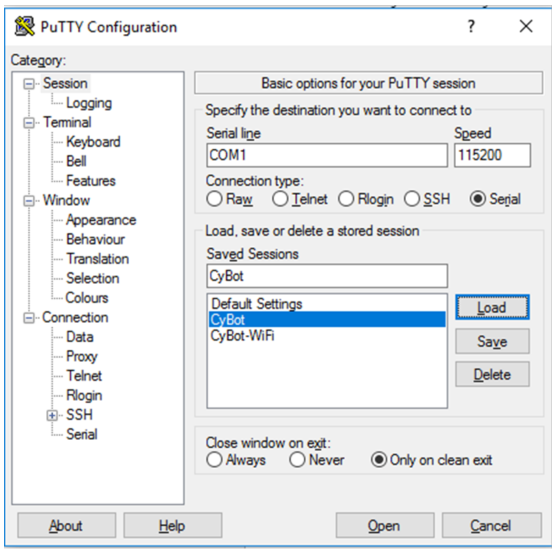
PART 1: CYBOT COMMUNICATION USING PUTTY

You may have completed CyBot-PuTTY communication during Lab 3. You should start by confirming that you can send messages to the desktop PC from the CyBot. The following instructions were given in Lab 3.

1. Look over cyBot_uart.h. This file describes functions you will want to use to communicate between the PC and your program on the Cybot. The communication will use a UART serial port on the Cybot (UART1), UART serial port on the PC, RS232 serial cable, and a PC application called PuTTy. Note: pay special attention to the description of the functions and the implications of their behavior.

RS232/UART serial cable:

PuTTy settings:



Basic options for your PuTTY session

Specify the destination you want to connect to

Serial line: COM1 Speed: 115200

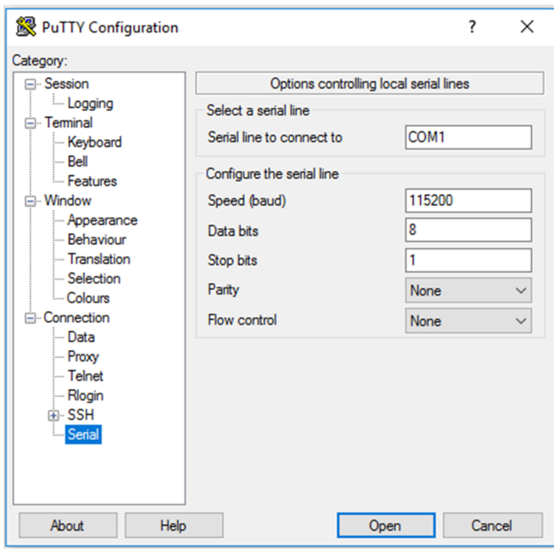
Connection type: ○ Raw ○ Telnet ○ Rlogin ○ SSH ● Serial

Load, save or delete a stored session

Saved Sessions: CyBot

Default Settings
CyBot
CyBot-WiFi

Close window on exit: ○ Always ○ Never ● Only on clean exit



Options controlling local serial lines

Select a serial line

Serial line to connect to	COM1

Configure the serial line

Speed (baud)	115200
Data bits	8
Stop bits	1
Parity	None
Flow control	None

2. Use the code from Lab 3 to test the UART communication, or write a short test program that sends messages back and forth.

CHECKPOINT:

You should be able to send and receive messages between the Cybot and PC.

PART 2: UART INITIALIZATION

In Lab3, you made use of a UART device to send information between PuTTY and the CyBot. You were given the function called `cyBot_uart_init()`. For this part, you will write code for the first half of this initialization process, which involves setting up the GPIO portion of the microcontroller. We are providing you with the last half, called: `cyBot_uart_init_last_half`. You will replace your call to `cyBot_uart_init()` with the following code that you need to complete.

```
cyBot_uart_init_clean(); //Clean UART initialization

// Complete code for configuring the GPIO PORTB part of UART1 init
SYSCTL_RCGCGPIO_R |= FIXME;
while ((SYSCTL_PRGPIO_R & FIXME) == 0) {};
GPIO_PORTB_AFSEL_R |= FIXME;
GPIO_PORTB_PCTL_R &= FIXME; //Force 0's at desired locations
GPIO_PORTB_PCTL_R |= FIXME; // Force 1's at desired locations
GPIO_PORTB_DEN_R |= FIXME;
GPIO_PORTB_DIR_R &= FIXME;  // Force 0's at desired locations
GPIO_PORTB_DIR_R |= FIXME;  // Force 1's at desired locations

cyBot_uart_init_last_half();  // Complete UART device configuration
```

The above code must be used to replace the call to `cyBot_uart_init` **you made in Lab 3. Your job is to replace the FIXME's with proper values. Make sure to copy libcybotUART.lib into your project.**

Suggestion: Make use of the last part of Lab 3 to help you complete the code for this part of the lab.

CHECKPOINT:

Send and receive data between the CyBot and PC, like in Lab 3, using the code you developed for initializing the first/GPIO half of the UART interface.

PART 3: SENDING SCAN DATA TO PUTTY

As part of the lab project, the CyBot will collect data about the environment around it using a servo motor that scans side to side while collecting distance information using two sensors, an ultrasound (PING) sensor and an infrared (IR) sensor. Copy **libcybotScan.lib** into your project, and take a look at the **cyBot_Scan.h** header file and notice the function to rotate the servo motor and collect sensor information in a struct. The function takes a reading from the PING sensor (distance in centimeters) and IR sensor (raw IR sensor information; refer to the IR sensor datasheet for more information on how to interpret its raw information).

When the Cybot receives an 'm' from PuTTY, have the servo motor scan from 0 degrees to 180 degrees. Take a measurement at every 5-degree increment. Send this information back to PuTTY as it is collected, and display it in the following format. Use a tab to separate each column.

Angle	PING distance	IR raw value
0	?	?
5	?	?
10	?	?
…		
180	?	?

Note: The function for scanning and collecting data has not been calibrated to your specific robot. So it is likely that 0 and 180 degrees are off from actual rotation of the motor (it could be off by a lot). In Lab 8, you will be calibrating your own version of the servo motor functionality.

Tip: In order to format your data in PuTTY, you can send '\r' to return, '\t' for tab, and '\n' for newline.

You can position one or more stationary objects in front of the CyBot, such as between 10 cm and 50 cm away. You can optionally set up the wireless WiFi connection with the CyBot using the instructions in the UART/WiFi Board Quick Reference Sheet document.

CHECKPOINT:

Send an 'm' command to the CyBot from PuTTY to start a scan, and display formatted sensor information in PuTTY.

PART 4: SCAN DATA ANALYSIS (OPTIONAL)

Data analysis is essential in order to use the sensor data to make decisions. See this video for a demonstration of an older lab project showing the CyBot navigate a test field, detecting, recognizing and avoiding objects, as appropriate, to reach its destination (white square): https://www.youtube.com/watch?v=ulidN0rs1NA
Notice the sensors scanning the region in front of the robot.

PING))) Sensor

The range of the PING))) SONAR sensor is 2 cm to 3 m. SONAR (SOund Navigation And Ranging) uses an ultrasonic burst (well above human hearing range) to determine the presence and distance of objects. SONAR is a term that is valid for both air and water. The frequencies of the pulses emitted are different for these two mediums. We will be implementing a primitive SONAR. The sensor emits 40 KHz pulses and estimates distance by using the fact that objects reflect sound. The distance is calculated by determining the time between emitting a sound pulse and receiving an echo. The echo is translated into distance given the speed of sound in a particular medium. In our case, the medium is air. While air temperature does affect the speed of sound, for our purposes we will assume the speed of sound to be constant at 340 m/s or 1130 ft/sec. The PING sensor will only report one echo for any given pulse. This one echo is the first received to meet the minimum threshold used to discriminate between noise and a proper echo. The pulse does spread as it travels away, so the first reflection may come from an object that is not directly in front of the sensor. Clutter **does** affect the usefulness of the sensor.

The composition and shape of objects affects the amount of sound that gets reflected back to the sensor. A soft material like fabric will absorb more sound than a hard material like steel. It is possible to angle a box so that sound waves will reflect away from the sensor, so the box may "disappear" like the B2 or F117 US Air Force airplanes.

IR Sensor

The InfraRed sensor is an electro-optical device that emits an infrared beam from an LED and has a position sensitive detector (PSD) that receives reflected IR light. A lens is positioned in front of the PSD in order to focus the reflection before it reaches the sensor. The PSD sensor is an array of IR light detectors, and the distance of an object can be determined through optical triangulation. The location of the focused reflection on the PSD is translated to a voltage that corresponds with the measured distance. The IR distance sensor is designed to measure distances from 9 – over 80 cm. However, the IR sensor can only fairly accurately display a distance value for an object 9 – 50 cm away. Objects must be placed far enough apart to provide a measurable gap between them. This gap may not be measurable by the PING))) sensor, but the tighter beam of the IR distance sensor will generate data consistent with a physical gap.

The datasheets for these sensors are provided as reference files. These sensors will be examined in more detail in Labs 6 and 7.

Using sensor data, we are interested in determining (1) distance to the object, (2) angular location of the object, and (3) size of the object. Being able to collect good quality data and analyze it is important, as it will be critical to successful navigation in the upcoming lab project. Object detection is done by distinguishing between distance measurements that pertain to objects in contrast to background noise. Distance measurements are needed to detect the presence of an object, and from these measurements, the edges of the object can be estimated at particular angular positions. These positions can be helpful in estimating the center of the object for distance, width, and location measurements, which will be useful for navigating the CyBot.

Transfer the data from PuTTY to a file and visualize it using graphs.

1. Configure PuTTY as you normally would but select “Session > Logging”
2. Under file name, click “Browse”, then navigate to where you want the output saved (the desktop might be a good option).

Now when you scan and send the data, it will also be output to the file that you specified. Once you have successfully saved the data to a file, use Excel, Matlab or another tool to plot the data. You can choose which data analysis tool you want to use.

Tip: Try plotting the data as a radar graph. Note that you may need to adjust the scale of the data that is displayed.

As you plot the data, look for:

1. Object distance
2. Angular location: For example, if the robot starts seeing an object at servo angle 30 degrees and stops seeing it at 35 degrees, its size is 5 degrees.
3. Object width: The actual linear width can be found using geometry/trigonometry principles. This will be important for the final project since objects with the same radial width value may have different linear widths (e.g., bigger object farther away than smaller object have the same radial width).

CHECKPOINT:

Graph a 0-180 degree set of scan data. You may want to plot the IR sensor data in a graph separate from the PING))) sensor data, separately view them, and then view them in the same graph. Do the data and graphs give you expected information about the objects in front of the robot?

DEMONSTRATIONS:

1. **Functional demo of a lab milestone –** The TA will announce the specific milestone at the start of lab. The milestone is based on the checkpoints.
2. **Debug demo using debugging tools to explain something about the internal workings of your system –** The TA will announce any specific debugging requirements at the start of lab; otherwise you will create your own debug demo based on your needs and interests in the lab.
3. **Q&A demo showing the ability to formulate and respond to questions –** This can be done in concert with the other demos.