### IR DISTANCE MEASUREMENT

## Introduction

This week in lab you will be working with the Infrared (IR) sensor, programming the Analog-to-Digital Converter (ADC) module, and demonstrating data acquisition and distance measurement using the ADC and IR sensor.



FIGURE 1: INTERNALS OF AN IR ELECTRO-OPTICAL DISTANCE SENSOR

The IR sensor is an electro-optical device that emits an infrared beam from an LED and has a position sensitive detector (PSD) that receives a reflected IR beam; see Figure 1. 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 detectors, and the distance of an object can be determined through optical triangulation (see Figures 2-4 below). The location of the focused reflection on the PSD is translated to a voltage that corresponds with the measured distance. An IR distance sensor is designed to measure distances in a specific range. The IR sensor used in the lab is designed for 9 - 80+ cm.

However, the IR sensor can only fairly accurately display a distance value for an object 9-50 cm away. As the distance increases, the voltage decreases. See the IR Sensor Datasheet to learn more about the operation of the IR sensor.

ADC (Analog to Digital Conversion) and DAC (Digital to Analog Conversion) allows our embedded programs to interact with realworld physical systems. Physical quantities such as temperature, pressure, distance, or light are analog and represented using continuously valued signals with infinite possible values in between. In contrast, a digital signal is a discretely valued signal having a fixed precision. Since analog is a continuous value, we need a way to convert a physical analog signal into an n-bit digital signal.

Our platform's microcontroller has two 12-bit ADCs. The IR sensor measures a distance and sets the voltage on the wire leading to the GPIO pin. The ADC then converts this voltage into a 12-bit digital value between 0 and 4095 ( $2^{12}$  values) and stores it in an ADC register. This digital value is called a conversion result or quantized value. The program then reads the digital (quantized)



FIGURE 2: OPTICAL TRIANGULATION FOR DISTANCE OF A NEAR OBJECT.



FIGURE 3: OPTICAL TRIANGULATION FOR A DISTANT OBJECT.

value and converts it into a distance. After an ADC conversion has been triggered and is complete, the conversion results can be obtained from the ADC Sample Sequence Result FIFO (**ADCSSFIFOn**) register. You may need to think about the correlation between the physical input signal and the digital results. Once you have set up the ADC to generate a digital result (quantized value), you will need to calculate the distances corresponding to the digital results and improve the accuracy of the distance calculations.

Note: The following figure illustrates the use of triangulation to determine the distance to an object based on similar triangles. The IR sensor does this triangulation for you, such that its output voltage represents the distance as shown in the datasheet.



**Figure 8** Principle of optical triangulation sensor. The unknown distance, D, is determined from the known distances E, F and the measured value of G—the distance to the pixel in the position sensitive detector (PSD) recording the image of the laser spot on the measured object.

#### FIGURE 4: PRINCIPLE OF OPTICAL TRIANGULATION.

Citation: Garry Berkovic, Ehud Shafir, "Optical methods for distance and displacement measurements," Adv. Opt. Photon. 4, 441-471 (2012); <u>https://www.osapublishing.org/aop/abstract.cfm?uri=aop-4-4-441</u>

### **Reference files**

The following reference files will be used in successful completion of this lab:

- 1. Lab Evaluation Form
- 2. lcd.c, a program file containing various LCD functions
- 3. lcd.h, the header file for lcd.c
- 4. timer.c, A program file containing various wait commands
- 5. timer.h, the header file for timer.c
- 6. TM4C123GH6PM Datasheet
- 7. IR Sensor Datasheet

In addition to the files that have already been included for you, you will need to write your own **adc.c** file and **adc.h** file and the associated functions for setting up and using ADC. Separate functionalities should be in separate functions for good coding quality and reusability purposes. This means that in your adc.c file you should write separate functions for initializing/configuring ADC and taking an ADC sample. Remember to use good naming conventions for function names and variables. For example, you may want to name your ADC initialization function **adc\_init** and name your function to take samples **adc\_read** as there will be other initialization functions you will write in later labs that will eventually have to be used together. Minimally, we recommend defining the following functions:

```
void adc_init(void);
int adc_read(void);
```

# Part 1: Initializing ADC and Displaying Quantized Value

Write a program for the ADC that will initialize and configure the registers needed to sample the analog voltage from the IR distance sensor. Print the digital result (quantized value) to the LCD screen.

The ADC uses Ports B, D, and E. While there are 12 input channels, in lab we will use AIN10 on PB4. See Table 13-1 in the Tiva datasheet for all channels. You may choose which ADC module to use (ADC0 or ADC1) and which Sample Sequencer unit to use (SS0, SS1, SS2, SS3).

Make sure the IR sensor is connected to the correct pins on the CyBot board, as shown on the right. There are three pins labeled **SNSR1** on the baseboard, which are marked as 3 rows: (1) Signal Row, (2) 5V Row (power), and (3) Ground Row. The Signal Row makes a connection to microcontroller pin PB4 (AIN10). The sensor connector should be oriented such that the white wire connects to Signal and the black wire connects to Ground. The red voltage supply wire is in the center.

Your program should be set up to repeatedly take samples. The displayed quantization values will be dynamic based on conversion results produced by the ADC. You will notice quite a bit of variability in the quantized value even when the CyBot and object are stationary. This is because of noise inherent with ADC operation. You may want to slow down how fast samples are taken as well as how frequently the display is updated.

As you are setting up the registers for ADC, you may find the ADC Module Block Diagram useful for understanding how samples are taken. You can also refer to Chapter 13 in the Tiva Datasheet, which contains sections on signal descriptions, functional descriptions, initialization and configuration steps, as well as the register descriptions. You may not need to read everything in these sections. Look for basic concepts and overviews first, ignore things that seem outside the scope of the lab (whether or not it may be needed later, ignore it for now, such as digital comparators or differential mode). Browse the steps given for initialization, sample sequencer configuration, etc.

#### Figure 13-2. ADC Module Block Diagram



Another resource that gives a nice overview and selected details of the ADC is Valvano and Yerraballi, Chapter 14: Analog to Digital Conversion, Embedded Systems: Introduction to ARM Cortex-M Microcontrollers, 2014.

Checkpoint: Demonstrate ADC conversion of IR sensor signals to your mentor. Display quantized values read from the ADC (raw digital conversion results). Do not calculate distance for this part. Explain why the quantized values appear to be valid. You will calculate and calibrate the distance measured in the next part.



# Part 2: Calibrating Distance Measurement

In Part 1, you configured the ADC to generate a quantization value based on the distance of an object. Due to the nonlinear sensor operation, there is not a simple linear transfer function between the distance being measured and the analog voltage output of the sensor. Thus, you will need to implement a technique to map quantization values to distance values. To do this, you will need to take some measurements. Your task is to accurately display a distance value for an object 9 - 50 cm away. In order to calibrate the sensor, you will need to collect several data points consisting of the known distance and quantized value at that distance. Based on these data points, you can find an equation for a best fit line/curve, or use a lookup table based approach. Your method will return an estimated distance value for a given quantization value. For full credit, the estimated distance calculated by your program must be within 2 cm of the actual distance. Print to the LCD both the quantization value and the estimated distance in cm.

You should also reduce the variability in values seen in Part 1 by averaging multiple samples. You can use either hardware averaging (Tiva Datasheet section 13.3.3) or software averaging to collect and average 16 samples to get a more stable sensor value. The variability observed in Part 1 can be reduced by averaging multiple samples and treating the average value as the estimated distance value. Use an averaging mechanism in your program.

Checkpoint: Demonstrate the distance measurement. Print to the LCD both the quantization value and the estimated distance in cm. Your distance readings should be within 2 cm of actual values. Additionally, you must implement an averaging mechanism in your program. Explain and justify your calibration methods with a detailed description of how your choices were made or implemented

## Part 3: Extended Simple Mission: ADC integration

Now that you have the ability to configure and use the ADC device to sample the IR sensor, and can compute calibrated distance values, integrate this code into your Extended Simple Mission. In other words, use your ADC code instead of the CyBot Scan library's IR raw value within your Extended Simple Mission.

Checkpoint: Demonstrate to your mentor that your Extended Simple Mission still works using your ADC code instead, of using the CyBot Scan library's IR raw values.

## Part 4: Controlling the Cybot using a simple Socket program

In this part you will extend your previous simple socket program (that you used to drive the CyBot) to also display scan data (in text and graphically), and switch between manual and autonomous modes.

### **Review the starter Socket program:**

i) The code: Review the provided starter Socket programs (simple-sensor-Socket-or-UART-client.py & simple-sensor-mock-Cybot-Socket-server.py - found on the resource section of the course website) with your lab partner. Give a brief summary of the steps the starter Socket programs are taking.

**ii**) **Network Sockets:** For a reminder on sockets, review the following Network Socket overview link given within the program (<u>https://realpython.com/python-sockets/</u>) with your lab partner.

### Running the socket program without a CyBot:

i) Server: Start the Server Program (simple-sensor-mock-Cybot-Socket-server.py).

ii) Client: Start the Client Program (simple-sensor-Socket-or-UART-client.py).

**iii**) **Echo/Scan:** Within the Client Program's prompt type messages to send to the server. Verify the Server sends the message back to you, and if you send an 'm' the Server it will reply with scan data that the Client displays and write to a file.

<u>Use the Socket Client program to control the Cybot:</u> Instead of having the Client communicate with the given Server program, instead have it communicate with your CyBot.

i) Client: Update your client program to use the <u>IP address</u> and <u>Port number</u> that you have been using to set up PuTTy communication

**ii)** Cybot: Use your Lab 7 Cybot program to receive commands from the Client program for driving the Cybot, switching between autonomous and manual mode, and collecting scan data

- **Note:** Your Lab 7 code needs to be compatible with the Client (or you need to modify the Client). So first review, and use the **Simple-CyBot-echo.c** C program (found on the resource section of the course website) to communicate between the Client and Cybot. Then modify your Lab 7 code based on this example code to make it compatible with the Client program.

**iii**) **Drive the Cybot**: Start your Client code, and use it to manually drive your Cybot. In other words, drive the CyBot using your Client program instead of PuTTy (You did this for Lab 5).

iii) Drive the Cybot, and graphically display scan data: Update your Client program so that it graphical displays scan data anytime the Cybot sends data in response to receiving an "m", or "h" command. <u>Tip</u>: Leverage the code found in previously given scripts to plot data.

Checkpoint: Give a brief explain of the Client code to your Mentor. Demonstrate that you can control your CyBot, collect scan data, plot scan data, and switch between manual and autonomous mode using the Client Socket program you modified.