TI-RSLK MAX

Texas Instruments Robotics System Learning Kit

Texas Instruments

Module 6

Lab 6: General Purpose Input Output

Lab: General Purpose Input Output

6.0 Objectives

The purpose of this lab is to interface a line sensor that the robot will use to explore its world, see Figure 1.

1. You will learn functions, conditionals, loops, and calculations in C.
2. You will use GPIO to perform input and output.
3. You will understand how light is converted to voltage, and how voltage is converted to binary.
4. You will interface a line sensor to the microcontroller.

Good to Know: General purpose input output (GPIO) is the simplest and most pervasive means of performing I/O on the microcontroller. The sensor you interface in this lab will allow a robot to explore its world.

6.1 Getting Started

6.1.1 Software Starter Projects

Look at these three projects:
GPIO (a very simple system that outputs to four pins),
InputOutput (simple system that inputs from switches and outputs to LEDs on the LaunchPad),
Lab06_GPIO (starter project for this lab)

6.1.2 Student Resources (Links)

Meet the MSP432 LaunchPad (SLAU596*)*
MSP432 LaunchPad User's Guide (SLAU597*)*
QTRX.pdf, line sensor datasheet

6.1.3 Reading Materials

Chapter 6, "Embedded Systems: Introduction to Robotics"
,

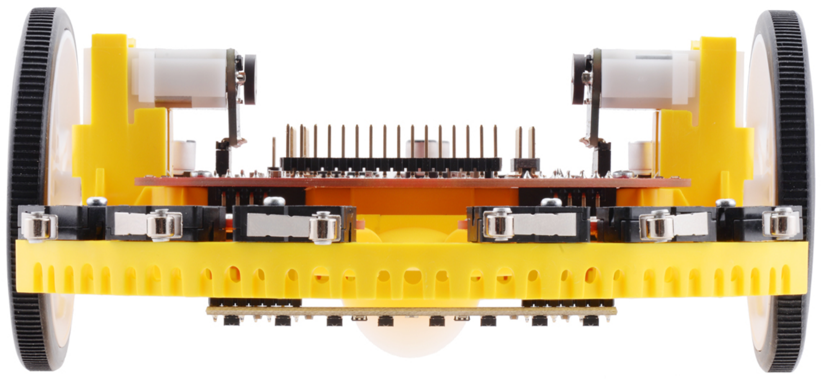

Figure 1. QTRX mounted on bottom of robot, 3mm above the floor. Notice, the sensor does not extend over the battery compartment.

6.1.4 Components needed for this lab

All the components needed in the lab are included in the TI-RSLK Max kit (TIRSLK-EVM kit). For this lab you will need, the six batteries that are not part of the kit. You can use disposable 1.5V alkaline batteries or rechargeable 1.2V NiMH batteries. Black tape is also not part of the kit. You can use the robot you build in lab 5.

Quantity	Description	Manufacturer	Mfg P/N
1	TI-RSLK MAX kit	Texas Instruments	TIRSLK-EVM

Table 1. Components needed for this lab

6.1.5 Lab equipment needed

Oscilloscope (one or two channels at least 10 kHz sampling)
Logic Analyzer (4 channels at least 10 kHz sampling
Note you also need *a black non reflective tape and a white surface*.

Lab: General Purpose Input Output

6.2 System Design Requirements

The ultimate goal of this lab is to design a sensor system that measures the position of the robot relative to a line. For example, black tape on a white surface could be used in a robot challenge of exploring its world.

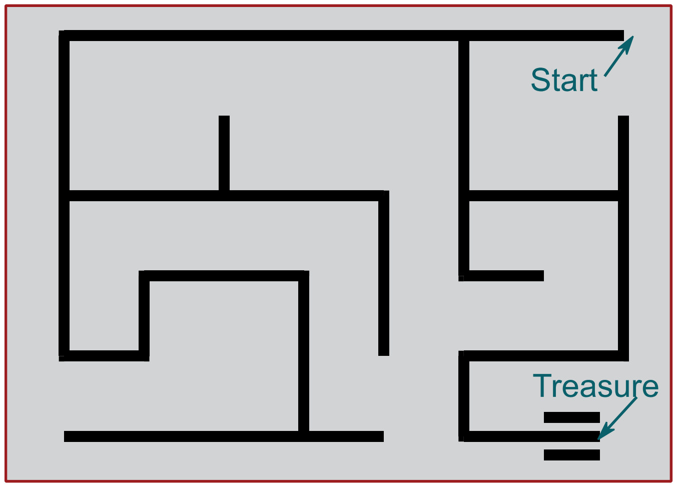

Figure 2. Possible final robot challenge of exploring the world and finding the treasure.

The robot will have eight sensors that detect the line, see Figure 3.

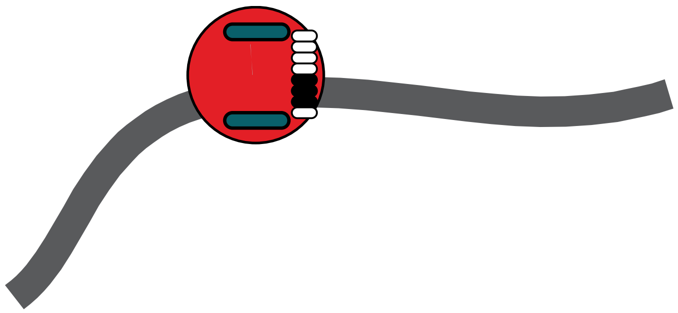

Figure 3. Robot with eight line sensors and two wheel motors.

Each sensor is binary, returning 1 if it sees black and 0 if it sees white. If the robot is properly positioned on the line, the middle sensors will see the black line. If the robot is a little off to the left or right, one or more outer sensors will see the black line. If the robot is completely off the line, all sensors will see white. We will use sensor integration to combine the eight binary reading into a single position parameter. We define **position** of the robot as the distance from the sensor to the line. The line sensor (QTRX) we will use is about 76.2 mm wide (with about 9.53mm between sensors), so we should be able to estimate the robot position of -33.4 to +33.4 mm from the center of the line.

Figure 4 shows the desired output of the sensor measurement system. Basically, you are asked to create an output that varies from -330 to +330 (units 0.1mm) as a function of the position of the robot relative to the line.

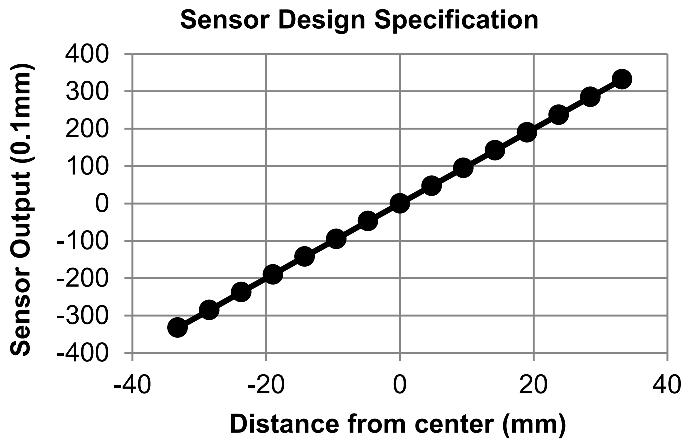

Figure 4. Desired output of the measurement system.

There are a number of performance measures with which the system could be evaluated. **Accuracy** is the difference between true line and measurement at a given time.

$$\text{Average Accuracy} = \frac{1}{N}\sum_{i=1}^{N}|x_i - T_i|$$

where x_i is the measurement and T_i is the actual or true line. Accuracy is interesting, but not very important for solving the explorer missions.

The system is **monotonic** if an increase in input never causes a decrease in output, i.e., as the input increases, the output increases or stays the same.

Similarly, as the input decreases, the output decreases or stays the same. Monotonicity will be important for the robot control system. You will determine if your system is monotonic by slowly sliding the sensor across the line and measuring the system output at each position.

Noise is also important for a control system. **Standard deviation** is quantitative measure of noise and given by:

$$\mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

$$\sigma = \frac{1}{N}\sum_{i=1}^{N} (x_i - \mu)^2$$

where in this case, x_i are measurements taken with the input fixed. The **coefficient of variation** is the standard deviation divided by the mean or the average value.

$$CV = \sigma/\mu$$

Specificity is a measure of relative sensitivity of the system to the desired signal compared to the sensitivity of the measurement to other unwanted influences. A system with a good specificity will respond only to the signal of interest and be independent of these other factors. The unwanted influence you will study is angle to the line, as defined in Figure 5. In particular, you will create response curves like Figure 4, for angles 90, 75, 60, and 45 degrees.

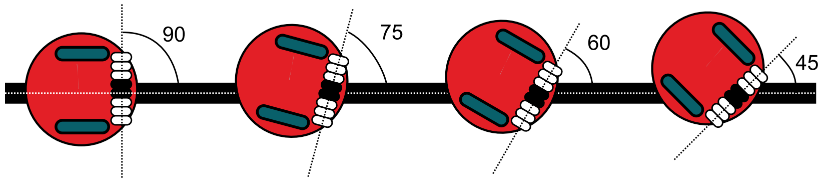

Figure 5. It is desired for the system to be able to measure relative distance to the center of the line for a wide range of angles.

6.3 Experiment set-up

The optimal sensing distance for this sensor strip is 3 mm (0.125"). **The distance between the line sensor and the floor should be about 3 mm.** During this lab, you could use two 3 mm objects attached to each end of the sensor to set the distance and make it parallel. A better approach would be to mount the sensor on the bottom of your robot (if you have purchased the kit and build it in lab 5) again positioning the sensor parallel to the floor 3 mm above the floor. See Figure 1.

You will need access to a mockup (small representative piece) of the challenge arena (similar to shown in Figure 2) in which to test the sensor. The sensor works well on a white reflective surface, marked with black non-reflective tape.

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.

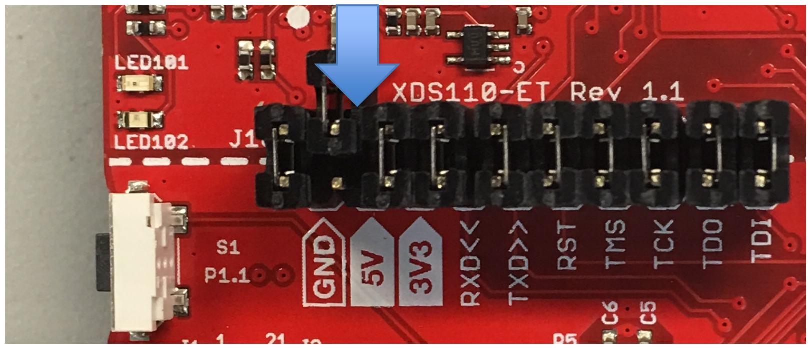

Figure 6. MSP432 LaunchPad with +5V jumper removed.

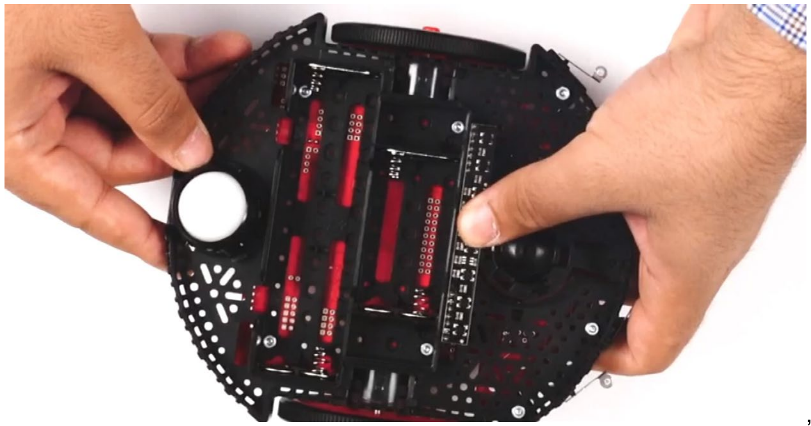

Figure 7. Line Sensor attached to the robot

Warning: *Be careful not to plug the sensor in backwards, see Figure 7*

Lab: General Purpose Input Output

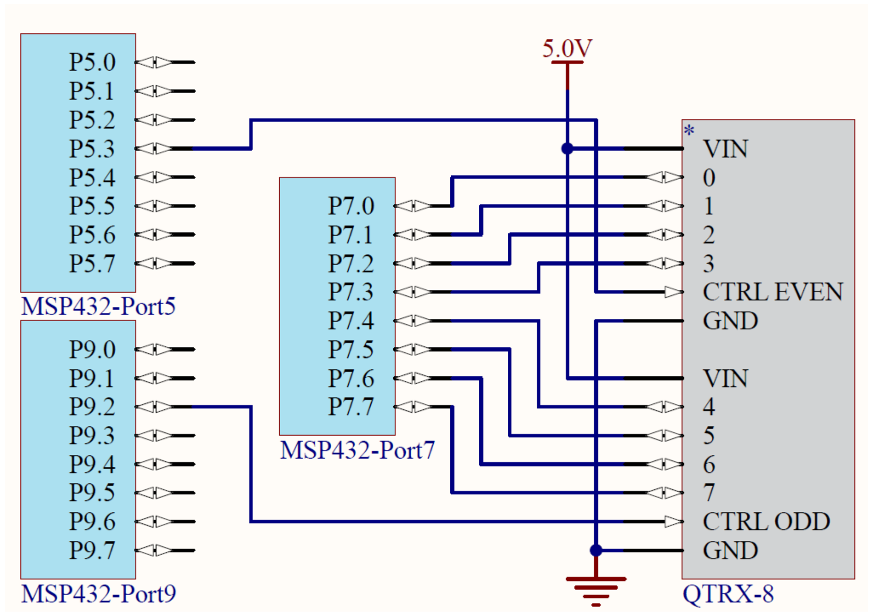

Figure 8 The 8-channel line sensor is interfaced to Ports 5 7, and 9. These connections are made on the TI-RSLK chassis board. Sensor 0 (P7.0) is on the right, and sensor 7 (P7.7) is on the left.

6.4 System Development Plan

6.4.1 Configuring the MSP432 ports as input and output

To initialize an I/O port for general use you will perform the following steps. First specify GPIO by writing zeros to the PxSEL0 and PxSEL1 registers. Secondly you will set the direction of the registers.

Open the **GPIO** project - Compile, download and debug this example. Using the debugger, observe the direction register and output register for Port 4 while you single step through the main program. Run the program and observe the output on Port 4 bits 3 – 0. Notice the use of a function to initialize Port 4. Is the output to Port 4 friendly or unfriendly? Notice the main2 version provides for abstraction.

Open the **InputOutput** project and notice that initialization can be done in different ways *(main, main2, and main3).* The main example is unfriendly. The *main2* example is friendly. The *main3* example is friendly and provides abstraction.

Using the **InputOutput** project, compile, download, and run the program. Using the debugger, observe the direction, input, and output registers for Ports 1 and 2 while you single step through the main program. Notice how the ports are initialized, how data are input, and how data are output. Within this project there are three versions. See comments in the project file.

Note: *The first version is "unfriendly", because it writes to the entire registers, even though it needs to affect just some of the bits. The second version is friendly, because it only alters the bits needed. The third version provides abstraction, referring to objects by their logical name (SW1IN, SW2IN, BLUEOUT, GREENOUT, REDOUT) rather than their physical implementation (P1->IN, P2->OUT).*

After following the above steps in the lab you will comment on the three versions with respect to the following:

- Ease of understanding
- Ability to integrate into larger system
- Portability, ease of implementing system on another microcontroller

6.4.2 Configuring the Low-level sensor input/output

You will begin by analyzing one of the eight sensors on the line sensor and discover how the sensor works.

We will use P5.3 as an output, connected to the EVEN IR pin of the line sensor. P5.3 turns on the infrared (IR) LED for sensors on P7.3, P7.2, P7.1 and P7.0. P7.0 is connected to one of the right most sensor of the QTRX, and P4.0 is set as a digital output (used in this section for testing). See Figure 9. In particular, the goal of this section is to observe how the reflectance of the given surface affects the voltage on P7.0, and how the microcontroller converts the voltage on P7.0 into binary data depending on the color of the reflective surface.

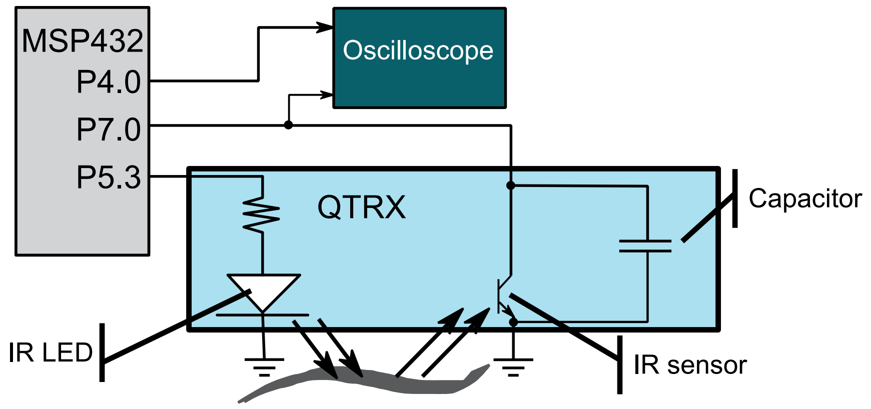

Figure 9.Hardware configuration for one sensor.

Your solutions will be placed in **Reflectance.c** so that the software will be usable for subsequence labs. See **Reflectance.h** for detailed specifications of software you will need to write. Place the sensor S1 (P7.0) 3 mm above a **white reflecting surface**. Perform this software initialization:

```
  1) Initialize the Clock system to run at 48 MHz, Clock_Init48MHz();
  2) Make P5.3 output, and set it initially low
  3) Make P7.0 an input
  4) Make P4.0 an output (you can use any unused pin)
```

Connect P7.0 and P4.0 to a dual channel oscilloscope. If you have a single channel scope, then first measure P7.0 and then measure P4.0. Since the input (color of the line) does not change, the two separate measurements can be aligned. The body of the main program executes steps 1 to 7 over and over:

```
main program
  0) Initialize
while(1){
  1) Set P5.3 high (turn on IR LED)
  2) Make P7.0 an output, and set it high (charging the capacitor)
  3) Wait 10 us, Clock_Delay1us(10);
  4) Make P7.0 an input
  5) Run this loop 10,000 times
     a) Read P7.0 (converts voltage on P7.0 into binary)
     b) Output binary to P4.0 (allows you to see binary in real time)
  6) Set P5.3 low (turn off IR LED, saving power)
  7) Wait 10 ms, Clock_Delay1ms(10);
}
```

Notice that you read the sensor 10,000 times, looking for the input signal to change from 1 to 0. You will observe the change on P4.0 using an oscilloscope or logic analyzer. Figure 10 shows the theoretical response, and Figure 11 shows an actual measurement.

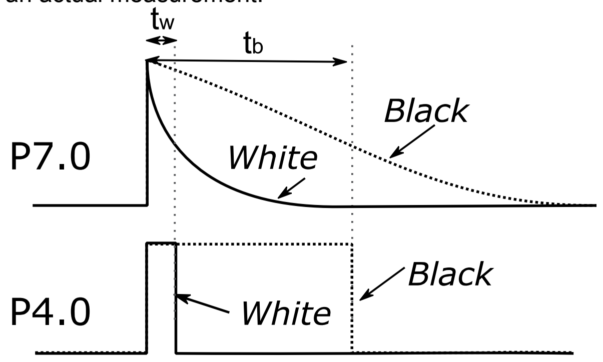

Figure 10. Conversion of light to voltage, and voltage to binary.

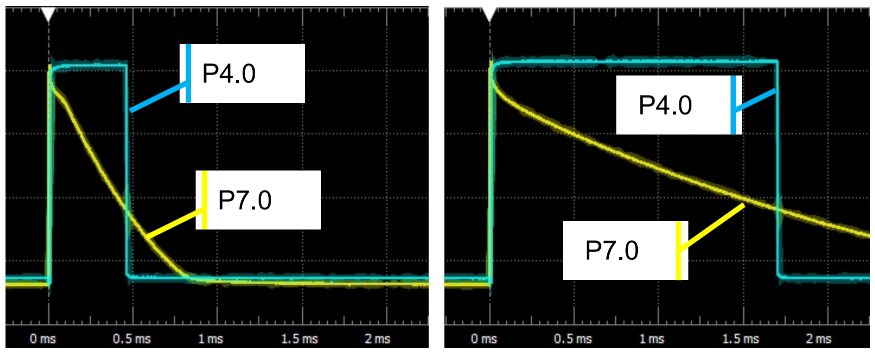

Figure 11 Left side measured with white surface, right side measurements with black surface. If you read P7.0 at 1ms, it will be low for white and high for black.

Run the system and observe the voltage on P7.0 and P4.0 on the oscilloscope. At what voltage does the binary switch from 1 to 0 on P4.0 for the white surface? Repeat the experiment with a black non-reflective surface.

Note: *Basically, what is happening is the 10us output pulse on P7.0 charges a capacitor on the QTRX board (see Figure 7). Once the microcontroller sets the pin to input, the reflected light on the light-sensitive transistor discharges the capacitor. The white reflective surface has more light on the base of the transistor, and conducts more current through the collector-emitter. This current discharges the capacitor. Notice the white surface discharges faster and P7.0 falls more quickly. Using the rise of P7.0 as a time reference of 0, measure the time P4.0 falls for the white and black surfaces, see Figures 8 and 9. For a white surface, measure the time t_w. For a black surface, measure the time t_b.*

6.4.3 Low-level sensor interface

Now that you understand how the sensor works, you will create a low-level software driver that measures all eight sensors at the same time. You will assume the MSP432 is running at 48 MHz clock. The initialization includes:
1) Make P5.3 and P9.2 outputs, and set them initially low
2) Make P7.7 – P7.0 inputs

Create a C function that measures all eight sensors. Let **time** be a parameter passed into this function, choosing **time** between t_b and t_w. Basically, if we wait 1000 µs, then the white will have decayed to a zero, while the black will still be high. This allows us to differentiate between white and black,

Perform these 8 steps in this sequence:
1) Set P5.3 and P9.2 high (turn on IR LED)
2) Make P7.7 – P7.0 outputs, and set them high (charging 8 capacitors)
3) Wait 10 us, **Clock_Delay1us(10);**
4) Make P7.7 – P7.0 input
5) Wait time us, **Clock_Delay1us(time);**
6) Read P7.7 – P7.0 inputs (converts voltages into binary)
7) Set P5.3 and P9.2 low (turn off IR LED, saving power)
8) Return 8-bit binary measured in step 6

Passing into this function makes it easy to retune the sensor if the robot or arena changes. **Time** will be about 1000 µs, as measurements from the oscilloscope have shown. You will test your system using a simple main program similar to Program6_1 and observe the **Data** variable with the debugger to make sure it matches expected behavior (a “0” means white and a “1” means black).

```
// Use this program to test the Read function
uint8_t Data; // QTRX
int Program6_1(void){
  Clock_Init48MHz();
  Reflectance_Init(); // your initialization
  TExaS_Init(LOGICANALYZER_P7);
  while(1){
    Data = Reflectance_Read(1000); // your measurement
    Clock_Delay1ms(10);
  }
}
```

Note: *You will notice this measurement consumes all the processor time. Time is wasted while it waits the* ***Time****=1ms for the capacitors to discharge (or not discharge). Time is wasted again in the 10 ms delay within the main loop. Once we learn interrupts in Module 10, we can recover this wasted time.*

6.4.4 High-level sensor integration

In this section, you will combine the eight binary measurements into a single parameter representing the amount of which the robot is away from the center of the line. We will assume the sensor REFL0 (P7.0) is positioned on the robot's right, 33.4 mm from midline. Furthermore, we assume the sensor REFL7 (P7.7) is positioned on the robot's left, -33.4 mm from midline. As mentioned earlier, another goal is to make this parameter **insensitive to angle**. If the sensor is operating properly, the 8-bit binary pattern stored in **Data** falls into four categories:

1) <all 0’s> (off the line or on white surface)
2) <some 0’s, some 1’s>, e.g., 00000111 (off to the left)
3) <some 0’s, some 1’s, some 0’s>, e.g., 00110000 (over the line)
4) < some 1’s, some 0’s>, e.g., 11110000 (off to the right)

Figure 10 (source: Pololu.com) Shows the sensors are about 9.53 mm apart, with the center of the robot aligned between sensors 3 and 4 (P7.3 and P7.4)

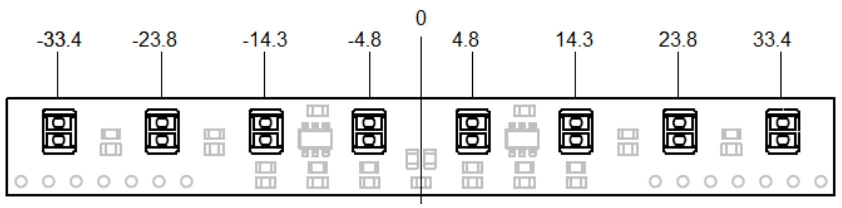

Figure 12. Position is defined as relative distance to the center of the robot.

Define an array of eight distance values in 0.1mm units, W_i for i = 0 to 7.
W **= {334, 238, 142, 48, -48, -142, -238, -334}**

Define b_i to be 0 (white) or 1 (black) for each binary bit returned by the **Reflectance_Read** function.

One possible sensor integration function is to calculate a weighted average of the eight sensor readings or binary results. Assuming there is at least one black reading, estimate distance **Reflectance_Position** as:

$$d = \frac{\sum_{i=0}^{7} b_i W_i}{\sum_{i=0}^{7} b_i}$$

In this section of the lab after connecting the MSP432 launch pad development board and the QTRX, you will mount as shown in Figure 1 and then you will define true distance according to the specifications illustrated in Figure 10.

Slowly slide the sensor across the line (Figure 2 shows a maze line), and plot the measured true distance and the estimated/calculated by **Reflectance_Position**, as a function of true distance, similar to Figure 4 (first at 90 degrees as defined in Figure 5).

Comment if the response is monotonic. Interpret if there any regions where the measurement is noisy. Repeat the experiment for other angles (Figure 5) to estimate the specificity of the measurement.

Note: *The sensor will be effective if the measurements are monotonic and low noise. Another issue is* ***offset****. Where is the robot when the readings are 0? Since the controller will attempt to drive the distance measure to 0, this is the position to which the controller will seek.*

6.5 Troubleshooting

Sensor doesn't work:

- Check the wiring as shown in Figure 5, including power and ground
- Look at signals P5.3 P9.2 and P7.0 as described in Section 6.4.2.
- Single step and verify the direction registers are correct
- Verify the computer is running at 48 MHz.
- Try another LaunchPad. Try another sensor

6.6 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.

- What is a function? How are functions used to simplify software?
- What are direction registers? How do we use them? Why do we have them?
- Is it possible to use the same pin as both an input and an output?
- What information is in the header file **Reflectance.h**? What is in the code file **Reflectance.c**? What benefits does this abstraction provide?
- What voltages does the MSP432 consider low? What voltages does the MSP432 consider high?
- What is the difference between a logic analyzer and a scope?
- What is monotonicity and why is it important?

6.7 Additional challenges

In this section, we list additional activities you could do to further explore the concepts of this module. You could extend the system or propose something completely different. For example,

- You could use this approach to measure the position of a slide pot or joystick. Place a capacitor in parallel with the potentiometer, use the pin as an output to charge the cap, then use the pin as an input to see how long it takes for the capacitor to discharge.
- Develop alternative methods to integrate the 8 sensor readings into a single parameter that determines the relative position of the robot on the line.

- Add verification checks to **Reflectance_Position** to make sure the sensor readings are one of the four expected patterns. In particular, add checks to determine if the robot is positioned over the treasure (see the end of Figure 2.)

6.8 Which modules are next?

The GPIO pins are a simple but common means to input data into the microcontroller, or output data from the microcontroller. In addition to this line sensor, GPIO will be used in the robot for bump sensors, motor direction, LCD output, tachometer input, ultrasonic I/O, Bluetooth Low Energy (BLE), and wifi. The following modules will build on this module:

Module 7):	Study finite state machines as a method to control the robot
Module 8)	Interface actual switches and LEDs to the microcontroller. This will allow for more inputs and outputs increasing the complexity of the system.
Module 9)	Develop a simple PWM output to adjust duty cycles
Module 10)	Learn SysTick periodic interrupts, so these measurements occur in the background in a very time-efficient manner
Module 12)	Connect the line sensor and motors to the robot.

6.9 Things you should have learned

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

- Use functions to provide software abstraction
- Perform conditionals, loops, and calculations in C
- Use the debugger to single step and visualize I/O registers
- Set the direction register for GPIO pins
- Perform friendly access to I/O registers
- Visualize an input pin converting voltage into binary
- Use an oscilloscope to visualize time behavior
- Perform incremental design and testing
- Conduct experiments to determine accuracy, monotonicity, specificity, and noise
- Integrate multiple sensor measurements into a single value

ti.com/rslk

Texas Instruments