The Mechatronics Revolution: Fundamentals and Core Concepts
Lab Assignment 6

Building a Remote-Controlled Obstacle Avoiding Robot

6.1 Objective

The main objective of this lab is to build an obstacle avoiding robot. All the modules and previous lab assignments studied in the course will come together in this lab to build a robot that can be remotely controlled, detect obstacles around it, and automatically avoid bumping to obstacles in front of it. The robot is controlled by sending remote commands to it through a PC/laptop via UART, and it senses the obstacles around it using an analog distance sensor and six bump switches installed at the front, in order to detect and avoid collisions with objects.

The detailed learning objectives of this lab include: i) to interface and control the two brushed DC motors of the TI-RSLK-Mechkit robot with the MSP432 LaunchPad, using PWM signals and an H-bridge motor driver, ii) to enable the TI-RSLK-Mechkit robot to be controlled by remote commands sent from a computer, iii) to construct a data acquisition system and transmit the collected data to a computer, iv) to detect objects in front of the robot using an analog distance sensor, sampled by a periodic interrupt, and v) to interface the bump sensors of the TI-RSLK-Mechkit robot with the MSP432 LaunchPad and detect collisions.

In Module 6, we studied how to build a mechatronic system considering the overall software workflow, microcontroller peripherals and motor controllers. Specifically, the fundamental knowledge required for this lab assignment were provided in Modules 1 to 6 of the course:

Module 1	Overview of Microcontrollers (MSP432 LaunchPad)
Module 2	Circuits and Electrical Components (Diodes, H-Bridges, etc.)
Module 3	Microcontroller Fundamentals (GPIOs, Interrupts, Timers)
Module 4	Motors and Servos (DC motors, PWM signals)
Module 5	Sensors and Microcontroller Interfacing (ADC conversion, UART communication)
Module 6	Building Mechatronic Systems

6.2 Setup

This lab requires the *Code Composer Studio* IDE, the *TI MSP432P401R LaunchPad*, the *TI-RSLK-Mechkit*, the *H-bridge motor driver circuit* built on a breadboard in Lab 2, a *Sharp*

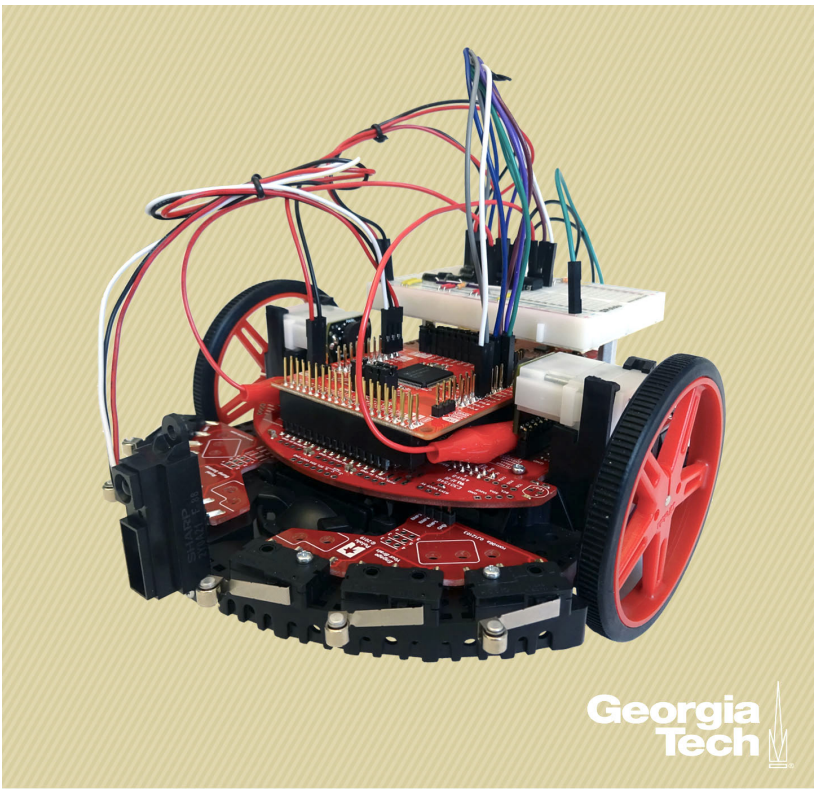

Figure 1: The obstacle avoiding robot using the TI-RSLK-Mechkit.

GP2Y0A21YK0F Analog Distance Sensor, and a *SSH terminal* (e.g. Putty for Windows, Terminal for MacOS, Putty or Terminal for Linux). The lab uses onboard features of the MSP432 LaunchPad including GPIOs, UART, ADC and Timers.

Note: For more details on how to setup a SSH terminal, please refer to the Labs Helper Guide, available under Course Resources on edX (Labs-Helper-Guide.pdf).

All components needed for this lab are included in the TI-RSLK-Mechkit. Details on how to assemble the TI-RSLK-Mechkit robot are presented in the Robot Assembly Manual, available under Course Resources on edX (Robot-Assembly-Manual.pdf). Six AA batteries (not included) will be needed to power your robot. You can use disposable 1.5V alkaline batteries or rechargeable 1.2V NiMH batteries.

Five predefined macros and and a setup function are declared in the `mechrev.h` header file, available under Course Resources on edX (mechrev-header.zip). The output text file generated by the macros will be used to grade the assignment. The motors' encoders data which will be collected automatically will also be used for grading purposes.

In this lab, the UART data is streamed through the USB debugging cable. Moreover, the output of the grading macros will be printed to the Code Composer Studio console under the debug mode. Therefore, the LaunchPad USB cable must be connected from robot to PC during the lab and when collecting the grading outputs.

Warning: Make sure that you have removed the +5V jumper on the MSP432 LaunchPad. Not removing this jumper will cause permanent damage to the LaunchPad and the TI-RSLK chassis board.

6.3 Problem Statement

Control the *TI-RSLK-Mechkit* robot by sending remote commands to it through a PC/laptop. The robot should automatically stop if it senses an obstacle in front of it using a distance sensor, or detects a collision through its bump sensors.

To this end, you need to write a program that receives data commands form a PC/laptop over `UART` and controls the direction and speed of two brushed DC motors accordingly. All motor speed control should be accomplished using *PWM signals* generated by `Timer_A0`. Specifically, the MSP432 could receive five different data characters from the SSH terminal, as described in the table below:

Command	Robot Behavior	DC Motors States
w	Drive forward	Both motors full-speed forward
z	Drive backward	Both motors full-speed backward
d	Turn right	Left motor full-speed forward, Right motor half-speed forward
a	Turn left	Right motor full-speed forward, Left motor half-speed forward
s	Stop	Both motors stop

Table 1: UART commands to control the robot.

Note: For this lab assignment, you are not required to set the full-speed of the motor as 100% PWM duty cycle. For example, you may decide to use 50% duty cycle for full-speed, and 25% for half-speed.

Moreover, when a carriage return character is received (generated on the keyboard by pressing the "enter" or "return" key, associated to the HEX code `0x0D`), the program should transmit the latest distance reading as ASCII characters to the SSH terminal, so that the distance value (in *cm*) is printed out on the terminal.

All other characters received from the SSH terminal should be ignored.

Furthermore, the program needs to read the analog output signal of the distance sensor (installed at the front of the robot) using `ADC` at 0.5 second intervals (using periodic interrupts generated by `Timer_A1`), and makes the robot stop (by turning off the DC motors) if the distance to an obstacle is less than 20 *cm* (8 inch).

Note: The LEDs on the MSP432 LaunchPad can be used for debugging purposes, for example to indicate receiving a character over UART.

Lastly, the program requires to detect the bump sensors press using GPIO interrupts, and stops the robot (by turning off its DC motors) upon detecting a collision.

Note: It is recommended to use the receive (RX) interrupt for reading the characters over UART sent via the SSH terminal. However, no interrupt is needed for the transmit (TX) functionality.

More details about hardware and software requirements will be discussed in the next sections.

6.4 Hardware

Build the *TI-RSLK-Mechkit* robot by following the instructions detailed in the Robot Assembly Manual, available under Course Resources on edX (Robot-Assembly-Manual.pdf).

The main hardware components used for the Obstacle Avoiding Robot are listed below. For more information about the hardware components, refer to the lab manuals for Labs 2, 3, 4, and 5.

- `Brushed DC Motors` (Lab 2 and Lab 4)
- `H-bridge IC` (Lab 2 and Lab 4)
- `Diodes` (Lab 2 and Lab 4)
- `Bumper Switch Boards` (Lab 3)
- `GP2Y0A21YK0F Distance Sensor` (Lab 5)
- `400-Point Breadboard` (Lab 2 and Lab 4)

6.5 Required MSP432 Peripherals

The MSP432 peripherals used for the Obstacle Avoiding Robot are listed below. For more information about these peripherals, refer to the lab manuals for Labs 3, 4, and 5.

- `GPIO interrupts` for bump sensors (Lab 3)
- `Timer_A0` for PWM signals (Lab 4)
- `Timer_A1` for periodic interrupts (Lab 5)
- `UART` for serial communication (Lab 5)
- `ADC` for analog-to-digital conversion (Lab 5)

6.6 Circuit Schematic

Figure 2 shows the schematic of the circuit for this lab assignment, including the H-bridge motor driver and the distance sensor. You have already assembled this circuit on the breadboard in Labs 2 and 4, and connected the distance sensor to the MSP432 LaunchPad in Lab 5. Note that the bumper switches are directly connected to the MSP432 LaunchPad through the TI-RSLK chassis board.

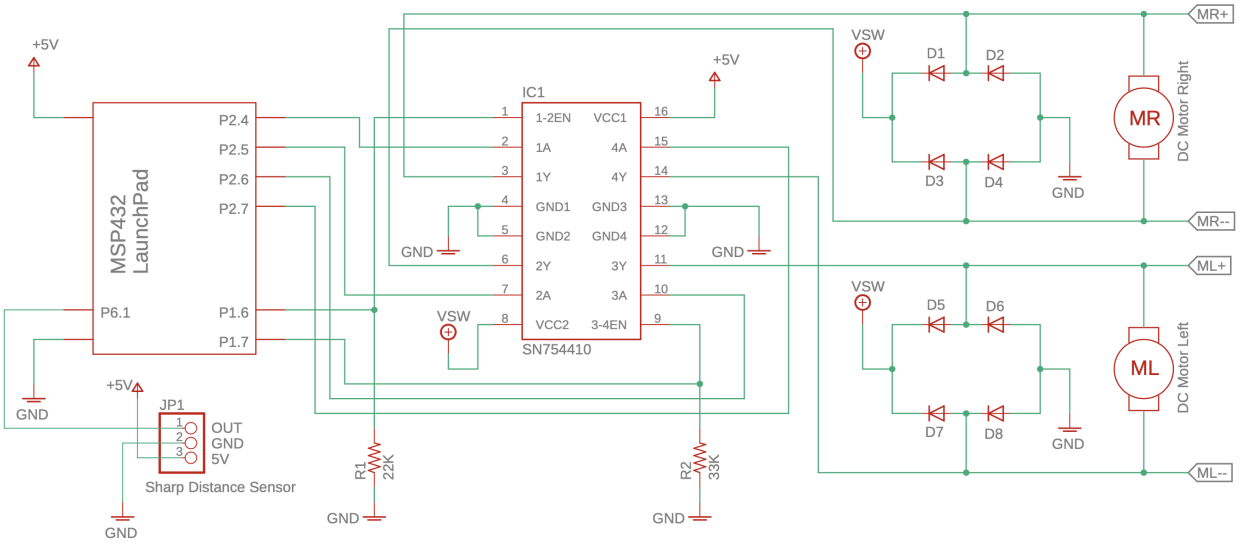

Figure 2: Circuit diagram for the distance sensor and the H-bridge interface with the TI-RSLK-Mechkit motors and the MSP432 LaunchPad.

The motors are powered via the six AA batteries installed on the robot, which provide a voltage in the range of 7.2V to 9V through the VSW pin of the TI-RSLK chassis board. We recommend to connect the VSW test point on the TI-RSLK chassis board to the (+) power rail of the breadboard using an alligator clip, and then use the power rail for all the pins connected to VSW. Similarly, the GND pin of the LaunchPad can be connected to the (−) power rail of the breadboard using a Male-Female jumper wire, so that all the pins connected to GND can share that power rail. The +5V required for the VCC1 pin of SN754410 (pin 16) can be taken from the 5V pin of the LaunchPad using another Male-Female jumper wire.

The distance sensor is powered by +5V, coming from the 5V pin of the MSP432 LaunchPad. The analog output of the sensor is connected to pin P6.1 of MSP432, corresponding to the `Channel 14` of the ADC14 module.

Note: The circuit shown in Figure 2 is the combination of the circuits built in Lab assignments 2, 3, 4 and 5.

6.7 Software Architecture

Note: To create a new CCS project for the MSP432P401R target device, and include the `mechrev.h` header file and the DriverLib library in the project, follow the procedure described in Lab Assignment 1. The `mechrev.h` file and the `driverlib` folder are available under Course Resources on edX (mechrev-header.zip and driverlib.zip).
To get you started with designing the program architecture, a code template for Lab 6 is available under Course Resources on edX (lab-code-templates.zip).

After disabling the watchdog timer by calling the `WDT_A_holdTimer()` function from DriverLib to avoid unnecessary watchdog timer time-out interrupts, call the `mechrev_setup()` function provided by the `mechrev.h` header file to bypass the motor drivers of the TI-RSLK chassis board. Calling this function will also enable the encoder readings, which will be used only for grading purposes.

Warning: Make sure that you have called `mechrev_setup()` function to disable the built-in motor drivers on the TI-RSLK chassis board. Not disabling the default drivers will cause a conflict with the SN754410 motor driver on the breadboard and will potentially damage the chassis board and/or the SN754410 IC.

Initialize the P1.6 and P1.7 GPIOs as output pins, used for enabling the H-bridge motor driver IC. We recommend to initialize these outputs to HIGH, in order to enable the driver channels while the program is running.

Initialize the P4.0, P4.2, P4.3, P4.5, P4.6 and P4.7 GPIOs as input pins with interrupt on falling edge, used for bump sensors. Remember to enable the PORT4 interrupt.

Initialize all four channels of `Timer_A0` module to generate the four PWM signals required for two motors: two PWM signal for each motor, one for each direction. That is, since `Timer_A0` is used for PWM generation, the channels `A0.1` (P2.4), `A0.2` (P2.5), `A0.3` (P2.6) and `A0.4` (P2.7) should be used to generate four PWM signals with different duty cycles. Make sure that `Timer_A0` channels are initialized to 0% duty cycle at the beginning.

As you learned in Module 4, after initializing the `Timer_A0` to generate the PWM signals, the duty cycle of a PWM signal can be changed by only changing the value of the `TAxCCRn` register associated to that timer channel. For example, to change the duty cycle of channel `A0.1`, we only need to change the value of `TA0CCR1` register (i.e. `x=0, n=1`).

Configure the UART module `EUSCI_A0` to enable transmit and receive and to establish a successful communication between the LaunchPad and your computer. Initialize the UART module using the functions from the DriverLib library (an example code was provided in Module 5, Topic 3, Lesson 4 of the course). First select a baud rate to use (standard baud rates include 9600, 38400, 57600 or 115200). The configuration parameters (clock divider and other register values) can be computed from http://software-dl.ti.com/msp430/msp430_public_sw/mcu/msp430/MSP430BaudRateConverter/index.html. Your UART configuration should use the above baud rate configuration parameters, the *SMCLK clock source*, *no parity*, *LSB first*, *UART mode*, *one stop bit*, and *oversampling*. For this lab, we recommend to use RX interrupt, but no interrupt is needed for TX functionality. Remember

to also configure the pins P1.2 and P1.3 with the primary functionality, associated to the `EUSCI_A0` module.

Configure the ADC14 module `Channel 14` in the *10-bit* single channel single conversion mode. Set the clock source to *SMCLK*, and allocate the register `ADC_MEM0` to hold the results from the channel `A14`, with *non-differential* functionality. Select the `AVCC` source as the voltage reference. Make sure to set the sample timer to *manual iteration*. Do not forget to enable the ADC14 conversion at the end of the initialization.

Configure `Timer_A1` such that it generates a periodic interrupt every 0.5 seconds (an example code was provided in Module 3, Topic 5, Lesson 2 of the course). Make sure to enable the `Timer_A1` interrupt and start the counter. Place the manual ADC conversion trigger along with the ADC reading procedure inside the `Timer_A1` interrupt handler (`TA1_0_IRQHandler`). You can also perform the ADC value conversion to the actual voltage and distance inside the `Timer_A1` ISR.

Design your code inside the UART ISR function (`EUSCIA0_IRQHandler()`) to read the received character for each RX interrupt. Upon receiving one of the specific robot commands (as listed in Table 1), change the state of the robot by adjusting the PWM duty cycles. You can use a global variable for storing the state of the robot, and check that variable inside the main `while(1)` loop to adjust the duty cycles.

Upon receiving the carriage return character (`0x0D`) via UART, transmit the latest distance reading (in *cm*) back to the PC via UART. Use the `sprintf()` function to convert the distance variable into ASCII characters to be sent over UART (an example code was provided in Module 5, Topic 5, Lesson 2 of the course). When you convert the data, convert the distance as a floating point number with two decimal place (this can be achieved using the format descriptor `%.2f` as an input to the `sprintf()` function).

```
/* convert the distance variable into an array of ASCII characters */
char stringData[10]; // to hold the ASCII characters to be transmitted out
int numChars = sprintf(stringData, "%.2f", distance_var);
```

Note: When data is transmitted from LaunchPad to the SSH terminal, your code could append a carriage return character (`0x0D`) and a newline character (`0x0A`) at the end to move the curser to the next line in the SSH terminal.

Note: Do not forget to clear the UART RX interrupt flag at the end of the ISR. Not clearing the interrupt flag results in the program not exiting the ISR.

Design your code inside the PORT4 ISR function (`PORT4_IRQHandler`) to detect bump sensors press. Upon a collision detection, change the state of the robot to "stop", as explained in the Problem Statement in Section 6.3.

Note: Do not forget to clear the PORT4 interrupt flag at the end of the PORT4 ISR. Not clearing the interrupt flag results in the program not exiting the ISR.

Design your code inside the `Timer_A1` ISR function (`TA1_0_IRQHandler()`) to read the distance sensor output using the ADC14 module at every interrupt. Remember that the `Timer_A1` interrupt is supposed to trigger at 0.5 second intervals. As discussed in Module 5, for a successful ADC reading, you can toggle the conversion trigger, wait until the ADC conversion is complete (you can achieve this by placing the `ADC14_isBusy()` function as the condition for a `while()` loop), and then get the ADC result from the configured memory (an example code was provided in Module 5, Topic 2, Lesson 3). Since the entire procedure takes only a few milliseconds, all these functions can be placed inside the timer ISR.

```
/* Read the ADC */
ADC14_toggleConversionTrigger();
while( ADC14_isBusy() );
adc_result = ADC14_getResult(ADC_MEM0);
```

Since we are using the ADC in 10-bit mode, the ADC converts the output voltage of the sensor (which is in range of 0 to 3.3V) to a digital number in the range of 0 to 1023. To convert the digital number to the actual voltage in your program, you can multiply the ADC results by $\frac{3.3}{1023} = 0.0032$ and store the result as a `float` variable:

$$voltage\ (V) = ADC\ value * \frac{3.3}{1023}$$

```
// convert the ADC value to the actual voltage
voltage = adc_result * (3.3/1023.0);
```

You can then convert the GP2Y0A21YK0F sensor output voltage (V) to an approximate distance (cm) by using the following equation:

$$distance\ (cm) = 27.726 * voltage^{-1.2045}\ (V)$$

which can be implemented in C by including the `<math.h>` header file and then using the `pow()` function, as:

```
// convert the voltage to the actual distance in cm
distance_var = 27.726 * pow(voltage,-1.2045);
// call the grading macro here when generating the grading outputs:
MACRO_LAB6_ADC_EVENT(distance_var); // see Section 6.10
```

This equation gives an estimate of the actual distance in centimeters, however the accuracy is enough for the purpose of this lab. For more details about the relationship between the sensor's output voltage and the inverse of the measured distance, refer to Lab Assignment 5.

If the actual distance is less than 20 cm, change the state of the robot to "stop", as explained in the Problem Statement in Section 6.3.

Note: Do not forget to clear the "Capture/Compare" interrupt flag at the end of the `Timer_A1` ISR. Not clearing the interrupt flag results in the program not exiting the ISR.

Note: After an ADC conversion is complete, you can print the results to the CCS console using the `printf()` function for debugging purposes. This can be done by placing the print statement inside the same `Timer_A1` interrupt handler. However, make sure to comment out the `printf()` line when using the grading macros and generating the output text file.

To use `printf()`, you must include the `<stdio.h>` header file. No additional configuration is needed beyond this to use the `printf()` function.

Design your code inside the main `while(1)` loop to check the robot state variable (which is changed inside the ISRs) and adjust the duty cycle of the `Timer_A0` channels to change the motors' direction and speed accordingly, as explained in the Problem Statement in Section 6.3.

Consult the DriverLib User's Guide and the MSP432 Technical Reference Manual for guidance as you set up your code, available under Course Resources on edX (MSP432-DriverLib-Users-Guide.pdf and slau356i.pdf).

6.8 Expected Performance

Connect the robot to the PC using the LaunchPad USB debugging cable, and open the SSH terminal with appropriate UART configuration.

Press the "w" key on the keyboard to send the forward ASCII command to the robot. The robot should start moving forward. Now press "a", "d", and "z" to move the robot to the left, the right and backwards respectively. Press the "s" key to stop the robot.

Press the "enter" or "return" key to send the ASCII code of `0x0D` to the robot. The program should return the actual distance (as ASCII characters) to the SSH terminal, and the distance (in *cm*) should be printed out on the terminal.

Move an object in front of the robot in the range of 10 to 80 cm, and press the "enter" or "return" key repeatedly to get the distance values. Make sure that the distance values are roughly correct.

Place an object in front of the robot in the range of 10 to 80 cm, and move the robot towards it. The robot should stop automatically when it gets to within 20 cm from the object.

Drive the robot toward an object such that the robot hits the object by one of the side bumper sensors. The robot should stop automatically upon detecting the collision.

Note: To be able to drive the robot for long distances while the USB debug cable is connected to the LaunchPad, you may use a USB extension cable between the robot and your PC.

Note: For more details on how to setup a SSH terminal, please refer to the Labs Helper Guide, available under Course Resources on edX (Labs-Helper-Guide.pdf).

6.9 Troubleshooting

A bump sensor(s) does not work:

- Verify that you have initialized the GPIOs and PORT4 interrupt correctly.
- Set breakpoints using the CCS debugger and verify that the application jumps to the interrupt handler when a bump sensor is pressed.
- Look at bump sensors signals with a voltmeter, oscilloscope or logic analyzer. The voltage on the microcontroller pin should be 0 when the associated bmop sensors is pressed and 3.3V when it is released. The operation of one switch should not affect the signals on the other switches.
- Look at the port registers in the debugger. With the debugger doing periodic updates, and the software running, you should see the port input register change with a switch press.

The chassis board or the LaunchPad or the SN754410 IC get hot:

- Double check the power (VSW and 5V) and GND wires connected to the breadboard.
- Verify the direction of flyback diodes.

Motors not do spin or get hot:

- Remove power and double check the connections.
- Recharge the batteries.
- Verify the six signals from the LaunchPad to the motor driver breadboard using a voltmeter, an oscilloscope or a logic analyzer.

One motor spins faster than the other:

- It is normal for the motor speeds to be ±20% of each other.
- Check for friction on the slower motor.

Timer interrupt does not occur:

- Verify that the timer interrupt is initialized correctly.
- Set breakpoints using the CCS debugger to check whether the program is stuck in any specific function.
- Check the `Timer_A1` registers using the CCS debugger and verify them with the MSP432 reference manual.

UART RX interrupt does not occur:

- Verify that the UART RX interrupt is initialized correctly.
- Make sure that the UART is configured with the correct baudrate and UART parameters during the initialization.
- Verify that the SSH terminal is using the right COM port.
- Verify that the SSH terminal is configured with the right UART parameters.

ADC readings are noisy, not correct:

- Verify that the ADC14 module is initialized correctly.
- After toggling the ADC conversion, make sure that the program waits for the conversion to be complete by checking the `ADC14_isBusy()` function before reading the results.
- Double check the conversion of the raw ADC value to the voltage value and then to the distance value by setting breakpoints using the CCS debugger.

Cannot open the COM port with the SSH terminal to MSP432:

- Make sure that the SSH terminal is using the right COM port.

6.10 Grading

An output text file with the file extension `.txt` should be uploaded to Vocareum in order for your lab assignment to be graded.

You can generate the grading output text file by the following procedure:

- Make sure that the `mechrev.h` header file is included in your main code.
- You must call `mechrev_setup()` during the initialization of your code (before the main `while(1)` loop). This will disable the chassis board motor drivers, and set up the encoders for the DC motors which are used for grading the lab assignment.
- You must call `MACRO_LAB6_INIT()` after initializing all the GPIOs, Timer, ADC, UART and other peripherals in your code (right before the main `while(1)` loop).
- You must call `MACRO_LAB6_ADC_EVENT(distance_var)` right after reading the ADC value and converting the raw data to distance. The macro input argument `distance_var` is the distance global variable (the variable name is optional) containing the last resulting distance value (in *cm*) from ADC module.
- You must call `MACRO_LAB6_UART_RX_EVENT(distance_var)` right after receiving a character via UART. The macro input argument `distance_var` is the distance global variable (the variable name is optional) containing the last resulting distance value (in *cm*) from ADC module.
  For example, if using the DriverLib functions to read UART RX, you need to call the macro after the function call `UART_receiveData(...)`:

```
// receive a character via UART
uint8_t rxData = UART_receiveData(EUSCI_A0_BASE);
// call the grading macro
MACRO_LAB6_UART_RX_EVENT(distance_var);
```

- You must call `MACRO_LAB6_UART_TX_EVENT()` right after transmitting a character via UART. No input argument is needed for this macro.
  For example, if using DriverLib functions to transmit through UART, you need to call the macro after the function call `UART_transmitData(...)`.
  Only use this macro after transmitting each ASCII character of the distance value, resulted from the `sprintf()` function. DO NOT call the macro after transmitting any appended "carriage return" or "new line" characters.

```
// transmit a character via UART
UART_transmitData(EUSCI_A0_BASE, stringData[i]);
// call the grading macro
MACRO_LAB6_UART_TX_EVENT();
```

- You must call `MACRO_LAB6_SWITCH_EVENT()` right after detecting a bump switch press inside the PORT4 ISR and before clearing the PORT4 ISR interrupt flag.
  Make sure that you call the macro BEFORE clearing the interrupt flag, otherwise the generated grading output will not be correct.

- After finalizing your program and placing the provided macros in the code, generate the grading output file by following the steps below:
  - Connect the USB debug cable to the MSP432 LaunchPad.
  - Turn the robot on by pressing the Power button on the TI-RSLK chassis board. Make sure that the blue LED on the chassis board is turned on.
  - Start the debug mode in the Code Composer Studio. The code will be paused at the beginning of the main loop. DO NOT press the Resume (F8) button in CCS yet.
  - Start your SSH terminal and connect to the associated COM port.
  - Now press the Resume (F8) button in CCS to start debugging the code. The grading outputs should start printing in the CCS console.
  - Test your code by driving your robot using UART commands sent from the SSH terminal, and evaluate the distance sensor and bump switches by driving the robot around obstacles. Make sure to try all the commands listed in Table 1. Press the "enter" or "return" key for at least 10 times over a period of time to read the distance values. Test your code for at least 30 seconds to collect enough data.
  - Stop the debug mode. Select all the outputs from the CCS console, copy them and paste them into an empty text file. Save the text file with an optional name (with file extension `.txt`) and submit it to Vocareum.

Warning: Make sure that the extension of your output text file is `".txt"`. Vocareum cannot open other file types, and would assign a zero grade to your work if the extension is other than `.txt`.

Warning: You can submit the lab assignment to Vocareum for up to 5 times. When *resubmitting* an assignment, make sure to first delete the old output file in the workspace (listed under the "work" directory) and then upload the new file, or use the same name for the new output file so it replaces the old one in the workspace.
DO NOT press the Submit button if there are more than 1 text file uploaded to the workspace, otherwise Vocareum would not be able to open the text file and would assign a zero grade to your work.

For more details on how to include header files, save data as text files, and submit the files to Vocareum, please refer to the Labs Helper Guide, available under Course Resources on edX (Labs-Helper-Guide.pdf).

6.11 Optional Modules (Non-graded)

6.11.1 Scan the environment using a servo motor and the distance sensor

You can scan the entire space in front of the robot by rotating the distance sensor using a servo motor. Add a servo motor to your design to change the direction of the distance sensor. The servo motor can be found at: https://www.pololu.com/product/2818

Servo motors are controlled by sending a PWM signal through the signal wire. The frequency of the control signal should be 50Hz (pulse period of 20ms). The servos can usually rotate 180 degrees (they have a physical limits of travel), and the width of pulse determines angular position of the servo. Generally pulses with 1ms duration correspond to 0 degrees position, 1.5ms duration to 90 degrees and 2ms to 180 degrees. Though the minimum and maximum duration of the pulses can sometimes vary with different brands and they can be 0.5ms for 0 degrees and 2.5ms for 180 degrees position.

You can use another Timer A module (e.g. `Timer_A2`) to generate an additional PWM signal. The PWM signal should have frequency of 50Hz , and the duty cycle should be varied from 5% to 10% (1ms to 2ms).

Figure 3: Servo motor (Image courtesy of Pololu).

6.11.2 Display the distance value in real-time

Add a basic graphic LCD display module to the TI-RSLK chassis board, and print the distance measurement on it in real time. One such character display is the Sparkfun 10168, which has a SPI serial communication interface.

To add the display to the chassis board, you first need to identify the appropriate pins at the rear of the chassis board. The board has been labeled to help identify the right section (the Sparkfun 10168 section is on top).

More information about attaching an LCD display to the TI-RSLK robot can be found at https://training.ti.com/attaching-display-ti-rslk-max

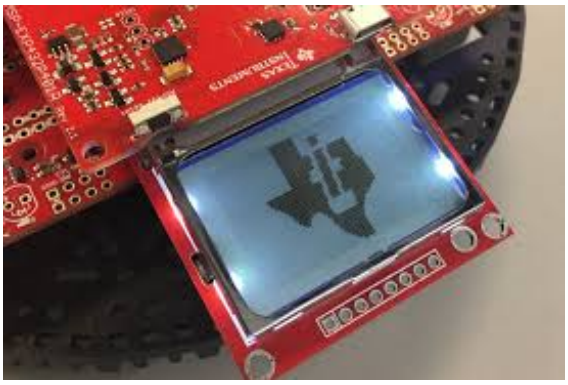

Figure 4: LCD display added to the TI-RSLK chassis board (Image courtesy of Texas Instruments).