Computer Systems Design Lab

SYSC 4805 (Fall 2022)

Instructor: Dr. Mostafa Taha
Carleton University
Department of Systems and Computer Engineering

Lab 4: Motor Driver Board and Wheel Encoder — Sept 29 and 30, 2022

In this lab, we will conduct the following Tasks:

- Task 1: Testing the Motor Driver Board
  - Experiment 1: Testing the Motor Driver Board using the AD2
- Task 2: Testing the Wheel Encoders
  - Experiment 1: Testing the wheel encoder using AD2
  - Experiment 2: Reading time using the Arduino Due Board
  - Experiment 3: Testing the wheel encoder using the Arduino Due Board
- Task 3: Completing the Robot setup and taking a first drive .
  - Experiment 1: Testing the loaded motors
  - Experiment 2: A 0-radius 90° turn

Report Instructions

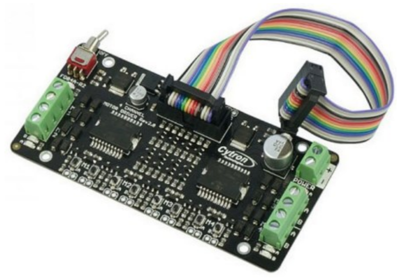
General:

- The report should be submitted before the beginning of the next lab. After that, there will be a penalty of 25% if the report submitted in the following 24 hours. If the report submitted 24 to 48 hours late, it would receive a penalty of 50%. No report will be accepted afterwards.
- The cover page should include your group names, the lab number, and the lab date.
- Organization is extremely important: Divide your report into sections, use captions for figures and headings for tables, …etc.

Lab 4: Your report should include the following

- All the Table completed.
- Answer to all the discussion questions in the experiments of each task.
- The required screenshots and pictures in the experiments of each task.

Task 1: Testing the Motor Driver Board



Introduction

The Cytron motor driver board does not have a plastic casing and should never be placed directly on top of the metal chassis of the robot. Doing so will create a short circuit inside the motor driver board.

Also, in all the experiment, we will separate between the 5V supply to power the board itself and the motors, and the 3.3V control inputs.

The Cytron FD04A Rev2.0 motor driver control board provides a 4-channel bi-directional control for 4 brushed DC motor. It provides the required power levels to operate the motors at a range of 7 - 25VDC with a maximum continuous current of 1.5A per channel, while being controlled by 3.3V or 5V logic level input. The board also provides test switches and test LEDs for each channel and each direction (total of 8 switches and 8 LEDs). The maximum PWM frequency that the board can support is 10 KHz.

How it works

This motor driver control board relies on two L298P motor driver chips. Each L298P chip holds two H-Bridge circuits. The operational concept of the H-Bridge is as follows. The rotation direction of a DC motor is controlled by controlling the direction of current flowing through its armature winding. For a DC motor with armature terminals A and B, if the current flows from A to B the motor will rotate in one direction, while if the current flows from B to A the motor will rotate in the opposite direction. Fig. 1 shows a typical H-Bridge. The H-Bridge circuit is used to deliver power to the DC motor (between V_{CC} and Gnd), while being controlled by digital control signals (En, In_1 and In_2) that defines the rotation direction of the motor.

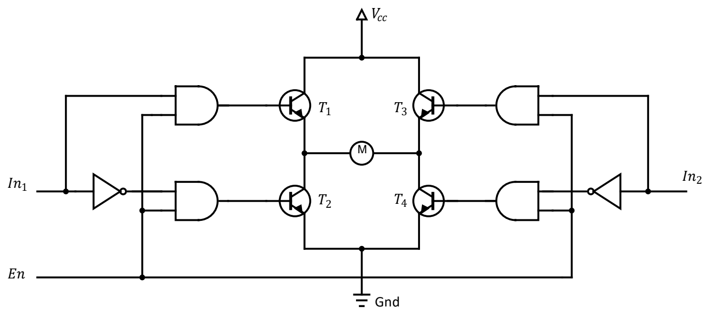


Fig. 1 A typical H-Bridge with enable input

The H-Bridge circuit is essentially based on the concept of using transistors as switches. T_1 and T_2 cannot be ON at the same time. Similarly, T_3 and T_4 cannot be ON at the same time. Let's always set $In_2 = \overline{In_1}$, then, if In_1 is HIGH, the transistors T_1 and T_4 will be ON to derive the motor in one direction. On the other hand, if In_1 is LOW, the transistors T_3 and T_2 will be ON to derive the motor in the other direction. The Enable signal can be used to turn off all the transistors and inhibits the current flowing regardless of the control signals. This Enable signal is typically used to control the motor speed using PWM signal.

The following images descript the connection header of the motor driver board, from its datasheet.

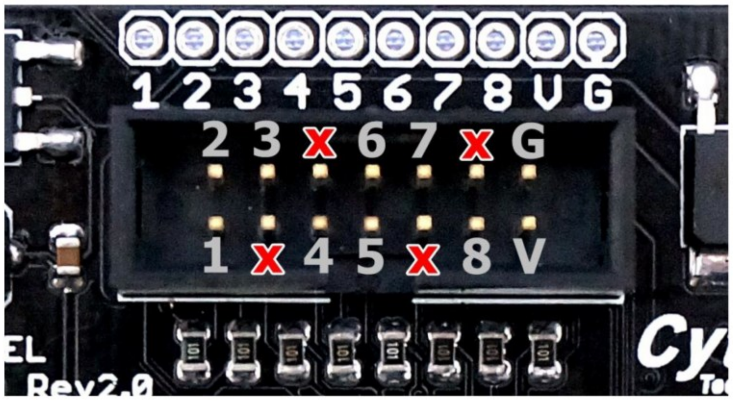


Figure show the pin orientation of the 7 x 2 IDE socket and extra pad. The function of each pin is described as below:

PIN	DESCRIPTION
1	Motor 1 direction
2	Motor 1 enable/Speed Control
3	Motor 2 direction
4	Motor 2 enable/Speed Control
5	Motor 3 direction
6	Motor 3 enable/Speed Control
7	Motor 4 direction
8	Motor 4 enable/Speed Control
V	Output voltage 5V
G	Ground
x	Not connected

Table 1

Enable pin	Direction pin	OUT A	OUT B
0	X	LOW	LOW
1	0	HIGH	LOW
1	1	LOW	HIGH

Note that the V pin is an output 5V. Similar to the Arduino Due Board, this is an output pin that is provided to power other circuits if needed, however it should never be used to power the motor driver board itself.

The direction pin is set to select the rotation direction of the motor, while the enable pin is used to control its speed using PWM.

A technical note on controlling the speed using PWM:

PWM signal is a periodic rectangular wave shown in Fig. 2. The signal has a period T_p and its amplitude stays at the high level for interval T_H and goes to low level for T_L such that:

$$T_p = T_H + T_L \tag{1}$$

The PWM signal is characterized by three main features:

- Frequency (F): Which is the reciprocal of the period $F = \frac{1}{T_P}$.
- Duty Cycle (D): Which describes how much time the signal stays at the high voltage with respect to the total period $D = \frac{T_H}{T_P}$.
- The Amplitude V_H.

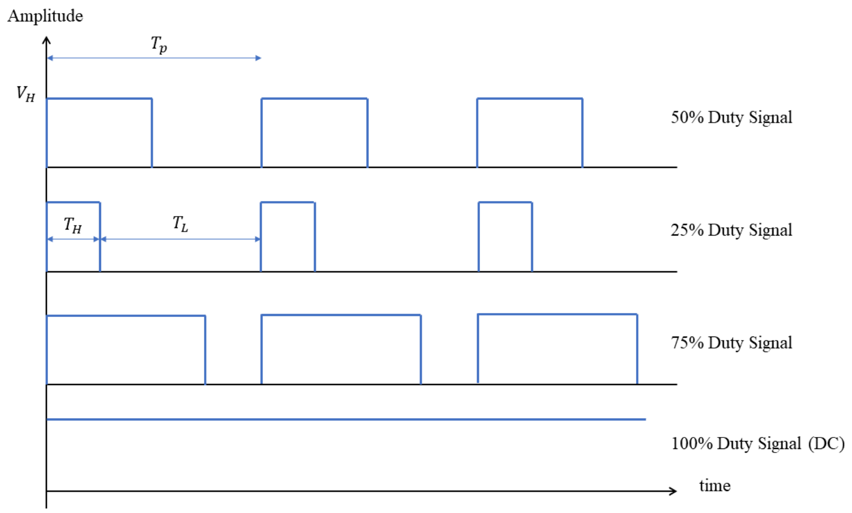


Fig. 2 Pulse Width Modulated Signal (PWM)

Suppose that a PWM wave is applied to a resistor with value R, the average power dissipated at the resistor can be calculated by calculating the energy dissipated in one period divided by the period width:

$$Power = P = \frac{1}{T_P}\int_0^{T_p} \frac{V^2(t)}{R}\,dt = \frac{1}{T_P}\frac{V_H^2}{R}T_H = \frac{V_H^2}{R}D \tag{2}$$

Equation (2) shows that the power delivered to the resistor is directly related to the duty cycle in a linear fashion. The equation reduces to the popular DC power $\frac{V_H^2}{R}$ in case of 100% duty cycle which is DC signal.

A PWM power signal can be applied to energize a DC motor. The power delivered to the motor is directly related to the duty cycle and so the torque and speed. One can argue that applying an ON/OFF signal like this to the motor will cause it to rotate and stop in an undesirable way and so the robot will be also moving in an erratic way. This is correct if the PWM signal is "slow", but the trick is indeed in the frequency. The motor is a mechanical device that has an inertia, it cannot rotate or stop rotating instantly. Instead, it has to accelerate or to decelerate to change its state. We can take advantage of this mechanical fact by applying a PWM wave with a relatively high frequency such that the motor inertia prevents it from following the electrical signal ON/OFF cycle mechanically. Instead, the motor will rotate with an average speed as if it is powered by a DC voltage with amplitude equal to $V_H\sqrt{D}$. A DC motor, due to its inertia, senses the change in the duty cycle as a change in the applied DC voltage. Hence, the output speed and torque of a DC motor can be effectively controlled by controlling the duty cycle of the PWM wave with suitable frequency.

Experiments

In the following experiments, we will investigate the basic functionality of the Cytron motor driver board.

Experiment 1: Testing the Motor Driver Board using the AD2

In this experiment, we will use the AD2 to test the Cytron motor driver board

Require Equipment

- Battery holder, with 5 batteries installed.
- Cytron motor driver board.
- The DFR Robot with motors & wheels mounted (we will demonstrate using one motor).
- Analog Discover 2.
- Jumper wires.

Procedure

1- Install the Batteries into the Battery holder.

2- Connect the Red +ve wire and the Black Gnd wire of the battery holder to the + and – of the Cytron motor driver board power supply terminal block (green in color). You can use jumper wires and bread board to extend the connections. Connect the ↓ Gnd wire of the AD2 to the Gnd wire of the battery to create a uniform Gnd.

3- Connect the Red and Black wires of one of the front motors (any motor without the wheel encoder to exclude any effect from the when encoder module) to the A and B terminals of Channel 1 (named M1 on the board). If you use the entire DFR Robot for this demonstration, you can put any object underneath it to raise it up to let the wheels run freely.

4- Testing using the provided pushbuttons: Use the motor driver board provided test pushbuttons of the M1 motor to perform an initial testing for the connection and rotation direction of the motor. You should notice that the A pushbutton moves the motor in one direction and lighting the LED A of the M1 port, while the B button moves the motor in the other direction.

5- Use the AD2 to supply 3.3V: Open the WaveForms software, open the “Supplies” module, and set the V+ channel to 3.3V.

6- Test using direct jumper wires: M1 motor should be controlled by the direction and enable/speed pins (PIN 1 and PIN 2, respectively). Connect the M1 direction pin, PIN 1, of the Cytron board to the Gnd line while connecting the M1 enable/speed pin, PIN 2, to the 3.3V output of the AD2 (wire V+). You should notice that the motor will start moving in one direction and M1 LED will be ON. The moving direction of the motor can be reversed by connecting the M1 direction pin, PIN 1, to the 3.3V output of the AD2.

Remember that if anything went against what you expect, you can always turn off the motor motion by using the power toggle switch on the Cytron board.

7- Test using the AD2: Connect the jumper wires as follows. Use bread boards whenever needed.
   a. Connect the Battery holder to the Cytron board.
   b. Connect the AD2 Gnd pin (PIN ⏚) as well as the 1- pin to the Gnd line. This will create a uniform Gnd to all the circuits. The 1- is the reference of Ch1 oscilloscope.
   c. Connect the M1 direction pin (PIN 1) to the Gnd pin (PIN G). This will keep the rotation direction of the M1 channel to one direction.
   d. Connect the M1 enable/speed pin (PIN 2) to the AD2 W1 pin (Wavegen output channel 1), as well as 1+ pin of the AD2. The AD2 function generator will provide the required PWM signal, while the oscilloscope Ch 1 functionality will monitor the PWM signal.
   e. The AD2 pins 2+ and 2- (the oscilloscope channel 2 wires) to the M1 output port of the Cytron board. You can keep the motor terminals connected for better demonstration.

8- From the Welcome Screen of the AD2:
   a. Select the wavegen module. Activate Channel 1 (W1)
   b. Configure the wavegen channel to generate a "Pulse" wave, with frequency 100Hz, Amplitude 3.3V, Offset 0V and Symmetry (duty cycle) of 50%. This is equivalent to use a "Square" wave, with amplitude 1.65V, and offset +1.65V. Run the Wavegen module.
   c. Select the Scope module, run both Ch1 and Ch2, with trigger from Ch1, rising edge at 2V

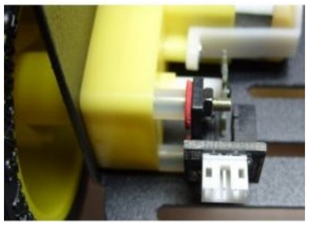
9- Test and take notes: In this experiment, we won't measure the actual speed of the motor since we are not using the wheel encoder. This is actually a task in the next experiments. However, we will just "listen" to rotation of the motor, and its relationship with the input and output signals.
   a. Change the duty cycle from 0% to 100%, with step 20%. What do you notice? Note that you can continuously change the duty cycle using the up and down arrows while keeping the cursor on the Symmetry input.
      Also, note that you can display the Wavegen window alongside the Scope window.
   b. Reset the duty cycle to 50%, and change the frequency from 10Hz, 50Hz, 100 Hz, 500 Hz, 1KHz, 20 KHz and 200 KHz. Change the time base of the Scope module to appropriate values for each change. What do you notice?
      Make special attention to the 1Khz case, as this is the default frequency of the Arduino AnalogWrite() function in the case of Arduino Due board.
   c. Write down in your report the frequency that leads to the highest speed under each select of the duty cycle.

10- Discussion Questions:
   a. At 20KHz, the Cytron motor driver board should still be providing the correct signal. You can check the motor driver board output on Ch 2 by reversing the M1 direction pin between 3.3V and Gnd which will flip the Ch 2 signal of the oscilloscope with respect to Ch 1. However, the motor may not be moving at all. Why?
   b. You should notice that at 50% duty cycle, the 1Khz default frequency of the AnalogWrite() function does not lead to the fastest motor rotation, as the case with frequencies in the range 50 to 200 Hz. 10 Hz and below the motor is skipping, and above 300, the motor is not receiving the appropriate power.

Take screenshots of all the testing cases and include them in your report.

Task 2: Testing the Wheel Encoders

Prepared by Mahmoud Sayed



Introduction

The motor encoder is used to measure and count the revolutions of the motor shaft. Its working principle is described in Fig. 8, where a gear is connected to the rotation shaft of a motor to rotate with it with the same speed. As the gear rotates, its teeth obstruct the light emitted from a light source in front of the gear, causing the photo detector behind the gear to receive the light signal in the form of pulses with frequency that is equivalent to the frequency of the teeth cutting the light path. Thus, the photo detector produces an electrical signal in the form of pulses that replicates the same frequency.

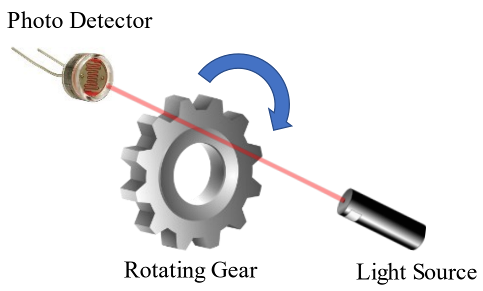


Fig. 3 Motor Encoder working principle.

Some motors come with built-in encoders. In case of geared motors, the encoder may be connected to the internal shaft for greater accuracy (more pulses per wheel revolution, depending on the gear ratio). Some motors come with quadrable encoders, where there are two output signals with phase difference that can be used to know the rotation direction in addition to measuring the revolutions.

Knowing the number of revolutions, the time spent, and the wheel diameter, the linear velocity of the robot can be directly calculated. The reading from the encoder is usually used as a feedback signal in a closed loop control algorithm/circuit to control the speed of the motor and the robot in general. However, extra care should be given while using the encoders for wheeled robots to calculate linear velocity because of the wheel slipping problem. If the wheel slips, i.e. the wheel rotates in place and the robot does not move, the readings of the encoders will be correct in term of wheels revolutions but not in terms of linear velocity.

Experiment 1: Testing the wheel encoder using AD2

In this experiment, you should test the basic functionality of the wheel encoder and confirm its operation using the Analog Discovery 2. The setup and procedure of this experiment is very similar to Task 1 Experiment 1, while observing the response from the wheel encoder on the AD2 oscilloscope.

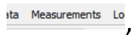
Required Equipment:

Same as Task 1, Experiment 1.

Procedure:

1- Connection: Use the same connection setup of Task 1, Experiment 1. Do the following changes:
   a. Connect one of the back motors, with the wheel encoder installed, instead of the front motors.
   b. Disconnect the Oscilloscope Ch2 inputs of the AD2 (2+ and 2- wires) from the M1 output port. Connect the reference 2- wire to the Gnd line. Keep the Ch1 as is connected to the AD Wavegen Ch1 (W1 wire) and Gnd as is.
   c. Connect the wheel encoder cable, with three wires. Connect the VCC supply Red wire of the sensor module to the V+ output of the AD2, making sure that the V+ channel of the "Supplies" module is activated and set to 3.3V, while V- channel is deactivated. Connect the Black line of the sensor module to the Gnd line. Connect the Green wire of the sensor module to the Ch2 input of the AD2 (2+ wire).

2- Supply the PWM signal: Run the Wavegen Ch1 (W1 wire) on "Pulse", frequency 100Hz, Amplitude 3.3V, Offset 0V and Symmetry (duty cycle) of 50%. This is equivalent to "Square", Amplitude 1.65V, with offset 1.65V.

3- Observe and Note: Open the Scope module of the AD2 and activate the 2 channels. Ch1 should be monitoring the PWM signal from the AD2, while Ch2 is monitoring the signal output from the wheel encoder.
   a. You should notice that, when there is no barrier between the light source and the detector, the voltage level on the signal output wire of the sensor module is high (3.3V). When there is a barrier, the signal level goes low. You can test this be turning the motor driver board OFF and turn the wheel by your own hands. You can also use a small piece of paper as a barrier.
   b. Turn the motor driver board back ON. On Ch 2, the signal from the wheel encoder sensor module, should show a square wave with an almost 50% duty cycle. The positive half is the time of no barrier in front of the light source, and the ground half is the time of one tooth from the gear. Since the wheel encoder has 10 gaps (please confirm this count on your sensor module), one full rotation of the wheel should show up on the Scope as 10 full cycles.
   c. On the top menu of the Scope, select Measurements: 

, Add, Defined Measurement, Channel1, Horizontal, Period. This should report the Period of the wheel encoder signal output.

4- Take Measurements
   a. Test different frequencies and different duty cycles by following Step 9 from Task 1, Experiment 1.
   b. What is the highest speed of your robot? You can find the smallest period at 100% duty cycle (that is a 3.3V DC signal), multiply the period by 10, to account for the 10 teeth of the wheel encoder gear. This will be the time for one full rotation. Divide the wheel circumference by the time to get the speed.
   c. Write down the maximum speed in cm/s in your report.

Experiment 2: Reading time using the Arduino Due Board

In this experiment we will let the Arduino Due measure the period of the wheel encoder output signal.

Of course, there are many methods that could be used by the Arduino Due board to measure the period, i.e., the time between having a rising edge on the wheel encoder output signal to having another rising edge. The experiments here are very similar to Lab 2 when we needed to create PWM signal using the Arduino Due board. In the following, we will go through these methods one at a time.

Required Equipment:

- Battery holder, with 5 batteries installed.
- Cytron motor driver board.
- The DFR Robot with motors & wheels mounted (we will demonstrate using one motor).
- The Arduino Due board.
- Jumper wires.

Procedure:

1- Connection: Connect the battery holder to the Cytron motor driver board.
Connect the Gnd of the Arduino Due board to the Gnd line. The Arduino will be powered by the USB connection.
Connect one motor with wheel encoder module to the M1 output of the motor driver board.
Connect the red and black wires of the wheel encoder module to the 3.3V of the Arduino board, the Gnd line respectively.
The green signal output wire of the wheel encode will be connected to different pins on the Arduino board depending on the experiment.

2- Test the timer: In every experiment, we will test the timer by moving the wheel by hands or by pressing on the M1 pushbuttons of the Cytron motor driver board. Using the pushbuttons will deliver a full power, DC signal to the motor.

Experiment 2.A: Measuring time by polling any digital pin

You can poll any digital pin and check whenever the input signal is HIGH. You can use the millis() function to capture the time between two consecutive HIGH inputs.
Needless to say, this method is not recommended as it completely blocks the CPU. In addition, the measurement is not accurate since the measurement is not triggered by the signal change, but rather when the CPU has some free time to checks this change. This won't make a big difference in a sample testing case, but in complex systems, this time is not predictable since the CPU may be busy performing interrupt service routes.

Nothing is required in this experiment.

Experiment 2.B: Measuring time by using Interrupts

You can connect this wheel encoder signal to any digital pin and assign this pin as a hardware interrupt. The interrupt will command the CPU to immediately process the interrupt service routine. Then, use the millis() function inside the interrupt service routine to measure the time. Note that, once inside an interrupt service routine, the millis() return value does not change. This method has a better performance since the signal change immediately commands a time

measurement, delayed by a couple of instructions to process the interrupt service routine.
- Connect the green signal output wire of the wheel encoder module to pin 2 on the Arduino Due board. Test the following code:

```
/*Connect the wheel encoder signal to pin 2 in the Arduino.*/
volatile unsigned long t = 0;
volatile unsigned long t_old = 0;
volatile boolean CaptureFlag = 0;
void setup() {
  Serial.begin(115200);
  pinMode(2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(2), time_ISR, FALLING); //+ve edge in PullUp.
}
void loop() {
  if (CaptureFlag) {
    CaptureFlag = 0;        //Reset the flag,
    Serial.println(t-t_old); //Print the time in msec,
    t_old = t;
  }
}
void time_ISR() {
  t = millis();
  CaptureFlag = 1;          //Inform the main loop of an update.
}
```

The code is self-explanatory. The Arduino provided attachInterrupt() function is used to attach a hardware interrupt on Falling edge of the digital pin 2 to the interrupt service routine timer_ISR().

Move the wheels by hand or use the pushbuttons on the motor driver board to test the code. Ideally, in full speed motor rotation (100% duty cycle), one full period can read between 50 and 60 msec. This corresponds to the time between the edge of one tooth and the next.

Note that, sometime, the timer readout is too small to be real, in the range of 0 to 20 msec. This is mainly noise due to a variety of hardware issues inside the sensor module as well as the Arduino interrupt handler.

In order to reduce the noise, a process called *debouncing*, we can ignore any reading that has an increment of less than 25 msec. We know that a normal reading can never be less than 25 msec. Update the time_ISR() function as follow:

```
void time_ISR() {
  if(millis() - t_old > 25){
  t = millis();
  CaptureFlag = 1;          //Inform the main loop of an update.
  }
}
```

Perform you own experiments, to record and update the debouncing period as necessary. In your report, write down the fastest speed, the debouncing period and screenshots.

Experiment 2.C: Measuring time by the Timer Counter unit.

You can configure a Timer Counter unit to measure the time by itself and store the time in a register for whenever the CPU is free to read it, or use the timer capture to raise an interrupt for the CPU to immediately read it.
In general, this is the best method to measure time in microcontrollers since the time is captured independently from any CPU activity at the resolution of the timer clock frequency.

This method is exactly similar to the experiment in Lab 2, Task 3 (Testing the Ultrasound Sensor), Experiment 3. While changing the loading time and triggering time to be both on the rising (or falling) edge of the timer input.

- Connect the green wire, wheel encoder signal output, to the pin A7 on the Arduino

Test the following code:

```
/*Connect the wheel encoder signal to pin A7 in the Arduino.*/
void setup() {
  Serial.begin(115200);
  PMC->PMC_PCER0 |= PMC_PCER0_PID28;     // Timer Counter 0 channel 1 IS TC1, TC1 power ON
  TC0->TC_CHANNEL[1].TC_CMR = TC_CMR_TCCLKS_TIMER_CLOCK1 // capture mode, MCK/2 = 42 MHz
                              | TC_CMR_ABETRG      // TIOA is used as the external trigger
                              | TC_CMR_LDRA_RISING// load RA on rising edge of TIOA
                              | TC_CMR_ETRGEDG_RISING; // Trigger on rising edge

  TC0->TC_CHANNEL[1].TC_CCR = TC_CCR_SWTRG | TC_CCR_CLKEN; // Reset TC counter and enable
}

void loop(){
  volatile uint32_t CaptureCountA;
  CaptureCountA = TC0->TC_CHANNEL[1].TC_RA;
  printf("L#, Group#: %f msec \n", CaptureCountA/42000.0); //the .0 is required to type casting.
}
```

Move the wheels by hand or use the pushbuttons on the motor driver board to test the code. While the code should be working properly as intended, it always print new results even if there are no changes in the input signal. This is because the loop() is continuously polling the TC.RA register, which always have the last reading stored.

A better code it to let the rising edge of the input signal to initiate an interrupt and read the TC.RA only within the interrupt service routine.

As previously introduced in Lab 2:

1- Copy the previous code.
2- Define two global variables, before the setup function, to be accessible from the interrupt function as well as the main loop function

```
volatile uint32_t CaptureCountA;
volatile boolean CaptureFlag;
```

3- Add these two lines at the end of the setup function to enable the interrupt.

```
TC0->TC_CHANNEL[1].TC_IER |= TC_IER_LDRAS; // Trigger interrupt on Load RA
NVIC_EnableIRQ(TC1_IRQn);                  // Enable TC1 interrupts
```

4- Modify the loop function and add a new interrupt handler function, outside the main loop, as follows.

```
void loop(){
  if (CaptureFlag) {
    CaptureFlag = 0;        //Reset the flag,
    printf("L#, Group#: %f msec \n", CaptureCountA/42000.0);} //the .0 is required to type casting.
}

void TC1_Handler() {
  uint32_t status = TC0->TC_CHANNEL[1].TC_SR; //Read status register, Clear status
  if (status & TC_SR_LDRAS) {                 // If ISR is fired by LDRAS then ....
    CaptureCountA = TC0->TC_CHANNEL[1].TC_RA; //read TC_RA
    CaptureFlag = 1;         //Inform the main loop of an update.
  }
}
```

Move the wheels by hand or use the pushbuttons on the motor driver board to test the code. You should notice that the timer reports are more accurate. If you notice any noisy reading, you will have to implement a debouncing code to ignore reports within too small time intervals.

Experiment 3: Testing the wheel encoder using the Arduino Due Board

As you may expect, in this experiment we will let the Arduino Due create the PWM output signal to control the speed of the wheel, while also measuring the wheel encoder output signal, do the math to report the speed in cm/s directly over the serial monitor.

Required Equipment:

- Same as Experiment 2.

Procedure:

1- Connection: Connect the battery holder to the Cytron motor driver board.
Connect the Gnd of the Arduino Due board to the Gnd line. The Arduino will be powered by the USB connection.
Connect one motor with wheel encoder module to the M1 output of the motor driver board.
Connect the Motor 1 direction pin (pin 1) to any output pin. Connect the M1 enable/speed pin (pin 2) to an appropriate pin on the Arduino Due board depending on the selected method of generating the PWM signal, following Lab 2, Task 3.
Connect the red and black wires of the wheel encoder module to the 3.3V of the Arduino board, the Gnd line respectively.
Connect the green signal output wire of the wheel encode to an appropriate pin on the Arduino Due board depending on the method of measuring the time, following Experiment 2 of this lab.

2- Generate PWM signal: Use any method tested in Lab 2, Task 3 (Testing the Ultrasound Sensor), Experiment 1 (Generating the trigger signal using the Arduino Due Board) to generate PWM according to the frequency and duty cycle used in Table 1.
3- Measure the speed: Use any method tested in the previous Experiment 2 of this lab, best by using the Timer Counter module, to read the time of a complete period of the wheel encoder signal output. Measure the wheel circumference and update the time measurement code appropriately to report the speed in cm/s (not just the time). The reported speed should be the running average of the last 10 readings, i.e, always store the last 10 readings and report their average. The average can be computed using the mean or the median. The mean is better if there are no noisy readings, while the median is better if you have occasional noisy readings.
4- Take measurements of unloaded motors: While the robot is raised above the ground, and the wheels are allowed to run freely, take measurements and fill in Table 1. Once again, 1KHz has a special importance since this is the default frequency of the AnalogWrite() function.

Table 1: PWM parameters effect on unloaded motors.

Duty Cycle	F = 10 Hz	F = 50 Hz	F = 100 Hz	F = 500 Hz	F = 1 KHz
0%					
20%					
40%					
60%					
80%					
100%					

Task 3: Completing the Robot setup and taking a first drive

A note on Robot Steering

There are different kinds of steering mechanisms for wheeled vehicles depending on the mechanical structure of the vehicle and the wheels. Number of wheels, their location, wheels type, and suspension type are examples of the factors that affects the steering mechanism. In our case, we have a 4-wheel-drive structure with 4 independent DC motors. Each wheel is connected to a motor through fixed rotating axis. Thus, the wheels mechanical orientation is fixed and directed towards the vehicle front. In this case, the steering type that can be used is called "***skid steering***", which depends on changing the wheels rotation velocity with respect to each other to change the robot's orientation. Fig. 6 shows describes how the robot's orientation changes due to difference in wheels rotation velocity.

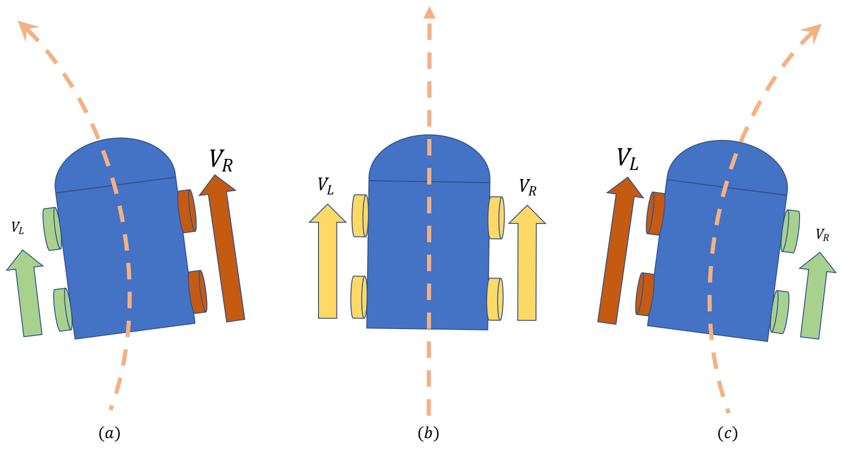


Fig. 4 Robot's skid steering. (a) steering to left $V_L < V_R$ (b) straight path $V_L = V_R$ and (c) steering to the right $V_L > V_R$

Let's assume that both left wheels (front and back) are rotating with velocity V_L and both right wheels are rotating with velocity V_R. If $V_L < V_R$ then the robot will turn to the left. If $V_L = V_R$ then the robot will follow straight path. If $V_L > V_R$ then the robot will turn to the right. Since the wheels are mounted on fixed axes, they cannot change their direction mechanically, they have to "skid" to steer the robot body and hence the name "skid steering". The skidding behavior becomes more obvious with small rotation radius, and it reaches its extreme in the cause of 0-radius turn at which $V_L = -V_R$ so the robot turns in place without linear movement. The skidding behavior makes predicting the robot movement giving the control inputs alone a hard task, as the robot movement depends on a combination of linear movement and skidding with the later one depending on several factor including the friction coefficient of the wheels. There are a number of models in the literature that were developed to describe the skid steering mathematically confirmed by simulation and experimental measurements. One reference[1] showed that the 4wd skid steering robot movement can be approximated by an equivalent 2-wheels differential steering robot is shown in Fig. 5.

[1] Wang, Tianmiao, et al. "Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor." *Sensors* 15.5 (2015): 9681-9702.

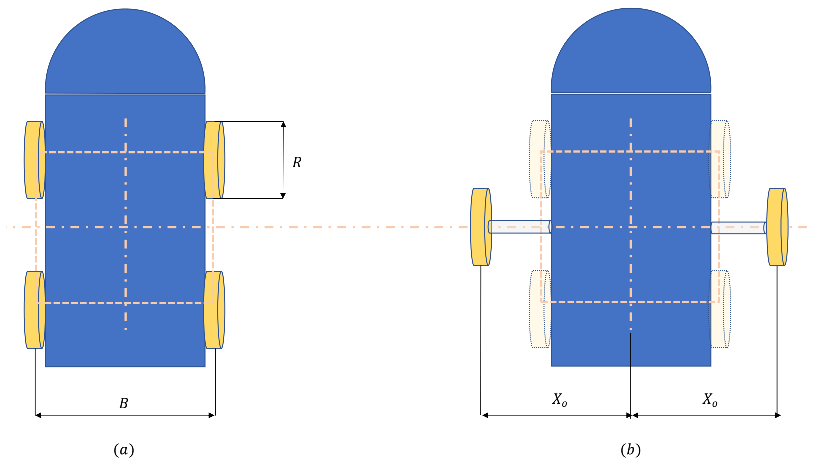


Fig. 5 (a) 4WD skid steering model and (b) equivalent 2-wheels differential steering model.

The differential drive steering is used for a robot that has only 2 motorized wheels mounted on fixed axes with no mechanical steering in addition to a passive 3rd wheel for balance (usually a castor wheel). Differential steering model provides easier path calculation as the turning process does not include skidding. For symmetrical skid steering model as the one provided in Fig. 5(a), its path would be equivalent to the path followed by the differentially steered 2-wheels robot according to the following equations [1]:

$$X_o = \frac{\chi B}{2} \tag{3}$$

where $\chi \geq 0$ is a dimensionless parameter that counts for the skidding effect. The value of χ does not depend only on the robot mechanical dimensions, it's also affected by the robot's moving kinematics. In particular, it depends on linear velocity of the right and left wheels as in the λ parameter:

$$\lambda = \frac{v_l - v_r}{-v_l + v_r} \tag{4}$$

Where v_l is the linear velocity of the left wheels and v_r is the linear velocity of the right wheels. The parameter χ has an inverse proportion with λ, and reaches its maximum of 1.48 when $\lambda = 0$, which means that the velocities of right and left wheels are the same but in opposite directions. In this case, the robot is called to be turning in a curve with 0-radius i.e. it turns "in place" around its center. This 0-radius turning is an important feature of skid, differential, and dual differential steering, as it allows the robot to turn in very small space which increase its maneuverability. Using the robot dimensions, equation (3), and the value of χ for 0-raduis turn, we can calculate the number of complete wheel revolutions to achieve a 0-radius turn with of an angle ω_o (rads).

$$\#Revolutions = \frac{Arc\ lengh}{\text{Wheel circumfrance}} = \frac{\omega_o \times X_o}{\pi R} = \frac{\omega_o \times \chi \frac{B}{2}}{\pi R} = \frac{\omega_o \chi B}{2\pi R} \tag{5}$$

Completing the Robot setup

We first begin with some important cautionary notes:

1- Make sure you always have a common ground to all your components, the battery holder, the Arduino board, motor driver board, all the sensors, the testing tools, and the different circuits that you build on top of the robot.

2- While working on your actual system, you should NEVER use the 3.3V or 5V output pins of the Arduino Due. The 3.3V output of the Arduino is good for testing a single sensor, but it can't provide all the current requirements to power all your sensors at the same time. A small Buck Regulator will be distributed to all the kits. If you are really careful, you can use the Buck regulator to supply some sensors, while the Arduino Due to supply *other* sensors, making sure that you never connect the two sources together. The source with lower voltage will be a load on the other source.

3- You have to make an accurate, detailed and complete power budget for your system. Refer to the appropriate datasheets to know the power capability of your power sources (the batteries, the buck regulator and the Arduino Due board if used), and the power requirement of the Arduino as a load and all the sensors in your system. Power deficiency is one of the most common reasons of weird and erroneous behaviour in battery-powered embedded systems. While, at low power, the motors will be visibly suffering, other electronic components and sensors will silently die.

4- Use A/D feature of the Arduino Due board to read the voltage out of the buck regulator supply, and report error message in the terminal and/or turn off the system when the voltage output is lower than what is required to run the system.

5- Note that you have two different power sources in your system. 6V from the battery holder (each rechargeable battery unit is rated at 1.2V) and 3.3V coming from the Buck Regulator. The 6V should only power the motor driver board, the Arduino Due board itself and buck regulator. The 3.3V should power all the sensor, including the wheel encoder.

6- The robot power switch should be between the +ve Red wire from the battery holder on one side and the Arduino Due board, the motor driver board and the buck regulator on the other side.

7- The motor driver board should never be placed directly on the metal chassis. The metal chassis will create a short circuit inside the motor driver board.

8- Use any painter's tape, can be provided in the lab, on top of the chassis before using any stick pads to setup your sensors. The painter's tape can later be removed without leaving any residue.

9- Whenever possible, use more than one sensor to read any experimental variable. Compare the sensor readings. If the readings are close to each other, report the average. If the readings are significantly different, report error and stop the system.

10- Try to use as clear and readable error codes as possible, as many as possible distributed everywhere in your code. The error codes must have unique ID's. Use the LED output, pin 13, to report visual signal on errors, and use the terminal to report the error to the PC. When the Arduino is working independently away from the PC, use the EEPROM to create a log of error messages for later debugging [here].

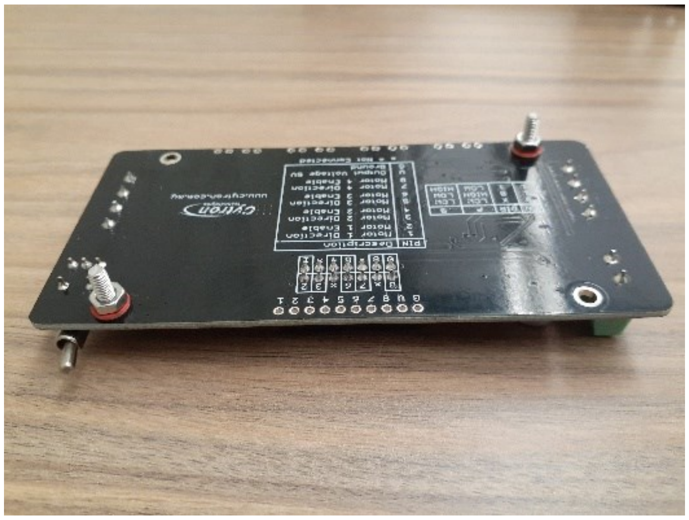
11- Use the solid core wires as much as possible, specially to connect all the boards on the surface of the chassis. Use jumper wires of appropriate length to connect sensors that are on different levels. Try to be organized as much as possible. Wire misplacement is another very common reason for weird and erroneous behaviour.

12- For the robot to move forward, all the motors have to drive forward, but the motors in one side are mounted in an opposite direction compared to the other side. This means, for your control code consistency, if you decided that the red wires will go to the A sockets for the right-side motors, then the red wires of the left-side motors should go to the B sockets.

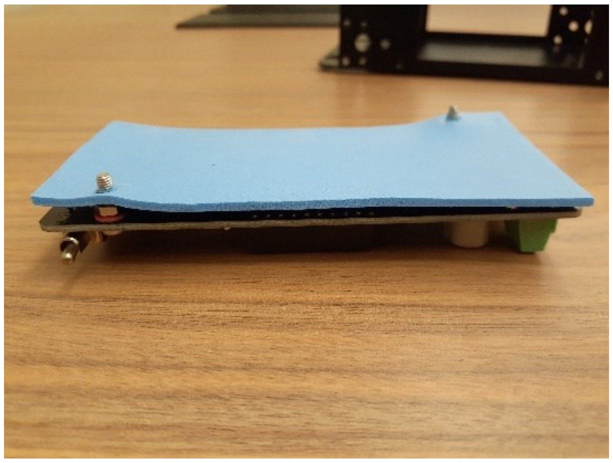
The following is a "suggested" placement of the motor driver board and the Arduino Due board. It is up to your team to decide on the final placement as your unique design:

1- You should collect the following from your TA:
   - 4x Philips Truss Screws (M3x8mm) or (M3x10mm) or (M3x12mm)
   - 2x M3 Washers
   - 6x M3 Nuts
   - 1x 2-inch x 4-inch foam sheet.

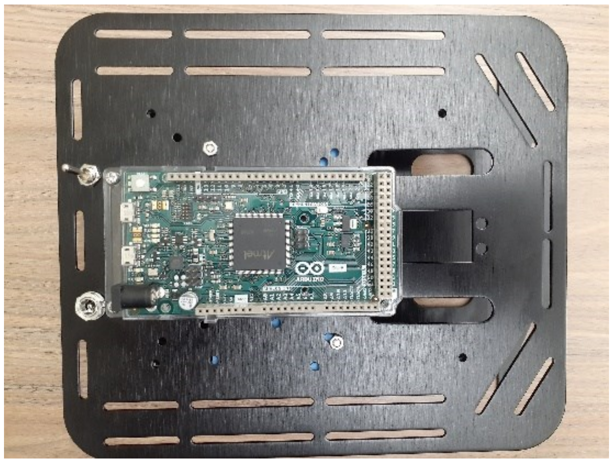
2- Use 2x Philips Screws and 2x washers as spacers underneath the Cytron motor driver board. Unfortunately, there are no holes on the chassis that can match all the four mounting screws of the motor driver board. Hence any two should be selected. One solution is in the following image.



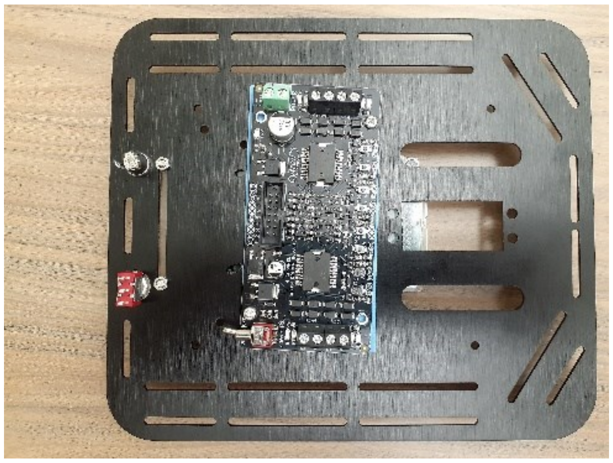
3- Put the foam sheet to entirely cover the actual boar. Since the board is fixed by only two mounting screws, the other edges of the board will slightly move allowing the electrical components to touch the metal chassis. The foam sheet is used to support the installation and as an electric insulator.



4- Find appropriate holes in the chassis to install the motor driver board and the Arduino Due board. One interesting placement is to attach the motor driver board underneath the top plate, facing downwards toward the battery holder. This will keep a significant number of wires between the two plates, away from the sensors on top. One solution is shown below, with facing up and facing down images.



Top view



Down view

5- Connect all the motors to the motor driver board
6- Connect the black wire from the battery holder to a common bread board, then to the power socket of the Arduino board, the motor driver board and the buck regulator.
7- Connect the red wire from the battery holder to the middle pin of the main power switch of the robot. Connect the right or left pins to a common bread board, then to the power socket of the Arduino board, the motor driver board and the VIN input of the buck regulator.

Experiment 1: Testing the loaded motors

While the robot is on the ground, and the wheels are carrying the robots load, take measurements and fill in Table 2. Once again, 1KHz has a special importance since this is the default frequency of the AnalogWrite() function.

Table 2: PWM parameters effect on loaded motors.

Duty Cycle	F = 10 Hz	F = 50 Hz	F = 100 Hz	F = 500 Hz	F = 1 KHz
0%					
20%					
40%					
60%					
80%					
100%					

Experiment 2: A 0-radius 90° turn

In this experiment, you should make your robot do an in-place 90° turn based on the mathematical model provided earlier and the encoder pulses as a measure of wheel revolution.

Required Equipment: Same as Experiment 1 + Encoders (should be already mounted)

Procedure:

1- Make Measurements and Calculations:

- Distance between left and right wheels B = …..
- For χ of 1.48, the value of X_o= ….
- The arc length of quarter a circle with radius X_o = ….
- The wheel diameter R = ….
- The wheel circumference = ….
- The number of revolutions per the quarter circle arc distance (per a 90° turn) = ….
- The number of pulses per revolution (encoder #teeth) = ….
- The number of encoder pulses per a 90° turn = ….

2- Configure the motors direction: For the robot to turn in place (0-radius turn), all the motors should rotate with the same speed (PWM duty cycle and frequency), but the right side motors should be rotating in a direction that is opposite to the left side motors. If you want the robot to turn right, let the right motors run backwards and the left motors run forward, and vice versa.

3- **Make you robot rotate a 90° angle:** By monitoring the number of pulses, keep the robot turning till the number of pulses reach the pre-calculated number in step 1. After that, stop the motors and you should find the robot rotated by an angle close to 90°. Of course, the rotation angle might not be accurate because of various aspects related to the model itself, wheels slipping, different motors rotating in slightly different speeds although powered with the same signal, motors that have different speed when rotating forward versus backwards, and a many other practical issues that could appear. Thus, you will have to correct the number of pulses according to the actual experiment.

End of Lab 4.