Light Following Robot – Part 2

LAB 2 ECE 3400 SPRING 2021

08 MARCH 2021

QUIZ (REPORT): DUE 30 MARCH BY 11 PM ITHACA TIME

Objectives

In this Lab 2, you will integrate what you did in Lab 1 with the CdS photoresistors with the motors and the h-bridge to actuate your robot and turn it into a light-following robot. Your light-following robot must do the following:

- When the robot will be in normal lighting conditions, or when there is too much light everywhere, the robot will turn around in place, not knowing where to go. During this, the onboard LED on the Arduino will blink (ON for 500ms, OFF for 500ms, etc.).
- When bright (brighter) light hits the robot from one side or the other, the onboard LED will stop blinking, and the robot will move towards the bright light until the bright light is turned off again or the robot has turned sufficiently so that it faces the light, at which point the robot will move towards it in a straight line.
- The above cycle is repeated depending on if the bright light is removed (i.e., returning the robot to normal lighting conditions), or if the bright light continues to shine (from straight ahead, or from either side of the robot), with the robot adapting as described above.
- There is a video uploaded to Canvas demonstrating what your robot will have to do at the end of the lab. Note that in that video, at the end when the robot faces the light, it wiggles left-and-right: your robot must NOT exhibit this wiggling behavior but rather must move smoothly towards the light as long as the light remains on.

Constraints

- You cannot use the function `delay()` in any part of your code.
- You cannot use any external library (i.e., have one or more line with `#include`)
- You cannot use interrupts

Notes

- As outlined in Lab 1, labs get progressively more difficult in the sense that less "recipe following" is given and more is left out for you to figure out. This starts with this lab. Obviously, the TAs and Carl are here to help you if you have questions/problems!
- It is OK to look up stuff on the Internet! Make sure to reference any such source.

- Ground yourself!
- In this lab, your final demonstration of the robot will use the 9V battery to power the Arduino, and the AA batteries to power the motors. You will use the USB cable to power your Arduino when you program it, and you can certainly leave the USB cable connected while testing your code. ***Clarification with regard to powering the Arduino***: it turns out that you can leave the USB cable connected to your Nano to power it while at the same time powering the Nano with an external power source (e.g., the 9V battery). The Nano Every is designed such that the power source on the board is automatically selected to the highest voltage source (in this case, it will be the 9V battery).
- Make sure that you disconnect everything when you are finished with your work.
- Everyone must have their own individual code. You can work as a team, but there will be no direct copying of code from one team member to another.

Materials

- 2 x CdS photoresistors
- 2 x 10kΩ resistors (possibly 2x3.3kΩ or 511kΩ resistors)
- L293D chip
- Jumper wires (they can be separated out into single wires by pulling/peeling them)
- AA and 9V batteries
- Possibly 2 x 3.3kΩ or resistors 511kΩ resistors
- Breadboard
- Your smartphone's or other flashlight
- Robot frame
- A floor space that is not too bright where you robot will move around, maybe measuring about 1m X 1m

Procedure

0. Playing with the ADC

We covered in class in some detail how the ADC (Analog-to-Digital Converter) works in the ATmega4809 microcontroller (the microcontroller that is the heart of the Nano). That is in chapter 29 of the ATMega4809 Datasheet that is on Canvas. One thing that we did not cover much and it is quite important is the timing of the ADC process, and that is discussed in some detail in section 29.3.2 of the Datasheet (read it!).

The microcontroller being what it is, i.e., being a digital device, its operation is centered around clock cycles. The conversion timing is discussed in section 29.3.2.3, with its figure 29-8 repeated on top of the next page in Fig. 1.

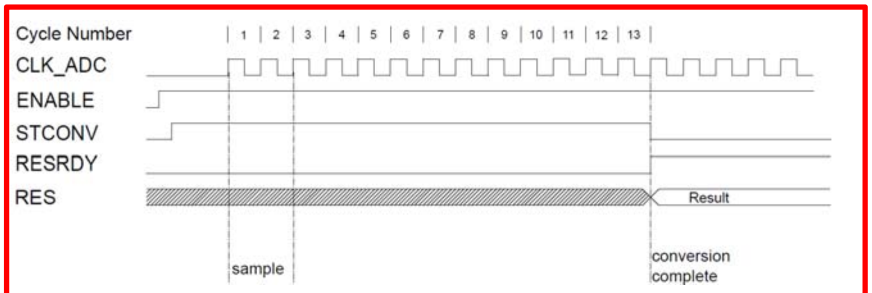

Fig. 1. ADC Timing Diagram for a single conversion

This timing diagram in Fig. 1 is easy to follow and gives you quite a bit of insight into the timing of a single ADC conversion, with section 29.3.2 describing the Operation of the ADC. Of particular interest is the CLK_ADC clock signal in Fig. 1 above. The ADC process is timed with the CLK_ADC signal which is generated from the CPU clock (CLK_PER). To generate the ADC clock signal, the microcontroller uses the CPU clock and divides it down by a configurable factor called the prescaler which is specified by the CTRLC register of ADC0 (the ADC peripheral of the ATmega4809). We can specify this value of the prescaler. Also, it has a default value.

A. Canvas submission: Default value of ADC prescaler

Write a new simple sketch similar to that used in Lab 1 to read bit values to read the appropriate bits of the CTRLC register to determine the prescaler used in the ADC.

Before we move on, let's clarify what CLK_PER is. In section 10 of the Datasheet, the Clock Controller is described in detail. From Fig. 10-1 in that section, the signal CLK_PER is generated from CLK_MAIN (which is one of several clock sources, the default being the 16/20 MHz[1] internal oscillator for the Nano), by dividing it down using (yet another) prescaler, this one specified by the MCLKCTRLB register of the CLKCTRL (Clock Controller).

B. Canvas submission: Clock prescaler

Adding to your code above, read the appropriate bits of the MCLKCTRLB register of CLKCTRL to determine the prescaler used by the Nano that fixes the value of CLK_PER. BUT… is that prescaler value even used? Read the value of the bit PEN of the MCLKCTRLB register of CLKCTRL. What do you conclude?

Now that you know what value of CLK_PER is (since you know the frequency of the clock signal), with the ADC prescaler value obtained above, you can calculate what the ADC period CLK_ADC is. The application of the prescaler for the ADC is nicely represented in Fig. 29.6 of the Datasheet. Now clearly, if we have control over the signal CLK_ADC, we can make the process of an analog-to-digital conversion as shown in Fig. 1 above go slower or faster. We obviously would want things to go as fast as possible (that's the point of fast microcontrollers and microprocessors!), but there is a limit to how fast we can go. In section 29.3.2.2 there is an important sentence near the bottom of the page, repeated here: "The ADC requires an input clock frequency between 50

[1] It turns out that the actual clock frequency of the Nano will depend on V_{DD} (check out pages 455-456 in the ATMega4809 Datasheet), but the frequency is set by the fuses (FUSE) memory (check out p. 55) and calibration files ensure that the frequency stays at 16 MHz.

kHz and 1.5 MHz for maximum resolution. If a lower resolution than 10 bits is selected, the input clock frequency to the ADC can be higher than 1.5 MHz to get a higher sample rate."

C. Canvas submission: Minimum ADC prescaler

Given the information at the end of the previous paragraph, what is the minimum allowable ADC prescaler that must be used for the CLK_PER signal corresponding to the 16 MHz signal of the oscillator, keeping in mind the allowed prescaler values in the CTRLC register?

You will now test various ADC prescaler values to give you an idea of how fast the ADC can be operated in reality. With your Arduino powered off, connect the +3.3V pin (pin # 2) to the A3 pin (that's the PD0 processor pin, also labeled D17, on the Nano Pinout diagram). Next, instead of using the analogRead() Arduino pre-programmed function, download the file ADC_SingleConvClass.ino from Canvas (in the Lab 2 Module) and open it in the IDE.[2] Note in passing that this file contains the (slightly modified) code from Example 9-1 in the "Getting Started with ADC" document on Canvas. Read through the code, understand what the code is doing, and if you compare it to section 29.3.1 in the Datasheet, that is what it is doing. Next, upload the code to your Nano. It will output the analog-to-digitally converted value of the +3.3V pin connected to PD0 on the Serial Monitor using a value of 16 for the ADC prescaler. Is the ADC result as expected?

What you will do now is use this code for all possible values of the ADC prescaler as defined in the CTRLC register of ADC. This is done on line 12 of the sketch with the statement `ADC_PRESC_DIV16_gc`. Simply change `DIV16` for any other one as needed (e.g., `DIV2`, `DIV4`, etc.). For each ADC prescaler value, note the value that is output in the Serial Monitor. What do you notice? What does the output fail to give you what it is expected to give?

D. Canvas submission: ADC values for various prescaler values

With the appropriately modified ADC_SingleConvClass.ino file, enter the various values of the ADC output value from your code for all of the possible ADC prescaler values. Why do you observe this behavior?

1. Familiarize yourself with the H-Bridge and test it

We covered in detail how the h-bridge operates and will be used. Your robot's motors need to be controlled with your Arduino. The motors' speed and direction of rotation of rotation are what you will control to make your robot go in a straight line, turn right or left or in place, or go backwards. As explained in class, the easiest way to do this is to use an H-Bridge. Consult the L293D's specification sheet on Canvas (under the Spec Sheets Module) to familiarize yourself with its peculiarities. The pages of interest are: 1-8, 11-12 (Figure 10 and Table 3 in our case).

[2] It *may* be that you will get an error when compiling this code. If that happens, make sure in the IDE that under Tools, Registers emulations, "None" is selected.

Understanding Figure 10 and Table 3 is critical to be able to control your robot's wheels. For ease of reference, Fig. 2 shows the pinout of the L293 H-Bridge – *notice the orientation of the chip with the notch*. You can easily refer to that picture to appropriately assign the L293D's pins to what they should be connected as we proceed.

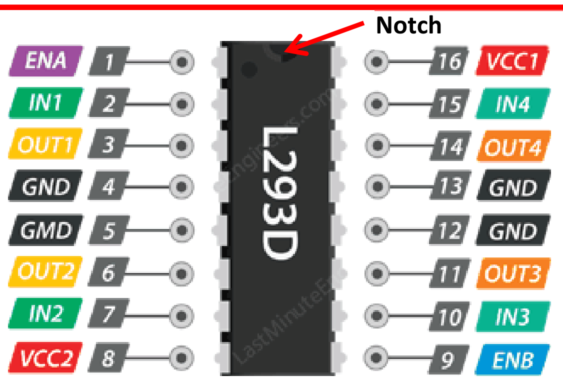

Fig. 2. Pinout of the H-Bridge chip.

Mount the L293D chip at the rear of the breadboard (the front of the breadboard is where the Arduino is mounted and should be mounted on your robot with the Arduino pointing to the front of the robot). That way, the motors are easily wired to the h-bridge, and so will the AA batteries that will power the motors. With no batteries connected to the breadboard and the Arduino powered off as well, make the connections between the h-bridge, motors and your Nano as described in class and as you see fit. Double-check everything, and take note of which pins you are using on the Nano as you will need those when you write your code.

Next, write your Arduino code that will allow you to test your h-bridge, i.e., that you can make the wheels turn one way or the other, or stop. This will require you to create several variables: four for the control pins (two per motor), two for the enable pins (for the PWM signals), and maybe one or two (or more) that will contain the speed (i.e., PWM duty cycle) at which you want to rotate the motors. It is suggested that you keep your robot near you with its wheels not touching the table because once they start spinning, your robot may fly off (it can be funny, but you can also damage components and your robot). You can put something under your robot to keep its wheels off the table.

For the motors, you need to connect the +4.5V red wire from your AA batteries pack to the appropriate pin of the L293D, and the back wire from your AA batteries back to ground.

Important: All of the grounds of everything must be connected together otherwise you will observe situations/results that don't make sense. This means that the black wire from your AA battery back, the 4 ground pins of the L293D, and the ground of your Arduino must all be connected together (when you will use the 9V battery, you will also need to connect its black wire to the common ground). Use the power rails on your breadboard for this, ensuring continuity between adjacent power rails (as discussed in Lab 1). You will not use the 9V battery to power the Nano for this part: its power will come from the USB cable.

E. Canvas submission: wheel actuation

Upload a 5 second video (not longer than that) that shows your robot's wheels (it's a good idea to have your robot on its back with the wheels free-turning in air) turning slowly in the following manner:

1. Both wheels turning together forward
2. Both wheels turning together backward
3. Both wheels turning in opposite direction of one another
4. Both wheels coming to a full stop.

The photosensors are not used for this part. In the above, this will involve you writing code that actuates the wheels as described, with each of the points above lasting at most about 1 second each. Your video must start with both wheels stationary, then go in sequence through points 1-3 above and end with point 4, all in one continuous video. This means that your code must make sure that each of the events above are timed appropriately by following the constraints set at the beginning of this handout.

F. Canvas submission: code & picture for section 1

Upload your code in Canvas in the appropriate section of the Lab 2 quiz, as well as a picture of your breadboard with connections.

Disconnect the AA batteries red wire as well as your USB cable.

2. Calibrate the motors

As is to be expected, because nothing is perfect, the motors that came with your robot are probably not exactly identical. They probably each will rotate at a different speed even if they are powered in the same manner, and each will not react the same way under a load (e.g., the weight of your robot). You probably observed this in part 1 above. This means that you need to calibrate the motors. This is not something that is quick and easy, it requires some trial and error.

For this part, you will need to set your robot in your designed floor space from lab 1 so that it can move about. Write some code (you can use part of the code that you wrote for part 1 above) to actuate both wheels (and hence your robot as a whole!) to move forward *slowly* (you don't want your robot to crash!). If your USB cable is long enough to follow your robot as it will be moving forward (either your desktop into which you connect your USB cable is close to your robot's navigation area, or you are using your laptop with which you can easily sit near your robot), that's fine. If not, you will need to power the Arduino with the 9V battery: connect the red wire from the 9V battery to the VIN pin (#15), and the black wire from the 9V battery to ground (anywhere in the ground power rail). You will of course need to connect the USB cable to the Nano to program it, but once it is programmed you can disconnect it. Remember that you can press the reset button on the Nano to have the code that is on the Nano restart from the beginning at any time you want. The photosensors are not used for this part.

When the robot moves forward, observe if it moves in a straight line (probably not), or if it turns one way or another. You need to compensate in software (i.e., by adjusting the PWM of either wheel) to ensure that your robot goes in a straight line.

G. Canvas submission: moving slowly in a straight line

Upload a short video (2-3 seconds long) showing your robot moving in a straight line. Upload your code used for this.

Now that you calibrated your robot for low speed, you need to repeat the above for “medium” speed. It is up to you to determine what medium speed is, but this is to ensure that your robot doesn’t move soooo slowly that it is painful to test and observe when it is doing something.

H. Canvas submission: moving at medium speed in a straight line

Upload a short video (2-3 seconds long) showing your robot moving in a straight line. Copy & paste your code used for this.

3. Incorporating the photosensors

Now you need to write code that puts everything together. That is, the robot will navigate by itself using what the photosensors measure. This means that your code must be segmented such that

- When the robot will be in normal lighting conditions, or when there is too much light everywhere, the robot will turn around in place, not knowing where to go. During this, the onboard LED on the Arduino will blink (ON for 500ms, OFF for 500ms, etc.).
- When bright (brighter) light hits the robot from one side or the other, the onboard LED will stop blinking, and the robot will move towards the bright light until the bright light is turned off again or the robot has turned sufficiently so that it faces the light, at which point the robot will move towards it in a straight line.
- The above cycle is repeated depending on if the bright light is removed (i.e., returning the robot to normal lighting conditions), or if the bright light continues to shine (from straight ahead, or from either side of the robot), with the robot adapting as described above.
- There is a video uploaded to Canvas demonstrating what your robot will have to do at the end of the lab. Note that in that video, at the end when the robot faces the light, it wiggles left-and-right: your robot must NOT exhibit this wiggling behavior but rather must move smoothly towards the light as long as the light remains on.

All of the above must be done following the constraints listed at the beginning of this handout.

So you need to think ahead of the flow of things in your sketch…

- There is the `setup()` part
- There is the `loop()` part where the following needs to happen
  - Continuous reading of the sensor values and calculation of parameters
  - Evaluation of conditions to determine if the robot must
    - turn around in place with the onboard LED flashing, or

- turn left or right to follow the bright light that is on the corresponding side of the robot with the onboard LED turned off, or
- go smoothly (not in a wiggling manner) straight towards the light once the robot has turned sufficiently so that it faces the light

You will probably use the Serial Monitor to display on it some parameters as you are going through this lab and coding (always use the Serial Monitor to debug!). You might have noticed that the TX LED on the Nano blinks on at times: this happens when the Nano writes something to the Serial Monitor (e.g., `Serial.println("CdS value # 1: ")`). This blinking might be annoying to you when your testing your code with your robot possibly in a not-so-bright environment. If it does annoy you, you need to comment out such commands.

I. Canvas submission: robot responding to different lighting conditions

Upload a video of your robot that is now fully programmed to respond to lighting conditions. Your 20 second long or so video must show the following, in the sequence written, in one continuous video:

1. Video starts with your robot turning around in circles with the onboard LED blinking (ON for 500ms, OFF for 500ms, etc.).
2. You will shine your bright light source *briefly* on the left side of your robot, to which the robot will respond by turning the onboard LED off and turning itself towards it. Once your robot responds by turning towards the light, you will turn the bright light off and the robot will turn around in circles again, not knowing where to go, with the onboard LED blinking again. It must be obvious that the robot responds to the light and turns towards it.
3. You will repeat step 2 above but with the light shining *briefly* on the right side. It must be obvious that the robot responds to the light and turns towards it.
4. With the robot turning around in circles, You will shine your bright light source on the left side of your robot, to which the robot will respond by turning the onboard LED off and turning itself towards it. Once the light is in front of the robot, the robot will move forward towards it smoothly in a straight line. After the robot will have moved in a straight line towards the light over a distance of 30 cm, you will turn the bright light source off, and the robot will turn around in circles again, not knowing where to go, with the onboard LED blinking again.

Also upload your final code used for the above video.

4. Report (Quiz) & Wiki

Enter all required material and answers on Canvas before 30 March 11PM Ithaca time. Update your wiki as you see fit, including with pictures & videos of your progress so far. Grading for this lab involves grading the Lab 2 quiz, as well as your wiki. At the end of the semester, points for your wiki will be given for the overall look/feel/ease of navigation of your wiki.

5. Wrap-up

Disconnect the USB cable from the Arduino, and disconnect all battery wires from the breadboard (make sure that they are not touching each other or the metal frame of your robot). Put your robot frame away securely to make sure that it will not fall on the floor. *It is fragile!*