WPI

NSF Workshop
Staying on track

Last modification: December 26, 2021

Workshop Note:
The document here includes several tasks that a student in a course would do – more than we have time for in the actual Workshop. For those who want to work through the preliminary activities, we provide some skeleton code, the instructions below, and nominal solutions.
In the Workshop, we will start in Section 5 and do the experiments needed to control the robot's motion. If you choose not to do the preliminary activities, you can start with the solution code that we provide.

1 Introduction

Motion control is an important function of a mobile robot.[1] Whether a wheeled ground vehicle, an autonomous drone, a legged robot, or a robotic snake, the robot will need to be able to control its motion to achieve a desired position. Advanced robots may use very sophisticated methods for motion control: for example, drones typically use feedback from inertial sensors to stabilize themselves and autonomous cars use vision or LiDAR to sense their surroundings and adjust their steering or speed. Can you think of other examples?

In each case, the robot needs to be able to gather and interpret sensor information, make decisions, and perform desired actions. The process of making and implementing those decisions is called *control*. In addition, the robot often observes the result of those actions to make the next decision – we call this *feedback control*, since the results of the actions are "fed back" into the control algorithm.

In robotics, control happens at many different levels. When we speak of *high-level control*, we're typically referring to system-level decisions: deciding to return to a docking station; deciding what piece to pick up in a warehouse; calculating the shortest path to a destination; and so forth. Conversely, *low-level control* generally refers to control of individual components or sub-systems: how much power to feed a motor to maintain a target speed; how far to move an arm to be able to grab something; how to angle the control surfaces of a drone to change attitude; and so on. Developing a control hierarchy, where each control function interacts predictably and efficiently with the other functions, is an important part of developing a robot, and we'll discuss that concept in future activities.

Focusing specifically on motion control, a simple method for controlling a wheeled vehicle is to use line-following. Many industrial robots, for example, follow fixed routes around the factory, which can be delineated using lines marked on the floor.[2] Given the relative simplicity of line following, the arena for your project uses several lines to help your robot keep itself on track.

[1] Any robot, really, but here we focus on mobile robots.
[2] Industry is moving away from marked lines. Why do you think that is?

In this lab you will develop basic motion control through line following and also add a higher-level control function, namely detecting and stopping at an intersection. This latter function will serve as the foundation for navigation, which we'll discuss next week. Upon completion, your robot will be able to:

- Follow a line, and
- Detect and stop at an intersection.

You will build on previous exercises to add this functionality.

1.1 Objectives

For successful completion of this unit, a student will be able to:

- Calculate results of analog-to-digital conversion for a voltage divider circuit,
- Implement proportional control for common robotics problems,
- Describe how feedback control is used for line following, and
- Describe how parameters in a control algorithm (e.g., the *gain*) affect system performance.

2 Overview

2.1 Problem Statement

Your goal is to enable your robot to drive a "figure-8" made from tape on a white surface. While driving, you must be able to adjust the speed of the robot using the IR remote. The robot must stop when it reaches the intersection, accept a turn command, and then resume travelling once it is again commanded by the IR remote.

3 Preparation

3.1 Resources

- You'll need this video on Braitenberg vehicles for the Pre-lab.

> **Workshop Note:** In our class, we use the Braitenberg vehicle to introduce control, including a homework assignment with a potentiometer in a voltage divider. You're encouraged to watch the video if you have time, as it introduces the notion that behaviors are determined by the control algorithm.

3.2 Materials

- Your Romi,
- The line sensor array,[3]
- A course for line following.

> **Workshop Note:** We typically use a figure-8 made of tape on a whiteboard, but any course made from tape will do. You will need to have some intersections for the last part.

4 The line sensor array

You have been provided with a line sensor array from Pololu. Pololu makes a huge variety of sensor arrays, which vary in the number of elements, spacing, and power, but there are two main variants: a digital version and an analog version. You have the analog version. The analog version pairs a fixed resistor with a *phototransistor*, which has similar functionality to a photoresistor: When light hits a photoresistor or phototransistor, electrons are energized such that current flows more easily – the resistance *decreases* with increasing amounts of light (energy). When paired with a fixed resistor in a *voltage divider*, voltage between the two elements will be dependent on the amount of light hitting the photoreactive component. A microcontroller can be used to read that voltage, which enables it to determine how much light – at least qualitatively – is hitting the sensor. To understand better how all that works, let's discuss some fundmental electronics.

4.1 The Voltage Divider

A *voltage divider* can be used to translate changes in resistance – such as when a photoresistor responds to light – to changes in voltage, which are easily read by a microcontroller. The voltage divider is far and away the most common circuit that you will encounter in electronics. Luckily, it is also a simple circuit, and you should get to the point where analyzing one is practically automatic.

Consider the circuit in Figure 1, where a differential voltage potential is applied to two resistors connected in series. The task at hand is to find the voltage in between the two resistors, $V_{\otimes}$. If current flows *only* through the two resistors – a practical occurrence in many situations – the current through each resistor must be equal, and it must be equal to the current through the entire circuit,

$$I_2 = I_1 = I_{12}$$

Applying Ohm's Law gives,

$$\frac{V^+ - V_{\otimes}}{R_2} = \frac{V_{\otimes} - V^-}{R_1} = \frac{V^+ - V^-}{R_1 + R_2}$$

where the resistance in the denominator of the last term is just the series resistance of the whole circuit. Note that when we refer to a "voltage", we really mean the voltage with respect to ground,

[3] Formally, your line sensor array contains six independent sensor elements, two of which you will connect to your Romi Control Board.

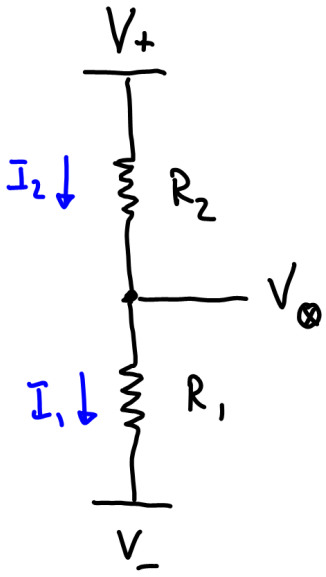


Figure 1: A voltage divider.

taken to be 0V for this activity. **Voltages are always measured as a difference between two points in a circuit.**

A little rearrangement of the second equality leads to,

$$\frac{V_{\otimes} - V^{-}}{V^{+} - V^{-}} = \frac{R_1}{R_1 + R_2}$$

or

$$V_{\otimes} = V^{-} + \left(V^{+} - V^{-}\right) \frac{R_1}{R_1 + R_2}$$

Usually, *but not always*[4], V^{-} is ground, which if taken to be 0 V yields,

$$V_{\otimes} = \frac{R_1}{R_1 + R_2} \cdot V^{+}$$

That is, the voltage at the junction is just the input voltage, V^{+}, times the ratio of R_1 to the total (series) resistance.

Care must be taken to ensure that no current leaves the junction through an additional component connected to the circuit; otherwise, the voltage at the junction will change. For example, if $V_{\otimes}$ is a reference voltage for another purpose, it is important that the current is negligible compared to that flowing through the resistors. The inputs on an oscilloscope and an Arduino pin set to `INPUT` have extremely high resistances, so this is not a concern in this activity. In other cases, you can construct a buffer circuit to prevent changes in voltage.

4.2 Analog-to-digital conversion

To read a voltage with a microcontroller, you'll use something called an *analog-to-digital converter*, or simply an *ADC*, which converts a voltage into a value (number) that can be used by the micropro-

[4] In particular, on quizzes...

cessor. For the microcontroller on the Romi, the ADC value is given by the formula,

$$\text{ADC value} = \frac{\text{sampled voltage}}{5.0\ \text{V}} \cdot 1024 \tag{1}$$

where the ADC value is rounded to the nearest integer[5] and constrained to be between 0 - 1023. For example, if the ADC were to sample 2 V, it would record a value of

$$\text{ADC value} = \frac{2\ \text{V}}{5\ \text{V}} \cdot 1024 = 409.6 \rightarrow 409 \tag{2}$$

The resolution of the ADC is the smallest change in voltage that is needed to change the result by 1. By substituting a 1 for the ADC value in Equation 1 and rearranging for the voltage, one can determine that for the microcontroller on the Romi, the resolution is given by,

$$\text{ADC resolution} = \frac{5.0\ \text{V}}{1024} \tag{3}$$

4.3 Reflectance sensor

Reflectance sensors have several uses, including line tracking or proximity detection. They emit infrared light and then detect how much light is reflected back using the voltage divider you just explored, except that they make their own (IR) light so that the lighting conditions are more consistent.

1. If you have not yet wired up your reflectance array, do so now.
2. In `platformio`, open the folder `act02-prep.line-sensors`, which reads both reflectance elements using `analogRead()` and prints the results to the Serial Monitor. Note that the pins for reading the line sensors are already defined as `LEFT_LINE_SENSE` and `RIGHT_LINE_SENSE`.

   > **Workshop Note:** Normally, we have students add code to the program from the previous activity, but you may wish to explore the sensors before you complete the Workshop, so we've given you a simple program to read/print the values.

3. Put a piece of dark tape on a white piece of paper. Move the sensors back and forth over the tape. How does the color of the surface affect the output values of the sensor? Record your observations in the Worksheet.
4. What happens when you lift your Romi up? Does that affect the readings?

5 Line following using proportional control

Proportional-integral-derivative (PID) control is a common method of controlling motors and other actuators. There are more advanced (and complex) methods that provide better performance, but PID control is easy to implement and works well in many contexts.

[5] Depending on the ADC, it may take the floor of the result. Regardless, it always produces an integer result.

Here, we only implement proportional control, where the control effort is proportional to the error.[6] Using motor speed as an example, we can define the *error* and the difference between a target speed and the current (actual) speed. The *effort* to the motor is the error times some gain, K_P. The greater the difference between the target speed and the current speed, the harder the motor works to speed up.

5.1 Motors

As noted in the previous activity, your Romi is driven using two brushed, DC motors. To control the speed of the motors, the microcontroller does not control the supply voltage directly, but uses something called *pulse width modulation* (PWM) to rapidly cycle the voltage applied to the motor. The percentage of time that the pin is `HIGH` – referred to as the duty cycle – will affect the *apparent voltage* applied to the motor. If the duty cycle is small, then only short bursts of current are fed to the motor and it turns slowly. For high duty cycles, the motor turns much faster.

Further, the microcontroller cannot produce nearly enough current to power a motor directly. To drive the motors with high currents, a *motor driver* is used. Different kinds of motors require different driver circuits – a brushed DC motor typically uses an *H-bridge*, so named because the schematic is laid out in the shape of the letter 'H'. When current flow through different legs of the 'H', the motor will turn in opposite directions. (needs a figure)

5.2 Line following with proportional control

Here, you will implement proportional control for the purpose of line following. Your approach will be to define the error as the *difference* between the two line sensor elements – when the Romi is centered on the line, both sensor elements will read some level of "gray," since they will be over the edge of the tape. When the romi deviates from the line, one sensor will read darker than the other – it will be up to you to program the Romi to correct itself.

Procedure

Starting with the code from the previous activity,

1. Add the `ROBOT_LINING` state to the list of states.
2. Define a variable, `baseSpeed`, and set it to 180 (which is in units of degrees per second).
3. Add code so that when the robot is in the `ROBOT_LINING` state, it calls a function, `lineFollow()` that takes a `baseSpeed` as a parameter.
4. Edit `handleLineFollowing()` to implement a proportional control algorithm.
   (a) Define an `error` term that is related to the *difference* between the sensor outputs and calculate the effort.
   (b) Define an `turnEffort` that is the `error` times some constant, `K_p`.

[6] Derivative control can improve performance by making turns sharper. Integral control is not necessary and generally lowers performance because due to overshooting.

   (c) Call `chassis.setWheelSpeeds()` using a combination of `baseSpeed` and the `turnEffort` you calculated above. You'll have to do some thought experiments to determine if you want to add or subtract from each wheel speed.

5. Using your figure-8 (or whatever course you have), test the basic functionality. Adjust the gain (`K_p`) to give qualitatively good results.

5.3 Speed control

One of the most useful features of implementing a state machine is that your robot can do multiple things at once – because none of the code blocks for extensive periods of time, you can respond to multiple inputs (events). You've already seen this in it's ability to both line follow and look for an intersection, but those two functions are so closely tied together that it doesn't make for a good example. Here, you'll add the ability to control the speed of your Romi while it's line following.

Procedure

1. In your `handleKeyPress()` function, add code that checks if the `CH+` and `CH-` buttons on your IR remote were pressed. Add or subtract 30 to/from `baseSpeed` so that the line following algorithm changes the average value passed to the motors.

2. Test your code. Can you control the speed? How fast can you go?

6 Intersections

With the code above, if your robot passes an intersection, it will likely do a short "hiccup" and then continue on its way. To be able to navigate the arena (covered in Activity 4), your robot will need to be able to detect intersections so that is can drive in the right direction. Here, you'll program your robot to stop at an intersection, where it will wait for another command. You'll write your program to detect the *event* of reaching an intersection, which will allow it to continue on when you press a new button.

7 Handling intersections

Procedure

1. Using your observations of the line sensor in Section 4, identify a suitable threshold value that will allow you to determine if a sensor element is over the tape or not. Describe your rationale in the Worksheet.

   Solution: Typically, the light surface will read less than 100 and dark tape, over 800. An intermediate threshold of 400 - 500 should work well.

2. Edit the checker function, `checkIntersectionEvent()`, to detect the *event* of reaching an intersection. You must use proper event checking (i.e., your function should only return `true` upon arriving at the intersection and not while your robot is sitting over the cross tape). Be careful with the logic!

Solution:

```
bool checkIntersectionEvent(int16_t darkThreshold)
{
  static bool prevIntersection = false;

  bool retVal = false;

  bool leftDetect = analogRead(LEFT_LINE_SENSE) > darkThreshold ? true : false;
  bool rightDetect = analogRead(RIGHT_LINE_SENSE) > darkThreshold ? true : false;

  bool intersection = leftDetect && rightDetect;
  if(intersection && !prevIntersection) retVal = true;
  prevIntersection = intersection;

  return retVal;
}
```

3. In your `ROBOT_LINING` state, add code to check for an intersection. When one is detected, command your vehicle to call the `handleIntersection()` function, which will command the robot to stop and return to the `ROBOT_IDLE` state.

4. Test your system to be sure that it both detects intersections and allow you to command the robot to start driving again. Can you command it to turn first?

8 Worksheet

Team exercise

1. Record the results of your experiments with the photoresistor.

Condition	Prediction	ADC reading	Equiv. voltage	Meas. voltage
Student 1	Light			
Student 1	Dark			
Student 2	Light			
Student 2	Dark			
Student 3	Light			
Student 3	Dark			
Student 4	Light			
Student 4	Dark			

Table 1: Record you predictions and observations here. Use as many lines as needed.

Solution: Light should give a large reading and comparatively high voltage. Measured and equivalent voltages will vary, since the ADC on the chip is not all that. The equivalent voltage will generally be a few 100 mV lower.

2. Record the results of your experiments with the reflectance sensors in Table 2.

Condition	ADC result (1)	ADC result (2)	ADC result (3)	ADC result (4)
Light (paper)				
Dark (tape)				

Table 2: Record the results of the experiment with the reflectance sensors here. Use one column for each student – as many as needed.

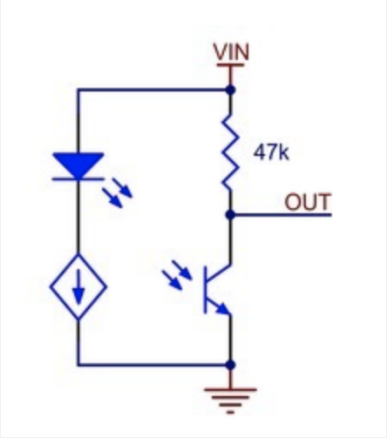
Solution: Reflectance sensors are wire opposite the circuit above. Dark usually results in an ADC of 2-3000. Paper is usually under 100.

3. Describe how the line sensor works.

   (a) Draw out a schematic.

   (b) Referring to the schematic, describe the behavior of the sensor for both light and dark surfaces. Give qualitative explanation of how and *why* the voltages change.

Solution: Schematic (from the Pololu website):



The IR LED shines light on the surface. If a lot is reflected, then the phototransitor passes current more easily (lower resistance). The voltage drop across the phototransistor will be small compared to the 47k resistor, leading to a low voltage at the `OUT` junction. For a dark (non-reflective) surface, the resistance will be high, leading to a large voltage drop and higher voltage at `OUT`.

4. Describe your control algorithm. How is the error calculated? How is the effort calculated?

Solution: The error should be the difference between the two sensors. Effort is just a constant, K_P, time the error.

5. Describe your rationale for the threshold for detecting the tape at an intersection. What happens if you make it too high? Too low?

Solution: If it's too high, then we may miss an intersection (classifies dark as light). If too low, a speck of dust or imperfection may trigger an intersection. A good threshold is probably half-way in between.

6. Describe the results of your testing. Did you meet your requirements? Explain how teleoperation can make it easier to build in autonomous behavior later.

Solution: Results will vary. You need to have a good description of your results. Teleop is useful because it allows for repeated tests without having to restart/re-upload code. Other explanations are possible.

7. Don't forget to submit your code.

Solution: OK

9 Pre-lab worksheet

1. Identify the key functions that your Romi needs to do to complete the prescribed activity in Table 3. Indicate if the function fits best with Sense/Think/Act and the component/process that will be used to meet that function. Try to break the functionality down into basic components, but you do not have to limit yourself to one of S/T/A.

   For example, for the LED in the first lab, you might write:

   ```
   Indicate state | A | Light LED
   ```

Function	S/T/A	How it's met

Table 3: Identify functionality here.

Solution: Some good answers are:

- Detect line / S / line sensor array
- Detect intersection / S / line sensor array
- Adjust effort / T / PID algorithm
- Speed up/slow down wheels / A / PWM to motors

One could certainly make the argument that some functions are more than one theme (e.g., "Position self" requires all three.

2. Identify two or three important requirements for this activity. While we normally want you to discuss ideas with others, in this case, make the list yourself – you'll combine your ideas with your team later. Just give a good try and you'll be fine.

Solution: Important reqs may include:

- The "wagging" of the vehicle as it follows the line should be kept to a minimum. It is difficult to measure, but *no perceptible wagging* is a good target.
- The faster, the better. There are lots of ways to put numbers on this, but it's a good example of merely indicating the direction you want to go (faster).
- It must detect the intersection every time.
- It must stop within a reasonable amount of distance at an intersection. For example, within 1 cm (specific numbers will vary).
- It must respond to speed inputs "immediately" (too hard to measure precisely).

3. With respect to the light-responsive vehicle described in the video on Braitenberg vehicles, (Section 3.1), in each of the figures in Figure 2, the lines represent the influence of the light (illumination of the *photoresistor*) on the motor. For example, in Figure (a), the the positive connection means that more light on the right sensor will lead to more effort to the right wheel.

   Describe the behaviour of each of the configurations.

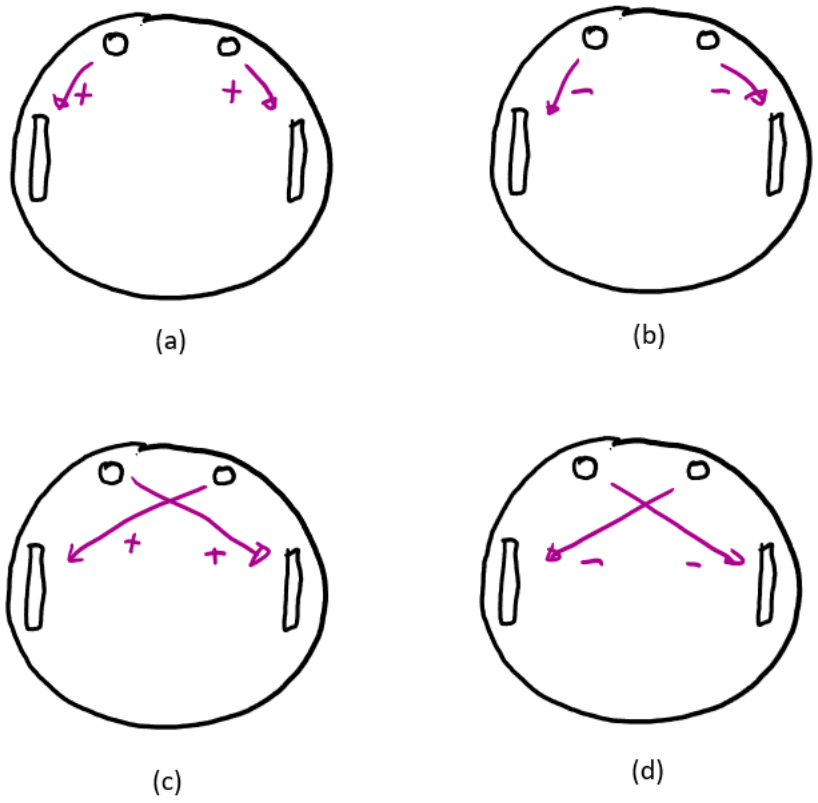


Figure 2: Four configurations for a Braitenberg vehicle.

Solution: (a) Evades light and slows in the dark. (b) Seeks light and slows as it approaches. (c) Seeks light and speeds up as it approaches. (d) Runs headlong into the darkness.

4. For the line (tape) following in this activity, which sensor *elements* do you think will provide the best performance? Why?

Solution: A good choice is two elements that are just about the width of a piece of tape. That way, a robot centered over the line will see "gray" from each sensor. Too wide and the vehicle will likely wobble; same for too narrow.