MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.081—Introduction to EECS I
Fall Semester, 2006

Assignment for Week 5

- Issued: Tuesday, October 3rd
- Prelab and Tutorial Due by 1pm Thursday, October 5th
- Postlab Due Thursday October 12th in class

The constraints view of circuits

In this problem set you will learn about resistor networks both as constraint systems and from a more intuitive perspective. The former is more useful when writing programs to determine the behavior of circuits, the latter is more helpful during design. You will get a chance to use your understanding of resistor networks to design a photoresistor interface so that your robot can “see”.

This problem set has four parts:

1. Pre-lab on using a constraint resolver to solve circuit problems.
2. Tutor problems on analyzing resistor networks.
3. Lab on interfacing a photoresistor so your robot can “go to the light”
4. Brief post-lab describing your in-lab work.

1. Pre-lab

You will be downloading and using *resolve_constraints.py*, which contains programs for creating lists of constraints and their associated variables, and then once the list of constraints and variables has been generated, determining values for the variables so that the constraints are satisfied. In particular, a user must first create an instance of the class *constraint_list.* Then the user makes multiple calls to *constraint_list*'s function *add_constraint* to append each of the constraints and the constraint's associated variables to the instance. By calling the function *resolve_constraints* with the *constraint_list* instance, values are determined for the variables so that the constraints are satisfied. The values of the variables can be printed by calling *constraint_list*'s function *display.*

The arguments to the function *add_constraint* are: a procedure whose input is a tuple of variable values, and a lists of strings which are the labels for the variables used in the constraint. The procedure should return zero when the input tuple satisfies the constraint, and should otherwise return a floating point number which indicates how far the tuple is from satisfying the constraint. For example, if the constraint is that $a = b^2$, then a constraint procedure could return $a - b^2$, $b^2 - a$ or even $\sqrt{b} - a$.

In order to use *resolve_constraints*, one needs an organized approach for generating the variables and constraints for a circuit. In class we discussed the nodal approach for accomplishing this task. The steps in the nodal approach were

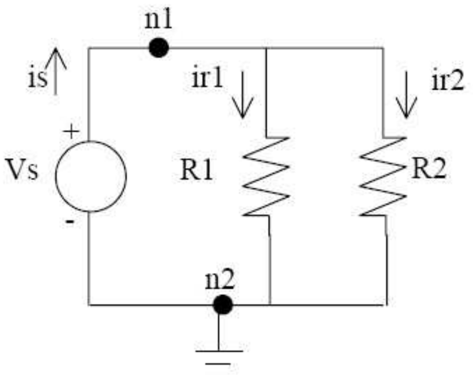
1. Label all the circuit node voltages and element currents (noting direction), and select a reference node.
2. For each element, write constitutive equations that relate element currents to the voltages at the element's terminals.
3. For each circuit node, except the reference node, write a conservation law. That is, insist that the sum of currents entering a node should be equal to the sum of currents leaving a node.

In order to use the constraint resolver to solve circuit problems, it is helpful to have functions which return constraint procedures associated with a circuit's constitutive equations and conservation laws. In the file *circuit_constraints.py*, are functions which return procedures which implement circuit related constraints. The *resistor* and *vsrc* functions return procedures which implement the constitutive relations associated with a resistor and a voltage source, the *kcl* function returns a procedure which implements the constraint that the signed sum of currents must equal zero, and the *set_ground* returns a procedure which implements a constraint forcing a value to zero (typically used to force the reference node voltage to be zero).

1.1 Download and test the constraint System

The software for this prelab is at the 6.081 web site on the calendar. Download and unzip the constraint resolver software (resolver.tar), and note that there are five python files: *resolve_constraints.py* and *circuit_constraints.py* described above, as well as three example circuit files, *circuitexample.py*, *circuit.py* and *circuit2.py*. Try running *circuitexample.py* by typing `python circuitexample.py`.

The circuit corresponding to this example looks like this:



As you can see in the code, after importing the relevant modules, we must make an instance of the class *constraint_list*, which we have called `ckt`:

```
ckt = constraint_list()
```

Going through the nodal approach steps:

1. Label all the circuit node voltages and element currents (noting direction), and select a reference node. As you can see in the picture above, we have labeled voltage nodes *n1* and *n2* and currents *is*, *ir1*, and *ir2*. The arrows denote the direction of each current. In our code, we add a constraint for the ground:

```
ckt.add_constraint(set_ground, ['n2'])
```

2. For each element, write constitutive equations that relate element currents to the voltages at the element's terminals. There are three elements: *Vs*, our voltage source, and *R1* and *R2*, two resistors. In our code, we write the following:

```
Vs = 5
R1 = 20
R2 = 10
ckt.add_constraint(resistor(R1), ['n1', 'n2', 'ir1'])
ckt.add_constraint(resistor(R2), ['n1', 'n2', 'ir2'])
ckt.add_constraint(vsrc(Vs), ['n1', 'n2', 'is'])
```

   The first line says that resistor *R1* has a resistance of `R1` (which equals 20 ohms). It connects node *n1* to node *n2*, and the current running through it is *ir1*. Resistor *R2* is similar. The third line says that the voltage source has voltage `Vs` (which equals 5 V) and connects node *n1* to node *n2* with current *is*.

3. For each circuit node, except the reference node, write a conservation law. That is, insist that the sum of currents entering a node should be equal to the sum of currents leaving a node. In this case, we have only one circuit node (*n1*) besides the reference node (*n2*). The corresponding line of code is as follows:

```
ckt.add_constraint(kcl([-1,1, 1]), ['is', 'ir1', 'ir2'])
```

   In this line, we are saying that $is = ir1 + ir2$.

Finally, we call the constraint resolver on our *constraint_list* instance and display all the voltages and currents found:

```
x = resolve_constraints(ckt())
ckt.display(x)
```

Now try running the other two circuit examples (*circuit.py* and *circuit2.py*, and notice that each example generates a different error. In circuit.py, there are more constraints than there are variables because there is an excess kcl constraint. Comment one out to make it work, then draw the resistor and voltage source circuit diagram associated with this example. You may occasionally find that in the course of generating all the variables and constraints for your circuit, you have entered more constraints than variables. This happens especially if you put in a conservation equation at the ground node when none is needed. When this happens, you must remove enough redundant constraints to make the number of constraints equal the number of variables.

In the second example circuit file, the reference node voltage has not been set. Insert a line to set the reference node voltage, then draw the resistor and voltage source circuit diagrams associated with this example.

1.2 Use the constraint system to analyze a complicated circuit

Use the constraint resolver to solve the for the voltages and currents for the circuit given in the third tutor problem for week five.

1.3 Add the non-ideal voltage source

The voltage source is an idealized model of a battery. All real batteries have some internal resistance, and this resistance restricts the current that can be provided by the battery. We can model a real battery by creating a more complicated circuit in which we add a resistor in series with an ideal voltage source, but one can also generate a single non-ideal voltage source constraint. Try adding such a constraint to the file *circuit_constraints.py*, and then use your new non-ideal voltage source to reduce the total number of constraints in the *circuit2.py* file. Notice that you will eliminate a node voltage as an explicit variable and should also eliminate its conservation law, but you should still compute the same source current.

1.4 ONLY FOR FUN

You might try adding a nonlinear resistor constraint to the system (one in which the current is a nonlinear function of the voltage). One example nonlinear resistor has a current-voltage constraint given by

$$R_0 i_R - (v_{n_1} - v_{n_2}) - \beta\,(v_{n_1} - v_{n_2})^3 = 0.$$

where β is typically less than one, and larger values of β correspond to more nonlinear resistors. Can you find an interesting circuit using such a resistor?

1.5 Do the tutor problems for week five

The tutor has circuit examples, one of which is too annoying to analyze by hand. Use a combination of your understanding of circuits and the constraint resolver to solve the tutorial problems.

2. In Lab - Helping your robot to see

For this lab, along with a robot and a laptop you will need:

- National Instruments (NI) Interface Box and USB cable.
- Robot mountable protoboard.
- Several 1k, 10k, and 100k resistors.
- A photoresistor.

We will be using the NI box in a number of labs to interface to circuits, sensors and motors. The NI box can be used to measure voltages and convert those voltages to numbers that can be accessed using python. We have provided a python program which will read numbers generated by the NI box, specifically the numbers that are related to the voltage at the NI box terminal labeled *AI0*. Note that for the NI box, the voltages are measured with respect to the voltage at the terminal labeled *GND*. Our program, *singlereadings.py*, is an infinite loop which keeps reading the voltage at *AI0* and printing the results. The program is available from the course web site; download the py-ni-test.tar.gz archive and uncompress it.

You will also be using a protoboard to connect components together. A protoboard is used for making easily modifed electrical connections between wires and circuit elements. If you look at your protoboard, you will notice many rows of holes, where each row has five holes. These holes are electrically connected, so if you plug two wires in to two holes in the same row, the wires will be connected electrically. Resistor leads can also be plugged directly in to the protoboard holes. Also, each protoboard has several long columns of holes which are used for nodes in a circuit that have a large number of connections. These columns are often used for ground and power. If you have never used a protoboard, have one of the staff members demonstrate the board's use.

2.1 - Try using the NI Box

Download the *singlereadings.py* program, connect the NI box, and try using the program to read the voltage at *AIO*. Please notice that we have connected wires to the *AI0* and *GND* terminals to make it easier to connect circuitry to the NI Box. You should also take note of the fact that the NI box has terminals which generate two reference voltages, +5 volts and +2.5 volts.

Try connecting *AI0* to *GND* and to the two reference voltages and note the readings returned by the *singlereadings.py* program. The results may surprise you, but you will determine the explanation in the next section.

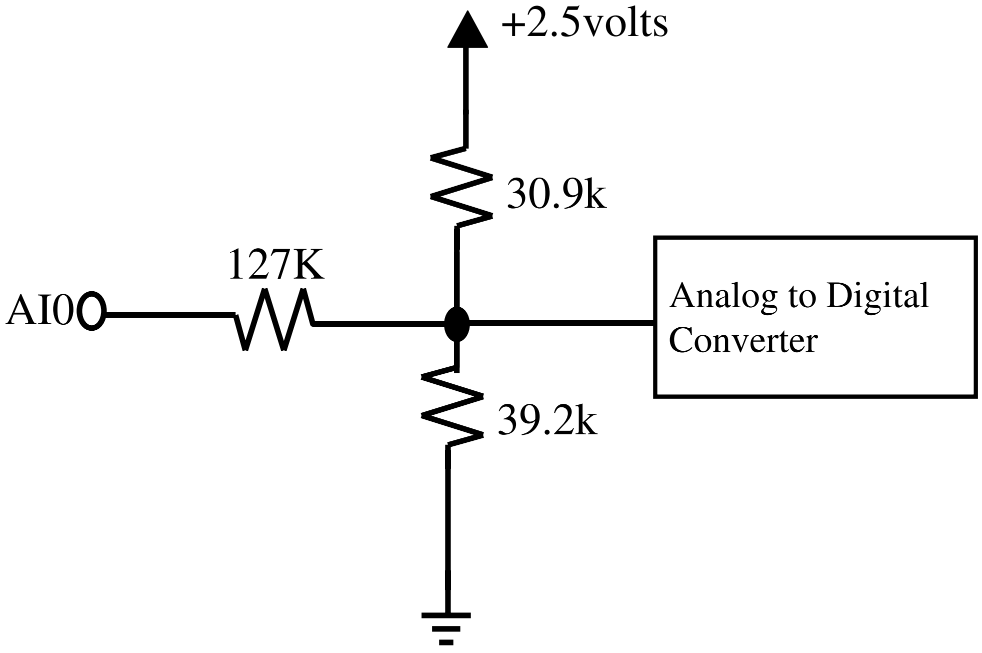
Checkpoint: 2:00 PM

- Demonstrate that you can use python to read voltages from the NI box.

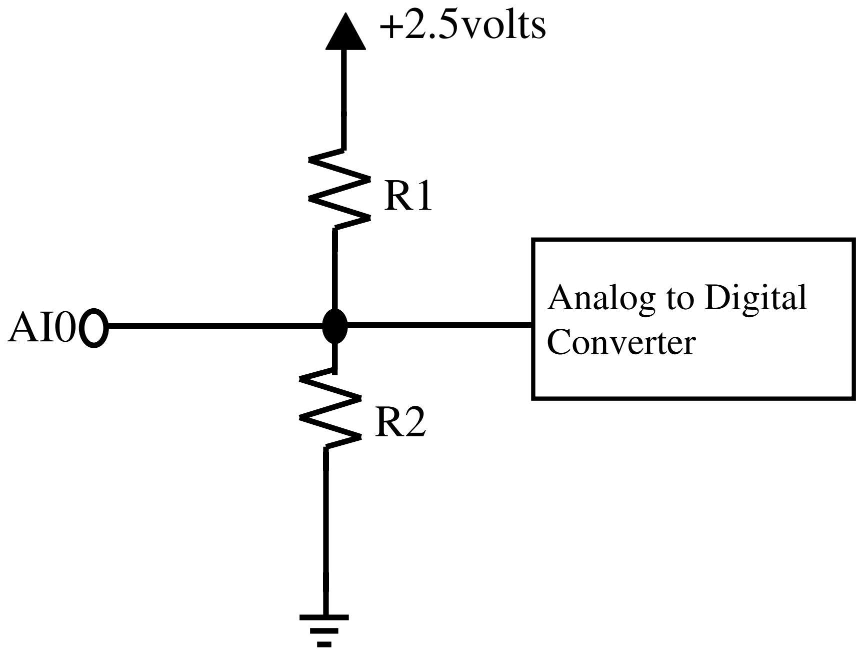
2.2 - Determine the correct input description

When connecting the *AI0* input to various reference voltages (including zero or ground), you certainly noticed that the computer printed a number that was NOT equal to the value of the voltage you applied to *AI0*. The reason is that the terminal *AI0* is connected to a resistor network, and the analog to digital converter which generates a digital number representing the voltage has been adjusted so that when there is no connection to *AI0*, the returned value is zero.

The documentation for the NI box is unclear about the input network connected to *AI0*, but one page of the documentation suggests that the circuit is given by



but other somewhat incomplete information suggests that the input network is



Your task will be to experiment with the NI box and try to determine which input circuit is correct, and to determine the value of any unknown resistors. The tests you performed above will certainly be helpful, but you will need to run additional experiments. Ask the staff if you need resistors. As a handy reference, here is the chart of resistor band colors:

Black	*Brown*	*Red*	*Orange*	*Yellow*	*Green*	*Blue*	*Violet*	*Gray*	*White*
0	1	2	3	4	5	6	7	8	9

When reading a resistor value, look for the side with three color bands closely packed. Read the numbers for the first two color bands, and then add the number of zeros corresponding to the third color band. For instance, brown-black-brown = 10 * 10 = 100 ohms.

Checkpoint: 3:00 PM

- Demonstrate that you have the correct circuit and resistor values.

2.3 - Interface a photoresistor

As one might expect from the name, photoresistors are resistors where the resistance varies with light intensity. Ask a staff member for a photoresistor and design some experiments to determine how the photoresistor resistance varies with light intensity. You can ask the staff for needed components and lamps. Then use the results of your model of the NI box input to design a circuit and a modification of *singlereadings.py* so that the program returns an indication of light intensity.

Note that the NI Box can supply five volts; use that five volts as the power supply for your circuit.

Checkpoint: 4:00 PM

- Demonstrate that you can sense the difference between ambient light and a direct illumination from a nearby lamp.

2.4 - Write a brain which moves toward an intense light

Install the protoboard with your photoresistor circuit on the robot. Then write a SOAR brain that will cause the robot to respond to a sudden change in light intensity by moving forward to the light. You can assume the light will be directly in front of, but above, the robot. The robot will not need to turn, but it should be able to stop once automatically it has passed under the lamp. Next week you will work on far more sophisticated light sensing with the photoresistors.

Checkpoint: 5:00 PM

- Demonstrate that you can make the robot move under a lamp and stop.

3. Post Lab Write Up, Due Thursday, October 12th in lab

Write a brief description (2 pages) of:

- The experiments you used to determine the correct NI box input circuit. Be sure to explain why your tests were conclusive.
- Describe the circuit you used to interface the photoresistor to the NI box, and also explain your software modifications.
- Describe your SOAR brain for moving to the light.