Geo-Bot 1.1
January 1, 2002

Geo-bot is a robot that can draw polygons and other shapes. It is a good project for teaching “Logo-like” (turtle math) procedures for drawing various shapes, solving mazes, and similar explorations. It is also a great taking-off point for exploring the area between mathematics and engineering. This is largely due to the fact that Lego robots tend to wobble, not turn precisely, and behave very differently depending on such real world conditions as floor surface, battery life, motor speed idiosyncrasies, gear friction and other such “gremlins”. The first step is to make a suitable vehicle. The next is to program it to draw specific shapes or polygons.

The basic requirement is a two motor design where one motor drives each wheel or tread. If using wheels, a three wheel model, with a swivelling centered front (or back) wheel, or two wheels and a smooth skid is best. This eliminates the curving turn that all 4 wheeled vehicles make. To draw nice polygons, your vehicle needs to be able to “turn on a dime”.

I have worked with three different vehicle designs. One, a tractor tread model, draws the most regular and repeatable polygons, deals with floor surfaces well, and is quite reliable, but is time-consuming to make and is slow and clunky. Another, a simple robot using wheels, is strong and flexible, but doesn’t draw geometric shapes as precisely; it is more susceptible to the real-world interventions mentioned above. Finally, the simple “Pathfinder” robot from the basic Lego Constructopedia works well enough for most purposes.

Once the robot has been built, the next step is programming. If students are already familiar with programming in any dialect of Logo (such as GeoLogo from TERC), it is good to refer to procedures and show how to adapt them for robotics.
So, if we ask students to develop a procedure to draw a square, they go through a lot of trial and error to find right turn 90 (a command that makes a 90 degree turn). They figure out that a procedure can be written:

To square
forward 50 right turn 90
forward 50 right turn 90
forward 50 right turn 90
forward 50 right turn 90
end

Then they can be shown the magic of the “repeat command:

To square 50
repeat 4 [forward 50 right turn 90 degrees]
end

and finally, a general procedure to draw squares:

To square :size
repeat 4 [forward :size right turn 90 degrees]
end

Typing “square 50”, draws a square that is 50 units on a side.

Finally, some students will figure out a general polygon procedure such as:

To polygon :size :sides
repeat :sides [forward :size right turn (360 / :sides)]
end

And some will also figure out that “polygon 1, 360” draws a good circle!

The Logo literature stresses experimentation by students to discover this general procedure themselves. Although, it’s not clear though that “360 / :sides” (defining an exterior angle) is that useful mathematically; interior angles show up far more often in Geometry studies. But exterior angles are essential for robotics! Also in robotics, turns by numbers of degrees are hard to specify; think of the history of sea exploration! You need a guidance system, like a gyroscope or a reference to a fixed location (compass, stars). And you need to be very careful and precise.

Furthermore, the use of 360 degrees defining a complete circle is a rather lucky, but arbitrary decision that came from ancient Babylonian number systems using base 60 and 12 - both factors of 360.

So, as every “roboteer” soon learns: to make a robot turn on a dime, you run the wheels in opposite directions. And the time you do this is directly proportional to the angle turned. The trick is calibration: figuring out the time required when the motors are running in opposite directions for various angles for polygons. You start with an educated guess, do some measurement, and tune it via trial and error.
To calibrate my robot, I made a giant protractor:

First, I put the robot in the middle of the protractor facing along the 0 degree line and then ran a program that made the robot turn. I had the robot wait 1 second (to give me a chance to get my hands off the robot and out of the way) and then turn in place. Students will have to experiment with the actual values for their robots due to flooring, battery condition, quirky qualities of motors, gear friction, etc. I then read degrees off the protractor to estimate how to get the values I want. There are two methods to do this: One is to see how far it actually turned and then tweaking the values in RoboLab code until you get a reliable turn for the angle. This requires estimating how far off you are and how much, or how little change gets you to the value you want. Alternately, you could measure an actual rotated angle, plot it on a graph of time vs. angle and, assuming the relationship between time and angle is linear (it more or less is), read other values, such as “90” degrees off the graph. I did this with my tractor tread robot, and it was linear enough to produce reasonable octagons when I used 45 degree angles.

For example: with vehicle type 2 I put the robot on the protractor and entered "100" as the input to an “n/100 timer”. It spun almost 180 degrees, well, I wanted to draw squares, so I needed to change that. By the way, even without knowing what “90 degrees” means, students should know what a right angle is. I tried “75”. This time it was close, but still went past a right angle. Eventually I found “55” was the correct value for my robot to make nice right angles.
Here is the RoboLab code:

Note: This stuff is right in the TERC books. It's actually some of the stuff that the 9th graders at Boston Arts Academy had trouble with. So, it's better to learn it now.

The RoboLab code for a polygon is a very simple extension:

(1) make both motors run forward for a time (figured out by the students in an earlier project).
(2) make one motor switch direction for a time (figured out using my giant protractor, or with floor tiles (which make right angles)).
(3) repeat 1 and 2 for the number of sides you want (4 for a square).

Note: Inventor Level 3 is required to draw polygons in this manner.

Students enter a time delay for the motors going in the same direction which defines the length of a side. They then enter a number into the other clock corresponding to the time both motors are running in opposite directions, making the robot turn the required number of degrees.

Using values of 4 seconds for the first, and 55 100ths of seconds for the second, my wheeled robot drew a 4 foot or so square on my dining room floor.

Decimal note: I used an "n/100 timer" for the delay. So if you enter 55 you are really running the motors in opposite directions for .55 seconds. This could be part of a decimal challenge: a 1.5 second delay (which would make my robot turn about 270 degrees) would be entered into the RoboLab code as "150" (150 hundredths of seconds). Alternately, one could use a regular timer and the students would enter the actual time in seconds, so my 90 degree turn would be ".55" and the 270 degree turn would be "1.5".

One can then have an extend the discussion to get into the relationship between mathematics and robotics: Drawing a square can be done with a Logo-like recipe, using an abstract thing called 90 degrees (as in Geo-Logo). But here we are making a square from a right angle, what's a right angle, well, it's the angle of a square and, as important: it's also the angle my robot turns when I run the motors full power in opposite directions for .55 seconds! That's as valid a
measure of something as anything else! One could then launch into a talk about standards, repeatability and precision. Should the world use my standard to define right angles? Why? Why not? Could one get to the moon with my standard? The astronauts fire a jet engine for a "timed burn" which makes the spaceship turn a certain angle so it heads to Mars rather than Jupiter…

So, to start, students all make a square. They have to figure out the times need for their robot to turn and make a right angle, then they make the robot do it. Did it? Sometimes the time to turn is different depending on whether the robot is already running or just starting up (inertia - we are after all dealing with real objects!) Wheeled robots differ in this regard from tread robots, and wheel size and motor speed make a big difference here. My robot actually took .60 seconds to draw a right angle on my protractor, but when drawing a square, it kept overshooting the angles and needed to be tweaked to .55 seconds. That's engineering!!!!!

Next, students try other polygons. For that they have to use the number of sides and the number of degrees to turn, and then figure out the “motor-on” times needed. So, to draw an octagon, I know there are 8 sides. 360 degrees divided by 8 equals, 45 degrees. Students can also use small protractors and actually measure the angles from drawings of polygons. My 90 degree turn needed .55 seconds, so the octagon procedure loops 8 times, with a "time of turn" delay of about half of .55, or, .23. seconds.

Table 1. Polygons

<table>
<thead>
<tr>
<th>Shape</th>
<th>Sides</th>
<th>Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>Triangle</td>
<td>3</td>
<td>120</td>
</tr>
<tr>
<td>Square</td>
<td>4</td>
<td>90</td>
</tr>
<tr>
<td>Pentagon</td>
<td>5</td>
<td>72</td>
</tr>
<tr>
<td>Hexagon</td>
<td>6</td>
<td>60</td>
</tr>
<tr>
<td>Heptagon</td>
<td>7</td>
<td>51.4</td>
</tr>
<tr>
<td>Octagon</td>
<td>8</td>
<td>45</td>
</tr>
<tr>
<td>Enneagon</td>
<td>9</td>
<td>40</td>
</tr>
<tr>
<td>Decagon</td>
<td>10</td>
<td>36</td>
</tr>
<tr>
<td>Hendecagon</td>
<td>11</td>
<td>32.7</td>
</tr>
<tr>
<td>Dodecagon</td>
<td>12</td>
<td>30</td>
</tr>
<tr>
<td>Icosagon</td>
<td>20</td>
<td>18</td>
</tr>
</tbody>
</table>

See:
http://www.math.com/tables/geometry/polygons.htm#names
http://www.ac-noumea.nc/maths/amc/polyhedr/polyg_draw_.htm

Possible extensions of this are: (1) rewrite the polygon procedure so that students need only enter "90" into an “angle” box in RoboLab and "4" into a “number of sides” box to draw a square, or "45" and "8" to draw an octagon; (2) use a light sensor and the protractor to actually time, and thereby measure and store, the value required to turn of a certain number of degrees. Autocalibration and using software to scale values to more “normal” values (such as 90 degrees instead of .55 seconds) are interesting topics and worth doing. Lots of tweaking is still required (there would need to be a “fudge factor” box that fed into the calculations needed to convert a number like 90 degrees into the time delay that the robot motors need).
Note: There is an especially good project on the Tufts CEEO site, that involves kids figuring out the time required for their robot to make a trip across a large (10-20 foot) map of the USA. The project should be done with a simple robot with one motor so it goes straight. Once they figure out, let’s say, how to get from Boston to Reno, they can then be asked how long it would take to get to Chicago instead. REQUIRES DECIMALS!!!! And this can be used to get “ballpark” numbers for the lengths of sides of polygons (assuming same size wheels, power level and gearing).

See: http://www.ccee.tufts.edu/curriculum/cgi-bin/activity_index.cgi and look for Cross Country Trip or Solar System.

**Special Note:** a nice extension to this is to attach this “draw-bot” assembly to the reversing motor port on the RCX (in this case “A”) and run the same polygon code. A regular orderly “square drawer” becomes a wild and crazy guy!