

# CheapBot Navigation Exercise

by NearSys

Now that you have a working robot, it's time to determine its navigation characteristics. The exercises below will show how your robot behaves when it drives. This information will prepare you for a dead reckoning competition.

## Driving and Steering Robots

The CheapBot robot base has two motors and wheels. For stability, a third point, the tail dragger, is located at the rear of the robot base where it behaves like a tripod. To make a CheapBot robot drive forward, both the left and right motors must turn in the same direction (that is from the robot's perspective). From the servo motor's perspective, one is turning forward and the other is turning backwards. However, since the motors are oriented opposite to each other, the wheels are turning in the same direction from the robot's perspective and the robot travels forward. If both motors spin in reverse, the robot still travels in a straight line, but this time backwards.

To turn the CheapBot robot, one wheel must turn in the opposite direction from the other. In other words, one wheel spins in the forward direction and the other spins in the backward direction. If you reverse the direction of spin of both motors, the robot still turns, but this time in the opposite direction (clockwise versus counter clockwise).

The H-Bridge takes two commands from the robot controller and converts it into a direction of rotation for the motor connected to it. The commands are how you set the two I/O pins connected to it. On the CheapBot-14, the two I/O pins connected to the first H-Bridge is 2 and 3. For the second H-Bridge, the I/O pins connected to it are 4 and 5 (on the CheapBot-18 the I/O pins are 0 and 1 for the first motor and 2 and 3 for the second motor). You send commands to the H-Bridge by setting the two I/O pins connected to it to either a HIGH (+5 volts) or LOW (0 volts). Since there are only two settings for each I/O pin (either HIGH or LOW) and there are two I/O pins, there are four combinations or commands you send to the H-Bridge. A truth table for the H-Bridges will look like this.

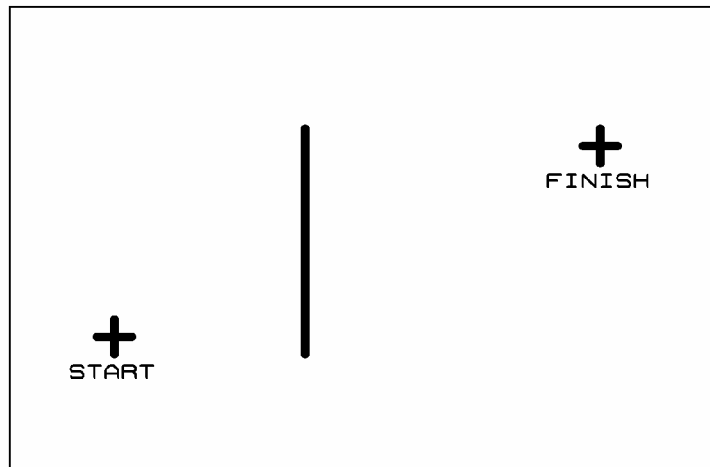
I/O-2 State	I/O-3 State	Rotation Direction
low	low	no rotation
low	high	clockwise
high	low	counter clockwise
high	high	brake

### Dead Reckoning Navigation

The simplest way to get your robot to drive from one location to another is by dead reckoning. Dead reckoning means navigating your robot by specifying a direction and a pause length. The robot receives no feedback as to its position or heading. Therefore, when robots navigate by dead reckoning, they tend to drift out of the place you expect. Careful programming will help limit the drifting, however, there are too many unpredictable behaviors in even the simplest robot. Try out Project 1 to see this for yourself.

## Project 1 Dead Reckoning by a Robotist

With tape, mark a starting point on the floor. Then with a second piece of tape, mark a destination about ten feet away. Between the starting and finishing points, place a strip of tape as a barrier that you're not allowed to walk over. The exact placement is not important, just as long as you can safely walk between the start and finish points. Your test arena should look similar to this.



Have a friend with you, as you're going to pretend to be your robot and you'll need someone to look out for your safety and to keep track of how well you did. Now, walk from the Starting Point to the Finishing Point. As you do, keep track how many steps you took and when and how you made a turn. For example, your walking directions could look something like this,

Walk six steps forward  
Turn 90 degrees to the left  
Walk four steps forward  
Turn 90 degrees to the right  
Walk three steps forward

After you memorize your directions, walk back to the Starting Point. Orient yourself like you did the first time and then close your eyes. Following your directions, walk to the Finishing Point without opening your eyes. Your friend will make sure you don't walk into something.

After you get to your destination, open your eyes and see how close you are. Ask your friend if you crossed the tape barrier that you were not supposed to cross.

**How close were you to the Finish Point?**  
**Did you avoid walking across the tape Barrier?**

This is the problem your robot faces when it dead reckons. It will drift out of line as it drives. Without a control loop that looks at the robot's current position and heading, there is no way to correct the errors that will build up. However, the Dead Reckoning competition asks you to understand the mechanical behavior of your robot the best you can. To have the best chance to dead reckon through a maze to your destination, you'll carry out the next two projects.

## **Project 2**

### **Time Versus Distance: How far will the robot go?**

Your robot drives because your program calls a subroutine. The subroutines used in navigation set the four I/O pins connected to the H-Bridges to either +5 volts or ground (or called high and low). When the I/O pins are set, they trigger electronic switches inside the H-Bridges and those control the direction in which current flows through the motor attached to them. To keep the motors turning, you just need to leave the H-Bridges alone. One way to do this is to use a pause command (there are other ways and you will use one with the line follower). The pause command stops the robot controller from doing any other work for the length of time specified. So if you typed the following commands...

```
GOSUB FORWARDS  
PAUSE 1000  
GOSUB FREEZE
```

The robot will drive forward for one second until it is commanded to stop. Just how far your robot will drive depends on the charge in its motor batteries (the logic batteries don't affect how fast or far the robot will drive) and the size of the wheels (bigger wheels make the robot travel faster and therefore farther). To successfully compete in the dead reckoning competition you need to know how far your robot will drive with a given pause. So type the following commands and your FORWARDS subroutine in the PICAXE Editor.

```
PAUSE 1000  
GOSUB FORWARDS  
PAUSE 500  
GOSUB FREEZE  
END
```

```
FORWARDS:  
:  
:  
:  
:  
RETURN
```

**Note:** You must type the four HIGH/LOW commands of your Forwards subroutine where the colons are located.

The first pause command is a delay to let you get your hands away from the robot before it starts up. So don't change the **PAUSE 1000**. The second pause is the delay that allows your robot to drive forward. In this project, you'll change the length of this delay and measure how far your robot drives.

- Make sure there is a space of at least five feet or more that the robot can drive down
- Mark the beginning of the range with a piece of tape
- Have a yard stick or tape measure handy
- Program your robot with the code above (with the motor power turned off)
- Turn off the logic power and unplug your robot
- Place the front of the robot at the tape mark
- Power up the motor power
- Power up the logic power
- After the robot stops, measure how far it traveled
- Enter the distance in the table below

<b>PAUSE</b>	<b>DISTANCE</b>
<b>500</b>	
<b>1000</b>	
<b>1500</b>	
<b>2000</b>	
<b>2500</b>	
<b>3000</b>	
<b>3500</b>	
<b>4000</b>	
<b>4500</b>	
<b>5000</b>	
<b>5500</b>	
<b>6000</b>	
<b>6500</b>	
<b>7000</b>	
<b>7500</b>	
<b>8000</b>	
<b>8500</b>	
<b>9000</b>	
<b>9500</b>	
<b>10000</b>	

- Change the 500 millisecond delay to 1000 and repeat the test
- Record the distance your robot traveled
- Repeat by adding 500 milliseconds to the pause length and measuring how far the robot drove

Once you're done, transfer the results you recorded into the graph at the end of this project. Keep this project with you, as you'll need this data for the dead reckoning competition

### **Project 3**

#### **The Perfect Right Angle**

Turns for the dead reckoning competition only need to be right angle turns (right angles are 90 degree angles). Therefore, you won't be creating a table of pauses and angles. Use the tape on the ground you used for the last project. However, this time, line your robot up on the line so you can determine the angle of the turn it makes. You don't need super accuracy, so don't worry about using a protractor. Download the following code into your robot and then shut off the power.

```

PAUSE 1000
GOSUB RIGHT
PAUSE 100
GOSUB FREEZE
END

```

**RIGHT:**  
:  
:  
:  
:  
**RETURN**

Carry your robot back to the tape marker and align it as carefully as you against the tape. Power up your robot and see how far it rotates. Adjust the 500 millisecond pause upward a little at a time until you find the proper pause length to get a good right angle turn. You may want to watch the robot turn a few times to see how consistently it rotates 90 degrees. After you find the proper pause length, right the pause length into the table on the last page.

We can't assume your robot turns symmetrically left and right, so repeat this activity again with the turn direction changed to left.

### **Project 4 Dead Reckoning Practice**

Now that you have collected the behavior data fro you robot, let's see how good you are at measuring a maze and programming your robot. Lay out some wooden blocks on the floor to form the walls of a maze. A good board to use is one by fours. Set the pattern of walls so your robot must make some left and right turns to navigate the maze. Also, be sure to place the walls at least eleven inches apart so your six inch by six inch robot can turn without bumping into the walls.

Now measure the distance your robot must travel forward and the direction of each turn. Take that data back to the PC and create a program to drive your robot though your maze. Always begin the program with a one second pause (**PAUSE 1000**) so it remains stationary long enough for you to get your hand out of the way. Always end your program with **END** so it stops at the end of the maze. Between the **PAUSE 1000** and **END**, you'll write a bunch of **GOSUB**s. The CheapBot-14 can only use 16 **GOSUB**s, so if you need too many, substitute the actual commands for Freeze for the **GOSUB FREEZE** command using the Copy and Paste commands.

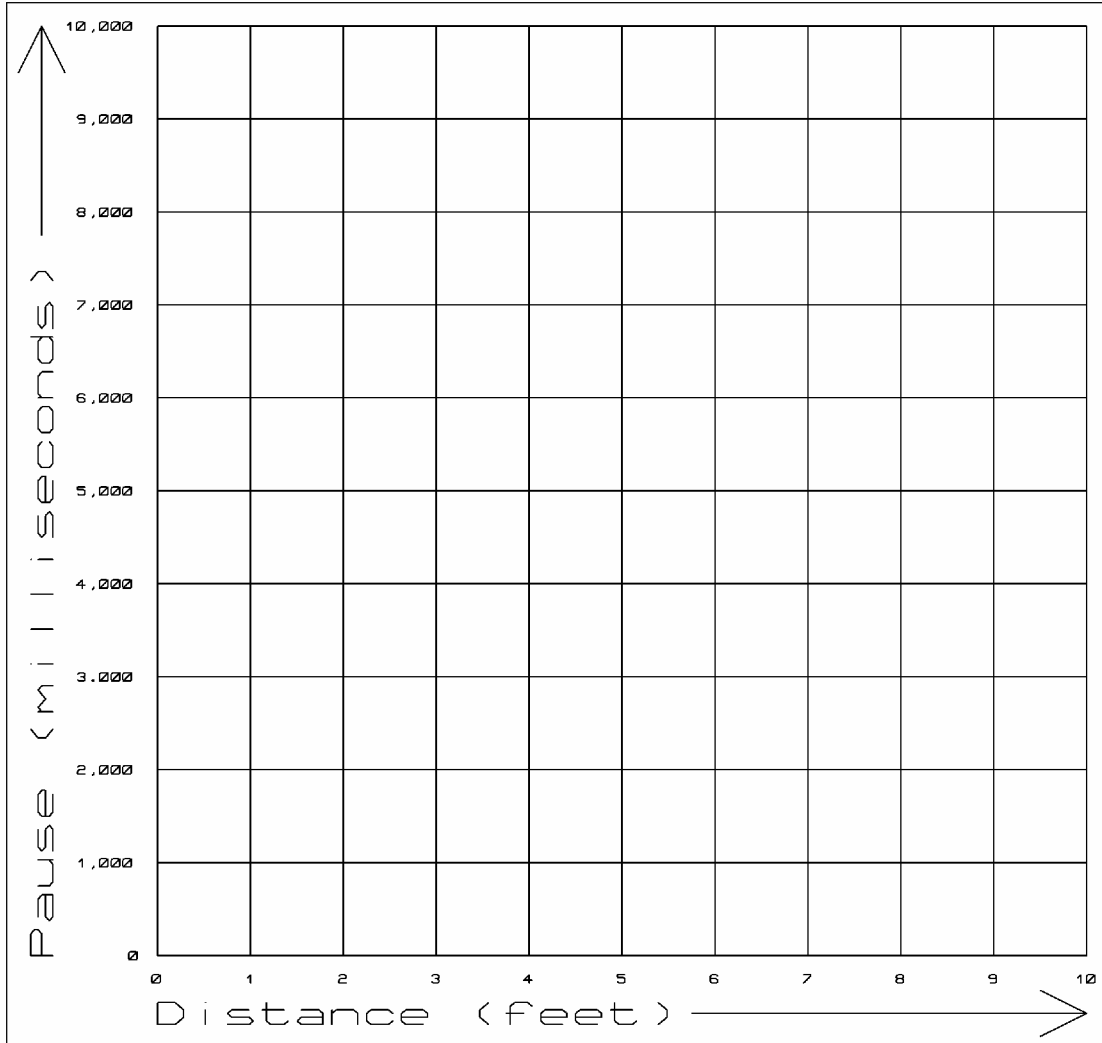
**GOSUB FREEZE** becomes **LOW 2**  
**LOW 3**  
**LOW 4**  
**LOW 5**  
**PAUSE 500**

However, don't replace all your **GOSUB FREEZE**s, or else you'll run out of memory.

A different trick you can use to save both memory and GOSUBs is to modify the FORWARDS subroutine by adding to the beginning of the subroutine a call to the FREEZE subroutine or adding the LOW commands found in the FREEZE subroutine.

## Robot Data

### Distance versus Pause Length



### Perfect Right Angle Turns

	Left Turn	Right Turn
<b>Pause Length</b>		