Subsystem for Simple Robotic Path Planning

There are many ways for a robot to interact with its surroundings, ranging from simple bumper switches to vision systems. Most of these systems have limitations in their field of view and/or they have cumbersome interfaces that use up valuable processor time. Sometimes, the information of an impending collision arrives too late or not at all. What is needed is a cheap and simple way to map th e surroundings. And that is the subject of this article.

A Simple Design

The design that I came up with uses very few components and a one-wire interface to reduce I/O overhead on the robot's main processor. The rotating head scans continuously back and forth, using a physical limit as a reference for position.

The limit is simply a bolt sticking out of the motor flange that the rotating head drives up against to locate "home." This is simpler and cheaper than sensing home with a switch or optical sensor (though I did leave an input available for a future home sensor, if needed). An array of distance readings, representing one full rotation, is updated with every sonar reading so that the robot can poll the sensor head at will and get the most recent data. Data is transmitted serially.

Sensor Motion

Ultrasonic sonar transducers have a perfect range for this application — from about six inches to 30 feet. If the sensor can be mounted to the top of the robot on a rotational head, it will have an unobstructed field of view of its surroundings. Stepper motors are commonly available in a 1.8 degree/step type that allows 200 steps/revolution.

The use of "half stepping" achieves 400 steps/revolution, which is enough resolution to distinguish relatively small objects at a considerable distance.

Mounting

I mounted the sensor in a blue plastic sensor enclosure from SensComp (see the table of sources for more information.) It has snaps molded in that capture the sensor very nicely and reduce the need for a lot of filing and final fitting. For tall robots, the sensor may need to be tilted down a bit. Four wires to the sensor carry power and two signals. This wire is dragged back and forth with each rotation, so it should be as flexible as possible. I used a cable from an old mouse. More adventurous folks may want to design a commutator system for transferring signals to the head, but that's mechanically tricky and fraught with problems.

The Stepper Motor

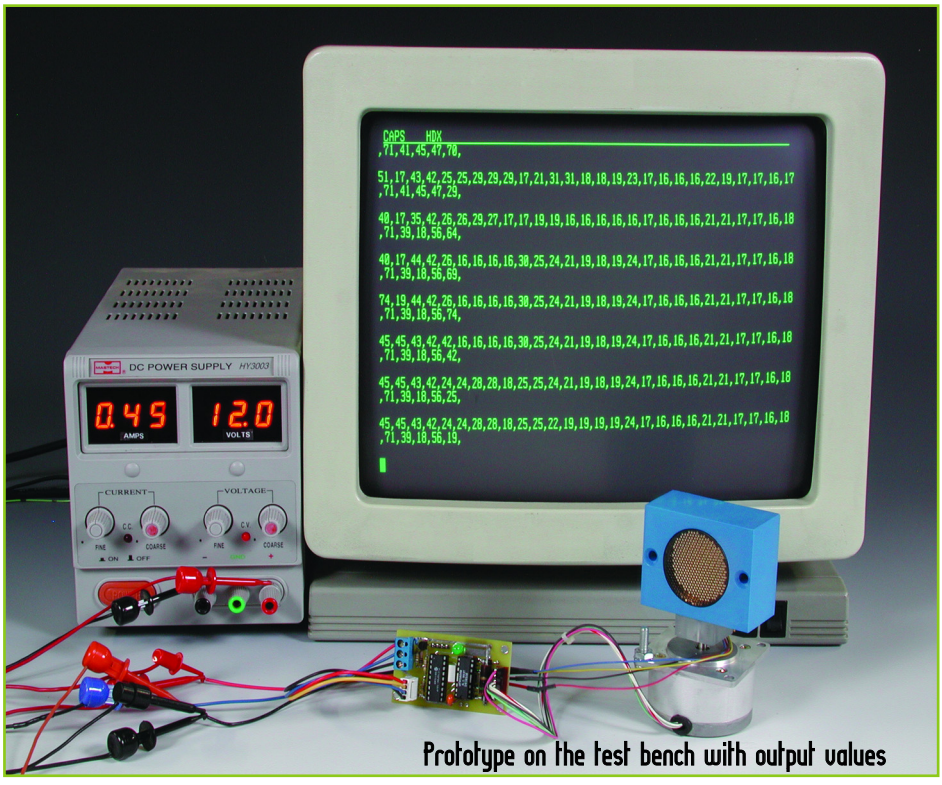
There are many types of stepper motors with a variety of coil and wiring configurations. The one I used is a six lead bipolar type, that has four coils arranged as two center tapped coils. Eight wire motors have separate wires per coil, and can be used in this application by tying the two pairs of coils together in series — but it's tricky to determine the correct phasing.

Steppers have many ways of identifying the coils with wire colors. Use an Ohmmeter to figure out which coils are connected to each other and which wires are common to each pair of phases. If you hook up one coil in reverse, the motor will just chatter and not rotate. It's a simple matter to transpose two wires and discover the correct phasing — no harm will be done to the driver chip.

Switching these coils in various sequences causes the motor to increment by steps. (See surplus sources for a list of suppliers.) Step motors with larger step angles could be used, but at the cost of decreased angular resolution.

The most common surplus stepper motors are printer pull-outs that have a 15 degree step angle — or a 7.5 degree half step yielding 48 steps — which may seem to be sufficient. These would be tempting to use at the cost of some resolution. However, most of these motors may lack the power to move the relatively large inertial mass of the head.

My design requires a motor that is a 1.8 degree per step type that can run on



Prototype on the test bench with output values

the robot's main batteries (usually 12 volts) and has reasonable power consumption.

The motor I used is rated at 20 ounce-inches at its rated 10 volts and 0.5 amp per phase (coil). This is a standard size that measures 2.25-inches in diameter by 1.625-inches deep, and usually comes with a 1/4-inch shaft. Steppers are designed with many different coil resistances to optimize them for various voltages.

Driver IC U1 is capable of driving 1.5 amp loads at up to 35 volts. However it is wise to limit the current to less than 0.5 amp, both to save battery power and to eliminate the need for a heatsink on the chip.

Since the driven load is almost entirely inertial, the power requirement is quite low. Power is controlled by a limiting resistor (R5) that can be tweaked for the desired power/performance trade-off. Calculate the value of R5 using Ohm's Law to find a combined series resistance of the motor coil and R5 that limits the total current to the desired amount.

You may find as I did that a substantially lower current than the motor's specification will still allow the motor to operate with sufficient power while conserving energy.

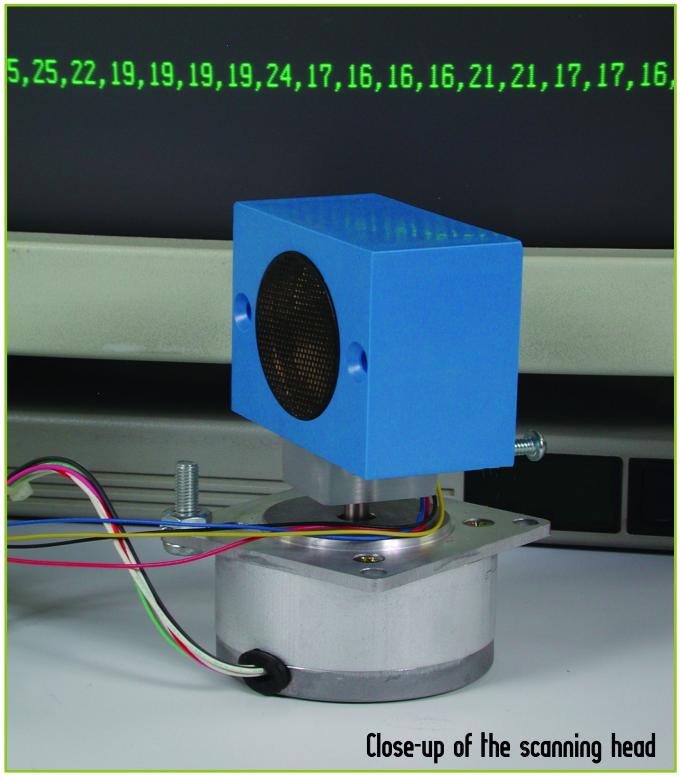
With R5 at 82 ohms, and a coil resistance of 20 ohms, my coil current was 125 mA — or about half the current specified for the motor. Stepper motors have more power at slower speeds and very high holding torque. Experimenting with the stepping speed and limiting resistance will yield an optimal overall performance for any given motor application. Be sure to use a resistor with more than sufficient wattage to handle the power — Ohm's Law states that watts = volts squared, divided by resistance.

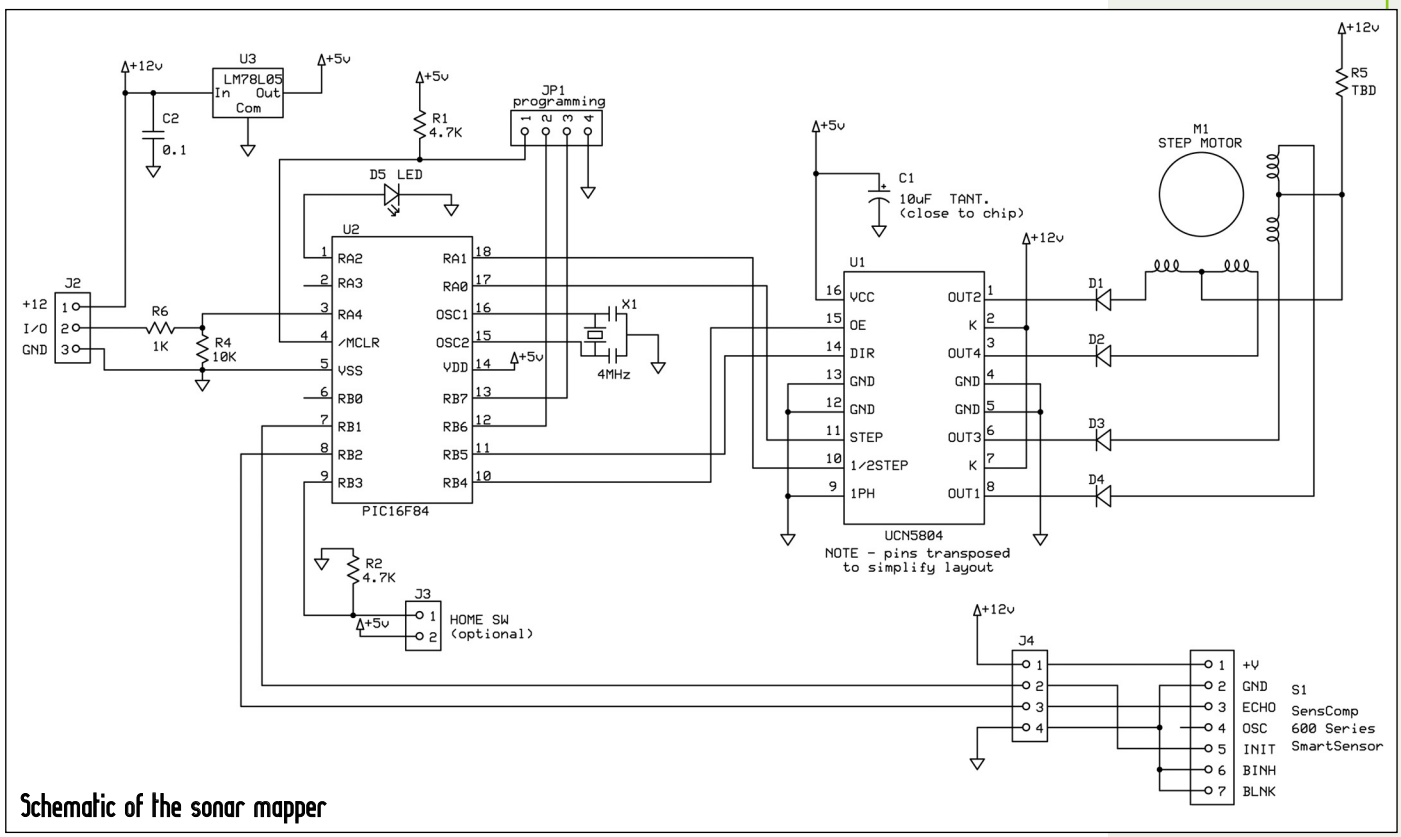
Motor Control

My favorite stepper motor control chip is the UCN5804 (U1) because it's cheap and simple. This part only requires four blocking diodes to limit the back EMF from the coils, and a current limiting resistor.

A tantalum filter capacitor across the logic power protects the chip from power spikes. Controls for this chip include OE (Output Enable), which is active low. I left this line LOW in my code to keep the motor active, so it won't lose position. It could be used to enable a power-saving scheme, if needed. However, the motor would need to be homed after each power down. The DIR input changes rotation direction based on level. The STEP input requires a short pulse to increment the motor one step. The 1/2STEP input, in combination with the PH input, can set up a variety of driving schemes.

By grounding PH and keeping 1/2STEP high, the motor is driven in half step mode (400 steps/rev). Setting


Close-up of the scanning head



Schematic of the sonar mapper

1/2STEP low will allow full stepping for faster movement of the motor for homing.

Sonar Signals

I used a SensComp 600 series Smart Sensor that is a re-packaged version of the "classic" Polaroid OEM kit sensor (see Resources). The two control lines are INIT and ECHO. The other inputs BINH and BLNK can be left open or grounded on the "classic" sensor. To take a reading, INIT is taken high which triggers the sensor to "ping" sixteen pulses at 49.4 kHz. The software routine then waits for the return ECHO to go high while looping to time how long it takes to arrive. Sound travels at about 0.9 mS/foot. By allowing for the return trip that is about 2 mS per foot at average temperature and humidity, distance can be calculated as a function of the echo delay time. The sensor "ping" sound is heard to the human ear as a click. The clicks are quite closely spaced as a full turn of the sensor takes around four seconds, so the perceived sound is a buzzing at about 50 Hz.

Sonar Sampling

While this design can take nearly 400 readings per turn (minus the size of the home stop), that much data is more than is necessary for basic mapping and collision avoidance. A data array of that size would also be beyond the capabilities of most small microcontrollers. My trick is to use only 372 of the 400 steps to take 12 readings in 31 vectors that are saved in a 31 byte array. Only the closest reading is stored in the array variable for that vector as each reading is taken.

Over Sampling Theory

The reason for all this "over sampling" is to ensure that small distant objects can be seen. The Polaroid type ultrasonic sensors claim to have a viewing cone of approximately 15 degrees, and can sense a flat one foot square object at their maximum range of 30 feet — smaller objects are less easy to resolve. Small round objects like broom handles placed at five or more feet present a poor reflector and may not be readily seen with every reading.

Additionally, the resolution of any given scan will be compromised by the turning motion of the robot, so it is best to over sample than under!

If vector #1 starts at the home sensor (which I placed at the rear of the robot) then the first value in the array represents a wedge of space of approximately 11 degrees facing the left rear. Subsequent array variables increment around the robot such that vector #15 should be almost dead ahead. The host controller can extrapolate a map from this data, or simply use it to avoid the closest object detected.

Graphic Diagram

I converted each reading to approximate inches to keep the value within a byte. Any arbitrary conversion can be used as long as you can figure out how to interpret it! I used two separate loops, one to sweep forward, and

the other to reverse, with an few extra motor steps added after the return sweep to ensure that the sensor head slams firmly against the home stop.

Code Details

I used a PIC16F84 (U2) and programmed it in PBasic, from microEngineering Labs. In circuit programming made the development much simpler, and I have made a four pin adapter to my programmer that allows me to use a four pin header to connect the programmer in circuit. Both PBASIC source and the HEX image are available for download from the *SERVO Magazine* website, **www.servomagazine.com**

Accessing the Data

During the scan, the code checks the data line to the host processor for a high signal. Once that signal is seen, the code waits for it to return low then switches to output mode after a brief pause and sends the 31 byte array in serial format. This I/O pin of the PIC is protected with a 1K current limiter, and a 10K pulls down at the chip. The data line reverts to an input after transmitting the data. Since the PIC is fully occupied while sending the data, it stops operating the scanner motor. This brief pause in rotation gives a clear indication of when data is being accessed by the host controller. I also added an LED (D5) that lights during the data dump for a secondary visual indicator.

This is a very simple design that can give robots a detailed map of their environment for very low cost — the rest of the job is making good use of the data provided by the sensor. SV

PARTS LIST

U1	UCN5804
U2	PIC16F84
U3	78L05
R1-R3	4.7KΩ
R4	10KΩ
R5	See text
R6	1KΩ
C1	10 µF tantalum
C2	0.1µF filter
D1-4D	1N4002
D5	Red LED
M1	12 V stepper motor, best under 1/2 amp, 1.8 degrees/step, 20+ oz-in of torque
S1	SensComp 600 series sensor, or Polaroid Ultrasonic OEM kit (includes two complete sensors) **www.polaroid-oem.com**

SOURCES

Stepper Motor:

Part #SSM8951 is what I used, and similar ones can be found on their site.
C&H Surplus
2176 E. Colorado Blvd., Pasadena, CA 91107
800-325-9465 — **www.aaaim.com/CandH/**

Also investigate the surplus offerings at Herbach & Rademan — **www.herbach.com**

Sonar Rangefinder:

SensComp 600 series smart sensor and mouting box (part number 619395.)
SensComp
36704 Commerce Rd., Livonia, MI 48150
734-953-4783 — **www.senscomp.com**

OEM Ultrasonic Kit:

Polaroid Corp.
www.polaroid-oem.com/ultrason.htm

Also, the Ultrasonic Owl Scanner — Sonar Sensor Kit has a 180 degree field of view using a hobby servo:

Ultrasonic Owl Scanner Kit, part #3-705 for $129.00
Sonar Explorer Kit, part #3-740 for $74.95
Robot Store / Mondo-tronics, Inc.
124 Paul Dr., Suite 12
San Rafael, CA 9490
415-491-4600 — **www.robotstore.com**

ABOUT THE AUTHOR

Guy Marsden is a Renaissance Man who designs and makes wood furniture and artwork, electronic art, and custom electronic prototypes. See his extensive website at **www.arttec.net** or Email him directly at guy@arttec.net