PROGO

A Beginning Language for Programming

Autonomous Mobile Robots

by
Keith L. Doty* and Scott Jantz**

*Professor (Emeritus) University of Florida **Machine Intelligence Laboratory, University of Florida
C.E.O. Mekatronix, Inc

Abstract

To alleviate the stress of program development for young students and beginning programmers of all ages who face the challenge of coding behaviors for intelligent autonomous mobile robots, the author has developed the PROGO™ language. PROGO™, defined completely by C Macros (#defines), establishes a set of standard functions for each type of Mekatronix robot and eliminates almost all the (painful!) details in programming robot behaviors in the C Language. PROGO™ offers a non-cryptic syntax that has an English-like flow. While PROGO™ does not implement all the functionality of C, the user can seamlessly integrate correct C language statements anywhere within a PROGO™ function without generating compiler errors. PROGO™ is easy to learn and use, six graders have learned the language in several work sessions.

Introduction

Students of the Machine Intelligence Laboratory at the University of Florida, typically program intelligent, autonomous mobile robots in the *C* programming language [Kernighan and Richie, 1978]. They do not possess a standard set of shared functions and must repeatedly reinvent the wheel. Non-standard *Include* files and macro definitions proliferate, making code sharing even more difficult and often forcing the students to focus on programming issues rather than robotic ones. While this is not all bad from a pedagogical standpoint, after all, students do learn from reinvention, such a development process does slow progress and limit the extent and sophistication of implemented behavior programs and robot control.

While many computer engineers enthusiastically engage in *C* programming, many roboticists do not share that enthusiasm. The cryptic syntax of *C* often irritates initiates to a high level of frustration. Even experienced *C* programmers often apply "=" in a conditional statement when "= =" is what they meant.

The problem of introducing students to robotics carries with it the burden of also introducing a computer language. If the student is not already familiar with the computer language chosen, the potential barrier to success can be overwhelming. To lower this barrier and alleviate the stress of code development for Mekatronix robots, the author has developed the PROGO™ language. PROGO™, defined completely by *C* Macros (#defines), establishes a set of standard kernel of robot functions for each of its robots and eliminates all *C* programming details [Qaiyumi, 1999]. The PROGO™ solution has broad applicability and is not confined to any particular company's robots.

PROGO™ offers a non-cryptic syntax that has an English-like flow, allowing the programmer to focus on robot behavior and not programming syntax and semantics. While PROGO™ does not implement all of the functionality of *C*, the user can seamlessly integrate any *C* language statements anywhere within a PROGO™ program as required. For example, PROGO™ only supports *int* data types. If the programmer requires floating point, a *C* declaration of *float* can be used within the PROGO™ code without error.

PROGO™ was designed so middle school children and beginning programmers could quickly learn it. This constraint had a powerful influence on the language design, to wit,

PROGO™ statements were designed to read like a type of formal English. This makes it easy for initiates who speak English to understand and absorb the syntax and semantics quickly. Further, *C* macros and functions absorb all the *C* syntactic support structures such as *include* files and robot functions such as *go()*. PROGO™ code is typically much shorter and easier to read than an equivalent *C* program, even though the language is more verbose.

As the student becomes proficient in PROGO™, we expect their transition to *C* will be much easier, but this has not been verified.

A Brief Description of PROGO™

To illustrate the simplicity of PROGO™, this paper provides a complete, ostensible definition of the language with some sample programs for the TJ PRO™ robot.

Structure of a PROGO™ Program

Every PROGO™ program (Figure 1) optionally begins with a *User Function List* of user defined functions, a **Dictionary** declaration of user defined integer variables followed by **Program_begin**. The program, consisting of PROGO™ (or *C) statements*, corresponds to *main()*in *C* and must be terminated by **Program_end**.

PROGO™ Functions

User defined Functions must appear in a list before **Program_begin**, the *User Function List*. This requirement maintains compatibility with the *C* language and makes for good programming practice as well. The *User Function List* syntax is:

```
Function <function_name1> used
                    ...
Function <function_nameK> used
```

The keyword **used** must be present after each function named.

Anywhere after **Program_end** the user may define a function with the following syntax.

```
Function <function_name>
Function_begin
  <block>
Function_end
```

To call a function in a program or another function, use the function's name followed by the keyword **call.**

```
<function_name> call
```

PROGO™ functions do not have parameters and cannot return a value except through global variables. While extremely limited, functionality does not suffer too much, although clarity might. Remember the language was defined with inexperienced programmers in mind. Experienced programmers can type PROGO™ functions just like *C* functions, although not technically defined in the PROGO™ kernel. For example, to make a function return a floating point number just type *float* before the word function and a return statement somewhere in <block>,

```
float Function <function_name>
        Function_begin
          <block>
        Function_end
```

To write functions with parameters in PROGO™, one must resort to *C* syntax.

PROGO Statements

An Appendix provides a brief synopsis of PROGO™ syntax, including the syntax for the robot specific motion and sensing commands. All PROGO™ statements begin with a capital letter. The only exception is a user-defined function where the user has chosen to begin a function name with a lower case letter.

Any *C integer expression* is valid in PROGO™, an extremely powerful feature. PROGO™ expressions utilizing arrays and structures can also be constructed, but programmers will first have to understand these data constructs and learn to declare the variables. Perhaps at that level of complexity, the user should learn *C*, although mixing statements causes no problem to the compiler, it might be confusing to the reader of the code!

Even if a student is not acquainted with a programming language, the readability and almost self-documenting aspects of PROGO™ statements makes learning it easy, after a brief

preliminary session explaining its syntax and semantics.

To convey what is involved with programming the TJ PRO™ robot, the next section illustrate a PROGO™ and *C* solution to the problem of making the robot trace out a 20inch square. With coaching and program examples, computer literate middle school students can begin modifying example programs and composing their own robot programs within 15 to 30 minutes.

Geometry Made Fun: Make the Robot Trace Out a Square

We have discovered that programming the robot to trace out simple geometric figures generates interest and excitement for middle and high school students and does not overwhelm beginners. We tape large pieces of white paper on the floor and attach pens with washable ink onto the robot. As the robot moves, the robot plots the geometric pattern. The robot becomes a movable plotter [Qaiyumi et. al. 1998]. From geometric figures we have noticed that participants of all ages branch into free form and artistic figures, ones pleasing to the eye.

While the triangle is the simplest polygon, beginners with little understanding of geometry may find programming a square easier. Right angles of the square can be taught as right turns, avoiding the introduction of unknown concepts. A turn of 120 degrees, the turning angle required to generate an equilateral triangle, is not intuitive and difficult to explain to one with no understanding of angles. Before continuing with drawing the square, I will digress some on the concept of angle and turning directions.

After a few "geometric" sessions on the robot, many young children pick up the abstract concept of angle quickly. We introduce them to clockwise turns (negative rotations), counterclockwise turns (positive rotations), a full circle, 360 degrees, a half circle, 180 degrees, and left (right) turns, 90(-90) degrees. After introducing these concepts, we ask middle schoolers to program the TJ PRO™ to go straight-ahead 36 inches, turn around and return to the same spot. A correct PROGO™ program to do this is:

```
Program_begin
  Forward 36 inches
  Turn_right 180 degrees
  Forward 36 inches
Program_end
```

Middle schoolers often use 360 for 180 in the **Turn_right** function. When the program executes and the robot spins 360 and continues onward, they understand their error immediately and correct it. Students also discover that another PROGO™ function **Turn_left 180 degrees** will work, illustrating that –180 degrees is the same as +180 degrees, a difficult concept indeed for young children!

We now return to the square-drawing problem. What should the algorithm be? Most children come up with the idea of drawing four sides and make four right turns,

```
Program_begin

  Forward 20 inches
  Turn_right 90 degrees
  Forward 20 inches
  Turn_right 90 degrees
  Forward 20 inches
  Turn_right 90 degrees
  Forward 20 inches
  Turn_right 90 degrees

Program_end
```

The above solution opens up the introduction of the **Repeat** statement. Instead of writing the sequence

```
Forward 20 inches
Turn_right 90 degrees
```

four times, the student learns to iterate with the **Repeat** statement,

```
Repeat 4 times
  Forward 20 inches
  Turn_right 90 degrees
Repeat_end
```

At this point, although the utility is not clear in this case, you can suggest the student write a function,

```
Function square_side
  Function_begin
```

```
    Turn_right 90 degrees
    Forward 20 inches
Function_end
```

and have the program call the function four times with the **Repeat** statement.

The resulting program, written in PROGO™ and *C*, appears in Figure 2 and Figure 3, respectively.

Examination of the *C* program should convince a novice programmer that *C* is not the language of choice when just beginning. After some experience with PROGO™, however, the student should be ready to take on *C*. The migration should be fairly smooth.

An Aside

If the robot traces out its trajectory on white paper, you will probably get a plot like one of the patterns shown in Figure 4. Because the robot does not turn 90 degrees precisely, it will not draw a true square. The drawings, produced graphically, illustrate what to expect if the angle is off by 10% in either direction. When I did the experiment, the angle error was under 10%.

Can the robot be programmed to perform a truer square? Sure, just play with the turn angle until you get the best results. Because of the limitations of the motors and their control circuits you will still not get a perfect square. These results provide a strong dose of reality therapy for young students not accustomed to thinking in terms of engineering tolerances. By the way, I programmed a $30,000 robot (not to be named!) to do the same thing. It did not draw a true square either. Ok, so it was a lot better (about 1% error), but the TJ PRO™ costs under $300!

Conclusion

Syntax rules in programming languages demand strict adherence. As the expressive power of a programming language increases, the complexity of its syntax typically increases. Novice programmers find complex syntax a barrier to learning and an obstacle to making the robot perform. The teaching language PROGO™, developed at Mekatronix, works around the steep learning curve of other languages and provides a tool suitable for beginners. A *C* language preprocessor converts PROGO™ programs directly into *C*. This *C* piggy-back feature insures the portability and wide spread applicability of PROGO™ as well as functional usefulness. The PROGO™ design offers a "reasonable" tradeoff between expressiveness and simplicity, but yet retains the important language features of conditionals, iteration, recursion, logic, arithmetic and so on. An example program in PROGO™ and *C* demonstrates that even beginners can learn quickly to program the robot to do something interesting using only PROGO™. As a by-product of all the fun, the student learns important programming concepts and principles.

Intelligent autonomous mobile robots offer a vehicle for technical training of young students in behavior-based robotics. Experience with bright middle school children demonstrates the claims of PROGO™'s ease of use and utility as a beginning robot programming language. Development of a working program suggests how the robots might be used to teach children basic geometric concepts as well. Bonus!

Finally, simply watching a robot execute a program and behave as you planned produces tremendous satisfaction…a satisfaction that tends to encourage further investigation, exploration and learning by young children.

References

1. [Doty, 1997] Doty, Keith L. *Autonomous Mobile Robots in Engineering Education: An Intelligent Machines Design Laboratory Course.* Florida Conference on Recent Advances in Robotics, Florida International University, Miami, FL, April 10-11 1997.
2. [Kernighan and Richie, 1978] Brian W. Kernighan and Dennis M. Ritchie 1978. *The C Programming Language*. Prentice Hall Software Series.
3. [Qaiyumi, 1999] Mekatronix Homepage, http://www.mekatronix.com
4. [Qaiyumi et. al. 1998] Aamir Qaiyumi, Scott D. Jantz, A. Antonio Arroyo, J. Andrew Bagnell and Patrick O'Malley Robots in the Classroom: Using Mobile Autonomous Agents to Stimulate Interest in Science and Engineering. Florida Conference on Recent Advances in Robotics Florida Institute of Technology.

```
/*User Function List*/
   Function <function_name1> used
    ...
   Function <function_nameK> used

/*Declare integer variables*/
   Dictionary
     <variable>,
     <variable>,
     ...
     <variable>
   ok


/*Program Brackets*/
  Program_begin
   <block of PROGO™ statements>
  Program_end

/*Function Defintions*/
   Function <function_name1>
      Function_begin
        <block of PROGO™ statements>
      Function_end

    ...
    ...
   Function <function_nameK>
      Function_begin
        <block of PROGO™ statements>
      Function_end
```

Figure 1. A schematic view of a PROGO™ program.

```
/*
  Make the robot trace a square.
*/

Function square_side used

Program_begin

  Repeat 4 times
    Repeat_begin

      square_side call

    Repeat_end

Program_end

Function square_side
  Function_begin

    Turn_right 90 degrees

    Forward 20 inches

  Function_end
```

Figure 2. This PROGO™ program also commands the TJ PRO™ robot to trace out a square 20 inches to the side.

```
/*****************************************************************
 *                MEKATRONIX Copyright 1998                      *
 * Title          square.c                                       *
 * Programmer     Keith L. Doty                                  *
 * Description                                                   *
 *    Make the robot trace a square.                             *
 *****************************************************************/

/************************** includes ****************************/
#include <tjpbase.h>
#include <stdio.h>
/****************************************************************/

/************************** prototypes **************************/
void square_side(void);
void turn_right(int);
void forward(int);
/************************ end prototypes ************************/

void main(void)
/************************** main ********************************/
{ int i;
/*Initialize the resources of the robot*/
  init_analog();
  init_motortjp();
  init_clocktjp();
  init_serial();
  IRE_ON;
  START;
  for(i=0; i<4; i++) square_side();
}
/************************ end main ******************************/

void square_side(void)
{
 turn_right(90);
 forward(20);
}

void turn_right(int deg)
{. . .}
void forward(int dist)
{. . . }
```

Figure 3. A *C* program to command the robot to trace a 20 inch square.

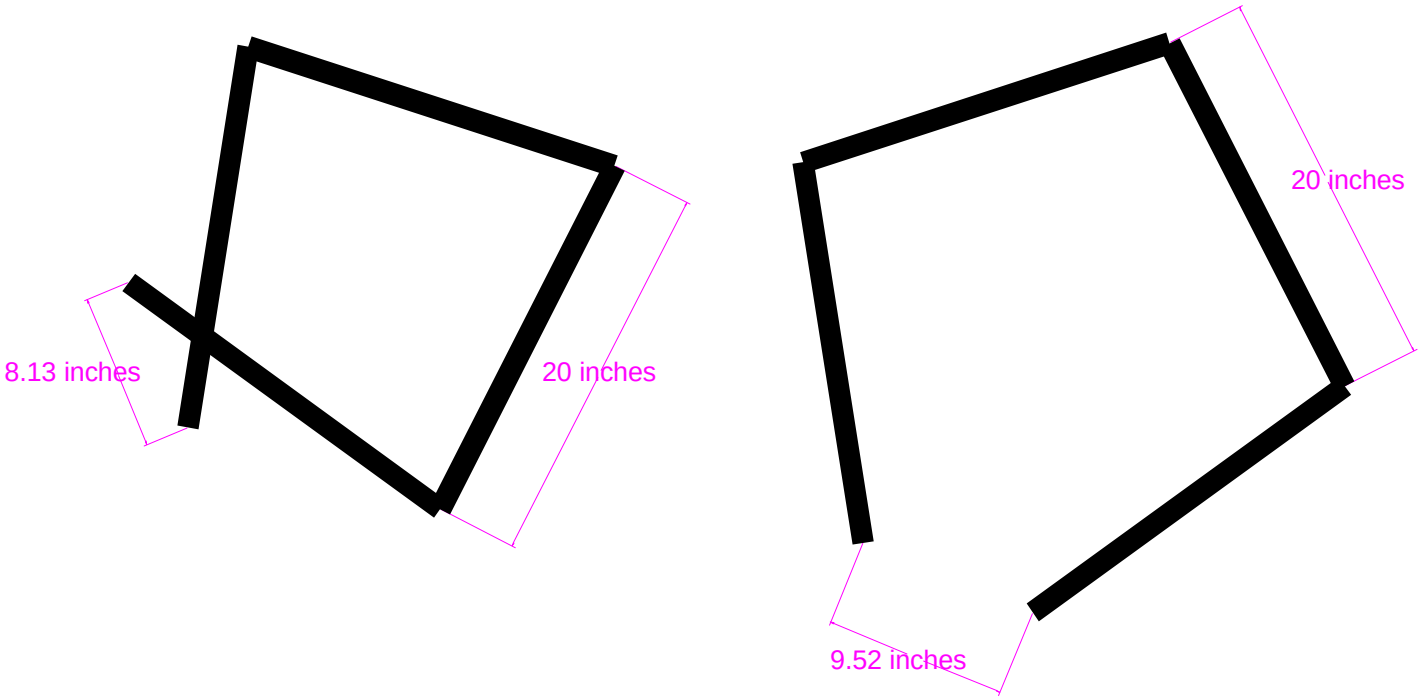

Figure 4. When the TJ PRO™ robot traces out the square, it does not execute true right angle turns. TJ PRO™ traces the “square” on the left when each turn is 99 degrees and the one on the right when each turn is 81 degrees, 10 % angle error in the positive and 10% angle error in the negative direction., respectively.

APPENDIX

PROGO™ SYNTAX
Copyright 1998 by Mekatronix, Inc.

PROGO™ LANGUAGE CONSTRUCTS

User Function List

```
Function <function_name1> used
 ...
Function <function_nameK> used
```

Declare integer variables

```
Dictionary
  <variable>,
  <variable>,
  ...
  <variable>
ok
```

Program Brackets

```
Program_begin
Program_end
```

Function Defintion

```
Function <function_name>
   Function_begin
     <block>
   Function_end
```

Function Call Statement

```
<function_name> call
```

Relational Operators

```
greater_than
greater_than_or_equal_to
less_than
less_than_or_equal_to
equal_to
not_equal_to
and
or
not
```

Bitwise Logical Operators

```
bit_and
bit_or
bit_not
bit_xor
```

Arithmetic plus, minus, times, divide and modulus

```
+, -, *, / , modulo
```

PROGO™ STATEMENT SYNTAX

Assignment Statement

```
Set <variable> to <expression> ok
```

Repeat Statement

```
Repeat <number> times
  <block>
Repeat_end
```

While Statement

```
While <test_expression>
perform
  <block>
end
```

Endless While Statement

```
Do_forever
  <block>
end
```

If Statement **If** <test_expression> **then** <block> **end** **or_else** <block> **end**	*If Short Form:* ***If*** *without* ***else*** **If** <test_expression> **then** <block> **end** *Function Call Statement* <function_name> **call**
Output from Robot to Personal Computer **Display** "<character string>" **on_screen** **Clear_screen** **Home_screen** **Write** <variable> **on_screen**	*Input to the Robot from the Personal Computer* **input_number** **input_character**

ROBOT MOTION FUNCTIONS

Move Robot Forward a Specified Distance **Fwd** <number> **inches** **Forward** <number> **inches** *Move Robot back a Specified Distance* **Back** <number> **inches** **Backward** <number> **inches**	*Turn Commands* **Pivot_right** <angle> **degrees** **Pivot_left** <angle> **degrees** **Turn_right** <angle> **degrees** **Turn_left** <angle> **degrees** **Spinccw** *Spin counterclockwise* **Spincw** *Spin clockwise*
Robot wheel move commands **Left_wheel** <number> **percent** **Right_wheel** <number> **percent** **Move** <number> **lws** <number> **rws** (**lws** means left wheel speed) (**rws** means right wheel speed)	*Simple Motion Commands* **Go** **Reverse** **Stop** *Delay Command* **Wait** <time_in_ms> **ms**

ROBOT SENSOR FUNCTIONS

Analog Bumper function **BUMPER** *Returns bump contact reading* *Logical bumper tests* **FRONT_BUMP** *True if a front bump* **BACK_BUMP** *True if a back bump*	*Read IR sensor values* **RIGHT_IR** *Read right IR sensor value* **LEFT_IR** *Read left IR sensor value* *Control IR emitters* **IR_ON** *Turn on all IR emitters* **IR_OFF** *Turn off all IR emitters*