前言

为什么要学习步行机器人

步行机器人在教育与娱乐机器人领域是一个相对未经开垦的领域。造成这一现象的原因有许多：首先是步行机器人的设计存在固有的缺陷，使其很难处理诸如楼梯这样各种复杂的场景；其次步行机器人的编程需要更为复杂的算法和基于传感器的反馈；再者就是要让步行机器人获得期望的运动，还需软件和硬件更紧密地协同。

这也许就是我们大多数机器人都采用轮子移动的原因之一。然而，人们还是有一种自然的倾向来赞美步行机器人：他们看起来更像人类；他们能够为人们提供更多的娱乐价值因为他们看起来更有趣；而且使一个具有双足的机器人成功行走也是一个值得追寻和挑战的概念。如果你能成功的使用本手册介绍的步行机器人，那么你就能成为一个受过良好训练的机器人专家。

除了学习编程技术外，步行机器人还能应用到许多真实的困难场合。具有多条腿和多个自由度的机器人最终能够处理更为复杂的场合。

本书介绍的步行机器人简化了步行机器人的概念。尽管本手册中介绍的步行机器人肯定不能照顾老人，不能帮你到厨房拿饮料，不能给你的房间吸尘，也不能驾车送你到商店等，但它确实能够为你提供一个可编程双足机器人的第一个映像和概念。通过本教材的学习，你将发现给步行机器人编程是一个复杂但是很有价值的任务。该步行机器人用一种积极的、有趣的和友好的方式引入并介绍了嵌入式控制概念、方法和实现过程。

从步行机器人获得的教育概念

老师们通常都会问帕拉斯，他们能从帕拉斯不同的课程当中学到什么。可以说步行机器人是一个高级的机器人学项目，无论是老师还是学生，都能够从中学到如下的概念和技能：

机械和电气系统的相关性和依赖性，即机电一体化的基本概念；

调整硬件和软件以获得预期结果的能力；

使用 BASIC Stamp2 的高级编程技能：一个高效的步行机器人程序要涉及 Stamp 的 DATA 语句和 EEPROM 指针，变量别名，以及程序子函数的使用技巧，他们在程序执行前中能够重复使用和配置；另外还涉及到通用的声音编程实践；

包括实时编程、分类结构和状态机的高级机器人编程技术；

如何一步步从基本的程序编写到完成更复杂并最终有用的程序的整个过程。

如果需要帮助，请直接致电或者发邮件给帕拉斯在中国的唯一合作伙伴—德普施科技寻求支持。我们将非常高兴地帮助你，让你的步行机器人按照你所希望的方式行走。

教师和学生指南

14 岁左右的学生应该能够组装和编程帕拉斯的步行机器人。因为该步行机器人用到了更复杂机械结构和编程技术，我们相信能够成功使用该套件的最年轻的学生可能是 12 岁左

右。如果您在这方面有其它成功的经验，请通过公司邮箱让我们知道。该教材没有提供教师指南。如果需要，我们可以选择一些教材中提出的挑战课题提供答案。

教育机器人论坛

德普施科技在其网站 www.depush.com/bbs 中开通了 BBS 论坛，其中有一个关于教育机器人的专题。如果你有问题，也可以同过该论坛进行讨论。

目录

第一章 装配步行机器人

认识机器人家族中的最新成员

尽管它看起来很简单，但你将很快将看到这种使用两个伺服电机进行两足移动的步行机器人比两轮机器人（宝贝车）复杂的得多，它通过机械机构和 BASIC Stamp 控制来实现行走。实际上，步行机器人使用到许多传感器作为反馈元件，这其中包括能检测到一定距离内是否有物体的红外线发射器及红外线检测器，缓冲传感器（闪光脚趾）和用以测量倾斜参数的加速度计。（注意，闪光脚趾和加速度计不是步行机器人的标准配件，用户需另外采购。）

如果你耐心调节步行机器人的硬件和软件，步行机器人可以完成轮式机器人所能完成的所有动作。步行机器人不仅比轮式机器人有趣很多，而且通过学习步行机器人行走控制，也能更加熟练的掌握控制程序的编写。步行机器人的编程将会引导你学会如何设计PBASIC程序，包括如何使用常量和变量、程序的指针以及存储数据的EEPROM（电可擦除只读存储器）。程序设计是否良好的其中一个标准是，在对一些机械装置调整后，不需重新编写整个程序，只需对程序作简单修改就能实现所要求的功能。

步行机器人的运动由两个伺服电机控制（这有点类似于操纵飞机的螺旋桨）。两个伺服电机有各自的作用，顶部的伺服电机控制机器人的重心位置在 1.5 公分的范围内摆动，而底部的伺服电机控制机器人的前后行走。步行机器人的腿和脚踝之间采用了一个简单的平行连接件，确保双脚能够平行的站在地面上。两条腿都连接在同一个电机上，所以一只脚向前移动时，另一只脚就会向后移动。

单独控制一个电动机，步行机器人能够完成前进、后退、左转、右转等动作。综合控制步行机器人两个伺服电机的运动，能实现更加协调、更加平稳的行走。

步行机器人的伺服电机和传感器由一个 BASIC Stamp2 微型控制器来控制。微控制器 BASIC Stamp2 是教学系统中运用很广泛的一种芯片，它提供了较大的程序空间、存储空间供机器人使用，并且处理速度快。

搭建步行机器人

根据机器人在地面上行走的方式，步行机器人的装配方法有很多。合理的装配方式取决于加装的传感器及其他硬件设施和机器人行走的速度等因素。恰当的装配对步行机器人的行走是很重要的，它将有利于程序的设计和步行机器人的行走。

需要的工具

搭建步行机器人需要以下工具：

- 小尖嘴钳(包含在零配件中）
- 小号螺丝刀（包含在零配件中）
- 中号十字螺丝刀（包含在零配件中）

附录A中附有完整的配件清单供参考。如果您的配套工具中少了应有的配件，我们会免费提供。在日常的生产过程中，我们有着严格的质量分析和质量管理机制，但是偶尔也会出现一点失误。如果您缺少配件，请您马上联系我们。如果您的配件破损了或者您想为步行机

器人增加一些配件，您也可以与我们联系定购您所需要的配件。

步骤 1：安装顶部的“倾斜”伺服电机

需要以下配件：

- 4颗M3x8盘头螺钉
- 4颗普通M3 螺母
- 一个步行机器人的支架
- 一个步行机器人的伺服电机

用螺丝刀取出电机连接部件和电机输出轴之间的螺钉，将电机连接部件从电机输出轴上取下来。保存好螺钉及电机连接件，以备后面使用。

如图1-1所示，正确安装步行机器人的伺服电机。用4个M3x8盘头螺钉和4个普通M3螺母将其固定。最简单的方法是用一只手扶螺母另一只手拧螺丝刀。

图1-1，安装倾斜伺服电机

步骤 2：安装底部的“跨步”伺服电机

需要以下配件：

- 4颗M3x8盘头螺钉
- 4颗普通M3螺母
- 1颗步行机器人的伺服电机

用螺丝刀取出电机连接部件和电机输出轴之间的螺钉，将电机连接部件从电机输出轴上取下来。保存好螺钉及电机连接件，以备后面使用。

如图1-2所示，正确安装步行机器人的底部伺服电机。用4个M3x8盘头螺钉和4个普通M3螺母将其固定。

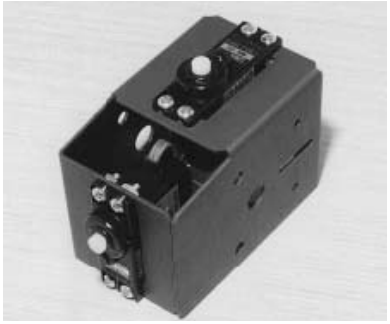


图1-2　安装跨步伺服电机

步骤 3：伺服电机归中的电气调整

需要以下配件：

- 电池盒
- 4只AA电池
- 串口线（RS-232 DB-9 and straight-through）
- BASIC Stamp软件

伺服电机归中的调整应该在步行机器人的其它组装之前进行，这样可避免之后编写程序时出现一些问题。切不能缺省这一步，它可使后面的调整更加容易。

将两个伺服电机的连接插头插入步行机器人电路板内的X6和X7。字母B对应着伺服电机的黑色导线，参考图1-3。

接下来，将4节电池装入电池盒中，并将电源的开关拨至0位。电池盒的白线与步行机器人接合板“+”端相连，黑线与接合板上“-”端相连，用螺丝刀将其固定。

使用一根串口线，将步行机器人的电路板与计算机相连。

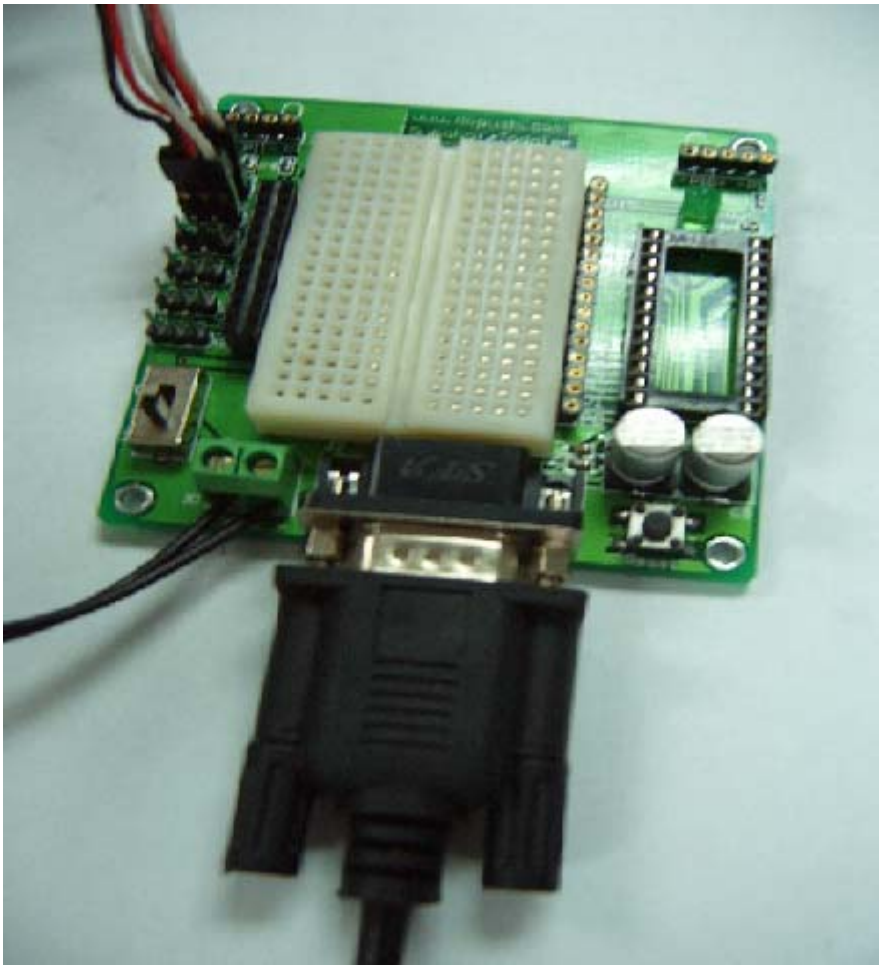


图1-3 连线示意图

步行机器人的电源开关有3种位置。每一位置对应着一种状态。三种位置中的“1”对应给伺服电机外所有电路提供电源。

位置“0”——不提供电源

位置“1”——为步行机器人电路板提供电源

位置“2”——为电路板、伺服电机等所有元件提供电源

在后面的章节中，你会发现电源开关的这3个位置非常有用。在调试机器人过程中，要在不移动步行机器人的情况下，单独调试机器人的传感器，可通过选择电源的这3个位置来实现。

下一步，利用BASIC Stamp编写程序。打开BASIC Stamp的编辑窗口，写下并载入下面调试伺服电机的程序：

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 1.1 - Servo Centering.bs2
' Electrically center Toddler servos during robot construction
' {$STAMP BS2}
' {$PBASIC 2.5}

' -----[ Declarations ]-------------------------------------------------------

TiltServo               CON     12      ' Tilting servo on P12
StrideServo             CON     13      ' Stride servo on P13

' -----[ Main Routine ]-------------------------------------------------------

Center:                                 ' Center servos with 1500 us pulses
  PULSOUT TiltServo, 750
  PULSOUT StrideServo, 750
  PAUSE 25                              ' Wait 25 ms
GOTO Center                             ' Repeat
```

程序编写完成后，将电源开关位置拨至“1”，通过菜单Run/Run或点击工具栏上的“▶”按钮来运行程序并将程序写入芯片。再将电源开关拨至“2”，这个程序设置的是一个死循环，当两个伺服电机停止运行时，表明电机的调试工作完成，断开电源（拨至“0”位），断开伺服电机与电路板的连线，取出电池盒中的电池，并拆除电池盒连接到电路板的导线。

步骤 4：安装伺服角

需要以下配件：

- 两个伺服电机连接部件
- 2颗将伺服角安装到伺服电机上的黑色小螺丝

如图1-4所示，伺服角与伺服电机通过齿形槽连接，利用步骤1和步骤2中取出的螺丝将连接件固定在伺服电机上。

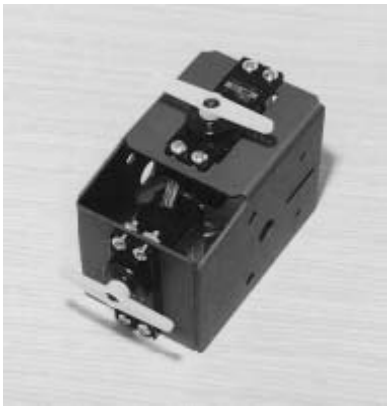


图1-4 安装伺服电机的连接部件

步骤 5：在跨步伺服电机上安装黄铜夹头

需要以下配件：

- 2个黄铜夹头（黄铜支架，固定螺丝和金属扣眼--也称为E-Z连接器）
- 2个M2x5螺丝

将两个黄铜夹头安装在跨步伺服电机连接件最外端的两个孔上（如图1-5所示，安装在伺服电机白色塑料连接件的两端）。用钳子将橡胶夹头压入黄铜夹头内，再用两个M2x5螺丝将其固定，使其不会脱离。（如图1-6所示）。

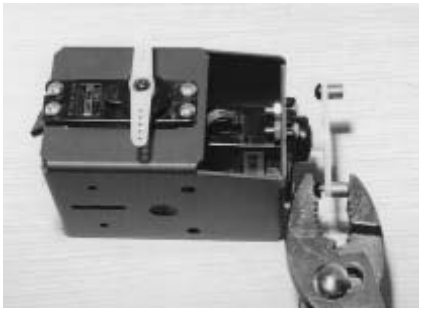


图1-5 在踏步电机上安装黄铜夹头

图1-6 固定黄铜夹头

步骤 6：安装顶板

需要以下配件：

- 步行机器人的顶板
- 4颗M3x8平头螺钉
- 4颗普通M3螺母

如图1-7所示，在步行机器人支架上安装顶板是相当容易的。一只手将M3x8平头螺钉从顶板上方穿过顶板及机器人身躯上对应的孔，另一只手将M3的螺母旋入螺钉，然后将其拧紧。重复以上步骤直至顶板安装完毕。注意：这一步用到的是M3x8平头螺钉，而不是平底的圆头螺钉。

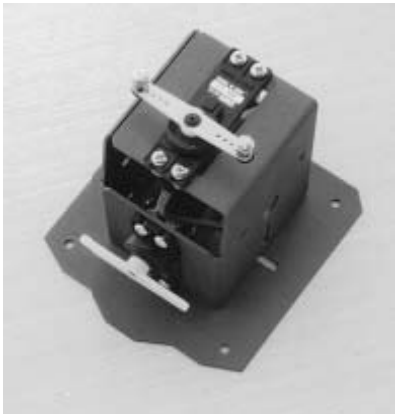


图1-7 安装顶板

步骤 7：安装身躯与腿之间的不锈钢连接杆

需要以下配件：

- 77毫米长金属圆杆
- 四个Φ5塑料垫圈

如图1-8所示，将两根金属圆杆分别穿过步行机器人支架上对应的两个孔，然后在杆的两端分别上一个垫圈。垫圈可减小步行机器人腿与身体之间的摩擦。

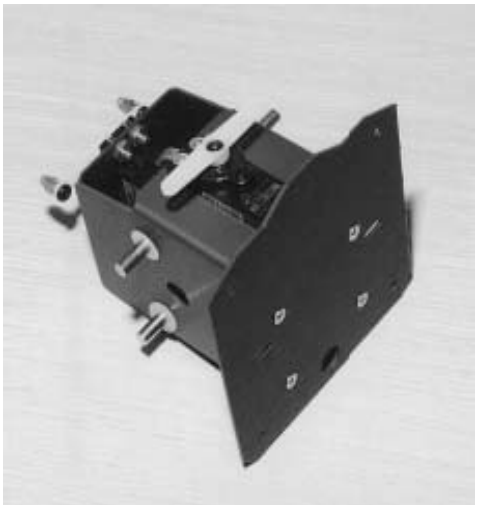


图1-8 安装连接杆

步骤 8：在步行机器人的身体上安装双腿

需要以下配件：

- 四支步行机器人的腿

如图1-9所示，将步行机器人的4条腿分别安装在两根金属圆杆上。

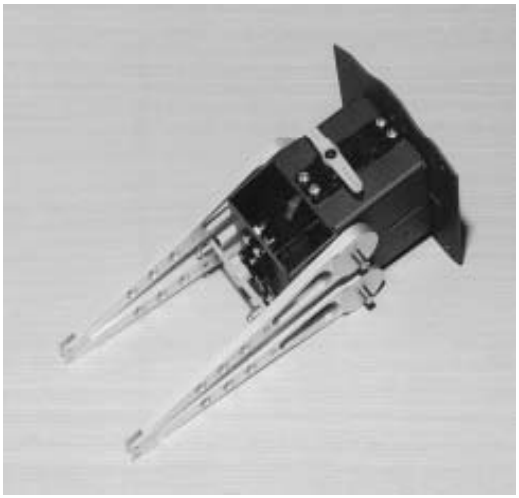


图1-9 安装四条腿

步骤 9：用轴环固定腿部

需要以下配件：

- 四个带有固定螺丝的轴环
- 4颗M3x5螺丝

在配件中找到金属轴环并将其套在金属圆杆的两端。如图1-10所示，用M3x5螺丝将每一个轴环固定在金属杆上。如果螺丝拧的不够紧，则在感觉到有阻力的情况下再转过一个很小的角度，这样的松紧程度最为合适。但如果拧的太紧，它可能阻碍伺服电机的正常工作，致使步行机器人不能很好的行走。

图1-10 安装轴环

步骤 10：安装两腿间的连接件

需要以下配件：

- 2个白色塑料垫圈（也称为E-Z调节角质托架）
- 2颗M3x15盘头螺钉，2颗M2x5螺钉
- 2颗M3防松螺母
- 2个黄铜直角金属杆

这一步分为两个子步骤：

首先，如图1-11所示用一颗M3x15盘头螺钉和一颗防松螺母将机器人左腿与塑料直角托架连接起来，再用一颗M3x15盘头螺钉和一颗防松螺母将机器人右腿与塑料直角托架连接起来。拧螺丝时，注意确保托架与螺丝在两腿部件上有足够的旋转空间，可进行180度的旋转。

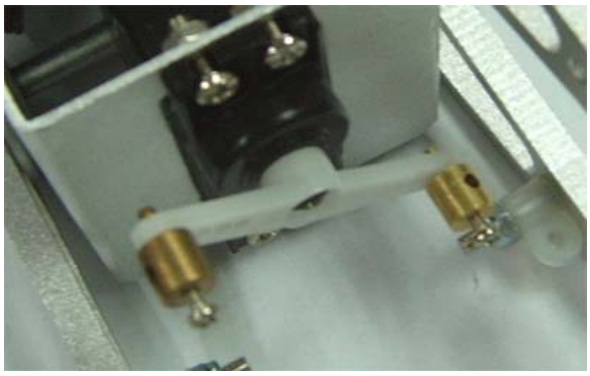


图1-11 加装直角托架

取出两根直角黄铜线，将黄铜线较长的一端穿过黄铜垫圈，如图1-12所示。然后将黄铜线较短的一端穿过塑料直角托架。用手扶住黄铜线，并用M2x5螺丝将其固定到黄铜夹头上。

调整链结，保证两条腿没有倾斜，并能垂直的站立。假如你需要或者伺服电机被移动过，你可以再次校正伺服电机的中心位置。重复以上步骤，调整机器人的另一只脚。

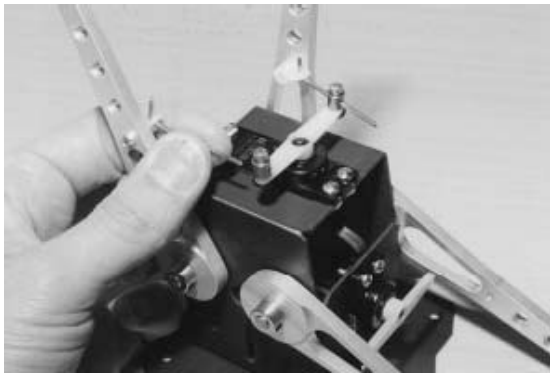


图1-12 连接四条腿

步骤 11：加装步行机器人的脚踝

需要以下配件：

- 2个脚踝配件
- 4个M2x10盘头螺钉
- 普通M2螺母

如图1-13所示，用4颗M2x10盘头螺钉将脚踝加装到步行机器人腿上。脚踝较长的一端应朝向步行机器人腿的后方，这样有利于步行机器人前进过程中重心的调整，可以更好的控制机器人。

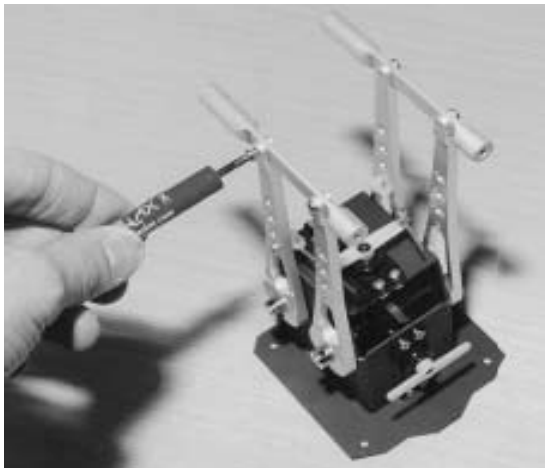


图1-13 加装脚踝

步骤 12：安装步行机器人的双脚

需要以下配件：

- 步行机器人的左脚和右脚

- 4颗普通M3x8螺钉
- 中号十字螺丝刀

如图1-14所示，将步行机器人脚踝安装在步行机器人脚掌上的第2个大孔上。如果太紧，可以将双脚稍微向外侧弯曲。

用两颗M3x8螺钉（这种螺钉不宜安装过紧，安装完毕后用手转动一下，确保能够容易转动）在步行机器人脚踝上加装左脚底板。然后以同样的方法安装步行机器人的右脚底板。为其写入程序前，确保每一只脚都能自由倾斜。

图1-15 加装脚板

步骤 13：安装倾斜杆

需要以下配件

- 2根直径2mm不锈钢杆，两端有螺纹
- 4个带有螺纹的半圆形接头（图1-15）

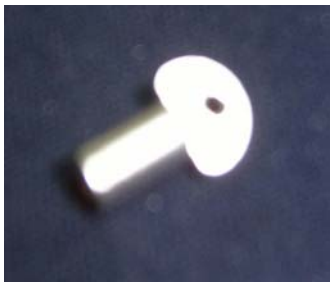


图1-15 带螺纹的半球形接头

在不锈钢杆的两端分别旋入2个半球形接头，调整半圆形接头旋入程度，二者之间的连接距离比步行机器人直立状态下电机连接件最外侧孔到脚掌最外侧小孔间的距离稍长一点最为合适。这样有利于机器人拐弯时脚紧贴在地面上。

图1-16 加装好的倾斜杆

以相同的方法加装另一根倾斜杆。

步骤 14：固定倾斜杆

需要以下配件：

- 4颗M2x10螺钉
- 4个M2螺母
- 4个Φ3塑料垫片

图1-17 安装倾斜杆

如图1-17所示，用M2x10螺丝将倾斜杆的半圆形接头固定到电机连接件最外侧的孔上，注意在电机连接件和半圆形接头之间垫上Φ3塑料垫片，并且螺丝不能拧太紧。两根倾斜杆分别装到电机连接件两侧的孔上。

图1-18 半圆形接头和脚掌的连接

如图1-18所示，用M2x10螺丝将倾斜杆的另一半圆形接头固定到步行机器人脚掌最外侧的孔上，注意二者之间需垫上Φ3塑料垫片，然后将两颗M2螺母拧到螺丝上，用2颗螺母是由于它们自身可以形成自锁，这样机器人运动时螺母不易脱落。用相同的方法将另一根倾斜杆连接到机器人另一脚掌上。

步行机器人的微调：当步行机器人的身体略往后倾斜时，步行机器人的性能是最佳的。也就是说，当两台伺服电机都处在中心位置时，步行机器人的双脚是平站在地面上的或者略微的有一点向后倾斜。

检查机器人时，首先要确定步行机器人的双脚平站在地面上，或者略向前倾斜。如果需要再一次调整其他硬件，可以利用BASIC Stamp重新调整伺服电机的中心位置。

步骤 15：安装电池盒

需要以下配件：

- 电池盒
- 2颗M3x8平头螺钉

- 2颗普通M3螺母

将步行机器人直立摆放。

如图1-19所示，用两颗平头螺钉和螺母将电池盒固定在步行机器人的顶部。拧紧平头螺钉，使平头螺钉嵌入电池盒中，并且从电池盒另一侧穿出。

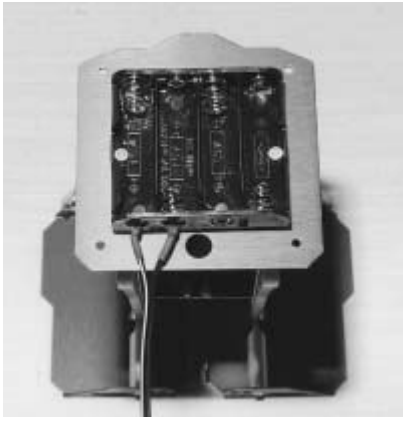


图1-19 安装电池盒

步骤 16：安装四个支柱

需要以下配件：

- 4根支柱
- 4颗M3x8盘头螺钉

如图1-20所示，用M3x8盘头螺钉在步行机器人顶板上安装4根支柱。

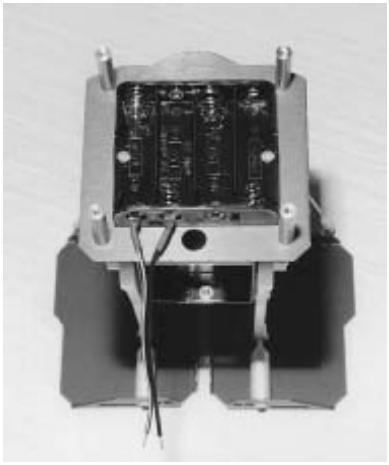


图1-20 在顶板上安装支柱

步骤 17：连接步行机器人的电路板和电池盒之间的导线

需要以下配件：

- 步行机器人的BASIC Stamp电路板

电池盒的白色导线与机器人电路板上的“+”接线头连接，另外一根导线与“–”接线头连接。用螺丝刀将它们固定（图1-21）。

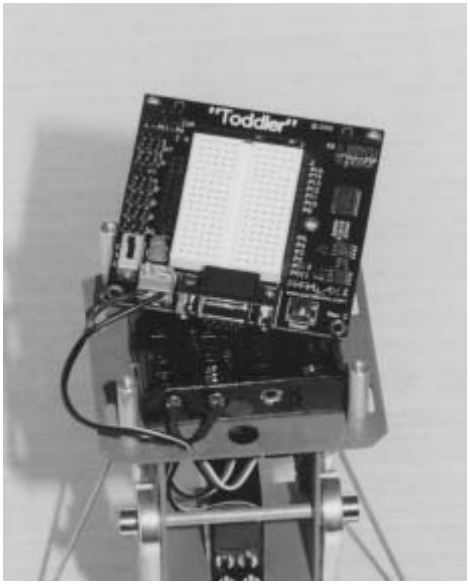


图1-21连接电源线

步骤 18：装入 4 个 AA 电池

需要以下配件：

- 4节AA电池

如图1-22所示，安装4节电池。然后将步行机器人电路板上的电源开关拨至“1”位，检验电源是否连接正确。如果正确，绿色的电源指示灯将变亮。

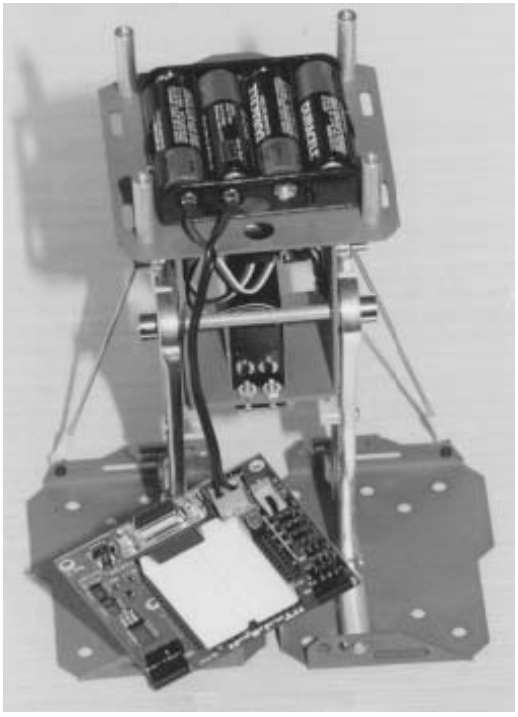


图1-22 安装4节电池

步骤 19：安装步行机器人电路板

需要以下配件：

- M3x8平头螺钉

如图1-23所示，在步行机器人的顶板上，用平头螺钉将机器人的电路板固定在4个支柱上。将步行机器人下方的伺服电机（踏步伺服电机）连接到电路板的P12处，而将上方的伺服电机（倾斜伺服电机）连接到电路板的P13处。注意黑色线对应“B”插头，红色线对应“R”插头。

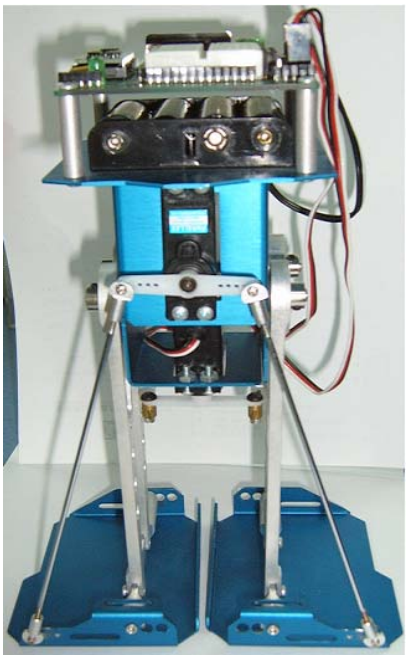


图1-23 安装完成的步行机器人

步骤 20：再次归中伺服电机并作调整

这是装配步行机器人最终的一步，重复前面步骤3—对伺服电机进行中位调整，调整后步行机器人的双脚应该均衡平坦地站在地面上。检查机器人的上部时，步行机器人的双腿应略向前倾斜。步行机器人的最佳行走状态是步行机器人的双脚略微向后倾斜1至2度而不是直立在地面上，以保证重心处在两足之间。

第二章：步行机器人的行走编程

简单控制及第一个程序

步行机器人有36种不同的动作，因此要实现步行机器人行走需要一些耐心。本章将教你学会如何编写几个程序让步行机器人向前走和向后走。

在掌握了步行机器人的前进和后退运动后，我们将在第三章介绍步行机器人转弯运动。在学习中，你将注意到连接机器人的后续动作必须考虑它的前一动作，例如，如果步行机器人左脚要往前移，需要它已经提起来。

当掌握这些基本要领后，你将学习如何存储你的运动及子运动到EEPROM中，以编写更有效的程序。在这一章中，我们进行的实验都是开环实验，也就是说没有反馈信号确定你的机器人是否走得过左或走得过右，或者是否碰到障碍物。

伺服控制基础

伺服电机是如何工作的

伺服电机被广泛运用于无线电遥控的汽车、轮船和飞机的运动方向及油门系统的控制。这些伺服电机被设计用来控制某一物体的特定位置，例如无线遥控飞机的方向舵。它们的控制范围一般从90度至270度，特别适应于要求价格低廉、精度较高、力矩较大的位置控制场合。我们利用一种信号来控制这些伺服电机的位置，我们称这种信号为脉冲序列，在下面我们马上将学习到这部分的内容。伺服电机内安装有一个机械限位装置，防止电机转动超出其设定的运动范围。伺服电机内还装有一个位置反馈装置，这样伺服电机内的控制电路才能知道在响应脉冲序列时转到哪。步行机器人使用的是未经修改的伺服电机，这种伺服电机能转过的最大角度大约是270度。

时间测量和电压

在这本学习指南中，我们将多次提到几个重要的时间单位秒（s），毫秒（ms），微秒（us）。通常，我们将秒用小写字母“s ”表示，所以，可以将1秒写成1s。毫秒则用“ms”表示，1毫秒等于1/1000秒；微秒用 “us” 表示，1微秒等于1/1000000秒，从毫秒、微秒与秒的关系你是否已经推出：1毫秒=1000微秒。在表2-1中用分式和科学计数法说明了秒、毫秒、微秒三者之间的关系。

表2-1：毫秒和微秒

$$1\,\text{ms} = \frac{1}{1000}\,\text{s} = 1\times10^{-3}\,\text{s}$$

$$1\,\mu\text{s} = \frac{1}{1{,}000{,}000}\,\text{s} = 1\times10^{-6}\,\text{s}$$

电压的单位是伏特，用大写字母V表示。步行机器人电路板上有几个插孔标着Vss,Vdd和Vin。Vss被称为系统地或称为参考电压。Vin通过一条导线与电池盒中的AA电池组正极连接，电压为6V。Vdd是一个5V的调整电压，它与Vss一起向步行机器人的电路板及面包板提供电源。

表2-2：电压与PCB上标识
Vss = 0 V (ground) Vdd = 5V (regulated) Vin = 6 V (unregulated)

从BASIC Stamp中发送出来的一组控制伺服电机的控制信号被称为“脉冲序列”，如图2-1所示。BASIC Stamp能够通过编程产生这样的信号波形，而且还能用它任意的一个I/O口进行信号的输出。在下面的例子中，BASIC Stamp向 P12（跨步伺服电机）和P13（倾斜伺服电机）各发送一个1500微秒的脉冲信号。在1500微秒的高电平送出后，BASIC Stamp继续发送一个25毫秒的低电平给该引脚，产生一组脉冲序列。

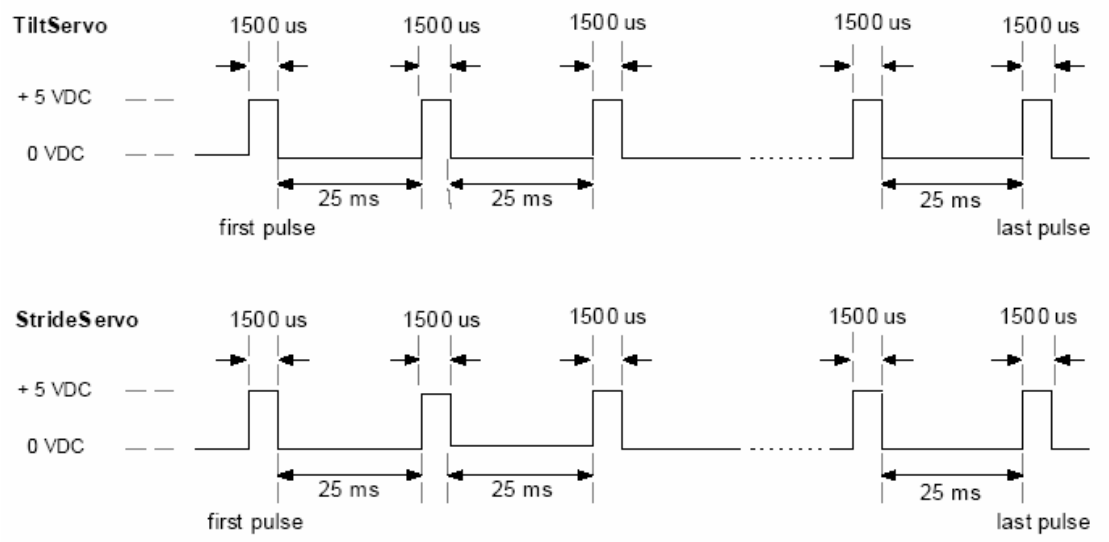


图2-1：调中脉冲序列示例

如图所示，这个脉冲序列由1500微秒的高电平和25毫秒的低电平组成。伺服电机的控制主要由1500微秒的高电平来控制，我们通常称这一段时间为脉宽。脉冲信号由低电平到高电平这一变化过程我们称为上升沿。同理，由高电平到低电平的变化我们称为下降沿。

我们这里所用的伺服电机，伺服脉冲之间的理想间隔时间为10—40毫秒。如果高于或者低于这个范围，将影响伺服电机的正常运行。

BASIC Stamp 2中的PULSOUT指令用来发送以2微秒为单位的高电平脉冲数。例如要向P13脚发送1500微秒的高电平信号，程序如：

PULSOUT 13，750　　在P13口输出1.5ms的脉冲

步行机器人的各种运动编程方式

编程不仅是一门科学还是一门艺术。通常有许多不同的编程方法都能达到同样的结果，有的程序更节约程序空间，而有的程序则执行起来更有效率。

在本章中，我们将看到步行机器人各式各样的运动，这其中包括向前走、向后走。我们

使用的步行机器人可以实现36种不同的运动。我们将利用这36种不同的运动来实现步行机器人的行走。36种运动方式都有一些关联，也就是说，无论哪个动作都有一系列可能的之前动作，或者有一系列可能的之后动作。

我们将通过许多不同的编程方式编写程序来实现步行机器人的运动。这里，我们将介绍三种不同的编程方式。在这三种编程方式中，最后一种编程方式使用起来非常的复杂，但它是更加的灵活，更加的容易掌握，所以被大多数人所接受、使用。有经验的程序员可以直接学习最后一种编程方式，但我们建议三种编程方式都是值得学习的。

方式 1：野蛮方法

这种方式应用了大量的子程序，每一个子程序对应着一个动作。通过调用和组合这些子程序来完成步行机器人复杂的运动。这种方式简明易懂，但一一列出36种运动方式极大浪费了宝贵的存储空间，更加不利于程序的修改。例如，要改变运动速度这一参数，就必须修改程序中所有的运动速度参数，极不方便。

方式 2：数据表格

这种方式通过找寻36种运动之间的共性，特别是一些具有相似功能的子程序，他们通过带入不同的参数实现不同的动作。把这些参数归纳到参数表中，是简化程序设计的一种手段。构造参数表与编写各子程序相比更简洁，因为数据表只包含参数。

方式 3：状态迁移

前面两种编程方式中，程序员必须记下步行机器人脚步的位置以及需要调用的例程，并且要在表格中填入适当的参数值。而状态迁移方式不同于前面的两种编程方式，因为步行机器人跟踪的双脚的运动轨迹。迁移动作用于从一个状态运动到另一个状态，基本动作是3种倾斜动作和3种跨步动作，这就明显少于其他方式中用到的36种基本运动。

步行机器人的运动原理

人类的步行十分自然，但实际的动作相当复杂。人将视觉和触觉感受到的复杂信息传回大脑，由复杂的大脑协调各个肌肉的运动来实现行走。步行机器人的行走是另一种极端。它仅有两部伺服电机，仅能够实现有限运动，而且基本上没有反馈信号。虽然步行机器人无法自己学会行走，但一旦你掌握基本方法，编程指挥它行走比较简单。

人类通常采用可控的前倾来实现行走。首先，身体略微向前倾斜，然后朝前迈出一只脚以防跌倒。人们就是通过不断重复这个过程来实现向前行走的，这一过程在跑步时更加明显。由此很容易明白人如果没有调整好行走的步伐就会跌倒在地面上。

人能象步行机器人一样向前行走，但是累了一些。这时是先把脚伸出去，再把身体拖过去，试一下。向前倾斜你的身体，走起来容易多了，但是这其实是防止你跌倒。

步行机器人通过倾斜和平衡运动，来实现行走的。这就需要一系列简单的运动作为运动基础。步行机器人基本上能实现双脚向任何一边倾斜，或者平稳的站在地面上。无论是左脚

还是右脚都能实现向前迈步，一只脚向前，一只脚向后，或者前后的反复运动。步行机器人有9个基本的脚步位置、从一个位置到另一个位置都有4种的过渡方式，共计36种迁移状态。

通过组合这些有限的运动，步行机器人能在平地上相对平稳的移动。由于步行机器人有限的运动方式，限制了步行机器人只有2种基本的运动：直线行走和原地旋转。利用这两种基本运动，仍然可以让步行机器人从A点移动到B点。

步行机器人的行走的分4个步骤：

a. 向一边倾斜
b. 移动被提起的那只腿
c. 向另一边倾斜
d. 移动被提起的那只腿

以上步骤实现一步行走。腿的运动方向决定了机器人的运动的行走方向，腿的移动距离决定了机器人的行走距离。步行机器人移动的速度，则主要取决于动作的执行频率，以及每次移动的距离。

假定步行机器人往一个方向没有倾斜太大，则机器人将保持平衡。这意味着这个过程中，另一边腿的运动过程能够在任意位置停止或者重新启动。这与人类的行走不同，人向前倾斜行走时，向前迈进的脚必须落到地面上才能防止摔倒。

在写程序控制步行机器人的运动中，动量扮演了一个很重要的角色。伺服电机精确的控制双脚的位置。通过对伺服电机的控制，能够快速或慢速的移动步行机器人的双腿，并能让它在移动中的任意位置停止运动。但是如果步行机器人的移动速度快，由于存在的较大的动量，步行机器人倾斜度将受到限制，否则很难保持平衡。如果提高双腿的移动速度，那么步行机器人双腿的动量也随之变大。要让步行机器人在指定的位置上停止，步行机器人的动量不能过大。过大的动量会使步行机器人跌倒。

人类通过膝盖和脊椎关节来实现转弯的，而步行机器人没有类似的结构，它只能通过原地旋转来实现转弯。步行机器人的双脚总是向前的，所以它不能通过转动脚来改变方向。步行机器人通过原地旋转实现转弯的，步行机器人从A点直线运动到B点，在B点原地旋转，转向C点的方向，然后直线运动至C点。

原地旋转也有4个步骤

a. 向一边倾斜
b. 移动被提起的那只腿
c. 两条腿同时着地
d. 向相反的方向移动双腿

这个过程靠摩擦力起作用。实际上，步行机器人的原地旋转是在最后一步（步行机器人的双脚同时在地面上）完成的。它基本的工作原理是一只脚向前运动时，另一只脚向后运动，通过双脚与地面间的摩擦里来实现原地旋转。转动的角度受腿的运动幅度和两脚与地面之间的摩擦系数的影响。过低的摩擦系数，导致机器人转动很小。过大的摩擦系数则会使步行机器人跌倒。

步行机器人限于在平坦的表面行走。地面的类型影响机器人与平面之间的摩擦力。步行机器人适合在木板、硬地毯和厨房的油毡地板上行走。致密地毯最适合步行机器人的行走。可以通过在步行机器人的脚板的底部加贴不同材质的胶带来增加步行机器人与地面间的摩擦系数，这里我们用胶带（电胶带，一片很小的紧固胶带）。关于摩擦系数的选择，没有一个硬性的规定。你可能需要通过调整程序来控制步行机器人适应某种地面的行走。例如，你不能设定两次原地旋转，就让步行机器人旋转的角度达到90度。

这里涉及到一个精度问题。步行机器人能够很好的进行移动，但是它的运动精度比轮式

机器人（宝贝车机器人）的运动精度差。如果步行机器人先向前走六步，再向后走六步，它几乎不可能回到原来出发前的位置。它可能是在原来位置的附近，但这个位置并不确定。增加一些转弯程序和并消除所有的意外，要步行机器人走一个正方形也几乎不可能。可以很容易地编写一个走正方形的程序，但因为步行机器人与地面之间的摩擦因数和机械运动精确度的影响，无法使步行机器人按编程设定路线准确行走。除非你给轮式机器人增加编码器，或者要解决迷宫问题，我们并不是很关心步行机器人的重复性精度的问题。

对于大多数的实验来说，精度并不重要。我们可以为步行机器人加装一个罗盘来追踪步行机器人方位，但是要追踪移动过的距离却是一个非常困难的任务。这个问题将不在本书讨论，但你可以在其他的地方查到许多相关资料。机器人，无论它通过轮子还是腿进行行走，都有一个非常具有挑战性的问题：“我的机器人现在哪里？”。

这里，步行机器人的双腿上缺少了关节连接件来越过障碍物，并且步行机器人无法在一个斜坡上行走。通过加装传感器，步行机器人能够避开障碍物。在后面的实验中，我们将使用一个步行机器人配套的红外线检测设备来识别障碍物。

任务 1：基本的步行运动：野蛮编程

步行机器人是一个行走的机器人，所以让机器人运动起来是学习编程的一个好起点。

图2-2显示了步行机器人向前行走几步的一个可能动作序列。

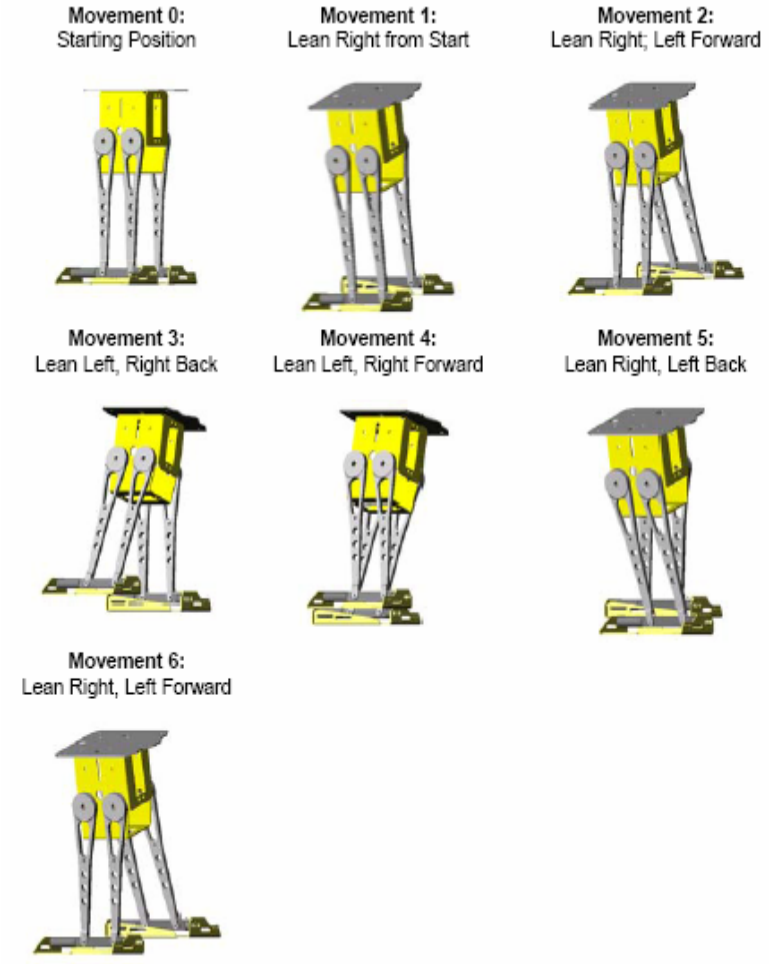


图2-2：前进运动的几个动作

一旦步行机器人完成运动0、1、2，重复运动3、4、5、6就能实现直线行走。接下来的几页中为你展示前三个运动的代码。运动1和2十分的相似，只是一个调整倾斜伺服电机另一个是跨步伺服电机。运动1、3、5使用了相同的代码，只是他们的参数值不同。同样，运动2，4，6使用相同的程序，参数值不同。

M0例程是特有的。无论双脚当前处在什么位置，M0都是让步行机器人在开始运动前，让双脚并列站在地面上。如果双脚运动前还没有在或者没有接近这个平衡位置，运行M0程序会导致一个跳动的动作。

例程M1、M2是程序中所有其它运动例程的典型样例。例程M1是使步行机器人向右倾斜，为此，需要同时向两台伺服电机发送脉冲控制指令。首先，发送相同的脉宽脉冲信号给跨步伺服电机，使跨步伺服电机保持静止，双脚没有向前或向后移动。然后，给倾斜伺服电机发送一个与M0种不同脉宽的脉冲信号，使倾斜伺服电机旋转，从而带动步行机器人的倾斜。

例程M2与M1相似，只是在M2中，倾斜伺服电机保持不动，而通过改变跨步伺服电机的脉宽值来使得跨步伺服电机转动，从而带动一条腿的向前，实现步行机器人的向前移动。

程序使用了许多定义为常量的伺服电机的脉宽限位值范围，而例程中的小段代码直接采用数值常量值。这两种编程方式虽然效果一样，但是如果要改变不同的参数进行实验时，采用定义的常量可以大大减少程序的修改量。

在下图的例子中，FOR...NEXT循环使用了一个STEP修改符。STEP修改符不是步行机器人的物理步长。在程序中它是一个步长，它用来发送一个脉冲增量给伺服电机来实现所要求的一系列指定的运动。

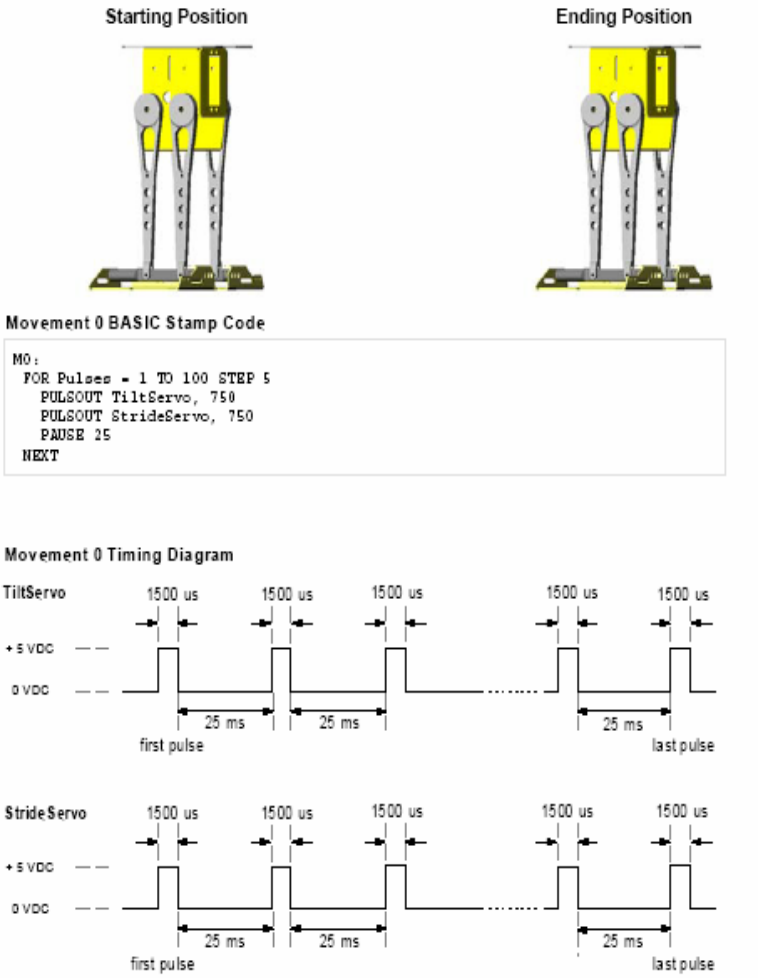


图2-3：动作0示例

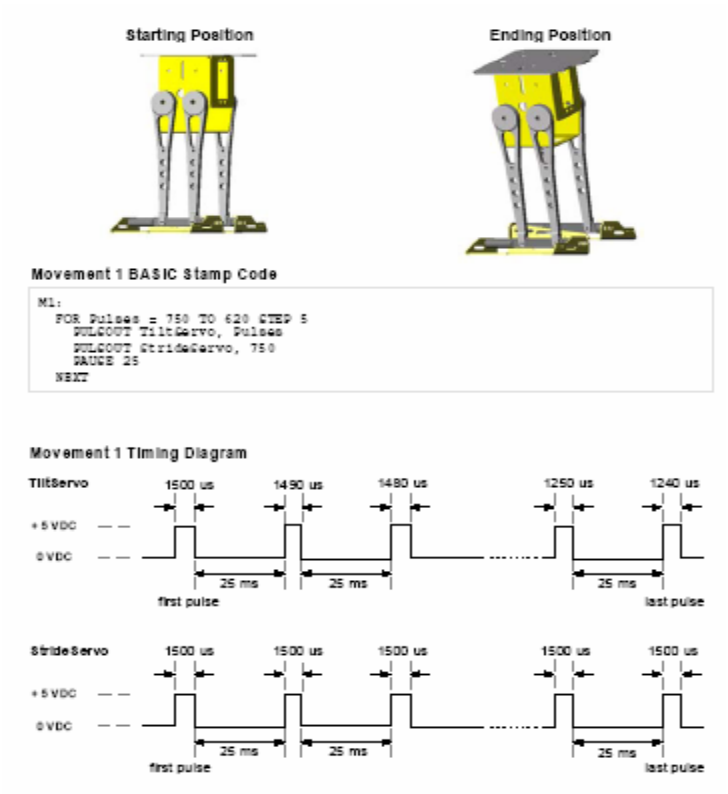


图2-4：动作1示例

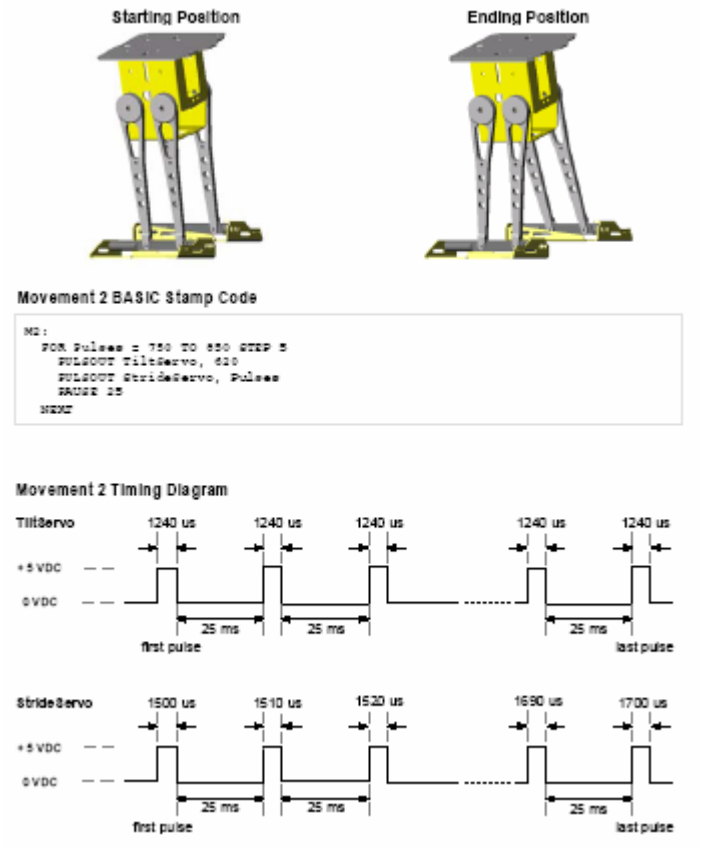


图2-5：动作2示例

下面是一个步行机器人向前行走的程序。你可以通过改变程序中的常量来改变步行机器人的行走速度，或步伐大小。在修改前要注意，过大的参数值，可能使步行机器人跌倒。

该程序在执行完后会使步行机器人归位，以使步行机器人的双脚并列平稳的站在地面上。这样，我们清楚了解步行机器人当前所处的位置，有利于执行其它程序。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 2.1 - First Steps Forward.bs2
' Run Movement Patters M0 to M8 to take several steps
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ Declarations ]-------------------------------------------------------
TiltStep CON 5 ' TiltServo step size
StrideStep CON 5 ' StrideServo step size
MoveDelay CON 25 ' in micrcoseconds
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightForward CON 650 ' Stride limits
StrideCenter CON 750
LeftForward CON 850
StrideServo CON 12 ' Stride servo on P12
TiltServo CON 13 ' Tilt servo on P13
MoveLoop VAR Nib ' Repeat movements
Pulses VAR Word ' Pulse variable
' -----[ Initialization ]-----------------------------------------------------
' -----[ Main Routine ]-------------------------------------------------------
' Take three full steps.
Main_Program:
GOSUB M0 ' center servos
GOSUB M1 ' tilt right
GOSUB M2 ' step left
FOR MoveLoop = 1 to 3
GOSUB M3 ' tilt left
GOSUB M4 ' step right
GOSUB M5 ' tilt right
GOSUB M6 ' step left
NEXT
GOSUB M3 ' tilt left
GOSUB M7 ' center feet
GOSUB M8 ' center servos
END
' -----[ Subroutines ]--------------------------------------------------------
M0:
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
```

```
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
RETURN
M1:
FOR Pulses = CenterTilt TO RightTilt STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
RETURN
M2:
FOR Pulses = StrideCenter TO LeftForward STEP StrideStep
PULSOUT TiltServo, RightTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
M3:
FOR Pulses = RightTilt TO LeftTilt STEP TiltStep
PULSOUT TiltServo,Pulses
PULSOUT StrideServo, LeftForward
PAUSE MoveDelay
NEXT
RETURN
M4:
FOR Pulses = LeftForward TO RightForward STEP StrideStep
PULSOUT TiltServo,LeftTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
M5:
FOR Pulses = LeftTilt TO RightTilt STEP TiltStep
PULSOUT TiltServo,Pulses
PULSOUT StrideServo, RightForward
PAUSE MoveDelay
NEXT
RETURN
M6:
FOR Pulses = RightForward TO LeftForward STEP StrideStep
PULSOUT TiltServo,RightTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
```

```
NEXT
RETURN
M7:
FOR Pulses = LeftForward TO StrideCenter STEP StrideStep
PULSOUT TiltServo,LeftTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
M8:
FOR Pulses = LeftTilt TO CenterTilt STEP TiltStep
PULSOUT TiltServo,Pulses
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
RETURN
```

注意在使用串口线下载程序时，应将步行机器人的电源开关拨至下载（“1”位）或者运行（“2”位）处。如果PC的串口线足够长，而且不影响步行机器人自由运动，串口线可以与机器人保持连接。此时你最好用手抓着串口线，减少对步行机器人稳定行走的影响。当你要断开串口线前，要先将步行机器人的电源关闭。此后，你只须打开电源开关，步行机器人就会根据下载到存储器上的程序自主的进行行走。你也可以先将步行机器人的电源开关拨至“1”位置，按下复位键，再将电源开关拨至“2”处，机器人同样会自动行走。

要让步行机器人更快行走，我们可以改变MoveDelay、TiltStep和StrideStep三个常量。减少MoveDelay值意味着发送给伺服电机的脉冲间隔将变小。增大TiltStep和StrideStep将使得伺服电机的脉冲宽度变大。

如果步行机器人不以双脚稳定的站在地面上的状态开始运动，或者你想让你的步行机器人走更大的步长，你可以修改CenterTilt和StrideCenter值，同时也需要修改两台伺服电机的左极限值和右极限值。

该你了

修改例程中的代码。尝试以下实验：

- 调整TiltStep和StrideStep到一个较小值，观察步行机器人的行为。调整相同常量到一个较大值，进行同样的观察。你能想象出伺服电机的时序图，并解释步行机器人不同行为的原因吗？
- 减小MoveDelay值，并观察步行机器人的行为。为你的步行机器人挑选适当的MoveDelay、TiltStep和StrideStep值，让步行机器人按照你的要求进行动作。记录这些参数值，为将来的程序设计做准备。

任务 2：向后行走：野蛮编程

与向前行走一样，步行机器人能够实现向后行走。但要注意的是，不仅仅只是将简单地前面程序中的步骤顺序颠倒过来，就能实现后退功能。步行机器人以一种相反模式运动，但实现此功能的函数不同。采用7个例程，最后的例子程序是相当简单的。修改该程序以便处理不同的运动方向并不是太困难。记住，我们在本章的后面还会介绍两种方法来完成这些任

务。

在前面的程序中，每一步的子程序都是按顺序编号。在这个程序中，步骤将会有略微的不同，所以我们需要使用不同的例程名。起步的运动与先前的向前行走的起步步骤相同，但第二步开始就不同了，我们将用到运动9，那对应M9例程。

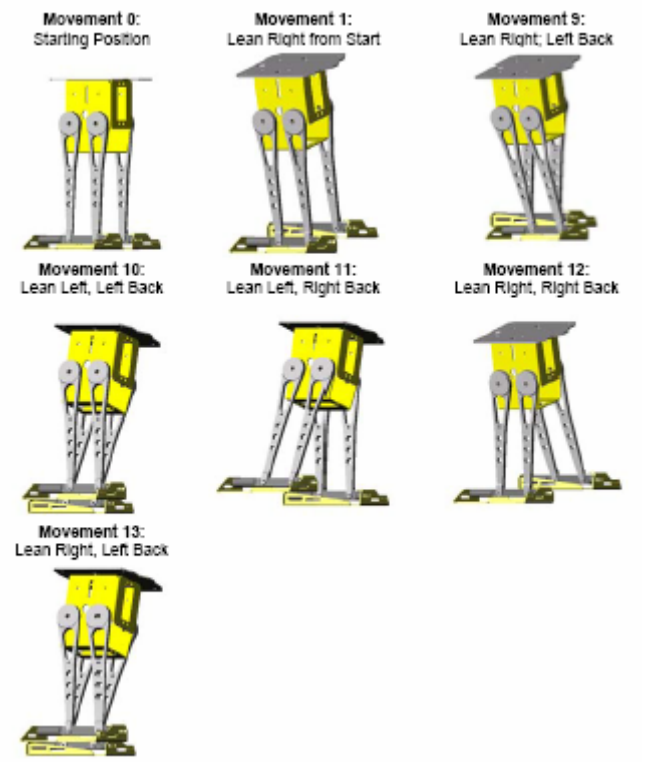


图2-6：后退的几个动作

与步行机器人向前行走的程序一样，步行机器人的起始步是运动0、1和9。通过运动10、11、12和13的重复实现步行机器人的向后直线行走。下面是一个例子程序。在程序中调整不同的常量值，使你的步行机器人走的更快、步长更长。这个程序同样是以两脚同时并立站在地面上这种状态作为步行机器人行走的结束。

注意该程序中与第一个例子程序中具有相同名称例程是直接从第一个程序中抽取过来的。采用这种方法编写的程序如果需要更复杂的运动，将需要更多的例程，甚至用到所有36种例程。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 2.2 - Walking Backwards.bs2
' Run Movement Patters M9 to M13 to walk backwards
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ Declarations ]-------------------------------------------------------
TiltStep CON 5 ' TiltServo step size
StrideStep CON 5 ' StrideServo step size
MoveDelay CON 20 ' in micrcoseconds
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
```

```
RightForward CON 650 ' Stride limits
StrideCenter CON 750
LeftForward CON 850
StrideServo CON 12 ' Stride servo on P12
TiltServo CON 13 ' Tilt servo on P13
MoveLoop VAR Nib ' Repeat movements
Pulses VAR Word ' Pulse variable
' -----[ Initialization ]----------------------------------------------------
' -----[ Main Routine ]-------------------------------------------------------
Main_Program:
GOSUB M0 ' center servos
GOSUB M1 ' tilt right
GOSUB M9 ' step back
FOR MoveLoop = 1 to 3
GOSUB M10 ' tilt left
GOSUB M11 ' step left
GOSUB M12 ' tilt right
GOSUB M13 ' step right
NEXT
GOSUB M10 ' tilt left
GOSUB M14 ' center feet
GOSUB M8 ' center servos
END
' -----[ Subroutines ]--------------------------------------------------------
M0:
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
RETURN
M1:
FOR Pulses = CenterTilt TO RightTilt STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
RETURN
M8:
FOR Pulses = LeftTilt TO CenterTilt STEP TiltStep
PULSOUT TiltServo,Pulses
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
```

```
RETURN
M9:
FOR Pulses = StrideCenter TO RightForward STEP StrideStep
PULSOUT TiltServo, RightTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
M10:
FOR Pulses = RightTilt TO LeftTilt STEP TiltStep
PULSOUT TiltServo,Pulses
PULSOUT StrideServo, RightForward
PAUSE MoveDelay
NEXT
RETURN
M11:
FOR Pulses = RightForward TO LeftForward STEP StrideStep
PULSOUT TiltServo,LeftTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
M12:
FOR Pulses = LeftTilt TO RightTilt STEP TiltStep
PULSOUT TiltServo,Pulses
PULSOUT StrideServo, LeftForward
PAUSE MoveDelay
NEXT
RETURN
M13:
FOR Pulses = LeftForward TO RightForward STEP StrideStep
PULSOUT TiltServo,RightTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
M14:
FOR Pulses = RightForward TO StrideCenter STEP StrideStep
PULSOUT TiltServo,LeftTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
RETURN
```

该你了

尝试以下实验：

- 改变程序中的常量为以下各数值并实验。说明步行机器人的运动为什么与先前的运动不同。

```
TiltStep CON 2 ' TiltServo step size
StrideStep CON 2 ' StrideServo step size
MoveDelay CON 12 ' in micrcoseconds
```

- 写一个程序，实现步行机器人向前走，然后向后回到原地。

任务 3：采用数据表存储动作进行编程

之前的两个程序长度是相似的，但是更复杂的程序需要更多的运动例程，这就使得程序长度增加更快。一个有效的简化编程任务的方法是将运动信息移到一个表格中。下面的样例就既使用了固定大小的数据表，也使用了变长度的数据表。

BASIC Stamp的数据指令（DATA）用来存储表格数据。这个指令在下载程序时让你将数据写到Stamp的EEPROM存储器中。如果使用恰当，这个数据表格将使你的PBASIC程序更加的高效，更加的简短。

注意：

DATA命令在BASIC Stamp编程手册中附有许多范例，可从www.depush.com上下载到。基本要点如下：

- DATA能够将数据以byte(8位)或word(16位)的形式写入BASIC Stamp2EEPROM存储器中，从地址0—2048顺序存储。而你编写的PBASIC程序则倒序存储在地址2048到地址0上的。
- 当用DATA命令致使数据覆盖程序代码时，编辑软件不会检测到它的发生。
- 一旦下载并运行步行机器人的程序2.3——First Steps Forward Using Tables.bs2，你可以选择Run｜Memory Map来查看数据和程序是如何存储在EEPROM中。

这个例子采用了两个数据表来开发步行机器人的行走运动。第一个表是定义运动的动作列表。例如，从之前的两个程序可以知道，M1动作后接着M2动作将使步行机器人开始行走。而动作M2之后通过重复3、4、5和6动作就可实现步行机器人的直线行走。

```
StartForward DATA 1, 2, 0
WalkForward DATA 3, 4, 5, 6, 0
FinishForward DATA 3, 7, 8, 0
```

在BASIC Stamp的EEPROM中，一个字节大小的数值1存储在地址0，数值2存储在地址1，数值0存储在地址3，数值3存储地址4中，如此类推。

这是一个可变长度数据表，被用于存储一系列动作。这使你可以定义任意长度的复杂运动。通过PBASIC的READ指令读取表中的数据。

第二个表是定义动作（Movement）的数据表，表中包括了控制步行机器人倾斜伺服电机和跨步伺服电机运动所用到的数据值。每一个动作都各有一个倾斜伺服电机的开始值和结束

值，以及跨步伺服电机的开始值和结束值。在这个数据表中，M1实际上就是存储CenterTilt地址。因为前面的三行代码已经占用了12个字节的空间，一个字长的CenterTilt常量将被存储在EEPROM的地址12和地址13中（EEPROM的起始地址是0，不是1）。

因为该表中的常量都大于255这个字节（Byte）定义的最大值，所以用WORD（16位字）修正符定义常量。

```
M1      DATA   WORD CenterTilt, WORD RightTilt,
               WORD StrideCenter, WORD StrideCenter
M2      DATA   WORD RightTilt, WORD RightTilt,
               WORD StrideCenter, WORD LeftForward
M3      DATA   WORD RightTilt, WORD LeftTilt,
               WORD LeftForward, WORD LeftForward
M4      DATA   WORD LeftTilt, WORD LeftTilt,
               WORD LeftForward, WORD RightForward
M5      DATA   WORD LeftTilt, WORD RightTilt,
               WORD RightForward, WORD RightForward
M6      DATA   WORD RightTilt, WORD RightTilt,
               WORD RightForward, WORD LeftForward
M7      DATA   WORD LeftTilt, WORD LeftTilt,
               WORD LeftForward, WORD StrideCenter
M8      DATA   WORD LeftTilt, WORD CenterTilt,
               WORD StrideCenter, WORD StrideCenter
```

该程序中只用到了8种动作方式，但我们可以很容易地将所有36种动作添加到表格中去。与第一种野蛮编程法相比，将动作用数据表存储在EEPROM中的编程方式大大减少了用于存放代码的程序空间。

程序中主程序Main_Program用来完成函数调用管理，子程序Movement完成动作过程Main_Program中，首先调用M0函数，对步行机器人的起始位置进行调整，使两台伺服电机停留在中心位置，步行机器人的双脚平稳的站在地面上。

接下来，将StartForward的值调入Mx中，这实际上就是数据表格中的地址0存储的数据。子程序Movement中，通过READ指令读取EEPROM中Mx数据表中数值，并存入Dx变量中，Dx现在的值为1。Mx则增加1以便为下一次循环做准备。

第一次运行LOOPUP的索引为1，将M1的地址值存储到Dx中。然后将地址Dx中的四个值读取出来，并赋值给TiltStart, TiltEnd, StrideStart和 StrideEnd四个变量，通过PULSOUT指令使步行机器人动作。

代码行：

```
IF TiltStart = TiltEnd THEN MovementStride
```

是检测步行机器人是不是不倾斜就让步行机器人的一只脚向前移动。在这种情况下，步行机器人保持前面的TiltStart值不变使倾斜电机稳定，只让跨步发生。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 2.3: First Steps Forward Using Tables
' Movement routines stored in EEPROM using DATA statement
' {$STAMP BS2}
```

```
' {$PBASIC 2.5}
' -----[ I/O Definitions ]-------------------------------------------------
' Connect Vs1 and Vs2 to Vss to ground both servos
StrideServo CON 12 ' Stride servo
TiltServo CON 13 ' Tilt servo
' -----[ Constants ]-------------------------------------------------------
TiltStep CON 10 ' TiltServo step size
StrideStep CON 10 ' StrideServo step size
MoveDelay CON 14 ' Servo pause (ms)
RightTilt CON 625 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 875
RightForward CON 600 ' Stride limits
StrideCenter CON 750
LeftForward CON 900
' -----[ Variables ]-------------------------------------------------------
MoveLoop VAR Nib ' Loop for repeat movements
Pulses VAR Word ' Pulse variable
Dx VAR Pulses ' Stores Mx movement index
Mx VAR Word ' Movement index
TiltStart VAR Word ' Start tilt value
TiltEnd VAR Word ' End tilt value
StrideStart VAR Word ' Start stride value
StrideEnd VAR Word ' End stride value
' -----[ EEPROM Data ]-----------------------------------------------------
' Indicates which movement routines are executed to comprise steps
' Extend WalkForward with 3, 4, 5, 6, for longer walks
' The number 0 marks the end of a list
StartForward DATA 1, 2, 0
WalkForward DATA 3, 4, 5, 6,3, 4, 5, 6,3, 4, 5, 6,3, 4, 5, 6,3, 4, 5,
6, 0
FinishForward DATA 3, 7, 8, 0
' Movement routines
M1 DATA WORD CenterTilt, WORD RightTilt,
WORD StrideCenter, WORD StrideCenter
M2 DATA WORD RightTilt, WORD RightTilt,
WORD StrideCenter, WORD LeftForward
M3 DATA WORD RightTilt, WORD LeftTilt,
WORD LeftForward, WORD LeftForward
M4 DATA WORD LeftTilt, WORD LeftTilt,
WORD LeftForward, WORD RightForward
M5 DATA WORD LeftTilt, WORD RightTilt,
WORD RightForward, WORD RightForward
M6 DATA WORD RightTilt, WORD RightTilt,
```

```
WORD RightForward, WORD LeftForward
M7 DATA WORD LeftTilt, WORD LeftTilt,
WORD LeftForward, WORD StrideCenter
M8 DATA WORD LeftTilt, WORD CenterTilt,
WORD StrideCenter, WORD StrideCenter
' -----[ Main Routine ]------------------------------------------------------
Main_Program:
GOSUB M0 ' Center servos
Mx = StartForward
'DEBUG ? Mx
GOSUB Movement
FOR MoveLoop = 1 to 3
Mx = WalkForward
GOSUB Movement
NEXT
Mx = FinishForward
GOSUB Movement
END
' -----[ Subroutines ]---------------------------------------------------------
Movement:
READ Mx, Dx ' Read state table number
Mx = Mx + 1
'DEBUG "Movement routine = ", DEC Mx,cr
IF Dx = 0 THEN DoReturn ' Skip if no more states
LOOKUP Dx,[M1, M1, M2, M3, M4, M5, M6, M7, M8],Dx
READ Dx, WORD TiltStart, WORD TiltEnd,
WORD StrideStart, WORD StrideEnd' Read table entry; store
IF TiltStart = TiltEnd THEN MovementStride
FOR Pulses = TiltStart TO TiltEnd STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, StrideStart
PAUSE MoveDelay
NEXT
GOTO Movement
MovementStride:
FOR Pulses = StrideStart TO StrideEnd STEP StrideStep
PULSOUT TiltServo, TiltStart
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
GOTO Movement
' ----- M0: Move feet to initial center position -----
M0:
FOR Pulses = 1 TO 100 STEP StrideStep
```

```
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, StrideCenter
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

在该程序中增加向后走的代码比较容易，只需要做3处改动：首先必须在固定尺寸表（即第二张表）中增加额外的动作参数表，并将表的索引增加到LOOKUP指令中，其次必须在动作列表中增加一个向后行走的可变长表；最后，在子程序Movement中增加表格的名字及调用表格的指令。显然，这种编程方式比起在程序中加入许多子程序的编程方式要好得多。

该你了

尝试以下实验：

- 修改TiltStep, StrideStep 和 MoveDelay的值，使步行机器人行走尽可能平滑，但并不需要很快。
- 修改WalkForward数据表，增加步行机器人行走的步数。

任务 4：使用状态迁移描述动作进行编程

现在我们要用一种完全不同的方式来控制步行机器人的行走运动。前面的两种方法编程要求程序将一系列的运动控制的子程序通过适当的顺序连接起来。在这种情况下，编程人员必须知道步行机器人之前所处的状态，以及哪些与它相关的动作可以被使用。这种编程方式用来完成一些基本的运动较为简单，但当运动变得复杂时，编程就会变得非常麻烦。

原有的编程方式必须知道从运动起始状态到运动结束状态的整个过程，而状态迁移编程方法则只需要跟踪当前状态，并据此采取动作。因为步行机器人仅有两种真正意义的动作：倾斜和跨步，以及三种逻辑方向，从而构成总共9种状态，所以状态迁移方式可以大大简化编程过程。图2-7显示这些状态。

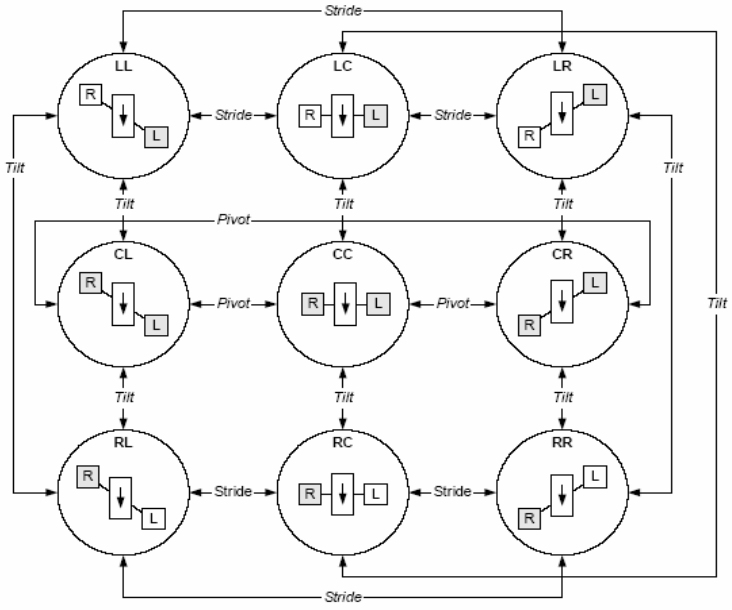


图2-7：状态迁移方法

图中每一个圆对应一种状态。圆中心的方框代表了步行机器人的腿和身体的姿态。方框中的箭头则标明了步行机器人的前进的方向。黑色背景的矩形框表示步行机器人相应位置的腿与地面接触。而白色背景的矩形框则表示相应步行机器人的腿悬空。圆形顶部的两个字母提供了步行机器人定位的表示方式。第一个字母表示倾斜伺服，第二个字母表示跨步或哪只脚在前面。字母可以是L，C或者R。其中C表示伺服电机处于中间位置。以倾斜为例，C表示步行机器人的双脚均站在地面上。对跨步而言，字母C则表示机器人两只脚处在过机器人对称中心轴线的垂线上。下面我们将会明白这种命名的原因。

图中的双向箭头表示从一种状态到另一种状态的有效迁移。箭头上的标志表示状态改变所需的动作。图中的Pivot与Stride（跨步）意思相同，因为他引起了步行机器人原地转动，所以我们用Pivot来表示。

九种状态，每一种都有4种可能迁移，于是产生36种不同的迁移。如果我们用步行机器人的起始状态和结束状态来命名一个迁移动作，并用格式TSxTS表示，那么前面定义的M0动作就对应CCxRC，表示机器人从双脚均衡的站在地面上运动到向右倾斜（右脚站立）。利用这种命名方法，可以将本章第一个例子程序代码简化如下。

```
 ' -----[ Main Code ]----------------------------------------------------
'
' Take three full steps.
Main_Program:
      GOSUB M0 ' center servos
      GOSUB CCxRC ' tilt right
      GOSUB RCxRL ' step left
      FOR MoveLoop = 1 to 3
          GOSUB RLxLL ' tilt left
          GOSUB LLxLR ' step right
          GOSUB LRxRR ' tilt right
          GOSUB RRxRL ' step left
      NEXT
      GOSUB RLxLL ' tilt left
      GOSUB LLxLC ' center feet
      GOSUB LCxCC ' center servos
END
```

这种命名方式改变带来的优点是显而易见的。前一动作的最后两个字母是下一个动作开始的两个字母，即前一个动作的最后的状态是下一个动作的起始状态。

当然，这样做仍然比较繁琐，而且如果没有明确的注释，动作很容易混淆。问题依然这种编程方法仍然需要知道每一个动作之前的状态。

现在我们来看下面的例子代码程序。这个程序去掉了注释（这不是一个好主意，仅是为了讨论方便），被调用的过程函数表明了程序所要执行的动作。

```
 ' -----[ Main Code ]----------------------------------------------------
'
' Take three full steps.
Main_Program:
      GOSUB M0 ' center servos
      GOSUB TiltRight
```

```
        GOSUB StrideLeft
        FOR MoveLoop = 1 to 3
            GOSUB TiltLeft
            GOSUB StrideRight
            GOSUB TiltRight
            GOSUB StrideLeft
        NEXT
        GOSUB TiltLeft
        GOSUB StrideCenter
        GOSUB TiltCenter
END
```

这个程序是如何运行的呢？首先，我们需要看一下下面的程序中，这些子程序是如何实现的，其中增加了有两个基本的变量，CurrentTilt和CurrentStride，来跟踪步行机器人双脚的位置状态。

这里只用到了6个子程序。虽然只有其中的4个子程序能在任意时刻改变状态，但它们可以以任意顺序调用。

这是为什么呢？因为6个子程序中的2个并不改变机器人的状态。例如，如果步行机器人向左倾斜站立，则调用TiltLeft子程序将仍旧使步行机器人停留在原来的位置。执行该子程序会有一个延时，但该延时通常很短，在观察机器人的运动时很难感觉到。

下面的程序中实现了前面代码段中用到的子函数，但也保留了在程序2.3中所用到一些思想。虽然这里使用了一些略微不同的编程风格，但这种小的修改还是值得。这个程序基本综合了之前两个程序所用到的编程技巧，既让程序员从数据表中读取一系列的动作进行连接，也从EEPROM中读取命令。在你修改这个程序时，你就会注意到程序是如何运用状态迁移图的。

看表2-3有利于你对不同的DATA代码意义的理解。该表有7个数据，其中第7个数据“XX”指示动作列表的结束。

表2-3：状态表码

TR	右倾
SL	左跨步
TL	左倾
SR	右跨步
TC	不倾斜
Xx	结束表

```
' -----[ Title ]-----------------------------------------------------------
' Toddler Program 2.4: First Steps Forward with State Transitions
' Keeps track of starting and ending states
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]--------------------------------------------------
TiltServo CON 13 ' Tilt servo on P12
StrideServo CON 12 ' Stride servo on P13
' -----[ Constants ]--------------------------------------------------------
MoveDelay CON 12 ' in micrcoseconds
```

```
StrideStep CON 5 ' StrideServo step size
TiltStep CON 5 ' TiltServo step size
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
' -----[ Variables ]-------------------------------------------------------
MoveLoop VAR Nib ' Loop for repeat movements
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
' -----[ EEPROM Data ]-----------------------------------------------------
' Take three full steps.
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
```

Chapter #2: Taking Your First Steps · Page 61

```
SC CON 4
SR CON 5
xx CON 255
WalkForward DATA TR, SL, TL, SR, xx
WalkBackward DATA TR, SR, TL, SL, xx
TurnLeft DATA TL, SR, TC, SL, xx
FinishForward DATA TR, SC, TC, xx
' -----[ Main Routine ]---------------------------------------------------
'
Main_Program:
GOSUB ResetCC
FOR MoveLoop = 1 to 3
Mx = WalkForward
GOSUB Movement
NEXT
FOR MoveLoop = 1 to 3
Mx = TurnLeft
```

```
GOSUB Movement
NEXT
FOR MoveLoop = 1 to 3
Mx = WalkBackward
GOSUB Movement
NEXT
Mx = FinishForward
GOSUB Movement
END
' -----[ Subroutines ]------------------------------------------------------
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = table index, table ends in xx
Movement:
READ Mx, Dx ' read next action
Mx = Mx + 1
IF Dx = xx THEN MovementDone ' skip if end of list
GOSUB DoMovement ' execute movement
GOTO Movement ' loop until done
DoMovement:
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
```



```
StrideCenter,StrideRight]
' fall through if invalid
' index
MovementDone:
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
```

```
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
Chapter #2: Taking Your First Steps · Page 63
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

噢，上当了！程序中还有一个地方是需要在下一章中进行详细介绍的内容，这里只是做了一个替换。之所以在这里包含原地旋转动作，原因是该动作没有用单独的子程序实现。取而代之的是用一个通用动作子程序来处理所有的动作。原地旋转动作被定义在TurnLeft表中。

变量表的处理也被修改成允许象TiltLeft的6个动作函数访问。每一个函数都实现为一个子程序，通过一个RETURN指令返回。因此在程序中使用GOSUB调用DoMovement子程序，而不是在循环中加入BRANCH语句。

还要注意LOOKUP语句的细节。命名为XX的常量用于指示可变长度数据表的结束，其值255超过了需要用到值的范围，因为表中第一个值是0。

这种编程方法将是本书中后续介绍的程序所采用的编程方法之一。拓展象WalkForward这种表相当容易，因为它只有6个有效的参数值。还有一种可能就是采用4个位来存储每一个数值，以节约存储空间。但是由于PBASIC的限制，将使数据表的定义和读取非常困难。不过，当程序代码空间受到约束时，这仍是一种可选方案。

该你了

- 运行上述例子程序

挑战课题

- 提高步行机器人的行走速度，并确定步行机器人能正常行走的最高速度。
- 让步行机器人在不同材质的表面上行走，如地毯，木板和瓷砖，观察机器人的性能是否一样。
- 步行机器人启动时可以先移动左脚或者右脚。尝试将左脚起步的程序改写成右脚起步的程序。
- 增加一系列更复杂的步骤，例如使步行机器人的脚在空中来回运动几个来回等，让步行机器人看起来象在跳舞。

第三章 转弯

滑动式转弯

步行机器人有点不大灵活。它只能向前或向后移动它的步子，不能相对它的身体旋转步子。但这并没有阻碍他的转弯能力，当走直线时，步行机器人和人走直线是相似的，而转弯就明显不同。步行机器人转弯和人最接近的是就象人穿着平底鞋站在冰面上转弯的情形。

人站在冰上右转弯的过程是相当简单的，把你的左脚向前迈一步放在地面上，然后把左脚向后滑行回收，这样你就可以以右脚为轴向右转了。如果冰是湿滑的话可能要多次转动才能转过 90 度。同样要左转就滑行右脚。

标准的步行机器人在冰上转弯的效果并不理想，但利用这种原理，在其它表面上可以转得很好。步行机器人的平滑金属脚提供了一个光滑的表面。当它站在能提供摩擦力的表面上来做转弯，效果会更加好。如果走的平面太光滑，那么对步行机器人的脚掌进行一些处理以便提供更大的摩擦力。最典型的做法就是在它的脚掌底下粘贴上一层胶带，不用覆盖整个脚掌只要有足够的摩擦力来转弯就可以了，所贴的胶带不该对走直线造成影响。

任务 1：做一个转弯的动作

步行机器人转弯是靠两脚平放在地面然后朝相反的方向滑行来实现的。仅仅朝相反方向移动双脚来实现转弯多少有些不尽如人意，因为有些作用力的因方向相反而抵消了。此时机器人的动作与其说是转弯还不如说是旋转。

步行机器人一次只能旋转一个较小的角度，他可能要用 5 到 10 次的运动来完成一个 90 度的转弯（除非你在它的脚掌底下粘贴胶带），10 到 20 次来转 180 度。一个简单的转弯过程是由如图 3-1 所示的四个动作构成的。左转和右转的原理是一样的，除了倾斜和脚的运动是相反的。

从图中我们可以看到前三步的运动是要把左脚迈出去，而最后一步才实现了转弯。下面的程序多次执行这个过程。它可以实现一个右转或左转。实现右转关键就是在程序 2.4 中增加了 TurnRight 程序条目。下一节里面增加了另外两个在此程序中出现的条目那就是 WideTurnLeft 和 PiotRight。

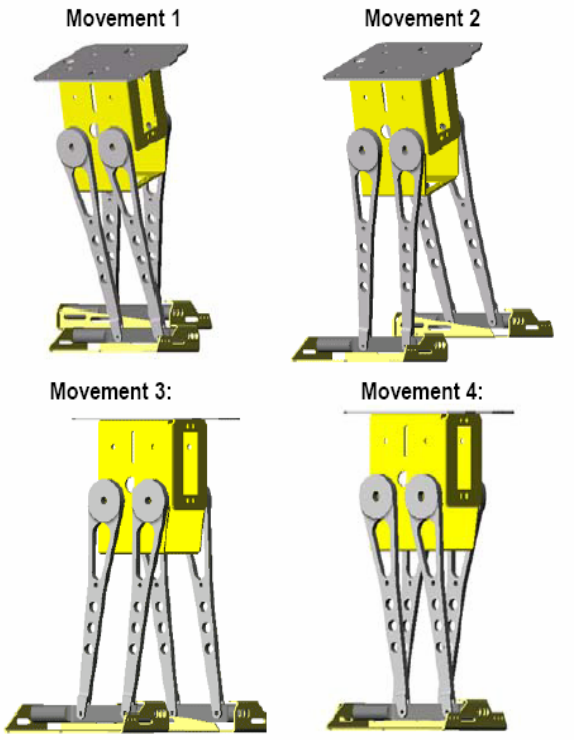


图 3-1 完整的右转弯行走的四个基本动作

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 3.1: Turning
' Demonstrates basic turning process which requires four movements
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]----------------------------------------------------
TiltServo CON 13 ' Tilt servo on P12
StrideServo CON 12 ' Stride servo on P13
' -----[ Constants ]----------------------------------------------------------
MoveDelay CON 15 ' in micrcoseconds
TiltStep CON 5 ' TiltServo step size
StrideStep CON 5 ' StrideServo step size
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
' -----[ Variables ]----------------------------------------------------------
MoveLoop VAR Nib ' Loop for repeat movements
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
' -----[ EEPROM Data ]--------------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255
WalkForward DATA TR, SL, TL, SR, xx
WalkBackward DATA TR, SR, TL, SL, xx
TurnLeft DATA TL, SR, TC, SL, xx
WideTurnLeft DATA TL, SR, TC, SL, TR, SL, TL, SR, xx
TurnRight DATA TR, SL, TC, SR, xx
PivotRight DATA TR, SL, TC, SR, TL, SL, TC, SR, xx
```

```
FinishForward DATA TR, SC, TC, xx
' -----[ Main Routine ]----------------------------------------------------
Main_Program:
GOSUB ResetCC
FOR MoveLoop = 1 to 5
Mx = TurnRight
GOSUB Movement
NEXT
FOR MoveLoop = 1 to 5
Mx = TurnLeft
GOSUB Movement
NEXT
FOR MoveLoop = 1 to 5
Mx = PivotRight
GOSUB Movement
NEXT
FOR MoveLoop = 1 to 5
Mx = WideTurnLeft
GOSUB Movement
NEXT
Mx = FinishForward
GOSUB Movement
END
' -----[ Subroutines ]-------------------------------------------------------
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = table index, table ends in xx
Movement:
READ Mx, Dx ' read next action
Mx = Mx + 1
IF Dx = xx THEN MovementDone ' skip if end of list
GOSUB DoMovement ' execute movement
GOTO Movement ' loop until done
DoMovement:
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
StrideCenter,StrideRight]
' will fall through if
' invalid index
MovementDone:
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
```

```
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

该你了！

- 运行程序
- 修改程序，让步行机器人先左转再右转。

任务 2： 不同的转弯

虽然基本的转弯动作能够让步行机器人到达它想要到的地方，但是在实现这个目的的过程中有很多的情况。例如当在一个狭窄的空间里转弯时原地转弯将很有必要。PiotRight程序段很清晰地说了这个是怎样完成的。在这种情况下，绕中心轴转是通过向前移动腿完成一个转弯来实现。紧跟着执行一个同样的动作，只是它是先后退一步。这样就产生了两个转向的动作，使步行机器人基本上是在原地转到了一个新的方向。

在此程序中另外一个子程序是WideTurnLeft。这部分程序让步行机器人在一个更大的转弯半径内实现转弯。 这种方式下每个转动动作后机器人都向前走了一步。仔细观察就会发现，步行机器人转弯的路径并不是一个完全的圆弧，而是一条有着圆角的类似多边形的曲线。不过它已经很接近圆弧了，人们通常认为它就走的是圆。

在程序3.1上做的简单的改进对于实现更好地转弯的意义是显而易见的。除了增加了两个子程序并调用它们，程序3.1和程序2.4几乎一样。

该你了！

- 修改程序3.1以实现不同的类型的转弯。

挑战课题！

- 示例程序中只给出了一部分类型的转弯。增加一些子程序入口来完成示例程序中没有包括的转弯。
- 步行机器人在结构和运动上都是对称的。它可以转弯然后前进或后退。编写程序可以完成程序 3.1 中的动作但是让机器人转弯后后退。
- 满步动作运用在示例程序中。如果把动作幅度变小些将会怎样呢。例如，试着从 CR 到 CC 动作而不是 CR 到 CL 。
- **WideTurnLeft** 让步行机器人朝左转，但是转弯的半径比 TurnLeft 大。现在让机器人的转弯半径更大。提示：可以从两个方面去考虑，一个与前进动作有关，另一个就是和转动的动作有关。

第四章 协调行走

实现各种运动的多重表

行走和转弯是步行机器人的基本动作，但仅有它们不能让机器人走得很远。简单地使用多个GOSUB语句来连接许多动作虽然可以实现更远的行走，但这让程序变得很冗长， 也不如本章实验里采用的方法有效。

第三章中例子程序使用了表来存储一系列基本动作。更复杂的运动可以用更长的表来实现，但一种更好的办法就是从更高层次来使用这些表。这种动作表不再指定步行机器人是向左倾斜还是向右倾斜，取而代之的是包括一系列诸如右转、后退10步、原地左转、前进10步等动作。

本章的第二个任务就是利用这种方法让步行机器人走比以前章节中更复杂的路线。但我们首先要看看如何确定一个表是是哪个命令集的一部分，这样就可以让Movement子程序确定一表是一个基本动作命令集，还是包含更复杂的动作命令集。这样Moevment子程序才能据此处理这些命令。

任务 1：哪个表？

步行机器人使用高级动作集比使用低级动作集更有优势，比如用执行左转弯命令替代先左倾斜然后让右脚向前迈，当然要来得简单。二者在实际运用中都需要，前面的例子已经告诉我们如何用低层一些的动作组成一个表来实现高层动作。要将这种方法提升到一个新的层次需要用到一个不同的数据表集，该表中的数据元素存储指向低层表的参考索引.

使用两种类型的表可能要用到两个不同的子程序，但只用一个子程序访问两类表有一个优势，它允许程序在其中自由地混合使用这两种类型的表。本任务程序就说明如何实现这一功能。

该程序并没有让步行机器人行走，但确实用BS微控制器执行了程序。它利用程序编辑器调试终端显示程序中由DEBUG语句产生的输出。大多数熟悉BASIC Stamp的编程者都已了解DEBUG语句，如果你不了解的话请查阅BASIC Stamp使用手册。同时注意在实验时让串行通讯线连接到步行机器人。

该程序假定两种表都将在程序中用到，而且每种表都将存储在他们各自的程序空间。不要求两种表一定要相邻，只要求它们必须分别在由BasicMovements常量声明定义的边界地址上方或下方。

	Advanced Movement Tables	高级动作表位于BasicMovements常量之前
BasicMovements	Basic Movement Tables	基本动作表位于BasicMovements常量之后
	Main program	

表 4-1 协调行走表的结构

程序使用了一个Movement子函数，该函数有一个存储在Mx变量中表素引。它利用DEBUG语句在PC机上的调试终端输出字符串，以指示该索引是高级动作还是基本动作。下一个任务中，Movement函数将修改为实际对表进行解释，让步行机器人运动。

理解数据是怎样被存储在BASIC控制器的EEPROM中非常有用。如果你选择Run-〉Memory Map菜单，你就可以看到数据从EEPROM地址0开始朝地址2048存储，而PBASIC程序则从位置2048向位置0方向存储，如图4-1所示。

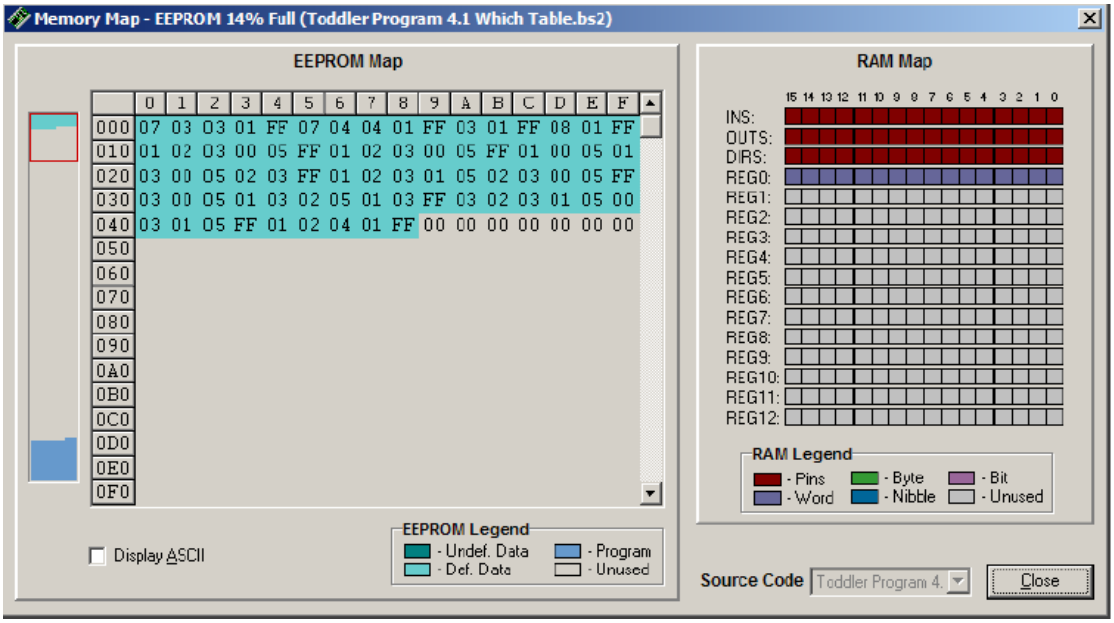


图4-1 EEPROM的内存映射

注意DATA语句将数据从位置0开始存储，PBASIC程序则从位置2048朝位置0存储，向下移动“Condensed EEPROM Map”中的红色方框来改变“Detailed EEPROM Map”显示区域，你可以看到PBASIC程序的存储情况。图4-2给出了程序执行时的DEBUG输出。

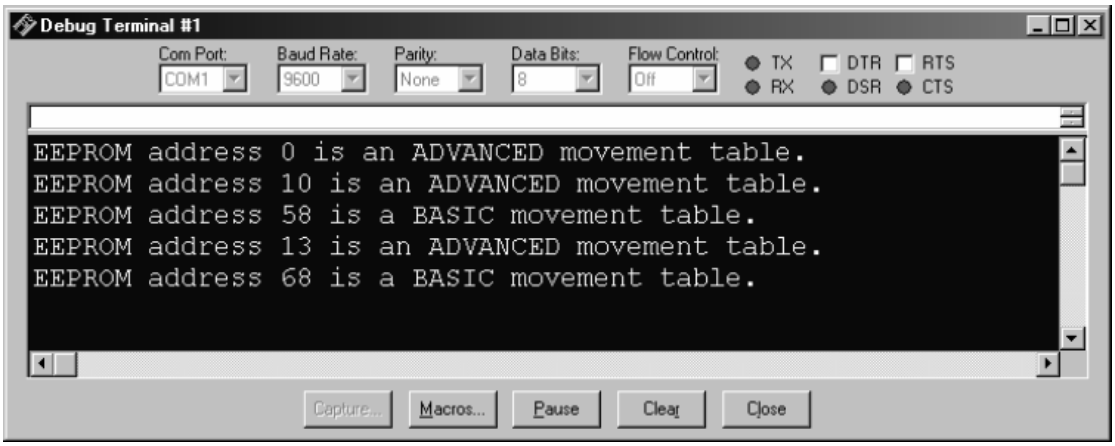


图4-2 调试终端中显示的EEPROM指针Mx

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 4.1: Which Table
' Demonstrates basic and advanced tables
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ Constants ]----------------------------------------------------------
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255 ' Movement table limit
' entry may be increased.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' -----[ Variables ]----------------------------------------------------------
Mx VAR Word
' -----[ EEPROM Data ]--------------------------------------------------------
'
' ---- Advanced Movement Tables ----
' Used for
LeftSemicircle DATA 7, bLeftTurn, bLeftTurn, bForward, xx
RightSemicircle DATA 7, bRightTurn, bRightTurn, bForward, xx
```

```
WalkForward3 DATA 3, bForward, xx
WalkForward8 DATA 8, bForward, xx
' ---- Basic Movement Tables ----
' These tables can contain movement support codes used in the advanced
' movement tables.
BasicMovements CON Forward
Forward DATA 1, TR, SL, TL, SR, xx
Backward DATA 1, TR, SL, TL, SR, xx
LeftTurn DATA 1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, TR, SL, TL, SR, xx
PivotLeft DATA 3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight DATA 3, TR, SL, TC, SR, TL, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
' -----[ Main Code ]----------------------------------------------------
' Walking routine is defined by the Mx value. Mix the basic and advanced
' movement tables to develop the walking routines for your Toddler.
Main_Program:
Mx = LeftSemicircle
GOSUB Movement
Mx = WalkForward3
GOSUB Movement
Mx = PivotRight
GOSUB Movement
Mx = WalkForward8
GOSUB Movement
Mx = Finish
GOSUB Movement
END
' -----[ Subroutines ]---------------------------------------------------
Movement:
IF Mx >= BasicMovements THEN BasicMovementTable
DEBUG "EEPROM address ", dec Mx," is an ADVANCED movement table.",cr
RETURN
BasicMovementTable:
DEBUG "EEPROM address ", dec Mx," is a BASIC movement table.",cr
RETURN
```

这种内存分隔方法考虑了PBASIC 给DATA语句按照程序定义顺序分配空间的方法。并非所有的程序语言都是这样，因此这个技巧只适应于具有相关技术支持的场合。这种方法在这里使用起来得心应手，而用别的方法分隔数据就更麻烦。如果不能恰当地组合各表，这种方法很容易产生编程错误，不过让这些**表**在程序文本中靠在一起就很容易发现这些错误。

该你了！

- 运行实例中程序
- 理解EEPROM调试终端显示的地址是如何与高级或基本动作相关联。

任务 2 走 8 字和方块舞

现在我们利用在任务1中创建的两类表来让步行机器人完成更加复杂的运动。一类表用来处理底层动作比如倾斜和迈步。另一类表用来处理一些高级运动比如拐个弯，或者走一个大圆。

在该任务中，将让步行机器人走一个“8”字和一个大的正方形。高级动作表的使用让步行机器人执行一系列更复杂运动变得很容易。此时，一个象LeftSemicircle的高级语句将导致步行机器人执行大量的基本脚部动作。

本任务的程序实现了一个比前面任务更复杂版本的Movement函数。这里的Mx变量包含的索引既可以是基本动作表，也可以是高级动作表。Movement函数据此执行相应的动作表。整个解释过程有点复杂，所以我们用一个DEBUG 语句来帮助呈现这一执行过程。DEBUG语句在下面的程序列表中实际上只是一个注释，但通过在编辑器执行一个“Replace All”的操作把“’ DEBUG”全部变为“DEBUG”，使其转变为可执行的代码行。相反的操作将把代码行变回注释。

DEBUG语句只有在步行机器人通过串行通讯线连接到个人计算机上才有用，因为信息只能显示在PC屏幕上。当步行机器人在一个较大范围内做一系列复杂的运动时，让步行机器人连接到个人计算机上也是可以的，只需要用一条更长一些的串行通讯线或者一台笔记本电脑就可以了。

当运行步行机器人上的电源切换开关处在下载模式的位置（位置2）时，我们可以用“DEBUG”替代“’ DEBUG”。这样，程序正常执行但电机却不转动。DEBUG语句可以让我们看到当电源切换开关处在运行模式（位置2）下步行机器人在执行什么。

当运行在下载模式时，步行机器人还是在不断地向电机发送脉冲，尽管电机没有被供电不能转动。虽然延时确实减慢了调试结果的显示，但可以紧跟在DoMovement标示后加一个RETURN命令来消除延时的影响，以使调试过程进行得更快。只是确认在让步行机器人真正运动之前，得将RETURN语句注释掉或者干脆删除该RETUN命令，否则步行机器人将不会运动。现在来看程序代码。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 4.2: Advanced Walking
' Demonstrates the use of basic and advanced tables for figure 8s
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]----------------------------------------------------
TiltServo CON 13 ' Tilt servo on P12
StrideServo CON 12 ' Stride servo on P13
LineFeed CON 10 ' Line Feed
' -----[ Constants ]----------------------------------------------------------
MoveDelay CON 18 ' in milliseconds
TiltStep CON 5 ' TiltServo step size
RightTilt CON 620 ' Tilt limits was 620
CenterTilt CON 750 ' was 750
LeftTilt CON 880 ' was 880
StrideStep CON 5 ' StrideServo step size
RightStride CON 650 ' Stride limits was 650
```

```
CenterStride CON 750 ' was 750
LeftStride CON 850 ' was 850
' -----[ Variables ]----------------------------------------------------
FigureLoop VAR Nib
MoveLoop VAR Byte ' Loop for repeat movements
MoveLoopLimit VAR Byte
SubMoveLoop VAR Byte ' Loop for repeat submovements
SubMoveLoopLmt VAR Byte
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
MxCurrent VAR Word
Sx VAR Word
SxCurrent VAR Word
' -----[ EEPROM Data ]---------------------------------------------------
'
' ------Movement Support Codes
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255
' ----- Movement Value Tables ------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section
LeftSemicircle DATA 7, bLeftTurn, bLeftTurn, bForward, xx
RightSemicircle DATA 7, bRightTurn, bRightTurn, bForward, xx
WalkForward3 DATA 3, bForward, xx
WalkForward8 DATA 8, bForward, xx
WalkForward1 DATA 1, bForward, xx
WalkBackward8 DATA 8, bBackward, xx
PivotLeft4 DATA 4, bPivotLeft, bForward, xx
' ------ Basic Movement Codes ------
```

```
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' ------ Basic Movement Tables ------
'
' These tables can contain Movement Support Codes.
BasicMovements CON Forward
Forward DATA 1, TR, SL, TL, SR, xx
Backward DATA 1, TR, SR, TL, SL, xx
LeftTurn DATA 1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, TR, SL, TL, SR, xx
PivotLeft DATA 3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight DATA 3, TR, SL, TC, SR, TL, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
GOSUB ResetCC
'DEBUG "Forward = ", HEX Forward, CR,LineFeed
'DEBUG "Backward = ", HEX Backward, CR,LineFeed
'DEBUG "LeftTurn = ", HEX LeftTurn, CR,LineFeed
'DEBUG "RightTurn = ", HEX RightTurn, CR,LineFeed
'DEBUG "PivotLeft = ", HEX PivotLeft, CR,LineFeed
'DEBUG "PivotRight = ", HEX PivotRight, CR,LineFeed
'DEBUG "LeftSemicircle = ", HEX LeftSemicircle, CR,LineFeed
'DEBUG "RightSemicircle = ", HEX RightSemicircle, CR,LineFeed
'DEBUG "WalkForward3 = ", HEX WalkForward3, CR,LineFeed
'DEBUG "WalkForward8 = ", HEX WalkForward8, CR,LineFeed
'DEBUG "WalkForward1 = ", HEX WalkForward1, CR,LineFeed
'DEBUG "WalkBackward8 = ", HEX WalkBackward8, CR,LineFeed
'DEBUG "PivotLeft4 = ", HEX PivotLeft4, CR,LineFeed
'DEBUG "Finish = ", HEX Finish, CR,LineFeed
' -----[ Main Code ]----------------------------------------------------------
Main_Program:
GOSUB ResetCC
' Make a Figure 8
FOR FigureLoop = 1 TO 5
Mx = LeftSemicircle
GOSUB Movement
Mx = WalkForward3
```

```
GOSUB Movement
Mx = RightSemicircle
GOSUB Movement
Mx = WalkForward3
GOSUB Movement
NEXT
' Make a big polygon
FOR FigureLoop = 1 TO 5
Mx = PivotRight
GOSUB Movement
Mx = WalkForward8
GOSUB Movement
NEXT
Mx = Finish
GOSUB Movement
END
' -----[ Subroutines ]-----------------------------------------------------
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
' or
' Mx = submovement table index, table ends in xx
'
' Note: All submovement tables come after the movment tables in this file.
Movement:
IF Mx < BasicMovements THEN SetupMovement
'DEBUG cr,"use submovements",cr
Sx = Mx ' setup for submovement table only
GOSUB StartSubMovement ' one pass through submovement table
RETURN
SetupMovement:
READ Mx, MoveLoopLimit ' read movement table repeat count
Mx = Mx + 1
StartMovement:
FOR MoveLoop = 1 TO MoveLoopLimit
'DEBUG cr,cr, hex (Mx-1), " Start Movement ", dec MoveLoop, " of "
'DEBUG dec MoveLoopLimit,cr
MxCurrent = Mx ' start of movements
MovementLoop:
READ MxCurrent, Sx ' read next submovment byte
MxCurrent = MxCurrent + 1
'DEBUG cr, " SX = ", dec Sx, " movement"
IF Sx = xx THEN MovementDone
' skip if end of list
```

```
LOOKUP Sx,[Finish,Forward,Backward,LeftTurn,RightTurn,
PivotLeft,PivotRight],Sx
' lookup submovement table index
GOSUB StartSubMovement
GOTO MovementLoop
MovementDone:
NEXT
RETURN
'--------
StartSubMovement: ' start executing submovement table
READ Sx, SubMoveLoopLmt
' read submovement table repeat count
Sx = Sx + 1
FOR SubMoveLoop = 1 TO SubMoveLoopLmt
'DEBUG cr, " SX = ", hex (Sx-1), " submovement ", dec SubMoveLoop
'DEBUG " of ", dec SubMoveLoopLmt, " "
SxCurrent = Sx
SubMovementLoop:
READ SxCurrent, Dx ' read next submovement action
SxCurrent = SxCurrent + 1
IF Dx = xx THEN SubMovementDone
' skip if end of list
GOSUB DoMovement ' execute movement
GOTO SubMovementLoop
SubMovementDone:
NEXT
RETURN
DoMovement:
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
StrideCenter,StrideRight]
' will fall through if invalid index
RETURN
' ---- Movement routines can be called directly as subroutines
TiltLeft:
NewValue = LeftTilt
'DEBUG "TL, "
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
'DEBUG "TC, "
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
'DEBUG "TR, "
```

```
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
StrideLeft:
NewValue = LeftStride
'DEBUG "SL, "
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
'DEBUG "SC, "
GOTO MovementStride
StrideRight:
NewValue = RightStride
'DEBUG "SR, "
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
'DEBUG "resetCC", CR, LineFeed
FOR Pulses = 1 TO 500 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

现在执行主程序Main_Program，步行机器人完成两个大的运动：走一个“8”字和一个正方形。Movement函数被调用来执行包含如左转的高级动作表。表中使用了bLeftTurn，因为这个例子中DATA语句只用来存储字节。这些bLeftTurn一类的值由Movement函数中的Lookup语句使用，以选择相应的低层动作表完成机器人的基本动作。一个高级动作表数据入

口将导致机器人执行很多底层的动作。

Movement函数仍然用任务1中提到的技巧来处理底层动作表。例如，Finish表就仅使用一些基本的动作。如果Movemet函数不处理两种类型的表，那么要么调用一个底层表，要么就要才创建一个高级动作表。

该你了！

- 运行示例程序
- 修改程序，让步行机器人在一个更小的范围内执行相同模式的动作。

挑战课题！

- 扩展 Movement 函数，使其可以处理如下语句定义的表。

```
SpecialMovement DATA 4, TL, SR, TC, SL, xx
DATA 2. TR, SL, TL, SR, xx
DATA 2. TL, SR, TC, SL, xx
```

为两种类型的动作表提供这种处理代码。

- 完成以上挑战。然后将下面多重调用 Movement 代码的程序

```
Mx = LeftSemicircle
GOSUB Movement
Mx = WalkForward3
GOSUB Movement
Mx = RightSemicircle
GOSUB Movement
Mx = WalkForward3
GOSUB Movement
减少成如下的代码
Mx = Figure8
GOSUB Movement
```

提示：走8字的表是上面所列四个表的组合。

- 编写一套对称有用的高级动作和低级动作表。高级动作必须包括TurnAroundLeft, TurnAroundRight, WalkForward1Foot, 和WalkBackward1Foot. 低级动作必须包括 ReversePivotRight 和 ReversePivotLeft.

第五章 光源跟踪

发挥你的创造性

我们所提供的元器件中光敏电阻可以让步行机器人探测到光线强度的变化。通过编写一定的程序，可以让步行机器人具有趋光和避光的特性。

要识别光的存在和强度，你必须在面包板上搭建一对光敏电阻电路。光敏电阻是一种阻值与光线强度有关的电阻，就像人的眼睛一样，它对光线很敏感。它里面与光有关的活性元素是CdS（硫化镉）。光线进入陶瓷半导体感光层并激发出自由的载流电子，从而产生了一个与光照强度成反比例的电阻也就是说光照越弱该电阻越大，光照越强该电阻越小。

这种很特别的电阻在我们提供的元器件里面就有。如果你还需要其他的光敏电阻，你可以从德普施或者帕拉斯公司元件商店或其他电子元器件提供者那里得到。见附录A：步行机器人零件目录和资源。光敏电阻的规格如表5-1所示。

Table 5-1: EG&G Vactec 光敏电阻规格。

Resistance (Ohms)					Peak Spectral Response nm	V_{MAX}	Response Time @ 1 fc (ms, typ.)	
10 Lux 2850K			Dark					
Min	Typ.	Max.	Min.	Sec.			Rise (1-1/e)	Fall (1/e)
20K	29.0K	38K	1M	10	550	100	35	5

Luminance（亮度）是用来度量光照的一个单位。通常亮度这个单位在英制中称作“foot-candle”（照度单位），在公制中称作“lux”（勒克斯）。但是这里我们没有必要去关注到底是多少“lux”，我们只要知道在某一方向光照是更强还是更弱就可以了。通过编程可以利用相对光强来给步行机器人导航。要了解更多关于用微型控制器测量光照的信息请参阅传感器的应用实验“light on Earth and Data Logging”。

任务 1：搭建电路并测试感光眼。

图5-1例出了在这个实验中要用到的电容和光敏电阻以及它们的电路符号。两个电容都是没有极性的，这意味着接线端1和2在电路中是可以互换的而不会对电路造成影响。除了电容之外，还需要二个220欧姆的电阻（红-红-棕）。

所需器件：

- 光敏电阻2个
- 0.1微法电容2个
- 0.01微法电容2个
- 220欧姆的电阻2个（图中没有给出）
- 跳线若干

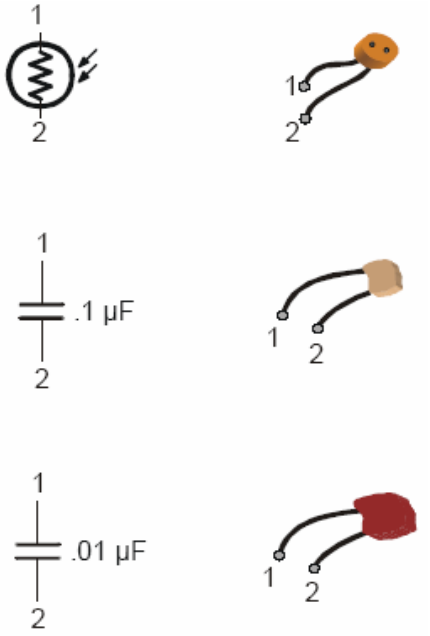


图 5-1 光敏电阻与电容的电路符号图与器件图

图5-2和5-3给出了每个光敏电阻的电阻/电容（RC）电路。光敏电阻是一种模拟器件，它的电阻值随着光照强度的变化而连续不断地变化。当它的感光表面在阳光的直射之下时，光敏电阻的输出电阻很小，随着光照强度的减弱，它的输出电阻增大。在完全黑暗的环境中，光敏电阻的阻值可以超过1M欧姆。虽然它是一种模拟器件，但它的阻值随光照强度的变化是非线性的。这就是说如果光照强度线性变化时，光敏电阻的输出并不一定线性变化。

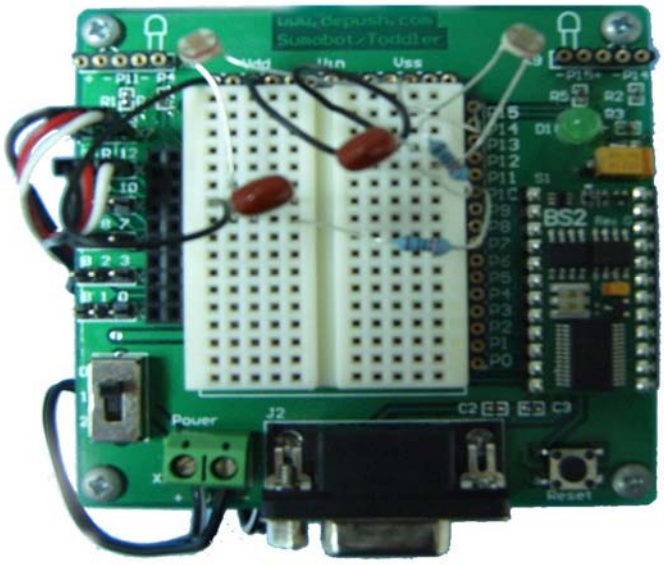


图 5-2: 光源跟踪电路接线实物图

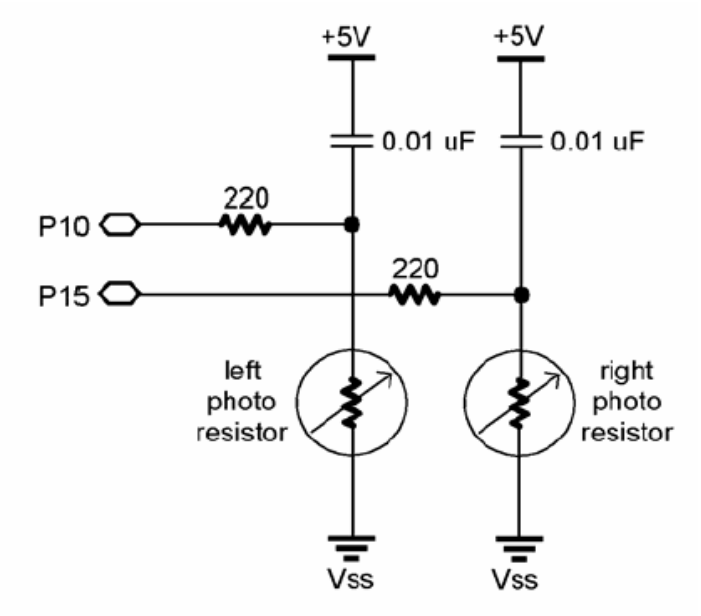


图 5-3 光源跟踪电路图

编程测量电阻

图5-2和图5-3所示的电路是为使用RCTIME 这条PBASIC命令而设计搭建的。这条命令可以用在一个RC电路中，电路中的一个参数R或者C变化而另一个参数保持不变。RCTIME命令能够测量变化值，因为它利用了RC电路的时间变化特性。

对于一个RC电路来说，要进行RCTIME测量的第一步是给电容低极板送上5V的电压。把通过一个220欧姆的电阻连接到电容低极板的I/O端口置成高电平，并持续一段时间。之后就可以用RCTIME命令了来测量电容低极板电压从5伏降到1.4伏所需要的时间。为什么是1.4伏呢？因为这是BASIC Stamp I/O端口的门限电压值。当加在I/O端口的高于1.4伏时，与这个I/O端口对应的位寄存器的值就是“1”。当电压低于1.4伏时，位寄存器的值就是“0”。

在这个电路中RCTIME测量一个光敏电阻电路中电容的低极板电压从5V降到1.4V所需要的时间，这个放电时间与光敏电阻的阻值成比例。由于光敏电阻阻值和亮度有关，因此放电时间也就与光的亮度有关。通过测量这个时间的长短，相对照度就可以推断出来。BASIC Stamp使用手册上有RCTIME命令详细的介绍。

RCTIME命令改变I/O端口的状态从输出变为输入。一旦当I/O端做输入用时，电容低极板的电压按照我们上面讨论的时间变化降低。BASIC Stamp 就开始以2us的间隔计时，直到电容器低极板的电压降到1.4伏以下。

（**注**：在寻找自然光的时候，光敏电阻在相对暗一些的光线下效果最好，直射的太阳光对光敏电阻来说太亮了。）

运行程序5.1。它给出怎样利用RCTIME命令读取光敏电阻的方法。程序中用到了调试终端，因此当程序运行时必须把串行通讯线连接到步行机器人电路板的串口上。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 5.1: Photoresistor RCTime Display
' Displays the R/C discharge time for two photoresistor circuits
' {$STAMP BS2}
' {$PBASIC 2.5}
```

```
' -----[ I/O Definitions ]-------------------------------------------------
LPhotoCircuit CON 10
RPhotoCircuit CON 14
' -----[ Variables ]--------------------------------------------------------
LPhotoVal VAR Word ' Stores measured RC times
RPhotoVal VAR Word ' of photoresistors.
' -----[ Initialization ]---------------------------------------------------
DEBUG CLS ' Open, clear Debug window
' -----[ Main Code ]--------------------------------------------------------
DO
' Measure RC time for left photoresistor.
HIGH LPhotoCircuit ' Set to output-high.
PAUSE 3 ' Pause for 3 ms.
RCTIME LPhotoCircuit,1,LPhotoVal ' Measure R/C time on left
' Measure RC time for right photoresistor.
HIGH RPhotoCircuit ' Set to output-high.
PAUSE 3 ' Pause for 3 ms
RCTIME RPhotoCircuit,1,RPhotoVal ' Measure R/C time on right
' Display RC time measurements using Debug Terminal.
DEBUG HOME, "Left = ", DEC5 LPhotoVal, " Right = ", DEC5 RPhotoVal, cr
LOOP
```

光敏电阻显示原理

程序中定义了两个word型变量RPhotoVal和LPhotoVal来存储左侧和右侧光敏电阻的RCTIME时间值。主程序用来测量和显示每个RC电路的RCTIME时间值。读取右侧RC电路时间的程序代码如下所示。首先I/O端口RphotoCircuit被置为高电平，且这个高电平保持在该端口上持续3ms，这让电容器有足够的时间来充电。3ms后，电容器的低极板电压足够接近5V就可以进行RC时间测量了。RCTIME命令从“1”状态（5V）开始测量I/O端口Rphotocircuit的RC时间，并把结果保存在Rphotoval变量中。注意存在Rphotoval中的值是一个数字。这个数字表明了在电容器低极板电压从5V降到1.4V门限电压时用了多少个2us。

```
HIGH RPhotoCircuit ' Set to output-high.
PAUSE 3 ' Pause for 3 ms
RCTIME RPhotoCircuit,1,RPhotoVal ' Measure R/C time on right
```

该你了！

- 试着用0.1uF的电容器代替其中一个0.01uF电容器。看看哪侧的电路在光线明亮的环境下效果更好，是有较大电容器那侧还是有较小电容器的那一侧呢？当环境光线不断变暗时又有什么样的影响呢？你能发现任何能告诉你哪侧电路在比较黑暗的环境下效果更好一些的蛛丝马迹吗，是有较大电容还是有较小电容的那侧呢？

任务 2：光源定位！

如果用一束手电筒光照着步行机器人的前面，那么上面所讨论过的电路和程序就可以让步行机器人旋转并朝着手电光。确认让光敏电阻对着光线以便能够让它们做一个光线强度的比较。除了与步行机器人中心线朝外成45度角外，光敏电阻还应与水平方向成45度角向下指。也就是说让光敏电阻的表面朝着桌面向下指，再用一束明亮的手电光让步行机器人朝着光的方向走。

编写程序让步行机器人指向光源

让步行机器人跟踪光源也就是通过编程测量和比较每个光敏电阻的值。我们知道光变暗时，光敏电阻的阻值增大。因此如果右侧光敏电阻的电阻值比左侧光敏电阻的阻值大时，这意味着步行机器人的左侧比右侧亮。这时Toddle机器人应该向左转。另一方面，如果左侧光敏电阻的RCTIME值比右侧光敏电阻的RCTIME值大，这就是说右侧比左侧亮，步行机器人应该向右转。

为了防止步行机器人转动太频繁，我们引入了DeadBand参量。DeadBand定义的是某一个范围内的值，在这些取值范围内步行机器人不会转弯。如果取值大于或小于这个范围内的值时，步行机器人会转弯。确定是否在DeadBand中最方便的方法就是用右侧的RCTIME值减去左侧的RCTIME值或者相反 ，然后取绝对值。如果绝对值在DeadBand范围内，那就什么也不做。否则采取相应的调整动作。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 5.2: Light Compass
' Points towards the most well-lit area in the room
' Adjust DeadBand for Toddler's sensitivity to light levels
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]----------------------------------------------------
TiltServo CON 13 ' Tilt servo on P13
StrideServo CON 12 ' Stride servo P12
LPhotoCircuit CON 10
RPhotoCircuit CON 15
' -----[ Constants ]----------------------------------------------------------
MoveDelay CON 15 ' in micrcoseconds
TiltStep CON 5 ' TiltServo step size
StrideStep CON 5 ' StrideServo step size
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
DeadBand CON 30 ' Light sensitivity diff
```

```
' -----[ Variables ]-------------------------------------------------------
LPhotoVal VAR word ' Stores measured R/C times
RPhotoVal VAR word ' of photoresistors
FigureLoop VAR Nib
MoveLoop VAR Byte ' Loop for repeat movements
MoveLoopLimit VAR Byte
SubMoveLoop VAR Byte ' Loop for repeat
SubMoveLoopLmt VAR Byte ' submovements
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
MxCurrent VAR Word
Sx VAR Word
SxCurrent VAR Word
' -----[ EEPROM Data ]-----------------------------------------------------
' ------ Movement Support Codes ------
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255
' ------ Movement Value Tables ------
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
' Note: ALL movement tables must be in this section
LeftSemicircle DATA 7, bLeftTurn, bLeftTurn, bForward, xx
RightSemicircle DATA 7, bRightTurn, bRightTurn, bForward, xx
WalkForward3 DATA 3, bForward, xx
WalkForward8 DATA 8, bForward, xx
' ------ Basic Movement Codes ------
'' Used in Movement tables.
' Referenced below using LOOKUP statement.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
```

```
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' ------ Basic Movement Tables ------
'' These tables can contain Movement Support Codes.
BasicMovements CON Forward
Forward DATA 1, TR, SL, TL, SR, xx
```

Chapter #5: Following Light · **Page 99**

```
Backward DATA 1, TR, SR, TL, SL, xx
LeftTurn DATA 1, TL, SR, TC, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, xx
PivotLeft DATA 3, TL, SR, TC, SL, xx
PivotRight DATA 3, TR, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
' -----[ Initialization ]----------------------------------------------------
GOSUB ResetCC ' Initialize feet
' -----[ Main Code ]----------------------------------------------------------
Main:
' Measure RC time for left photoresistor.
HIGH LPhotoCircuit ' Set to output-high.
PAUSE 3 ' Pause for 3 ms.
RCTIME LPhotoCircuit,1,LPhotoVal ' Measure R/C time on left
' Measure RC time for right photoresistor.
HIGH RPhotoCircuit ' Set to output-high.
PAUSE 3 ' Pause for 3 ms
RCTIME RPhotoCircuit,1,RPhotoVal ' Measure R/C time on right
' Measure difference between RPhotoVal and LPhotoVal, decide what to do
DEBUG home, "Left = ", dec LPhotoVal, " Right = ",dec RPhotoVal,cr
IF ABS(LPhotoVal-RPhotoVal) < DeadBand THEN main
IF LPhotoVal > RPhotoVal THEN turn_right
IF LPhotoVal < RPhotoVal THEN turn_left
'----- Navigation Routines -------
Turn_left: ' turn left towards light
Mx = PivotLeft
GOSUB Movement
GOTO main ' go back to main routine.
Turn_right: ' turn right towards light
Mx = PivotRight
GOSUB Movement
GOTO main ' go back to main routine.
' -----[ Subroutines ]---------------------------------------------------
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
```

```
' or
' Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.
Movement:
IF Mx < BasicMovements THEN SetupMovement
MxCurrent = Mx ' setup to use submovement
MoveLoopLimit = 1 ' table
GOTO StartMovement
SetupMovement:
READ Mx, MoveLoopLimit ' read movement table
MxCurrent = Mx + 1 ' repeat count
StartMovement:
FOR MoveLoop = 1 to MoveLoopLimit
Mx = MxCurrent ' Mx = start of movement
' table
'DEBUG DEC Mx, " Movement ", DEC MoveLoop, " of ", DEC MoveLoopLimit,CR
IF Mx < BasicMovements THEN MovementLoop
' skip if movement table
SxCurrent = Mx ' SxCurrent = submovement table index
GOTO StartSubMovement ' enter middle of loop
MovementLoop:
READ Mx, SxCurrent ' read next submovment byte
Mx = Mx + 1
IF SxCurrent = xx THEN MovementDone
' skip if end of list
'DEBUG " ", DEC SxCurrent, " movement",CR
LOOKUP SxCurrent,[Finish,Forward,Backward,LeftTurn,
RightTurn,PivotLeft,PivotRight],SxCurrent
' lookup submovement table index
StartSubMovement: ' execute submovement table
READ SxCurrent, SubMoveLoopLmt
' read submovement table
SxCurrent = SxCurrent + 1
FOR SubMoveLoop = 1 TO SubMoveLoopLmt
Sx = SxCurrent
'DEBUG " ", DEC Sx, " submovement "
'DEBUG DEC SubMoveLoop, " of ", DEC SubMoveLoopLmt,CR
SubMovementLoop:
READ Sx, Dx ' read next submovement
Sx = Sx + 1 ' action
IF Dx = xx THEN SubMovementDone
' skip if end of list
GOSUB DoMovement ' execute movement
```

```
GOTO SubMovementLoop
SubMovementDone:
NEXT
IF Mx < BasicMovements THEN MovementLoop
MovementDone:
NEXT
RETURN
DoMovement:
'DEBUG " ", DEC Dx, " action", CR
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
StrideCenter,StrideRight]
' will fall through if invalid index
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
```

```
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

光源定位是如何工作的！

程序 5.2 读取 RC 时间之后首先核对由 RCTIME 命令返回的两个值之差是否在 DeadBand 变量范围之内。它使用下面这个命令：

```
IF ABS(LPhotoVal-RPhotoVal) < DeadBand THEN main
```

如果两个RC时间的差在DeadBand内，则程序转到**Main:** 标志I处。如果两个RC时间差在DeadBand范围之外，两个IF…THEN语句决定调用哪个子程序，**Turn_left** 或 **Turn_right**。

```
IF ABS(LPhotoVal-RPhotoVal) < DeadBand THEN main
IF LPhotoVal > RPhotoVal THEN turn_right
IF LPhotoVal < RPhotoVal THEN turn_left
```

这些程序用的就是以前在前面章节就介绍过的运动程序。步行机器人能做出更小的转弯。

该你了！

- 进入并运行程序5.2
- 用一束光照着步行机器人前面，当你移动这个光束时，步行机器人也会跟着移动试图总是面对光束。
- 不再用手电光，用手遮住其中的一个光敏电阻，这时你会发现步行机器人会转圈并总是要离开你手带来的阴影区域。
- 在一个较暗的环境下，光敏电阻的值和他们之间的差值都会变大。你得增大死区值，让步行机器人能在较暗的环境中捕捉微弱的光线变化。光线越暗，需要PAUSE的时间值就越短。如果步行机器人的表现变差，这也许是因为脉冲之间的时间间隔超过了40ms。应对这种现象的第一招是把PAUSE的时间值减小到零。第二招是检查光敏电阻，第一个脉冲后检查右侧的光敏电阻，第二个脉冲后检查左侧的光敏电阻。你可以试着改进程序在挑战课题中来做到这些。
- 用上面的程序在不同的亮度的环境中做实验，测试比较亮的环境和比较暗的环境下的死区值。在亮一些的环境中，死区值可以取得小一些甚至可以为零，在暗一些的环境中死区值则要取大一些。
- 交换程序5.2中第二个和第三个IF….THEN语句的条件，再重新运行程序，这时步行机器人远离光源。

任务 3：光源跟随

实现让步行机器人跟随光源只需要对程序5.2做稍微的修改.主要的改变是当RC时间差值落在死区范围内时，在程序5.3将会使步行机器人产生向前的运动而在5.2中没有产生任何运动.让我们来看看程序是怎样工作的.

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 5.3: Follow the Light
' Marching toward the light
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]----------------------------------------------------
TiltServo CON 13 ' Tilt servo on P12
StrideServo CON 12 ' Stride servo on P13
LPhotoCircuit CON 10
RPhotoCircuit CON 14
' -----[ Constants ]----------------------------------------------------------
MoveDelay CON 18 ' in micrcoseconds
TiltStep CON 5 ' TiltServo step size
StrideStep CON 5 ' StrideServo step size
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
DeadBand CON 5 ' Photoresistor R/C DeadBand
' -----[ Variables ]----------------------------------------------------------
LPhotoVal VAR Word ' Stores measured R/C times
RPhotoVal VAR Word ' of photoresistors
FigureLoop VAR Nib
MoveLoop VAR Byte ' Loop for repeat movements
MoveLoopLimit VAR Byte
SubMoveLoop VAR Byte ' Loop for repeat submovements
SubMoveLoopLmt VAR Byte
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
MxCurrent VAR Word
Sx VAR Word
```

```
SxCurrent VAR Word
' -----[ EEPROM Data ]----------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255
' ------ Movement Value Tables ------
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section
LeftSemicircle DATA 7, bLeftTurn, bLeftTurn, bForward, xx
RightSemicircle DATA 7, bRightTurn, bRightTurn, bForward, xx
WalkForward3 DATA 3, bForward, xx
WalkForward8 DATA 8, bForward, xx
' ------ Basic Movement Codes ------
' Used in Movement tables.
' Referenced below using LOOKUP statement.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' ------ Basic Movement Tables ------
'
' These tables can contain Movement Support Codes.
BasicMovements CON Forward
Forward DATA 1, TR, SL, TL, SR, xx
Backward DATA 1, TR, SL, TL, SR, xx
LeftTurn DATA 1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, TR, SL, TL, SR, xx
PivotLeft DATA 1, TL, SR, TC, SL, xx
PivotRight DATA 1, TR, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
' -----[ EEPROM Data ]----------------------------------------------------
```

```
GOSUB ResetCC ' Initialize feet
' -----[ Main Code ]----------------------------------------------------
Main:
' Measure RC time for left photoresistor.
HIGH LPhotoCircuit ' Set to output-high
PAUSE 3 ' Pause for 3 ms
RCTIME LPhotoCircuit,1,LPhotoVal ' Measure R/C time on left
' Measure RC time for right photoresistor.
HIGH RPhotoCircuit ' Set to output-high
PAUSE 3 ' Pause for 3 ms
RCTIME RPhotoCircuit,1,RPhotoVal ' Measure R/C time on right
' Measure difference between RPhotoVal and LPhotoVal, decide what to do
IF ABS(LPhotoVal-RPhotoVal) > DeadBand THEN check_dir
' Check if difference between RC times is within the deadband
' If yes, then forward. If no then skip to check_dir subroutine.
walk_forward:
Mx = Forward
GOSUB Movement
goto main
' Select right_turn or left_turn depending on which RC time is larger
check_dir:
IF LPhotoVal > RPhotoVal THEN turn_right
IF LPhotoVal < RPhotoVal THEN turn_left
'----- Navigation Routines -------
turn_left: ' turn left towards light
Mx = PivotLeft
GOSUB Movement
goto main ' go back to main routine.
Turn_right: ' turn right towards light
Mx = PivotRight
GOSUB Movement
GOTO main ' go back to main routine.
' -----[ Subroutines ]---------------------------------------------------
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
' or
' Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.
Movement:
IF Mx < BasicMovements THEN SetupMovement
MxCurrent = Mx ' setup to use submovement table
MoveLoopLimit = 1
```

```
GOTO StartMovement
SetupMovement:
READ Mx, MoveLoopLimit ' read movement table
MxCurrent = Mx + 1 ' repeat count
StartMovement:
FOR MoveLoop = 1 to MoveLoopLimit
Mx = MxCurrent ' Mx = start of movement table
'debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr
IF Mx < BasicMovements THEN MovementLoop
' skip if movement table
SxCurrent = Mx ' SxCurrent = submovement index
GOTO StartSubMovement ' enter middle of loop
MovementLoop:
READ Mx, SxCurrent ' read next submovment byte
Mx = Mx + 1
IF SxCurrent = xx THEN MovementDone
' skip if end of list
'debug " ", dec SxCurrent, " movement",cr
LOOKUP SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,
PivotLeft,PivotRight],SxCurrent
' lookup submovement table index
StartSubMovement: ' start executing submovement table
READ SxCurrent, SubMoveLoopLmt ' read submovement table repeat count
SxCurrent = SxCurrent + 1
FOR SubMoveLoop = 1 to SubMoveLoopLmt
Sx = SxCurrent
'DEBUG " ", DEC Sx, " submovement ", DEC SubMoveLoop, " of ",
dec SubMoveLoopLmt,CR
SubMovementLoop:
READ Sx, Dx ' read next submovent action
Sx = Sx + 1
IF Dx = xx THEN SubMovementDone
' skip if end of list
GOSUB DoMovement ' execute movement
GOTO SubMovementLoop
SubMovementDone:
NEXT
IF Mx < BasicMovements THEN MovementLoop
MovementDone:
NEXT
RETURN
DoMovement:
'debug " ", dec Dx, " action",cr
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
```

```
StrideCenter,StrideRight]
' will fall through if invalid index
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
```

```
NEXT
DoReturn:
RETURN
```

光源跟踪程序是如何工作的

和以前的程序一样,第一个IF…THEN语句对左右两侧电路的RC时间值进行检测.语句已经被修改了以便当两侧RC时间差值在死区范围外时,程序将跳过Walk-forward子程序运行.另一方面，如果两侧RC时间差处在死区范围内,则执行前进的动作脉冲。之后,程序直接返回main处，又进行两侧RC时间的检测。

```
IF ABS(LPhotoVal-RPhotoVal) > DeadBand THEN check_dir

walk_forward:
     Mx = Forward
     GOSUB Movement
     goto main
```

如果两侧RC时间的差值不在死区范围内，程序将跳到check-dir标识处。该标识后的IF…THEN语句根据**RPhotoVal** 和**LPhotoVal**之间不等关系来决定步行机器人是左转还是右转。

```
check_dir:
     IF LPhotoVal > RPhotoVal THEN turn_right
     IF LPhotoVal < RPhotoVal THEN turn_left
```

挑战课题!

用手电光引导步行机器人。

- 让光敏电阻向上朝外指，而非将其表面直接指向机器人前面，步行机器人将在地面上漫游并总是寻找最亮的地方。
- 依据环境亮度的不同，你或许要增大死区的值来让步行机器人的动作平缓和连续一些，反之要减小死区的值让步行机器人反应更灵敏来找出最亮的地方。

第六章 红外避障运动

无线物体检测

如今最热门的产品看起来都有一个共性：具备无线通信功能。比如个人商务终端可以把数据无线传输到桌面电脑，无线遥控可以让我们自由选择频道。使用一些价格低廉随处可得的零件，BASIC Stamp微控制器也可以使用红外LED和检测装置检为行进中的步行机器人探测前方或者旁边的障碍物。

检测障碍物并不需要象机器视觉那样复杂的设备。一个很简单的系统就可以满足要求。许多机器人使用雷达（RADAR）或者声纳（SONAR或SODAR）。一个更简单的系统是使用红外光来照射机器人的路线，然后确定何时有光线从目标反射回来。由于红外远程控制技术的发展，红外线发射器和探测器已经很普及并且价格低廉。

步行机器人的红外物体检测方法有许多用途。步行机器人可以在碰撞到物体之前探测到他们。在进行线跟踪时，使用光敏电阻和红外线发射器，可以检测出白色和黑色。红外线还可以用于确定步行机器人到障碍物的距离。使用距离信息可以让步行机器人与被跟踪物体保持一固定距离，或者检测和避开高的台阶。

步行机器人的红外前灯

我们在步行机器人上建立的红外物体探测系统在许多方面就象小汽车的前灯。当汽车前灯的射出的光从障碍物体反射回来时，你的眼睛发现障碍物体，然后你的大脑处理这些信息，并据此控制你的身体驾驶汽车。步行机器人使用红外线两极管LEDs作为前灯。他们向前方发射红外光，在大多数情况下，红外线从机器人行进方向的物体上反射回。步行机器人的眼睛是红外检测器。红外检测器发出信号来表明它们是否检测到从物体反射回的红外线。步行机器人的大脑，BASIC Stamp微控制器据此做出判断并基于这个传感器的输入控制伺服电机。

IR（红外线）检测器有内置的光滤波器，它只允许需要检测的980 nm红外线通过。红外检测器还有一个电子滤波器，它只允许大约38.5 kHz 的光信号通过。换句话说，检测器只寻找每秒闪烁38,500次的红外光。这就防止了普通光源象太阳光和室内光对IR的干涉。太阳光是直流干涉光源（0Hz），室内光依赖于所在区域的主电源，闪烁频率接近100或120 Hz。由于120 Hz在电子滤波器的38.5 kHz通带频率之外，它完全被IR探测器忽略。

使用 FREQOUT 的技巧

IR检测器只能看到按照38.5kHz闪烁的IR信号，所以IR发射器必须按照这个频率闪烁。可以使用一个555定时器达到这个目的，但555定时器电路很复杂，而且功能也不如我们本章和下一章使用的电路。比如，下面介绍的IR检测方法可以用于距离检测，而555定时器则需要额外的硬件支持来完成距离检测。

该红外检测方法是有一对步行机器人爱好者发明的，这是一个非常有趣的方法，它无需使用555定时器。该方法使用不带RC滤波器的FREQOUT命令，RC滤波器通常用来将信号平滑成正弦波。尽管FREQOUT命令的最高设计发射信号频率只有32768Hz，但未经过滤的FREQOUT输出信号包括了一系列谐波，这些谐波包含38.5kHZ的谐波。当然，更有用的仍然是你能直接用象*FREQOUT Pin, Period, 38500*这样的命令来发射一个38.5kHz的谐波。

图6-1 （a）显示的是由命令*FREOUT Pin, Period, 27036*发射的信号。调整电子接收器，比如我们要使用的IR检测器，可以检测到被称之为谐波的信号成分。FREQOUT发出的信号的所包含的两个主要的低频谐波显示在图6-1的(b)和(c)中。(b)显示的基本谐波，(c)显示的则是三阶谐波。这些谐波实际上都是（a）中未经滤波的FREQOUT命令输出脉冲的组成部分。图(c)中显示的三重谐波频率可以直接通过输入如*FREQOUT Pin, Period, 38500*来控制。如需改变谐波频率，只需改变最后一个参数，如需40kHz谐波，输入*FREQOUT Pin, Period, 40000*即可。

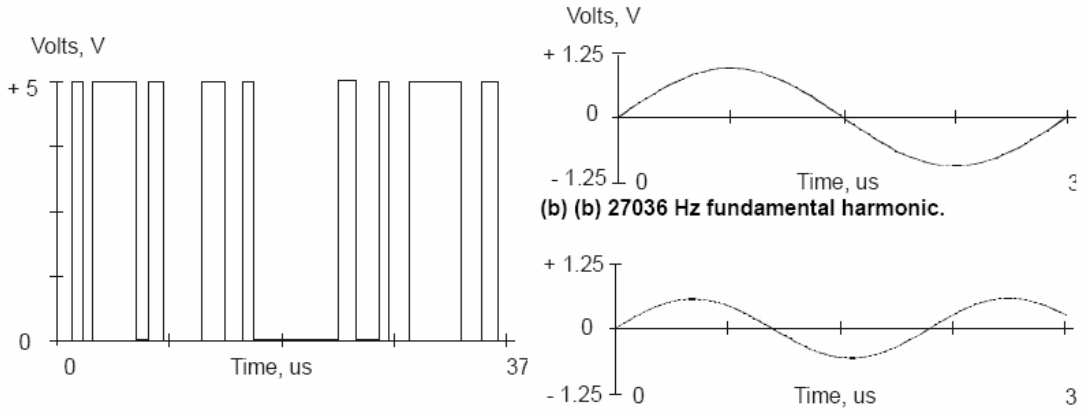


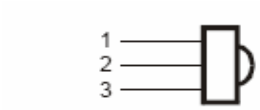
图 6-1 FREQOUT 的信号特性

尽管“FREQOUT”方法可用，但还有一个问题。BASIC Stamp不支持多任务。这之所以成为一个问题，是因为IR检测装置在接收到38.5kHz的红外信号时仅发送一个低电平信号来指示已经检测到物体，否则它一直输出高电平。幸运的是，检测装置能在返回高电平状态前足够长的维持低电平的时间，以便BASIC Stamp能够读取道该值。检测装置输出之所以能够维持这么长的时间，是因为该装置在收到象(a)中那样具有不对称的高低电平信号时，其响应时间会变慢。

任务 1：搭建并测试新的红外/检测装置

所需器件

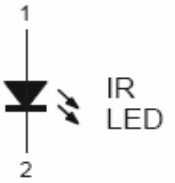
- 压堤电扬声器1个
- 封装好的红外二极管2个
- 红外检测器2个
- 一些连接导线



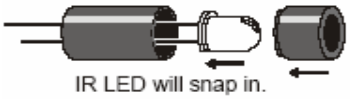
红外检测器电路符号图

红外检测器器件图

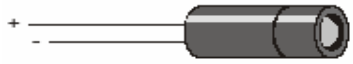
图 6-2 红外器件和它们的电路符号图



红外 LED 电路符号图



红外 LED 器件图



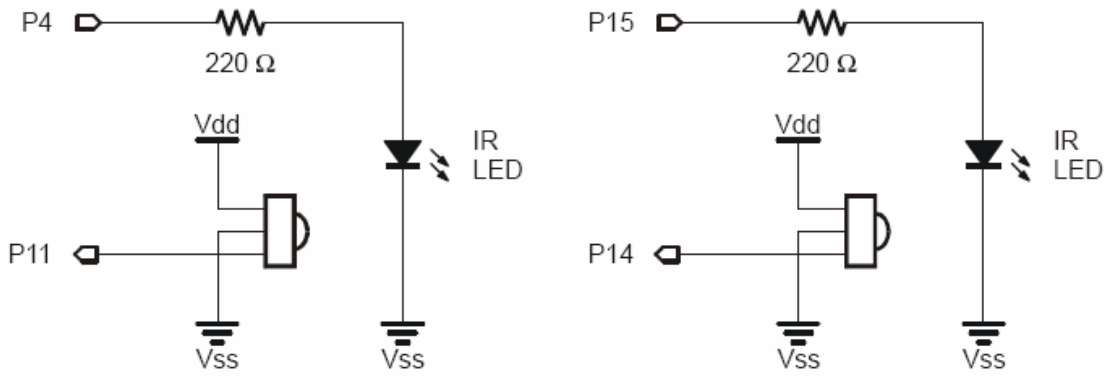


图 6-3 红外传感器接线图；注意：220 欧姆的电阻已经焊在 PCB 上了，只需要把两个 LED 插在前面的插座上即可，不再另需电阻

6-2是这些器件的电路符号图和实际器件图，你可以据此认出它们。图6-3是红外检测电路图。在你的步行机器人电路板上搭建该电路。注意220欧姆的电阻已经焊接在电路板上，因此只需将红外二极管插入，你的步行机器人就准备好了。

两个红外对分别安装在步行机器人的电路板的两个角落上。

测试红外线器件对

每个红外检测对的工作原理是：首先用FREQOUT发射1ms的未经滤波的38.5kHz的谐波，然后立即检测由红外检测器送回的信号，并存储。红外检测器没有检测到红外信号时的正常输出状态是高电平。当它检测到由红外二极管发射的38.5kHz的谐波信号时，其输出变为低电平。当然，如果如果没有物体反射红外线，红外检测器的输出就是高电平。程序6.1展示了这种读取检测器方法的编程实例。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 6.1: Infrared Pairs Display
' Test program for the infrared emitters / detectors
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]----------------------------------------------------
lEmitter CON 4
rEmitter CON 15
lInput VAR in11
rInput VAR in14
' -----[ Variables ]----------------------------------------------------------
lDetector VAR Bit ' Two bit variables for saving IR
rDetector VAR Bit ' detector output values.
' -----[ Initialization ]-----------------------------------------------------
OUTPUT lEmitter ' signals to function as outputs
OUTPUT rEmitter
' -----[ Main Code ]----------------------------------------------------------
DO
' Detect object on the left.
FREQOUT lEmitter, 1, 38500 ' Send freqout signal - left IR LED
lDetector = lInput ' Store IR detector output in RAM.
' Detect object on the right.
FREQOUT rEmitter, 1, 38500 ' Repeat for the right IR pair.
rDetector = rInput
DEBUG home, "Left= ", BIN1 lDetector
PAUSE 20
DEBUG " Right= ", BIN1 rDetector
PAUSE 20
LOOP
```

该你了！

- 输入并执行程序6.1。
- 该程序使用了BASIC Stamp的编辑器调试终端，因此在执行时请将串口线与步行机器人连接。
- 执行程序6.1时，让红外线检测装置指向一个使附近任何可能反射回来红外线都无法接收的方向。最好的办法就是让步行机器人指向天花板。调试终端窗口显示的左右红外检测返回的值都应该是“1”。
- 把你的手放到红外检测对的前面，相应检测对对应的调试终端显示窗口的检测值应该从“1”变为“0”。移开你的手，相应的检测显示应当返回“1”状态。这个现象应该对每个检测对都有效，你也应当可以把你的手放在两个检测对的前面，让两个检测输出显示都从“1”变为“0”。
- 如果红外检测对通过了这些测试，你就可以准备继续下面的实验，否则，检查你的程序和电路的错误。

红外检测对显示程序是如何工作的

首先声明了两个位变量来存储每个红外检测器的输出值。DO…LOOP 循环过程中的第一个 FREQOUT 命令不同。命令 FREQOUT lEmitter,1,38500 通过左边的红外二极管电路发射图 6-1 所示的开关模式信号，使其快速闪烁。包含在该信号中的谐波在碰到物体时就反射回来。如果信号被反射回来并被红外检测器检测到，红外检测器就送出一个低电平信号到 I/O 口 lInput,否则就送给 lInput 1 个高电平信号。FREQOUT 命令后的语句用于检测红外检测器的输出，该输出可以作为一个变量存储在 RAM 中。语句 lDetector=lInput 检测 lInput，并将检测值存储在位变量 lDetector 中。另一个红外对的检测就是重复这一过程，相应的输出值存储在 rDetector 中。然后 DEBUG 命令在调试终端显示这些值。

该你了！

- 采用比38.5kHz高的频率对红外检测对进行实验，比如39.0，39.5，40.0，40.5和41KHz。通过逐步向红外检测对移动一个物体，观察在哪个距离开始让红外检测对的输出从“1”变为“0”，确定不同频率下可以检测的最大距离。

任务 2 物体检测和避碰运动

红外线检测对提供了步行机器人所需的避碰运动信息。一个简单的避碰程序能让机器人在一个屋子里随意行走而不会发生碰撞。当然障碍物必须足够高，以使步行机器人的红外检测器可以检测到。

程序6.2检查红外检测传感器对，并且基于这些传感器的信息分配四种不同命令脉冲传中的一种。每次的漫游过程仅仅就是往Forward(前进)，Left_turn（左转），Right_turn（右转）或者Backward(后退)四个方向中的一个方向走上一步。一步走完后，再次检查传感器数据，然后据此再走一步，如此等等。该程序还使用了一些新的编程技术，这些技术你会发现非常有用。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 6.2: Object Detection and Avoidance
' Uses the infrared emitters and detectors
```

```
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]-------------------------------------------------
lEmitter CON 4
rEmitter CON 15
lInput VAR in11
rInput VAR in14
StrideServo CON 12 ' Stride servo on P12
TiltServo CON 13 ' Tilt servo on P13
' -----[ Constants ]-------------------------------------------------------
MoveDelay CON 18 ' in micrcoseconds
TiltStep CON 10 ' TiltServo step size
StrideStep CON 10 ' StrideServo step size
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
' -----[ Variables ]-------------------------------------------------------
Sensors VAR Nib ' variable is used to store
' lower two bits of detector values
FigureLoop VAR Nib
MoveLoop VAR Byte ' Loop for repeat movements
MoveLoopLimit VAR Byte
SubMoveLoop VAR Byte ' Loop for repeat submovements
SubMoveLoopLmt VAR Byte
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
MxCurrent VAR Word
Sx VAR Word
SxCurrent VAR Word
' -----[ EEPROM Data ]-----------------------------------------------------
' ------ Movement Support Codes ------
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
```

```
SL CON 3
SC CON 4
SR CON 5
xx CON 255
' ------ Movement Value Tables ------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section
LeftSemicircle DATA 7, bLeftTurn, bLeftTurn, bForward, xx
RightSemicircle DATA 7, bRightTurn, bRightTurn, bForward, xx
WalkForward3 DATA 3, bForward, xx
WalkForward8 DATA 8, bForward, xx
' ------ Basic Movement Codes ------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' ------ Basic Movement Tables ------
'
' These tables can contain Movement Support Codes.
BasicMovements CON Forward
Forward DATA 1, TR, SL, TL, SR, xx
Backward DATA 1, TR, SR, TL, SL, xx
LeftTurn DATA 1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, TR, SL, TL, SR, xx
PivotLeft DATA 3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight DATA 3, TR, SL, TC, SR, TL, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
' -----[ Initialization ]----------------------------------------------------
OUTPUT lEmitter ' signals to function as outputs
OUTPUT rEmitter
GOSUB ResetCC ' Initialize feet
' -----[ Main Code ]----------------------------------------------------------
DO
FREQOUT lEmitter,1,38500 ' Send freqout signal - left IRLED.
sensors.bit0 = lInput ' Store IR detector output in RAM.
```

```
' Detect object on the right.
FREQOUT rEmitter,1,38500 ' Repeat for the right IR pair.
sensors.bit1 = rInput
PAUSE 18 ' 18 ms pause(2 ms lost on freqout)
' Loads the IR detector output values into the lower 2 bits of the
' sensors variable, a number btwn 0 and 3 that the LOOKUP command can use
LOOKUP sensors,[Backward,PivotLeft,PivotRight,Forward],Mx
GOSUB Movement
LOOP
' -----[ Subroutines ]------------------------------------------------------
'
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
' or
' Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.
Movement:
IF Mx < BasicMovements THEN SetupMovement
MxCurrent = Mx ' setup to use submovement table
MoveLoopLimit = 1
GOTO StartMovement
SetupMovement:
READ Mx, MoveLoopLimit ' read movement table repeat count
MxCurrent = Mx + 1
StartMovement:
FOR MoveLoop = 1 to MoveLoopLimit
Mx = MxCurrent ' Mx = start of movement table
'debug DEC Mx, " Movement ", DEC MoveLoop, " of ", DEC MoveLoopLimit,cr
IF Mx < BasicMovements THEN MovementLoop
' skip if movement table
SxCurrent = Mx ' SxCurrent = submovement table index
GOTO StartSubMovement ' enter middle of loop
MovementLoop:
READ Mx, SxCurrent ' read next submovment byte
Mx = Mx + 1
IF SxCurrent = xx THEN MovementDone
' skip if end of list
debug " ", DEC SxCurrent, " movement",cr
LOOKUP SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,
PivotLeft,PivotRight],SxCurrent
' lookup submovement table index
StartSubMovement: ' start executing submovement table
```

```
READ SxCurrent, SubMoveLoopLmt
' read submovement table repeat count
SxCurrent = SxCurrent + 1
FOR SubMoveLoop = 1 to SubMoveLoopLmt
Sx = SxCurrent
debug " ", DEC Sx, " submovement ", DEC SubMoveLoop, " of ",
DEC SubMoveLoopLmt,cr
SubMovementLoop:
READ Sx, Dx ' read next submovent action
Sx = Sx + 1
IF Dx = xx THEN SubMovementDone
' skip if end of list
GOSUB DoMovement ' execute movement
GOTO SubMovementLoop
SubMovementDone:
NEXT
IF Mx < BasicMovements THEN MovementLoop
' exit if submovement table
MovementDone:
NEXT
RETURN
DoMovement:
debug " ", DEC Dx, " action",cr
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
StrideCenter,StrideRight]
' will fall through if invalid index
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
```

```
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

该程序声明了sensors（传感器）变量，这是一个半个字节（四位）长RAM变量。在传感器变量这四个位中，只使用了最低的两位。0位用于存储左边检测的输出，1位用于存储右边检测的输出。

主程序以FREQOUT命令开始，用于发射红外线信号，不过紧跟每个freqout命令之后的命令与前一程序中的命令有些不同。前面的程序将传感器输出的位值存储在一个不同的位变量中，而这里却把它们存储在sensors变量的位中。Sensors变量的0位存储IN8的二进制值，1位存储IN0的二进制值。在sensors变量的低两位都置好后，将获得一个“0”到“3”的十进制值。LOOKUP命令使用这些值确定将哪个程序标记发送给程序。

```
DO
     FREQOUT lEmitter,1,38500 ' Send freqout signal - left IRLED.
     sensors.bit0 = lInput ' Store IR detector output in RAM.
     ' Detect object on the right.
     FREQOUT rEmitter,1,38500 ' Repeat for the right IR pair.
     sensors.bit1 = rInput
     PAUSE 18 ' 18 ms pause(2 ms lost on freqout)
     ' Loads the IR detector output values into the lower 2 bits of the
```

```
        ' sensors variable, a number btwn 0 and 3 that LOOKUP can use
        LOOKUP sensors,[Backward,PivotLeft,PivotRight,Forward],Mx
        GOSUB Movement
    LOOP
```

四个可能的二进制数结果显示在表6.1中。表中同时显示基于传感器变量值触发的动作结果。

状态的二进制值	状态的十进制值	基于传感器输出状态值的分支动作
0000	0	lInput=0 和 rInput=0, 两个红外检测装置均检测到物体，机器人后退
0001	1	lInput=0 和 rInput=1, 左边的红外检测装置检测到物体，机器人右转
0010	2	lInput=1 和 rInput=0, 右边的红外检测装置检测到物体，机器人左转
0011	3	lInput=1 和 rInput=1, 两个红外检测装置均没有检测到物体，机器人前进

表6-1：红外检测器状态表

设置的Mx变量用来对应合适的动作表索引，然后动作函数执行相应的命令序列。

挑战课题！

你可以重新排列 LOOKUP 命令中的地址表，以使步行机器人碰到障碍物时执行不同的动作。一个有趣的实验是用 Forward 地址代替 Backward 地址。这样在 Lookup 地址列表中将有两个 Forward 实例，但这不是一个问题。另外，也可以交换 Left_turn 和 Right_turn 地址。

- 尝试上面讨论的改动。

现在你用起步行机器人应该是得心应手了。不过，如果没有办法进行距离检测，那么就不能编程使一个步行机器人就跟踪另一个步行机器人。如果前面的机器人停下来，后面的机器人就会撞上去。在下章中我们将此问题作为一个例子来解决。

第七章 在桌面行走

什么叫频率扫描？

通常，在你收听收音机节目时，查找你最喜欢的节目的过程就叫频率扫描。设定一个频率的电台，听听其节目内容。如果你不喜欢该节目正在播放的内容，就调到其它频率的收听其节目，直到找到你喜爱的节目。

任务 1：测试频率扫描

可以通过编程让步行机器人发射不同频率的红外信号，并检查每一个频率下检测到的物体。通过跟踪红外探测器报告的物体的频率，可以确定物体到步行机器人的距离。图7.1显示，当红外检测器接收频率大于38.5kHz的信号时，红外检测器电子滤波器的灵敏度随着频率的增大而降低。即电子滤波器使红外检测器在这些频率下检测红外线的能力降低。换一种思考方式就是如果你要想物体被这些灵敏度低（即检测距离近的）的红外线检测到，你必须把物体移到更靠近你的机器人。既然检测器的灵敏度变低，就得使用更亮的红外线使检测器能够看到物体，或者让物体更近。

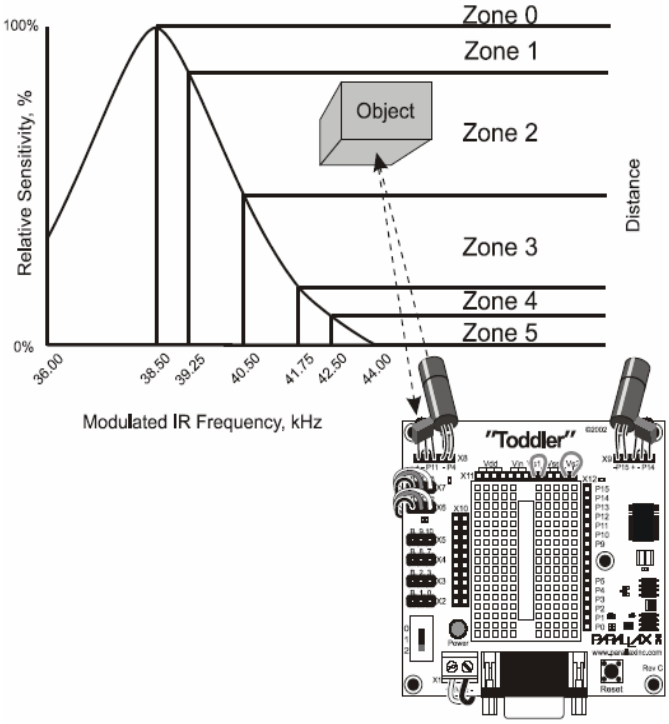


图7-1 不同频率红外线下的相对灵敏度

图7-1比较了红外检测器在不同频率红外线下的相对灵敏度。图的右半部分显示红外检测器的相对灵敏度与检测距离的关系。随着检测器灵敏度随着频率增高而降低，物体就必须

离光源更近才能被检测到。为什么是更近呢？原来当发射更高频率的红外信号使红外检测器的灵敏度降低时，就像给红外检测器撞上了一个更黑的透镜。就像手电光束照到离你更近的物体反射的光线越亮一样，从更近物体反射回的红外线对于红外检测器来说也越强。

图7-1右边部分显示了如何用不同频率的红外线来确定被检测物体所处的区域。首先用从38.5kHz的频率，可以检测出区域1到5是否有物体。如果没有物体被检测到，那它一定位于检测器可检测范围（区域0）之外。如果检测到有物体，再用39.25kHz的频率检测，可以收集到第一个关于距离的数据。如果38.5kHz检测到物体，而39.25kHz没有，那物体就一定位于区域1。如果物体被这两个频率检测到，而40.5kHz没有检测到，那物体就位于区域2。如果这三个频率都检测到物体，而41.75kHz没有，物体肯定位于区域3。如果上面四个频率都检测到物体，而42.5kHz没有，那物体就位于区域4。如果上述所有频率都检测到物体，那该物体就位于区域5。

本章介绍的频率扫描技术对于步行机器人来说非常好，所需的元器件成本也只是通常的红外距离检测传感器的零头。当然，用这种方法检测的精度也只是通用红外距离检测传感器的零头。对于那些需要一点距离感知功能的步行机器人基本任务而言，比如跟踪另一个步行机器人，这个有趣的技术就足够了。随着给步行机器人增加一些低精度的距离感知能力同时，也给学生提供了一些关于滤波器和频率响应的概念介绍。

所需器件

本任务使用第六章图6.4所示的相同红外探测电路。

红外距离检测编程

对BASIC Stamp编程发射不同的频率信号要用到DO…LOOP循环。Counter（计数器）变量用于让FREQOUT命令发射不同频率信号进行检测。该程序介绍了数列（Array）的用法。数列在程序7.1中用来存储不同频率下红外检测器的输出。左边红外检测区域0的输出存储在变量L_values的0位，区域1的输出存储L_values的1位，如此类推，一直到区域5。同样右边红外检测区域的输出的测量结果用同样的方法存储在R_values中。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 7.1: IR Distance Gauge
' Test of infrared sensors to show distance measurement
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]-----------------------------------------------
LeftIRLED CON 4
RightIRLED CON 15
LeftDetector VAR IN11
RightDetector VAR IN14
' -----[ Variables ]-----------------------------------------------------
Counter VAR Nib ' Counting variable
L_values VAR Byte ' Vars for storing freq
R_values VAR Byte ' sweep IR detector outputs
IR_freq VAR Word ' Frequency argument
```

```
' -----[ Main Routine ]-----------------------------------------------------
DO
L_values = 0 ' Reset L_values and
R_values = 0 ' R_values to 0
' Load sensor outputs into L_values and R_values using a FOR..NEXT loop
' and a lookup table, and bit addressing
FOR Counter = 0 TO 4
LOOKUP counter,[37500,38250,39500,40500,41500], IR_freq
FREQOUT LeftIRLED,1, IR_freq
L_values.lowbit(counter) = ~LeftDetector
FREQOUT RightIRLED,1, IR_freq
R_values.lowbit(counter) = ~RightDetector
NEXT
' Display l_values and r_values in binary and ncd format.
DEBUG HOME, CR, CR, "Left readings Right Readings", cr
DEBUG " ",BIN8 L_values, " ", BIN8 R_values, cr
DEBUG " ",DEC5 NCD(L_values), " ", DEC5 NCD(R_values), CR
LOOP
```

该你了！

- 输入并执行程序7.1。
  该程序使用了调试终端，因此在程序执行时请将串口线与步行机器人连接。
  当步行机器人面向一个距离3到5厘米的墙壁时，调试终端应该显示类似图7.2的内　容。将步行机器人向墙壁移近或移远，调试终端显示的值也应当相应增大或者减小。每个“1”都代表一个区域，以便当你看到五个“1”时表示物体离步行机器人最近。
- 将步行机器人放到使其红外二极管朝向墙壁并距离约1厘米远的地方。左右的读数都应当是“4”或者“5”。如果不是，检查红外检测器是不是与红外二极管指向相同的方向。

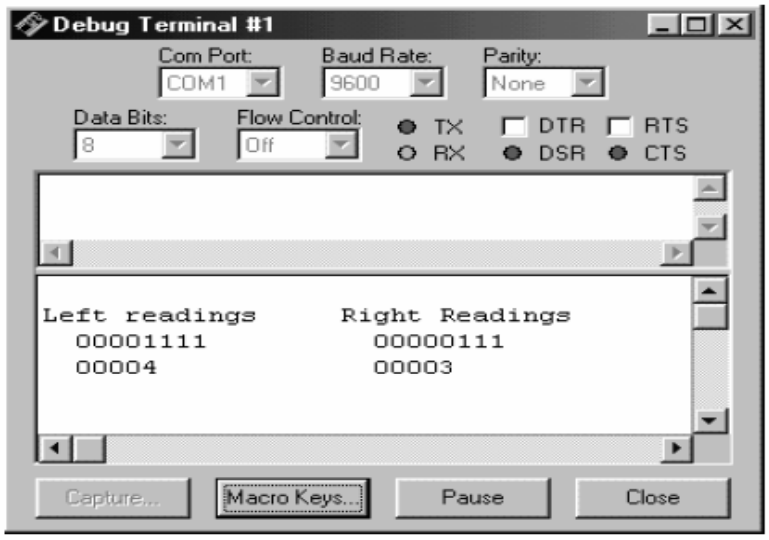


图7-2 频率扫描和用二进制与NCD格式显示的数据

- 逐步移动步行机器人，使其离墙壁越来越远。随着移动距离的增加，左右的读数应该逐

步减小到“0”。

- 如果两边的显示值总是“0”或者“1”，则表明你的程序或者接线有错误。此时，你必须拔下步行机器人的电池线，然后检查接线或者程序改正错误。

红外线最大的检测距离是20到30厘米，这依赖于墙面对红外光的发射率。对于一些更复杂的任务可能需要更为精确的距离测量结果，但就本任务而言，并不需要更高的精度。

距离测量程序是如何工作的

Counter 是一个四位的变量，用作 FOR…NEXT 循环的索引。FOR…NEXT 循环用于检查不同频率信号下红外检测器。L_values 和 R_values 变量存储不同频率信号下左右红外检测器的输出。每个变量存储了一个五位的测量结果。由于红外检测器的输出是检测的不同频率的红外线，IR_freq 就是一个用来存储不同频率值的变量。

主程序包括两个过程，一个用于频率扫描，另一个用于显示收集到的数据。频率扫表的第一步就是将 L_values 和 R_vlaues 设定为 0（初始化）。这很重要，因为变量中每一个位都将单独修改。然后，每个位都可以设置为“1”或者“0”，这依赖于红外检测器的检测结果。

```
DO
    L_values=0
    R_values=0
```

FOR…NEXT 循环就是进行频率扫描的地方。Lookup（查找）命令检查 counter（计数器）的值以确定将哪个频率值拷贝到 IR_freq 变量。当 counter 为 0 时，37500 被拷贝到 IR_freq，当 counter 为 1 时，38250 被拷贝到 IR_freq。随着 counter 的值被 FOR…NEXT 循环从 0 增加到 4，lookup 表中的值被逐一拷贝到 IR_freq。

```
FOR Counter=0 to 4
    LOOKUP counter, [37500, 38250, 39500, 40500, 41500], IR_freq
```

注意 lookup(查找)表以 37500（最灵敏）开始，以 41500（最不灵敏）结束。你可能要问，为什么 LOOKUP 表中的数据与图 7-1 中的数据不匹配。确实，如果 BASIC Stamp 能够在这些频率下发送出 50%占空比（即一个周期的脉冲信号中，高电平和低电平各占 50%）的脉冲序列串，测量频率将会与红外检测器滤波器指定的频率很好地匹配。然而，FEREQOUT 命令引入了其它的信号分量，这些分量信号影响了红外二极管发射的谐波信号的幅值。要预测可以使用的最优频率变量涉及到很高深的数学，超出了本书的范围。尽管如此，对一个给定距离的最优测量频率，仍然可以通过试验得到。我们这里使用的列表频率值是可靠的。

用 FREQOUT 命令发送当前 IR_freq 频率的信号，检查左边传感器的输出。然后，函数.lowbit()的参数变量用于确定 L_values 的每个相关位。当 counter 为 0 时，函数.lowbit(counter)提取到 L_values 的 0 位，当 counter 为 1 时，函数.lowbit(counter)提取到 L_values 的 1 位，如此类推。在将 IN8 的赋给 L_values.lowbit(counter)之前，即将输入值存入到 L_values 定义的位数列前，先用“非”运算符（~）将位的值进行转换。为 R_values 重复同样的过程。在 FOR…NEXT 的五次循环执行完后，红外数据位已经全部存入 L_values 和 R_values 中。

```
        FREQOUT LeftIRLED, 1, IR_freq
        L_values.lowbit(counter)=~LeftDetector

        FREQOUT RightIRLED, 1, IR_freq
        R_values.lowbit(counter)=~RightDetector
    NEXT
```

DEBUG 命令用了许多格式字符串来显示 L_values 和 R_values 的值。第一行用于显示文本标题，用于指示哪个读数对应右边的红外检测传感器，哪个读数用于对应左边的红外传感器。记住，左右是相对于你坐在步行机器人上而言。

```
DEBUG HOME, CR, CR, "Left readings          Right readings", cr
DEBUG " ", BIN8  L_values, "      ", BIN8   R_values, cr
DEBUG "  ", DEC5  NCD(L_values), "      ",DEC5 NCD(R_values),cr
```

第二行用于按二进制格式显示 L_values 和 R_values 的值。这样便于在物体距离变化时清晰地观察 L_values 和 R_values 的位变化情况。

第三行用于显示每个变量的 NCD 值。NCD 运算符返回一个反映变量中最重要位相关位置的值。当变量中所有位为 0 时，NCD 返回 0；当最右边的位为 1，其它所有为 0 时，NCD 返回 1；如果变量的位 1 包含一个 1，但其它所有位为 0 时，NCD 返回 2，依次类推。NCD 运算符实际上就是一个方便指示有多少 1 已经存入变量 L_values 和 R_values 的低位中的方法。更方便的就是 NCD 实际上直接告诉了你检测到的物体位于哪个区域。

当显示过程将数据发送到调试终端的任务完成后，程序控制返回到主程序(main 标记)。

该你了！

- 运行程序7.1，将步行机器人面向墙壁放置，并距离大约1.5厘米。为了获得最好的结果，在墙壁上贴一张白纸。
- 记录左边和右边传感器的读数。
- 慢慢将机器人移开。
- 每当其中的一个传感器的数据减小时，记录相应的距离。用这种方法，你可以确定每个红外传感器对检测区域。
- 如果一边的读数总是大于另一边的读数，你可以将返回较大值那边的红外二极管的指向稍稍向外调整一点点。例如，如果左边的红外传感器返回的读数总是高于右边的，尝试把左边的红外二极管和检测器的指向调整到离左边的红外传感器对稍远一些。

任务 2： 边缘检测

距离检测算法的一个应用就是检测边缘。例如，如果步行机器人在一个桌面上漫游行走，那它必须能够看到桌子的边缘，并在看到边缘后改变行走方向。为此，你需要将步行机器人的红外传感器对指向下方，使两个传感器对都能看到步行机器人前方的桌面。然后利用步行机器人的距离检测程序探测是否看到了桌面。当步行机器人靠近了桌子边缘时，一个或者两个检测器将开始报告它们不再看到桌面。这意味着步行机器人该从悬崖边转弯了。该程序在

浅色的桌面上效果很好。较黑的桌面会吸收更多的光线，反射红外线的能力较弱，因此效果不好。

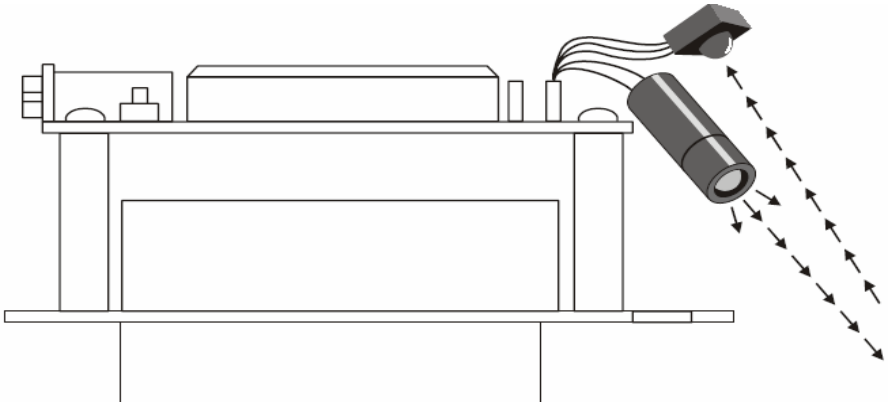


图7-3 IR发射器与接收器针对边缘检测的调整

- 将你的红外传感器对指向步行机器人前面的桌面，如图7.3所示。红外传感器对必须向下指向与水平面和机器人中心线各成45度角的地方。
- 在尝试程序7.2时，先用程序7.1进行下面的测试。
- 记录步行机器人直视台面时红外传感器对的输出值。让步行机器人直视桌面，如果红外传感器对的是3或者更大的值，表示红外检测器已经看到了它认为是桌面的东西。
- 让步行机器人看向桌子边缘以外的区域，记录红外传感器度的输出。如果这些输出值一直小于3，步行机器人就要准备运行程序7.2。
- 当机器人直视台面，如果步行机器人不能给出一个稳定而持续的3或者更大的读数，首先尝试调整红外传感器对的指向。同样，当机器人直视桌面以外的区域，而步行机器人也不能持续获得小于3的读数，也需要对红外传感器对进行一些附加的调整。
- 当机器人直视桌面能获得一个稳定3或者更大数输出，而看到边缘以外获得稳定的2或者更小的输出，步行机器人就可以准备执行程序7.2。

步行机器人执行程序7.2时，一定要密切观察机器人的行为。当机器人漫游走向桌子的边缘时要随时准备着将机器人拿起来。如果机器人快从边缘掉下，一定要在在它摔倒前把它拿起来。否则，你的步行机器人将成为一个烂宝贝。

当密切关注机器人避免从桌面掉下时，如果要拿起机器人就一定要从上面方向拿。否则机器人有可能看到你的手，而看不到边缘，从而导致机器人出现非预期的行为。

程序7.2用到了前面章节用过的forward,right_turn,left_turn和backward四个函数的修改版本，这四个函数从第二章开始每章都要用到。每个函数发出的脉冲数量都根据桌面的边缘进行了调整以获得更好的性能。Check_sensors子函数根据程序7.1红外距离测量的循坏进行距离测量。

执行和测试程序7.2。记住，随时准备着当机器人要从桌面掉落时机器人拿起来。

```
' -----[ Title ]-------------------------------------------------------------
' Toddler Program 7.2: Drop-off Detection
' Walking on a table avoiding the edges
' {$STAMP BS2}
' {$PBASIC 2.5}
' -----[ I/O Definitions ]-----------------------------------------------------
TiltServo CON 13 ' Tilt servo on P13
StrideServo CON 12 ' Stride servo on P12
```

```
' -----[ Constants ]-------------------------------------------------------
MoveDelay CON 25 ' in micrcoseconds
TiltStep CON 10 ' TiltServo step size
StrideStep CON 10 ' StrideServo step size
RightTilt CON 620 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 880
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
' -----[ Variables ]-------------------------------------------------------
FigureLoop VAR Nib
MoveLoop VAR Byte ' Loop for repeat movements
MoveLoopLimit VAR Byte
SubMoveLoop VAR Byte ' Loop for repeat submovements
SubMoveLoopLmt VAR Byte
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
MxCurrent VAR Word
Sx VAR Word
SxCurrent VAR Word
' -----[ EEPROM Data ]-----------------------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255
' ------ Movement Value Tables ------
'
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section
LeftSemicircle DATA 7, bLeftTurn, bLeftTurn, bForward, xx
```

```
RightSemicircle DATA 7, bRightTurn, bRightTurn, bForward, xx
WalkForward3 DATA 3, bForward, xx
WalkForward8 DATA 8, bForward, xx
' ------ Basic Movement Codes ------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' ------ Basic Movement Tables ------
'
' These tables can contain Movement Support Codes.
BasicMovements CON Forward
Forward DATA 1, TR, SL, TL, SR, xx
Backward DATA 1, TR, SR, TL, SL, xx
LeftTurn DATA 1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, TR, SL, TL, SR, xx
PivotLeft DATA 3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight DATA 3, TR, SL, TC, SR, TL, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
'----- Local Declarations --------------
counter VAR Nib ' For...next loop index variable
l_values VAR Mx ' R sensor vals for processing
r_values VAR Sx ' L sensor vals for processing
l_IR_freq VAR MxCurrent ' L IR freqs from lookup table
r_IR_freq VAR SxCurrent ' R IR freqs from lookup table
lEmitter CON 4
rEmitter CON 15
lDetector VAR IN11
rDetector VAR IN14
' -----[ Initialization ]----------------------------------------------------
OUTPUT lEmitter ' Set infrared emitters to outputs
OUTPUT rEmitter
OUTPUT 2
FREQOUT 2,500,3000 ' Signal program start
GOSUB ResetCC
' -----[ Main Routine ]------------------------------------------------------
Main: ' Main routine
' The command "gosub check_sensors" sends the program to a subroutine
```

```
' that loads distance values into l_values and r_values. So, when the
' fprogram returns rom the check_sensors subroutine, the values are
' updated and ready for distance based decisions.
GOSUB check_sensors
' The distances are checked for four different inequalities. Depending
' on the inequality that turns out to be true, the program either
' branches to the forward, left_turn, right_turn or backward navigation
' routine. The "3" value used below to test the boundary conditions
' may need to be changed depending upon the color of the walking surface
' and the angle of IR LEDs and detectors.
Boundary CON 2
IF l_values >= boundary AND r_values >= boundary THEN go_forward
IF l_values >= boundary AND r_values < boundary THEN left_turn
IF l_values < boundary AND r_values >= boundary THEN right_turn
IF l_values < boundary AND r_values < boundary THEN go_backward
GOTO main ' Repeat the process.
'----- Navigation Routines -------
go_forward: ' single forward pulse, then
Mx = Forward
GOSUB Movement
GOTO main ' go back to the main: label.
left_turn: ' eight left pulses, then
Mx = PivotLeft
GOSUB Movement
GOTO main ' go back to the main: label.
right_turn: ' eight right pulses, then
Mx = PivotRight
GOSUB Movement
GOTO main ' go back to the main: label.
go_backward: ' eight backward pulses, then
Mx = Backward
GOSUB Movement
GOTO main ' go back to the main: label.
' -----[ Subroutines ]------------------------------------------------------
' The check sensors subroutine is a modified version of Program Listing
' 6.1 without the debug Terminal display. Instead of displaying l_values
' and r_values, the main routine uses these values to decide which way to
' go.
check_sensors:
l_values = 0 ' Reset l_values and r_values to 0.
r_values = 0
' Load sensor outputs into l_values and r_values using a FOR..NEXT loop
' a lookup table, and bit addressing.
FOR counter = 0 TO 4
```

```
check_left_sensors:
LOOKUP counter,[37500,38250,39500,40500,41500],l_IR_freq
FREQOUT lEmitter, 1, l_IR_freq
l_values.lowbit(counter) = ~ lDetector
check_right_sensors:
LOOKUP counter,[37500,38250,39500,40500,41500],r_IR_freq
FREQOUT rEmitter, 1, r_IR_freq
r_values.lowbit(counter) = ~ rDetector
NEXT
' Convert l_values and r_values from binary to ncd format.
l_values = ncd l_values
r_values = ncd r_values
' Now l_values and r_values each store a number between 0 and 5
' corresponding to the zone the object is detected in. The program can
' now return to the part of the main routine that makes decisions based
' on these distance measurements.
RETURN
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
' or
' Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.
Movement:
IF Mx < BasicMovements THEN SetupMovement
MxCurrent = Mx ' setup to use submovement table
MoveLoopLimit = 1
GOTO StartMovement
SetupMovement:
READ Mx, MoveLoopLimit ' read movement table repeat count
MxCurrent = Mx + 1
StartMovement:
FOR MoveLoop = 1 to MoveLoopLimit
Mx = MxCurrent ' Mx = start of movement table
DEBUG DEC Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,CR
IF Mx < BasicMovements THEN MovementLoop
' skip if movement table
SxCurrent = Mx ' SxCurrent = submovement index
GOTO StartSubMovement ' enter middle of loop
MovementLoop:
READ Mx, SxCurrent ' read next submovment byte
Mx = Mx + 1
IF SxCurrent = xx THEN MovementDone
```

```
' skip if end of list
DEBUG " ", DEC SxCurrent, " movement",CR
LOOKUP SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,
PivotLeft,PivotRight],SxCurrent
' lookup submovement table index
StartSubMovement: ' start executing submovement table
READ SxCurrent, SubMoveLoopLmt
' read submovement table repeat
SxCurrent = SxCurrent + 1
FOR SubMoveLoop = 1 TO SubMoveLoopLmt
Sx = SxCurrent
DEBUG " ", DEC Sx, " submovement ", DEC SubMoveLoop, " of "
DEBUG DEC SubMoveLoopLmt,CR
SubMovementLoop:
READ Sx, Dx ' read next submovent action
Sx = Sx + 1
IF Dx = xx THEN SubMovementDone
' skip if end of list
GOSUB DoMovement ' execute movement
GOTO SubMovementLoop
SubMovementDone:
NEXT
IF Mx < BasicMovements THEN MovementLoop
' exit if submovement table
MovementDone:
NEXT
RETURN
DoMovement:
DEBUG " ", dec Dx, " action",cr
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,StrideCenter,
StrideRight]
' will fall through if invalid
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
```

```
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

别名变量

程序7.2中的边缘掉落检测程序相对数据内存而言是一个相当大的程序的开始。实际上，没有一点PBASIC的编程技巧，程序将不能编译。该技巧就是PBASIC具有给变量起别名的能力，因此一个变量可以使用另一个变量的存储空间。这允许程序用16个字的RAM空间运行程序。

程序7.2中下面的代码显示了如何给变量定义别名。

```
Counter     var     nib
l_values    var     Mx
```

```
r_values    var     Sx
l_IR_freq   var     MxCurrent
r_IR_freq   var     SxCurrent
```

第一个变量的定义正常，这是一个四位长的变量。后面的四个变量重用了不同的变量。作为别名变量，它们与元变量具有相同的字长。在使用别名变量时主要需要记住的是任何功相同一个存储空间的变量不能同时使用。换句话说，不能使用类似下面的语句。

```
l_values=1
Mx=2
```

因为程序中一个新的部分或者子函数使用原始的变量名不方便，因此别名被正常使用。PBASIC没有局部变量的概念，因此也就需要别名变量。

BASIC Stamp的集成开发环境（IDE）能够显示当前程序的内存映射。这提供了RAM和EEPROM的有用信息。步行机器人程序7.2的内存映射图显示在图7.4中。图中显示有5个字节的自由RAM。没有更多的了，但以经足够。这里包括了四个别名变量字的使用。如果这些变量不是用别名的方式定义，那么程序就需要8个额外的字节，还需要额外增加3个字节的RAM内存。

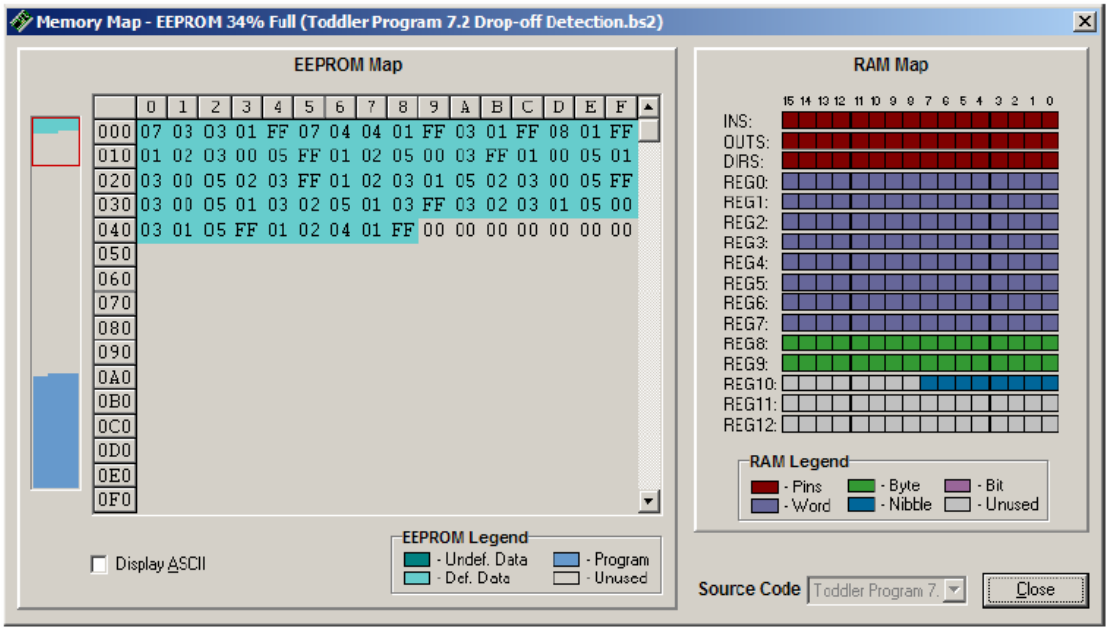


图7-4 EEPROM在程序7-2中的内存映射

别名使用起来要非常小心。当调试程序时，这是一个很大的问题来源。BASIC Stamp使用别名变量的优点是程序中只有很有限的变量数，因此问题发生时也很容易找到。

在这个例子中，包含Mx的变量初始集和用于程序的动作部分。只有Mx变量用在Movement（动作）函数的外面，它用来传递一个参数给该函数。包括l_IR_freq的别名变量集用于频率查询范围函数。既然这两个函数过程并不互相调用，因此很容易将这两个变量集合分隔开来。

边缘检测是如何工作的

现在我们已经讨论完别名变量问题，下面看看主程序是如何工作的。主程序做的第一件事是调用check_sensors子函数。注意check_sensors就是程序7.1去掉终端调试显示。这里也不再调试l_detect和r_detect的NCD值，取而代之的只是将这两个变量的值简单的用下面的语句转化为NCD值：

```
l_values=ncd l_values
r_values=ncd r_values
```

在调用完子程序check_sensors后，l_value和r_values取值为0到5之间的数。在程序从check_sensors子函数返回后，l_values和r_values被再次检查以确定桌子的边缘是否被检测到。

```
boundary CON 2
    IF l_values>= boundary AND r_values>= boundary  THEN go_foreard
    IF l_values>= boundary AND r_values < boundary  THEN left_turn
    IF l_values < boundary AND r_values>= boundary  THEN right_turn
    IF l_values < boundary AND r_values < boundary  THEN go_foreard
```

然后程序用相应的动作表索引赋给Mx变量，随后Movement函数使用表中的数值初始化步行机器人的腿部动作。boundary常量是边缘距离条件，该值可根据步行机器人行走桌面的颜色进行修改。该值必须设置正确以便步行机器人在向前移动时能够可靠的看到桌面。

现在安装在步行机器人电路板上的红外二极管和检测传感器的配置确实能检测到一个离边缘比较长的距离，因此步行机器人不会走到离桌子边缘的距离小于一步的地方。这也就意味着步行机器人需要一张相对比较大的桌子，有一个白色或者浅色的表面供步行机器人行走。

步行机器人在运动时将试图沿着桌面的边缘行走，尽管程序中并没有明确指明路径。理论上，如果步行机器人原来沿着垂直于某条边的方向行走，那么它将一直重复前进、后退，前进、后退这样的动作。但实际上这不可能发生，因为首先步行机器人的运动不是完全可重复的，随着它向前或者向后运动，它总会稍稍向一边转弯；其次两对传感器对的检测结果也会有差别，步行机器人可能用转弯来代替前进或者后退。

传感器本身也是导致机器人转向沿桌面边缘行走的另一个因素，比如，如果一个传感器比另一个传感器更灵敏。当然这种不同将总是在一个方向起作用，当另一个方向处理边缘检测时可能导致步行机器人向前多走一步。这不会导致步行器人走出桌面范围，因为程序总是试图让机器人停留在离边缘足够远的地方。额外的一两步不会造成问题。

特别当红外二极管和检测传感器指向正前方时，一个能成为问题的因素就是步行机器人将有一个有限的边缘视觉。步行机器人可能在沿着桌子边缘平行的方向运动时逐步漂向桌子的边缘。理论上靠近边缘一侧的传感器应该能够检测到边缘并让机器人转向远离边远的方向运动。但当桌子的边缘不是直线时常常会发生问题。将红外二极管和传感器对的指向稍微调向两侧一点可以消除这种可能发生的问题。

任务 3： 步行机器人的跟随运动

为了让一个步行机器人跟随另一个步行机器人行走，我们必须知道前面领路的机器人离跟随机器人的距离。如果跟随机器人落在了后面，就必须让它检测到并加速运动。如果跟随机器人离领路机器人太近，它也必须能够检测到并减速。如果跟随的距离正好，它就必须等待直到它再次检测到离领路机器人太远或者太近。

步行机器人与其同胞宝贝车不同，它采用离散的步伐运动，而不是轮子的小增量运动。在宝贝车的跟随运动中，使用了计算比例控制，而步行机器人就必须采用更离散的方式。可能采取的方式是比例步数，但由于步行机器人的动作精度的影响，使每个动作的细小改变对整个动作影响很小。另一方面，宝贝车能够在亚秒级（小于1秒的时间）时间使其一个或者两轮子移动小于1英寸的距离，而步行机器人走一步可能要花费1秒钟长的时间。

我们随后将证明，步行机器人的红外距离范围探测器对于跟踪另一个步行机器人非常有效。探测器输出的结果是离散值，且数据不大。如果不是这样，那就必须将检测数据转换到这种可以处理的量级。接下来的事情只是简单的选择合适的步伐类型和补数。

由于各种因素的影响，一个步行机器人将另一个步行机器人作为跟踪目标比较困难。为了提高红外传感器的检测效果，最好在目标步行机器人周围围上一个白色的盒子。该盒子可以是由普通的白纸或者硬纸板制成，并且可以用胶带或者其他方式步行机器人的框架上。盒子应当从步行机器人安装伺服电机的中央身体部分开始并能正好延伸到电路板的上方。红外传感器可以稍稍指向下方，以便它们能够检测到距其1步远的盒子中央部分。盒子不应当阻挡腿部的动作或者伺服电机的运动，而它也应当能够拓展几英寸尺寸大小。盒子夜不能太重获这太大，以致显著的改变机器人的重心位置，而不得不被迫调整步行行为。

对于一个步行机器人跟随另一个步行机器人而言，虽然这些改变并不是绝对需要的，但这些改变确实能够提高整个系统的性能。另外，漫游的区域必须没有障碍物和墙壁，跟随的步行机器人可能将把这些障碍物当作跟随对象。程序中没有对被探测的物体是否保持静止进行检测，虽然我们通过修改程序可能能够做到这一点。

对跟随机器人进行编程

程序7.3使用了一个额外的分支（branch）和查找（lookup）语句来基于距离探测结果调整机器人的位置。设计的动作就是让步行机器人对准要跟随的目标，该目标通常是另一个步行机器人，并且与目标保持一个离散的距离值。

该你了！

- 运行程序7.3。
- 让步行机器人朝向一张8.5×11英寸的白纸。步行机器人应当跟随移动试图与其保持一个平均距离。
- 试着移动白纸围绕步行机器人旋转。步行机器人应当跟随旋转。
- 尝试用白纸引导步行机器人随意走动。步行机器人应当能够跟随白纸运动。

```
' -----[ Title ]-----------------------------------------------------------
' Toddler Program 7.3: Shadow Walker
' Follows another Toddler with a piece of paper on his backside
' {$STAMP BS2}
```

```
' {$PBASIC 2.5}
' -----[ I/O Definitions ]-------------------------------------------------
StrideServo CON 12 ' Stride servo on P12
TiltServo CON 13 ' Tilt servo on P13
left_pin CON 4
right_pin CON 15
left_in VAR IN11
right_in VAR IN14
' -----[ Constants ]-------------------------------------------------------
MoveDelay CON 25 ' in micrcoseconds
TiltStep CON 20 ' TiltServo step size
StrideStep CON 20 ' StrideServo step size
RightTilt CON 630 ' Tilt limits
CenterTilt CON 750
LeftTilt CON 870
RightStride CON 650 ' Stride limits
CenterStride CON 750
LeftStride CON 850
' -----[ Variables ]--------------------------------------------------------
FigureLoop VAR Nib
MoveLoop VAR Byte ' Loop for repeat movements
MoveLoopLimit VAR Byte
SubMoveLoop VAR Byte ' Loop for repeat submovements
SubMoveLoopLmt VAR Byte
Pulses VAR Word ' Pulse variable
CurrentTilt VAR Word
CurrentStride VAR Word
NewValue VAR Word
Dx VAR Pulses
Mx VAR Word
MxCurrent VAR Word
Sx VAR Word
SxCurrent VAR Word
counter VAR Nib ' For...next loop index variable.
l_values VAR Mx ' store R sensor vals
r_values VAR Sx ' store L sensor vals
l_IR_freq VAR MxCurrent ' stores L IR frequencies.
r_IR_freq VAR SxCurrent ' stores R IR frequencies
' -----[ Movement Support Codes ]-----------------------------------------
'
' The following state tables are lists of movement state numbers.
' A xx indicates the end of a list.
' These are used with the Movement routine.
TL CON 0
```

```
TC CON 1
TR CON 2
SL CON 3
SC CON 4
SR CON 5
xx CON 255
' -----[ EEPROM Data ]----------------------------------------------------
'
' ------ Movement Value Tables ------
' These can be used with the Movement routine.
' The tables can contain Basic Movement Codes.
'
' Note: ALL movement tables must be in this section
TurnLeftForward DATA 1, bLeftTurn, bForward, xx
TurnRightForward DATA 1, bRightTurn, bForward, xx
PivotLeftForward DATA 1, bPivotLeft, bForward, xx
PivotRightForward DATA 1, bPivotRight, bForward, xx
BackwardPivotLeft DATA 1, bBackward, bPivotLeft, xx
BackwardPivotRight DATA 1, bBackward, bPivotRight, xx
Forward2 DATA 2, bForward, xx
Backward2 DATA 2, bBackward, xx
' ------ Basic Movement Codes ------
'
' Used in Movement tables.
' Referenced below using LOOKUP statement.
bFinish CON 0
bForward CON 1
bBackward CON 2
bLeftTurn CON 3
bRightTurn CON 4
bPivotLeft CON 5
bPivotRight CON 6
' ------ Basic Movement Tables ------
'
' These tables can contain Movement Support Codes.
BasicMovements CON Forward
Nop DATA 1, xx
Forward DATA 1, TR, SL, TL, SR, xx
Backward DATA 1, TR, SR, TL, SL, xx
LeftTurn DATA 1, TL, SR, TC, SL, TL, SR, TR, SL, xx
RightTurn DATA 1, TR, SL, TC, SR, TR, SL, TL, SR, xx
PivotLeft DATA 3, TL, SR, TC, SL, TR, SR, TC, SL, xx
PivotRight DATA 3, TR, SL, TC, SR, TL, SL, TC, SR, xx
Finish DATA 1, TR, SC, TC, xx
```

```
'----- Movement LOOKUP entries --------------
'
' These constants should reference the appropriate movement table.
' The constant syntax is lxry where x and y indicate the range from the
' left and right sensor respectively. A zero value indicates nothing
' is within range while a 5 indicates an object is within inches.
' In general, a 3 will be the closest desirable distance.
l0r0 CON Forward
l0r1 CON TurnRightForward
l0r2 CON PivotRightForward
l0r3 CON PivotRight
l0r4 CON RightTurn
l0r5 CON BackwardPivotRight
l1r0 CON PivotLeftForward
l1r1 CON Forward
l1r2 CON PivotRightForward
l1r3 CON PivotRight
l1r4 CON PivotRight
l1r5 CON BackwardPivotRight
l2r0 CON TurnLeftForward
l2r1 CON TurnLeftForward
l2r2 CON Forward
l2r3 CON Nop
l2r4 CON PivotRight
l2r5 CON BackwardPivotRight
l3r0 CON PivotLeft
l3r1 CON PivotLeft
l3r2 CON Nop
l3r3 CON Nop
l3r4 CON Nop
l3r5 CON BackwardPivotRight
l4r0 CON BackwardPivotLeft
l4r1 CON BackwardPivotLeft
l4r2 CON PivotLeft
l4r3 CON Nop
l4r4 CON Backward
l4r5 CON Backward
l5r0 CON BackwardPivotLeft
l5r1 CON BackwardPivotLeft
l5r2 CON BackwardPivotLeft
l5r3 CON BackwardPivotLeft
l5r4 CON Backward
l5r5 CON Backward
'----- Initialization ------------
```

```
OUTPUT 2 ' Declare outputs.
OUTPUT left_pin
OUTPUT right_pin
FREQOUT 2,500,3000 ' Beep at startup.
GOSUB ResetCC
'----- Main Routine --------------
main: ' Main routine
GOSUB check_sensors ' Distance values for each sensor
'debug "l",dec l_values,"r", dec r_values,cr
BRANCH l_values,[left0,left1,left2,left3,left4,left5]
left0:
LOOKUP r_values,[l0r0,l0r1,l0r2,l0r3,l0r4,l0r5],Mx
GOTO main_movement
left1:
LOOKUP r_values,[l1r0,l1r1,l1r2,l1r3,l1r4,l1r5],Mx
GOTO main_movement
left2:
LOOKUP r_values,[l2r0,l2r1,l2r2,l2r3,l2r4,l2r5],Mx
GOTO main_movement
left3:
LOOKUP r_values,[l3r0,l3r1,l3r2,l3r3,l3r4,l3r5],Mx
GOTO main_movement
left4:
LOOKUP r_values,[l4r0,l4r1,l4r2,l4r3,l4r4,l4r5],Mx
GOTO main_movement
left5:
LOOKUP r_values,[l5r0,l5r1,l5r2,l5r3,l5r4,l5r5],Mx
main_movement:
GOSUB Movement
GOTO main ' Infinite loop.
'----- Subroutine(s) -------------
check_sensors:
l_values = 0 ' Set distances to 0.
r_values = 0
' Take 5 measurements for distance at each IR pair. If you fine tuned
' frequencies in Activity #2, insert them in the lookup tables.
FOR counter = 0 TO 4
check_left_sensors:
LOOKUP counter,[37500,38250,39500,40500,41000],l_IR_freq
FREQOUT left_pin,1,l_IR_freq
l_values.LOWBIT(counter) = ~left_in
check_right_sensors:
LOOKUP counter,[37500,38250,39500,40500,41000],r_IR_freq
FREQOUT right_pin,1,r_IR_freq
```

```
r_values.LOWBIT(counter) = ~right_in
NEXT
l_values = NCD l_values ' Value for distance depends on MSB
r_values = NCD r_values
RETURN
' ----- Movement: Move feet using DATA table referenced by Mx -----
'
' Input: Mx = movement table index, table ends in xx
' or
' Mx = submovement table index, table ends in xx
'
' Note: All submovment tables come after the movment tables in this file.
Movement:
IF Mx < BasicMovements THEN SetupMovement
MxCurrent = Mx ' setup to use submovement table
MoveLoopLimit = 1
GOTO StartMovement
SetupMovement:
READ Mx, MoveLoopLimit ' read movement table repeat count
MxCurrent = Mx + 1
StartMovement:
FOR MoveLoop = 1 TO MoveLoopLimit
Mx = MxCurrent ' Mx = start of movement table
'debug hex Mx, " Movement ", dec MoveLoop, " of ", dec MoveLoopLimit,cr
IF Mx < BasicMovements THEN MovementLoop
' skip if movement table
SxCurrent = Mx ' SxCurrent = submovement index
GOTO StartSubMovement ' enter middle of loop
MovementLoop:
READ Mx, SxCurrent ' read next submovment byte
Mx = Mx + 1
IF SxCurrent = xx THEN MovementDone
' skip if end of list
'debug " ", hex SxCurrent, " movement",cr
LOOKUP SxCurrent,[Finish,Forward,Backward,LeftTurn,RightTurn,
PivotLeft,PivotRight],SxCurrent
' lookup submovement table index
StartSubMovement: ' start executing submovement table
READ SxCurrent, SubMoveLoopLmt
' read submovement table repeat count
SxCurrent = SxCurrent + 1
FOR SubMoveLoop = 1 TO SubMoveLoopLmt
Sx = SxCurrent
'debug " ", hex Sx, " submovement ", dec SubMoveLoop, " of "
```

```
'debug dec SubMoveLoopLmt,cr
SubMovementLoop:
READ Sx, Dx ' read next submovent action
Sx = Sx + 1
IF Dx = xx THEN SubMovementDone
' skip if end of list
GOSUB DoMovement ' execute movement
GOTO SubMovementLoop
SubMovementDone:
NEXT
IF Mx < BasicMovements THEN MovementLoop
' exit if submovement table
MovementDone:
NEXT
RETURN
DoMovement:
'debug " ", dec Dx, " action",cr
BRANCH Dx,[TiltLeft,TiltCenter,TiltRight,StrideLeft,
StrideCenter,StrideRight]
' will fall through if invalid index
RETURN
' ---- Movement routines can be called directly ----
TiltLeft:
NewValue = LeftTilt
GOTO MovementTilt
TiltCenter:
NewValue = CenterTilt
GOTO MovementTilt
TiltRight:
NewValue = RightTilt
MovementTilt:
FOR Pulses = CurrentTilt TO NewValue STEP TiltStep
PULSOUT TiltServo, Pulses
PULSOUT StrideServo, CurrentStride
PAUSE MoveDelay
NEXT
CurrentTilt = NewValue
RETURN
StrideLeft:
NewValue = LeftStride
GOTO MovementStride
StrideCenter:
NewValue = CenterStride
GOTO MovementStride
```

```
StrideRight:
NewValue = RightStride
MovementStride:
FOR Pulses = CurrentStride TO NewValue STEP StrideStep
PULSOUT TiltServo, CurrentTilt
PULSOUT StrideServo, Pulses
PAUSE MoveDelay
NEXT
CurrentStride = NewValue
RETURN
' ----- Move feet to initial center position -----
ResetCC:
CurrentTilt = CenterTilt
CurrentStride = CenterStride
FOR Pulses = 1 TO 100 STEP StrideStep
PULSOUT TiltServo, CenterTilt
PULSOUT StrideServo, CenterStride
PAUSE MoveDelay
NEXT
DoReturn:
RETURN
```

跟随机器人程序是如何工作的

主程序首先调用check_sensors子程序，子程序执行返回后，l_values和r_values都包含一个数值分别对应左右红外传感器对检测到的物体所处的区域。

```
Main:
    GOSUB check_sensors
```

下一行代码执行跳转，程序跳到许多LOOKUP语句中的一条。BRANCH语句使用左边红外传感器的状态，而LOOKUP语句则使用右边红外传感器的状态。这些语句用动作标的索引给Mx变量赋值，然后由Movement函数执行。

```
Branch l_values,[left0, left1, left2, left3, left4, left5]
    left0:
        LOOKUP r_values,[l0r0, l0r1, l0r2, l0r3, l0r4, l0r5],Mx
        GOTO main_movement
    left1:
        LOOKUP r_values,[l1r0, l1r1, l1r2, l1r3, l1r4, l1r5],Mx
        GOTO main_movement
    left2:
        LOOKUP r_values,[l2r0, l2r1, l2r2, l2r3, l2r4, l2r5],Mx
        GOTO main_movement
```

```
        left3:
            LOOKUP r_values,[l3r0, l3r1, l3r2, l3r3, l3r4, l3r5],Mx
            GOTO main_movement
        left4:
            LOOKUP r_values,[l4r0, l4r1, l4r2, l4r3, l4r4, l4r5],Mx
            GOTO main_movement
        left5:
            LOOKUP r_values,[l5r0, l5r1, l5r2, l5r3, l5r4, l5r5],Mx

    main_movement:
        GOUSUB Movement
```

LOOKUP语句中使用的那些值是在程序开头部分用CON这个常量定义符定义的。尽管也可以直接在LOOKUP语句中放入这些值，但这会使语句很长，而且这也会使我们很难看到在一个特定的状态下会采取哪个动作。而常量的定义为此提供了一种方法。现在我们很容易为一个如l3r3这样的特定状态关联一个如nop（即没有动作）这样的特定动作。类似地，l5r5指示步行机器人正好在某个障碍物的前面，而l0r0则指示在其检测范围内没有障碍物了。

一个动作执行完后，程序自动返回到main标记，并自动重复以上循环。

挑战课题！

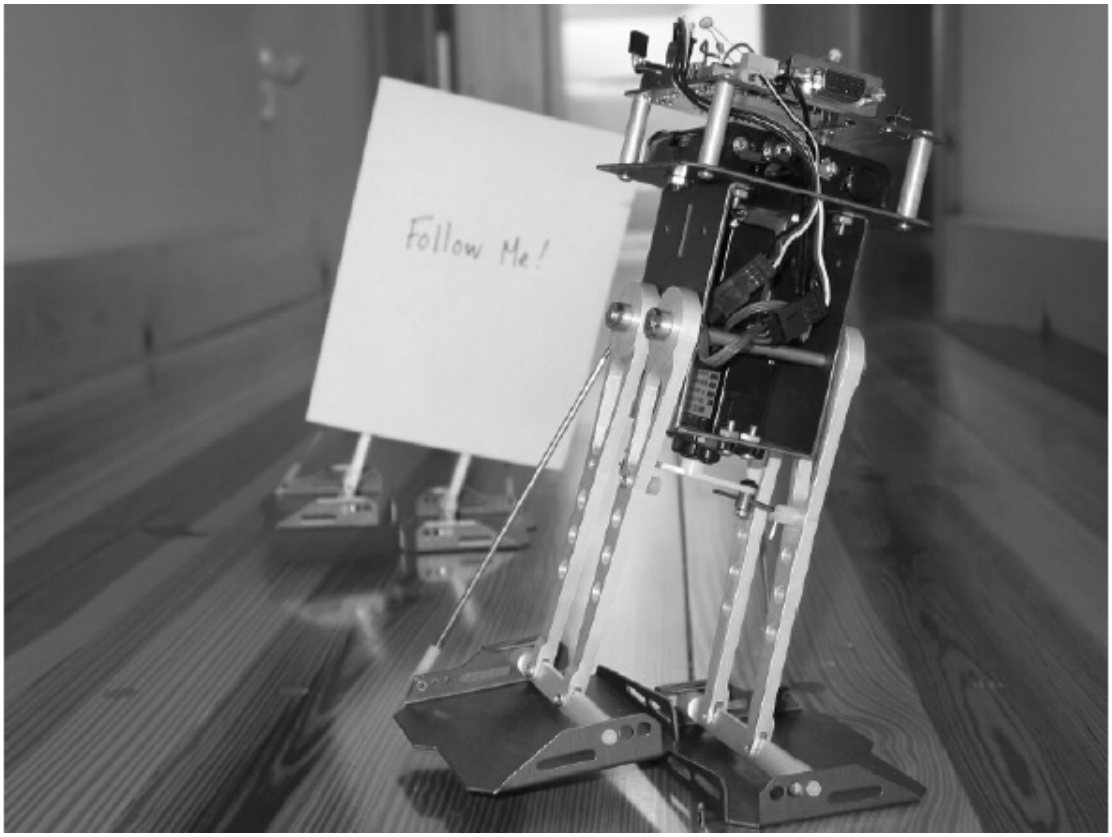


图 7.5 一个步行机器人跟随另一个步行机器人

图 7.5 显示了一个领路机器人后面有一个跟随机器人。领路机器人可以运行任意一个前面的程序，只需让调低其运动速度（增加 PAUSE 值或者减小 STEP 值），而跟随机器人执行程序 7.3。比例控制使跟随机器人成为一个很好的追随者。一个领路机器人可以带 2 个或者 3 个步行机器人形成一个队列，只需在领路机器人的后面增加一张 4×4 英寸的白纸。

- 如果你的班级有多个机器人，按照图 7.5 所示在领路机器人背后安装一个纸板。
- 如果你只有一个步行机器人，跟随机器人可以跟踪一张纸或者你的手，就像跟随领路机器人一样。
- 跟随机器人必须运行程序 7.3，程序不需任何修改。
- 如果有两个步行机器人运行各自的程序，将跟随机器人放在领路机器人的后面。只要跟随机器人没有被其它物体如手或者附近的墙壁吸引走，它将一直与领路机器人保持一个固定的距离跟随行走。

附录 A

步行机器人配件清单：

序号	产品名称		型号规格	数量
1	机械件	左脚		1
		右脚		1
		踝骨		2
		腿		4
		黄铜直角金属杆		2
		轴环		4
		步行机器人支架		1
		塑料垫圈	Φ5	12
		77mm 长金属圆杆		2
		黄铜夹头		2
		白色的塑料垫圈（E-Z 调节角质托架）		2
		步行机器人的顶板		1
		立柱		4
		电池盒		1
		伺服电机		2
		塑料垫圈	Φ3	8
		半圆形接头		4
		直径 2mm 不锈钢杆		2
2	标准件	盘头螺钉	M3x8	24
		盘头螺钉	M3x5	12
		盘头螺钉	M2x10	12
		盘头螺钉	M2x5	4
		盘头螺钉	M3x15	4
		平头螺钉	M3x8	10
		防松螺母	M3	4
		普通螺母	M3	18
		普通螺母	M2	12
3	工具	小号螺丝刀		1
		中号螺丝刀		1
		小号尖嘴钳		1

附录 B：步行机器人 PCB 原理图