# 设计实验室 3 6.01 –Fall 2011

# 全是胡萝卜，没有大棒

**目标：** 本实验室的总体目标是为机器人构建一种强大的能力，使其能够沿着由直线路径段组成的路线行驶。我们将分三个阶段来完成：

- 实现一个状态机，驱动机器人沿直线行进到在其里程计坐标系中指定的目标位置。
- 通过级联使机器人沿着一系列直线段行进

  生成目标点的机器与驱动至该目标点的机器

  目标；靶心；目的
- 通过添加一种“反射”机制，使机器人当其前进路径被阻塞时能够停下来，并等待直至路径畅通，然后再继续其行进轨迹，从而降低机器人对人类的危险。

## 1 材料

本实验应与搭档一起完成。每组搭档应有一台可靠的运行 SOAR 的实验笔记本电脑或个人笔记本电脑。运行 athrun 6.01 getFiles 以获取本实验的文件，这些文件将在 Desktop/6.01/designLab03/ 中；他们将为您提供这些资源：

**资源：**

- moveBrainSkeleton.py：大脑的主要文件，它导入了接下来的两个文件。
- ffSkeleton.py：用于您实现图形跟随状态机的模板。
- dynamicMoveToPointSkeleton.py：这是您实现驱动机器人移动到指定点的行为的模板。
- testFF.py：用于测试空闲状态下图形跟随**程序**。在运行大脑时未被使用。
- testMove.py：用于测试在空闲状态下向某点**移动的程序**。在运行大脑时未被使用。

**一定要把代码和数据寄给你的搭档。你们两个都需要把它带到第一次面试时。**

Some of the software and design labs contain the command athrun 6.01 getFiles. Please disregard this instruction; the same files are available on the 6.01 OCW Scholar site as a .zip file, labeled Code for [Design or Software Lab number].

本周，您应该阅读《**课程笔记》第 4 章**的全部内容。但在开始本次实验之前，请阅读：第 **4.2.1 节（级联组合**）、第 4.2.2 节（并行**组合）和**第 4.2.6 节（开关和复用）。

# 2 驱动

**目的:** 制作一个机器人大脑，能够沿着由一系列直线段组成的路径行进。

让我们通过创建一个由状态机组成的机器人大脑来探索状态机是如何组合的，这个状态机本身又是由更简单的状态机组成的，**如图 1** 所示。

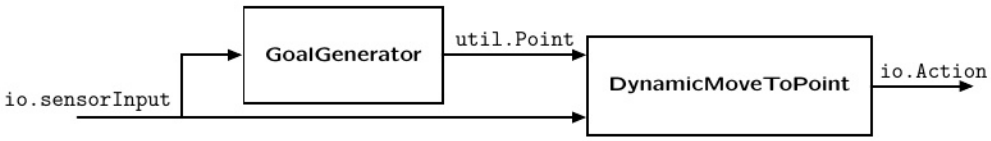



**图 1** 由两个更简单的状态机构建的机器人大脑（它是一个状态机）的架构。

我们将要构建的两个更简单的状态机是:

- **目标生成器**是一个状态机。在每一步中，输入是 io.SensorInput 的一个实例（其中包含机器人的声纳和里程计读数），输出是 util.Point 的一个实例，它表示机器人应该驱动前往的目标。
- **动态移动**到点的状态机接收一个输入，该输入是一个包含两个项目的元组：第一个是 util.Point 的实例，第二个是 io.SensorInput 的实例。在每一步中，该状态机生成一个 io.Action 的实例，该实例指定了向指定的 util.Point 迈进的一步。

**步骤 1. 详细指导**:（对于这部分不要定义任何新的类！）

使用 sm.Cascade、sm.Parallel、sm.Constant 和 sm.Wire（您可以在 6.01 主页的“参考”选项卡以及阅读材料中查阅相关**在线软件文档**）来构建一个**如图 1** 所示的组合机器。

目前，创建一个状态机（在**图 1** 中充当 **GoalGenerator** 的角色），它始终输出常量点 util.Point（1.0, 0.5）。另外，通过创建 **dynamicMoveToPointSkeleton.DynamicMoveToPoint** 类的一个实例来创建一个动态移动到点的状态机（**图 1**）。

将构建这个复合机器的代码键入 moveBrainSkeleton.py 中，替换掉第（行）处为 None 的内容。

```
mySM = None
```

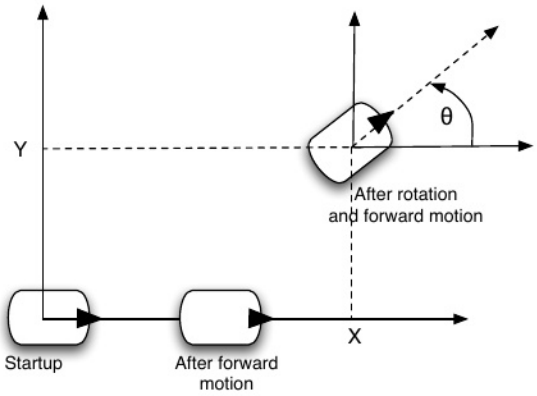
您可以通过在 SOAR 中运行 moveBrainSkeleton.py 来测试您的复合机器。选择 bigEmptyWorld.py 世界。每一步 **SOAR** 都应该打印一条消息（但不会有运动）

（机器人）。在后续步骤中，您将用移动机器人的代码来替换生成这些消息的代码。

## 2.1 里程计

在我们实现控制机器人的模块之前，我们必须了解坐标系以及机器人如何报告其当前位置。

机器人每个驱动轮上都有*轴编码器*，用于计算轮子的旋转次数（分数）。机器人处理器利用这些编码器计数来更新机器人在全局参考系中的估计位姿（一个表示位置和方向的术语）。该图展示了报告**机器人里程计**的参考系。




当机器人启动时，坐标系会被初始化，其原点位于机器人的中心，正x轴指向机器人的前方。现在您可以认为这个坐标系是涂在地面上的；当您移动或旋转机器人时，它会持续报告其在原始坐标系中的位姿。位姿具有x、y和θ三个分量：x和y是其在米级的位置，θ是向左的旋转角度，以弧度为单位。

请注意，如果您拿起真正的机器人并移动它，而轮子不转动，姿态读取值不会改变。它只知道它移动了，因为轮子转动了。并且要记住，里程计远非完美：轮子打滑，误差（尤其是旋转误差）会随着时间的推移而累积。

## 2.2 实用工具

我们将使用 util.Pose 类来表示机器人的姿态，您可以在**在线软件文档**中了解相关内容。**util 模块**还定义了一个 util.Point 类，用于表示二维空间中的点。假设 p0 = util.Point（x0, y0） 和 p1 = util.Point（x1, y1） ，那么

· p0.angleTo（p1） 返回从 p0 到 p1 的向量与 x 轴之间的角度（以弧度为单位）

· p0.distance（p1） 返回 p0 和 p1 之间的欧几里得距离，并且

· p0.isNear（p1, distEps） 返回一个布尔值，等于 p0.distance(p1) < distEps 。

假设 pose0 = util.Pose（x0, y0, theta0） ，那么
· pose0.point（） 返回一个等于 util.Point（x0, y0） 的点。

这些程序可能也会有用处:

- util. 近角度（角度 1、角度 2、角度容差）
- util. 修正角度（角度）使其在正负π之间
- util.clip(value, minValue, maxValue) 。

作为参考，您还应该查阅《*实验室基础设施指南*》。这是一个很好的去处，能提醒您了解机器人和大脑是如何工作的。

**警告！**小心角度运算。特别是要注意，π - θ（假设θ非常接近 -π + π/2，即 -179 度），但减去这些数会得到 2π - 2θ（假设θ 358°)! ）。使用 fixAnglePlusMinusPi 将接近 2π 的角度转换为接近 0 的等效角度。使用 nearAngle 来比较角度。

我们所有的角度操作程序都以**弧度来表示角度**。

## 2.3 驾车前往某地

**目的:** 实现**动态移动到目标点状态**机，该状态机引导机器人执行一项行动以到达目标点。

**详细指导:**

请记住，动态移动**到点状态机的输入**是一个元组（目标点，传感器），其中

- goalPoint 是 util.Point 的一个实例。这个点指定了机器人应该行驶的位置（在里程计的坐标系中）。
- sensors 是 io.SensorInput 的一个实例。您可以通过 sensors.odometry 获取当前的里程计值，sensors.odometry 是 util.Pose 的一个实例。

**动态移动**到点状态机的输出应该是一个io.Action实例，指定机器人下一步要采取的动作。例如，要创建一个具有前向速度0.2 m/s和旋转速度0.1 rad/s的动作，输出应该是io.Action（fvel=0.2, rvel=0.1）。

机器人应该从其当前位置沿着（近乎）直线行进至目标点。

**步骤 2.** 确定对于以下输入条件，**DynamicMoveToPoint** 状态机应输出何种动作。（答案可以是诸如“向前移动”或“向左旋转”之类的简单指令）

| 当前机器人位姿 | 目标点 | 行动 |
|---|---|---|
| （0.0，0.0，0.0） | （1.0，0.5） | |
| （0.0，0.0，π/2） | （1.0，0.5） | |
| （0.0，0.0，tan^(-1) 0.5） | （1.0，0.5） | |
| （1.0001，0.4999，0.0） | （1.0，0.5） | |

思考一种实现**动态移动**到目标点状态机的策略。解释该策略将如何驱动机器人移动到目标点。**您**的**状态机**的**输出应仅取决于输入**，而**无需使用状态**。

*检查 1.* 第 **3 周 2.1**：向一名员工解释您实施此行为的策略以及您对上述问题的回答。

**第三步**。在“动态移动到点骨架.py”文件中编写代码来实现您的策略。

- 首先，在 **idle 中测试您的代码**，如下所示。打开文件 dynamicMoveToPointSkeleton.py，注释掉

  从 soar.io 中导入 io 。

  并且取消注释

  导入 lib601.io 模块
- 现在，在 **idle 中测试代码**，通过运行 testMove.py 中的 testMove 程序来进行。

*自我检查 1.* 确保您明白此文件中测试用例的答案应该是怎样的，并且您的代码能正确生成这些答案。

- 在您的代码在**空闲状态下运行之后**，通过编辑 dynamicMoveToPointSkeleton.py 并注释掉代码，返回 **soar** 版本的 io.py 。

```
import lib601.io as io
```

and uncommenting

```
from soar.io import io
```

- 现在，使用 **soar 对其进行测试**，在模拟世界 bigEmptyWorld.py 中运行 moveBrainSkeleton.py 。**如果您收到一条错误**消息，**称“未连接到** soar”，则**需要按照上述**说明**更改 io 导入语句**。请注意，如果您用鼠标拖动机器人，机器人并不知道它已被移动。

若**要调试帮助**，请在“大脑”中查找“verbose = False”这一行并将其更改为“verbose = True”；这将导致 **SOAR** 打印出大量关于控制大脑的状态机的每一步的信息。

*自我检查 2.* 机器人总是从里程计读数（0, 0, 0）开始，所以它应该移动到靠近点（1.0, 0.5）的地方并停止。

# 3 时尚与复古共舞

**目的：** 让机器人沿正方形移动。通过定义一个新的状态机类 FollowFigure 来实现这一点（以及其他很酷的机器人动作），该类充当**目标生成器的角色**。

当我们初始化 FollowFigure 状态机时，会给它一个*航路点列表*，航路点是定义机器人应穿过的线性段序列的点。Follow-Figure 的任务是以 io.SensorInput 的实例作为输入，并生成 util.Point 的实例作为输出；它应该一开始就生成输入序列中的第一个点作为输出，并一直这样做，直到机器人实际的位置（通过 sensorInput.odometry 获取）接近该点；一旦机器人接近目标点，该机器就应该切换到生成下一个目标点作为输出，等等。即使机器人接近了最后一点，该机器也应该继续生成该点作为输出。

例如，如果我们使用下面的 FollowFigure 实例作为我们的目标生成器，它应该会导致机器人沿正方形移动。

```
squarePoints = [util.Point(0.5, 0.5), util.Point(0.0, 1.0),
                util.Point(-0.5, 0.5), util.Point(0.0, 0.0)]
FollowFigure(squarePoints)
```

如果机器人试图跟随上方的正方形，并且“FollowFigure”状态机从起始状态开始，并在接下来的顺序中看到里程计位姿作为输入，那么它在每一步的下一个状态和输出应该是什么？

| 当前机器人位姿 | 状态 | 目标点 |
|---|---|---|
| （0.0，0.0，0.0） | | |
| （0.0，1.0，0.0） | | |
| （0.499，0.501，2.0） | | |
| （2.0，3.0，4.0） | | |

**详细指导：**

**步骤 4.** 在 ffSkeleton.py 中定义您的 FollowFigure 状态机类。通过运行 testFF.py 中的 testFF 程序在**空闲**状态下对其进行测试，testFF.py 中包含了上述的测试用例。

**第 5 步。**在 moveBrainSkeleton.py 中，将 FollowFigure 类的一个实例替换到整体控制架构中的 **GoalGenerator** 中，并在 bigEmptyWorld.py 中进行调试。不要让你的机器人走正方形，而是让它跳一个酷炫的舞蹈！

任务 *2.* 第三周 **2.2**：向一位工作人员展示模拟机器人沿正方形或其他有趣图形移动所产生的粘液轨迹。解释它为何具有这样的形状。给粘液轨迹截屏并保存下来以备面试。将您迄今为止的代码和粘液轨迹邮寄给您的搭档。

# 4 避开行人

**目的：** 您当前构建的模拟机器人试图沿着指定的图形移动，并且完全忽略了其声纳传感器。我们希望定义一种新的行为，这种行为试图沿着图形移动，但会监测前方的声纳，如果其中任何一个读数小于 0.3，就会停止，直到障碍物消失，然后继续沿着图形移动。

**详细指导：**

**第 6 步。**使用 sm.Switch 状态机组合器来制作一个为行人停止的机器人。在阅读材料的**第 4.2.6** 节中了解关于 Switch 的内容。请记住，io.Action（） 操作会导致机器人停止。这应该只需要*添加少量*的额外代码。*不要更改* DynamicMoveToPoint 的定义。

您可以通过拖动机器人来模拟测试这个大脑；其里程计读数不会反映出它已被拖动（就好像您捡起机器人并“绑架”了它），所以如果您在它前面移动一堵墙，它应该会停下来；如果您把它拖走，它应该会再次开始移动。

*Checkoff 3.* **Wk.3.2.3**: Demonstrate your safe figure-follower to a staff member. Mail your code to your partner.

可选：在真正的机器人上进行操作。尽量别撞到工作人员。

# 5 我是一名芭蕾舞女演员

将你的图形跟随器中的 squarePoints 替换为 secretDance，让你的机器人优雅地按照秘密信息的模式起舞。这可能需要一段时间……

# 6.01SC 《电气工程与计算机科学导论》 2011 年春季