一种基于运行时交互约束的COTS构件集成测试用例生成方法

高 静[1,2] 兰雨晴[1] 金茂忠[1]
(北京航空航天大学软件工程研究所 北京 100083)[1]
(内蒙古农业大学计算机与信息工程学院 呼和浩特 010018)[2]

摘 要 COTS(Commercial Off-The-Shelf)构件的源代码不可得,而且规约通常缺少交互语义信息,使得集成了COTS的构件软件系统无法采用基于源代码和基于规约的集成测试用例生成方法。提出基于运行时交互约束的COTS构件集成测试用例生成方法。方法在构件软件系统的初始功能测试用例执行过程中,获取COTS构件在系统上下文环境中的运行时接口交互约束信息,建立COTS构件与系统中其它构件的交互约束模型,根据建立的构件交互约束模型生成集成测试用例。当替换、更新集成的COTS构件时,采用生成的集成测试用例验证重新集成的COTS构件与系统其它构件交互的正确性。实现方法的自动化工具原型,并采用真实的系统实例对生成的测试用例的有效性进行了初步评估。

关键词 运行时交互,交互约束建模,基于构件软件,集成测试

Method of COTS Component Integration Test Case Generation Based on the Run-time Interaction Constraints

GAO Jing[1,2] LAN Yu-qing[1] JIN Mao-zhong[1]
(Software Engineering Institute, Beihang University, Beijing 100083, China)[1]
(School of Computer and Information Engineering, Inner Mongolia Agricultural University, Huhhot 010018, China)[2]

Abstract Source code of COTS (Commercial Off-The-Shelf) component is not available and specification is usually lack of interaction semantic information. It makes the software system, which integrates COTS components, unable to use the method of integration test case generation which is based on the source code and specification. A method of COTS components integration test case generation based on the run-time interaction constraints was proposed. During the execution process of preliminary functional test cases for the whole system, it captures the interaction behavior of COTS components in the software system context, and establishes the interaction constraints model of COTS component, which extracts the interface interaction constraints of COTS component. The integration test cases can be generated automatically from the component interaction constraints model. Using the test cases generated, it is able to verify interaction of components when the COTS need to be updated or replaced. We had developed a tool. In addition to that, a preliminary evaluation for effectiveness of this method had also been conducted on a system.

Keywords Runtime interaction, Interaction constraints modeling, Component-based software, Integrating testing

构件软件系统由封装好的构件组成,构件间通过接口进行信息交互。构件系统中大部分的错误来源于构件间的交互[1,2]。集成测试是验证构件间交互正确性的重要方法,其设计人员根据构件接口规约中的交互约束信息来设计集成测试用例,集成测试用例通常为接口上的方法调用序列。完整的接口规约包括接口方法的符号信息和交互行为信息。符号信息说明了方法的名称、参数和返回值;交互行为信息说明了接口方法的交互约束,包括方法调用的顺序约束以及方法参数值的约束等。交互约束对于保证构件的正确使用和产生的集成测试用例的有效性非常重要,没有交互约束的指导,随机产生的方法调用序列可能为错误的序列或者在真实应用中不存在的序列。但现实情况是:通常构件的接口规约只有接口方法的符号信息,缺乏对交互约束的描述信息,而且由于COTS构件的源代码不可得,无法采用基于源代码的静态分析方法获得指导测试的规约信息。

本文结合动态分析技术和基于形式化模型的测试用例产生技术,针对以上问题,提出一个基于运行时交互约束的COTS构件集成测试用例生成方法。

1 与相关研究的不同点

构件行为约束信息包括方法调用顺序约束和数据约束两方面的信息。基于动态分析建模构件交互约束并应用于测试的研究分为两类:一类是建模方法调用顺序约束,并应用于单元测试和集成测试;另一类是对数据约束建模,应用于回归测

到稿日期:2008-04-28 本文受国家863基金项目(2006AA01Z186),2007发展基金项目"国产基础软件和硬件集成验证平台的开发与应用"资助。

高 静(1970-),女,博士生,主要研究方向为软件分析与测试,E-mail:gaojing@cse.buaa.edu.cn;兰雨晴(1969-),男,副教授,主要研究方向为软件测试和软件项目管理;金茂忠(1941-),男,教授,博士生导师,主要研究领域为软件工程。

试中。文献[3-5]将构件方法调用顺序约束建模为有限状态机，并用于集成测试用例的生成，但方法只建模了构件方法调用顺序约束，未建模构件间的数据约束，而且将整个构件软件系统表示为一个有限状态机，当系统规模较大时，存在状态爆炸的问题，无法用于较大规模的程序；文献[6,7]的方法将通过动态分析获得的数据不变量用于回归测试中，只考虑了数据约束，未考虑方法调用顺序约束。

本文方法在已有两类研究成果的基础上，在建模构件间接口交互约束和生成集成测试时，同时考虑了构件间交互的方法调用顺序约束和构件间交互的数据约束，提出了基于依赖网络有向图的构件交互约束模型和测试覆盖准则。在建立表示构件间的方法调用顺序约束的有限状态机时，根据构件提供的服务对方法调用序列分组，一个服务的执行对应一个有限状态机。一个服务所对应的方法调用序列规模很小，因此解决了将所有方法调用序列建模为一个有限状态机导致状态爆炸的问题。

2 基于运行时交互约束的 COTS 构件集成测试用例生成方法

本方法的思想为：在初始基于整体功能测试用例执行过程中，获取运行时 COTS 构件与系统其它构件之间的交互约束，并建立构件的交互约束模型；交互约束模型提取了构件在系统上下文环境中的接口交互约束，包括有限状态机表示的构件间交互方法调用顺序约束和布尔表达式描述的构件间交互的数据约束。在建立的交互约束模型基础上，基于形式化模型测试用例生成技术生成集成测试用例。更新或替换构件时，可采用生成的交互约束模型和集成测试用例完成回归测试。方法过程如图 1 所示。

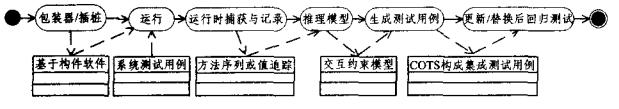


图 1 基于运行时交互约束的 COTS 构件集成测试用例生成方法

第一阶段，对构件软件系统中构件接口包装或插入探针，执行初始测试用例集。在初始测试用例执行过程中，通过包装器或探针获取构件间的交互信息，并记录方法调用序列和追踪交互的数据值。

第二阶段，根据记录的方法调用序列和交互的数据值的追踪来推理交互约束模型。

第三阶段，根据交互约束模型生成 COTS 构件与系统中其它构件的集成测试用例。

第四阶段，当集成的 COTS 构件被替换、更新或升级时，将生成的集成测试用例应用于 COTS 构件再集成后的回归测试，以验证替换或升级后 COTS 构件与系统中其它构件交互的正确性。

我们设计并实现相应于此方法的自动化测试工具原型 SafeCOTSIUp，初步实验结果显示了此方法的有效性。

2.1 定义

为叙述方便，假定基于构件软件系统为 A，包含一个集成的 COTS 构件 C，可以一般化为包含多个 COTS 构件。

定义 1（包含构件 C 的构件软件系统 A） 系统 A 为一个有向图表示的依赖网络，定义为一个三元组(SC,SR,SB)。其中 SC 表示图中的节点，对应系统所包含的构件集合；SR 为图中的边，连接系统中存在依赖关系的构件节点，节点 I 和节点 J 之间的边表示从构件 I 到构件 J 的控制和数据转换；SB 表示构件的交互约束，交互约束描述了构件间的数据和控制依赖，由对外提供服务接口的数据约束 DIC(Data Interaction Constraint)以及请求服务接口的控制约束 CIC(Control Interaction Constraint)组成。

定义 2（构件 C 与系统其它构件间的交互约束模型） 为系统 A 的依赖网络有向图的子图，由 COTS 构件节点的入度和出度覆盖的构件及构件间的交互约束(数据约束和控制约束)组成，如图 2 所示。采用 FSM 表示构件间的控制约束 CIC，采用布尔表达式建模构件间接口交互的数据约束 DIC。

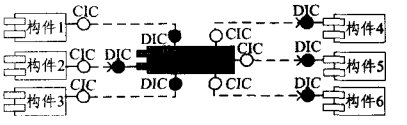


图 2 COTS 构件的交互约束模型

定义 3（方法调用顺序约束） 接口方法调用的先后顺序，例如 $C1.m1 -> C2.m2 -> C1.m2$。

定义 4（构件对象状态机） 采用有限状态自动机 FSM 建模构件间交互方法调用顺序约束，定义为一个 5 元组 $(Q, \Sigma, \delta, q0, QE)$。其中 Q 是一个有限状态集合，是构件的对象状态集；Σ 是输入字符集，是调用方法集，例如 $\Sigma = \{C1.m1, C2.m2, C2.m3, C3.m4\}$；$\delta$ 是状态转换函数，$\delta: Q \times \Sigma \rightarrow Q$，$\delta$ 建模状态间的变迁：$\delta(q1, m) = q2$ 表示调用方法 m 使得 FSM 从状态 $q1$ 转换为 $q2$；$q0$ 是开始状态，$q0 \in Q$；QE 是终止状态集，$QE \subseteq Q$。

定义 5（构件接口交互不变量） 构件接口交互不变量为构件接口方法入口处的前置条件和出口处的后置条件，以布尔表达式形式表示方法的数据值约束。

2.2 初始集成测试用例设计

集成测试主要测试构件之间的接口和接口交互，以及构件组合后形成的系统的整体功能。根据构件集成后形成的系统的功能/预期功能，设计测试大纲。测试大纲说明了集成后形成的系统的功能和提供的服务，在测试大纲的指导下设计测试场景。测试场景为包含外部输入和输出的输入/输出序列。每个测试场景对应一个测试用例，形成初始的集成测试用例集。

2.3 运行时交互行为信息获取

在执行初始集成测试用例时，观察被集成系统的外部输出是否为预期结果，同时观察系统内部构件接口交互行为，获得系统中构件间的方法调用序列和每个构件的输入输出序列。

根据实际情况可以通过对构件的接口进行字节码插桩，产生包装器，采用中间件的功能以及面向方面等技术，也可以采用一些工具，如调试器等收集运行时构件间交互行为信息。

运行时行为信息包括方法调用关系、方法占用的内存和 CPU 时间、构件间交互的数据值等。由于不考虑性能，我们收集的运行时追踪信息为构件间的方法调用序列和构件间交互的数据值。从运行时行为中提取两种与被监测 COTS 构件有关的信息：

(1) 构件对外提供服务接口的交互信息

由从系统其它构件发出到 COTS 构件的被请求的服务的信息和服务的输入数据，以及服务运行后产生的输出组成。

(2) 构件请求服务接口的交互信息

由当 COTS 构件的一个服务执行时从 COTS 构件发出的到系统其它构件的请求方法序列组成。

2.4 交互约束建模

构件交互约束模型建模了上下文环境中构件的交互约束。交互行为追踪记录了构件间接口方法调用序列和构件间交换的数据，从执行追踪自动提取交互约束模型。交互约束模型以布尔表达式形式描述了构件交互的值约束，以构件对象状态机 FSM 形式描述了方法调用顺序约束。

2.4.1 对外提供服务接口交互建模

获得对外提供服务接口的输入值和输出值，并将其集合输入到工具的推理引擎中。推理引擎中集成的 Daikon[8] 是一个程序不变量探测器，实现了不变量的动态探测。Daikon 提取的布尔表达式为对外提供服务接口的不变量信息，相应于对外提供服务接口的前置条件和后置条件。

例如，下面为一个实现栈功能的 Java 类 StackAr 程序的 Daikon 输出的一部分，显示了方法 constructor 和 isFull 的不变量信息。

```
Pre-conditions for the StackAr constructor
capacity >= 0
Post-conditions for the StackAr constructor
Orig (capacity) == this. theArray. length
this. topOfStack == -1
this. theArray [] elements == null
Post-conditions for the isFull method
this. theArray == orig (this. theArray)
this. theArray [] == orig (this. theArray [])
this. topOfStack == orig (this. topOfStack)
```

2.4.2 请求服务接口交互建模

使用有限状态自动机建模构件请求服务接口的方法调用顺序约束。对象的状态表示为状态机的节点，方法调用为节点间的转换。从正向样本追踪集推理 FSM 的著名的算法有 k-Tail[9]，k-Tail 算法的扩展算法[10,11]等。在实现原型工具的推理引擎时，我们选择了 Cook 和 Wolf[10] 提出的 k-Tail 扩展算法。因为 k-Tail 的两阶段推理算法会产生许多冗余的节点，增加了没有的追踪序列，而且不适用于处理循环节点。Cook 和 Wolf 提出的 k-Tail 扩展算法在两阶段基础上增加了第三个阶段，这一步骤合并了冗余的节点，并且增加了被观察到的追踪序列的一般化程度。

推理引擎从正向样本追踪集推理获得有限状态机模型，模型概括了当一个特定的请求接口方法执行时可能出现的方法调用序列。为每一个服务请求建立一个有限状态机，避免了状态爆炸的问题。

若要分析一个电子商务构件软件中的构件 Purchase 的交互行为，则 Purchase 构件管理用户发出服务请求，服务首先检索获得购物车和用户的详细信息，然后完成银行事务。如果事务失败，返回错误信息，否则修改库存内容。如果库存总量低，向生产商发出订购请求。服务的执行包含了与构件 cart，userMessage，transManager，storage，orderManager 和 catalog 的交互。观察运行时交互，获得的追踪文件如图 3 所示，图中数字表示同一方法重复执行的次数。其中

C = *cart. getCart*,

U = *userMessage. getUser*,

T = *transManager. bankTrans*,

S = *storage. removeProTotal*,

O = *orderManager. addProduct*,

G = *catalog. getItemDetail*.

$CUTSOS(3)\ G(3)$

$CUTSOSOSG$

CUT

$CUTSOSOG(2)$

$CUTS(5)\ G(5)$

$CUTSOS(2)\ G(3)$

$CUTS(3)\ OG(5)$

图 3 Purchase 构件的购买服务的交互追踪文件样例

将图 3 的交互追踪文件输入到基于扩展 k-Tail 算法的推理引擎，得到如图 4 所示的有限状态机。

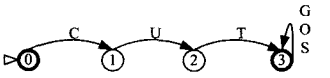


图 4 将图 3 的交互追踪输入到推理引擎获得的 FSM

2.5 COTS 构件集成测试用例覆盖准则和生成

COTS 构件交互约束模型表示了 COTS 构件与系统中其它构件间的交互约束。COTS 构件集成测试用例的覆盖准则为：覆盖 COTS 构件与系统其它构件间的所有交互约束。图 2 显示了必须覆盖的 COTS 构件与系统其它构件之间的所有交互，包括提供服务接口的交互以及请求服务接口的交互。提供服务接口交互的测试用例集覆盖了系统中其它构件到 COTS 构件的控制约束和数据约束。请求服务接口交互的测试用例集覆盖了从 COTS 构件到系统其它构件的控制约束和数据约束。

控制约束为有限状态机，可以采用不同的覆盖准则。简单的状态覆盖需要每个状态至少覆盖一次，复杂的路径覆盖需要每个路径至少遍历一次。我们采用转换覆盖原则，每个状态转换至少遍历一次。这个标准保证了交互构件的接口方法的执行。

数据约束为布尔表达式。对于布尔表达式的覆盖标准采用操作符覆盖准则，覆盖表达式中出现的所有操作符。单个操作符覆盖原则采用经典的边界值分析方法[12]，每个边界值、邻近边界值和边界值域范围内的都要执行一次，如表达式 person. getAge≥50，通过 3 个测试用例覆盖 person. getAge=50（边界值），person. getAge=51（邻近边界值）和 person. getAge>51（边界值范围内）。

2.6 工具原型 SafeCOTSIUp

实现方法的工具原型 SafeCOTSIUp，并集成到已有的自动化集成测试框架中。工具原型的总体架构如图 5 所示。

原型工具基于 Eclipse 插件技术开发，其中的交互行为获取及建模器由运行时交互记录器和推理引擎组成。运行时记录器记录运行时 COTS 构件的交互追踪，保存为交互追踪文件，并在数据库中建立相应的元数据；推理引擎（实现的推理引擎中集成了 Daikon[8]）推理获得构件间交互的数据约束和

控制约束，并存到数据库中。测试用例生成器从数据库中取出形成的交互约束模型，根据覆盖准则生成测试用例。

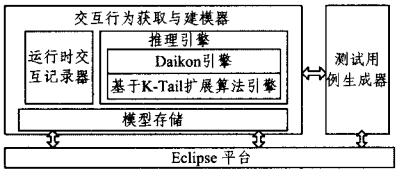


图5 工具原型的总体架构

3 初步实验对比结果

采用来源于 sourceforge 的电梯系统[13]进行了初步的实验。系统由若干 Java Bean 构件组成，各个构件实现不同的功能，这些功能共同完成电梯系统所具有的功能。关注电梯系统中的实现升降机功能的 Car 构件，采用产生包装器的方法，对 Car 构件进行了包装。在电梯系统的 62 个功能测试用例执行过程中获得 Car 构件与系统其它构件间的交互约束模型，并生成覆盖交互约束模型的集成测试用例 9 个。原测试用例集 62 个测试用例中的 21 个可以覆盖交互约束模型。

为了验证生成的集成测试用例的有效性和检测出系统集成故障的能力，选择基于故障的评估方法进行比较。比较标准为测试用例数和发现的集成缺陷数。基于故障的测试可以实现对软件测试充分性和测试集的评价，主要包括变异体测试和故障植入法[14]。由于构件通过接口与其他构件进行交互，因此接口变异体的设计是基于故障测试的研究重点。参考文献[14,15]故障植入的变异体设计方法，对 Car 构件的系统进行故障植入，主要集中在接口方法的参数和返回值。我们选择 15 种变异算子，随机地在调用 Car 构件的接口、被 Car 构件调用的构件接口和 Car 构件接口植入故障，共植入了 43 个故障，生成 43 个版本的变异程序，每个版本具有一个故障。在 43 个变异程序版本上分别执行原测试用例集(62)、生成的集成测试用例集(9)和原测试集中覆盖交互行为模型的覆盖测试集(21)。表 1 为各测试用例集故障检出百分比。

表1 交互约束模型产生的集成测试用例有效性对比

序号	变异体操作符	故障种子数	初始测试集(62)	覆盖测试集(21)	集成测试集(9)
1	RetStaDel	3	2(67%)	2(67%)	2(67%)
2	RetStaRep	4	2(50%)	2(50%)	2(50%)
3	FunCallDel	1	1(100%)	1(100%)	1(100%)
4	DirVarRepPar	2	2(100%)	2(100%)	2(100%)
5	DirVarRepLoc	5	4(80%)	4(80%)	4(80%)
6	DirVarRepExt	2	2(100%)	2(100%)	2(100%)
7	DirVarRepCon	5	5(100%)	5(100%)	5(100%)
8	DirVarAriNeg	1	1(100%)	1(100%)	1(100%)
9	DirVarLogNeg	2	1(50%)	1(50%)	1(50%)
10	DirVarBitNeg	2	2(100%)	2(100%)	2(100%)
11	ArgRepReq	2	2(100%)	2(100%)	2(100%)
12	ArgStcAli	1	1(100%)	1(100%)	1(100%)
13	ArgStcDif	4	3(67%)	3(67%)	3(67%)
14	ArgAriNeg	2	2(100%)	2(100%)	2(100%)
15	ArgIncDec	5	5(100%)	5(100%)	5(100%)

初步实验表明，原测试集 14.5%的测试用例即可覆盖 COTS 构件交互行为模型。由于 COTS 构件交互约束模型表示了 COTS 构件与系统其它构件的交互约束，因此尽管集成测试集是覆盖测试集的 43%，是初始测试集的 14.5%，但是和初始测试集以及初始测试集中的覆盖测试集检测出的故障数是一样的。

从实验结果可以看出，初始测试集、集成测试集和覆盖测试集都未检测出插入的所有故障。分析其原因，发现初始测试集并未覆盖系统的所有方法，只达到 30%语句覆盖和38.7%的条件覆盖，那些未检测出的故障为插入到初始测试集没有覆盖的代码中的故障。结果表明，将生成的集成测试应用于 COTS 构件更新或升级后的再集成回归测试，当初始测试集规模很大时，可以有效提高测试的效率。

结束语 本文提出一个 COTS 构件集成测试用例生成方法。在构件源代码不可得、规约缺少交互行为约束的情况下，自动生成 COTS 构件集成测试用例。初步实验结果表明，产生的 COTS 构件集成测试用例集可以有效地检测出构件交互错误。按照本方法建立的 COTS 构件接口交互约束模型，以及根据交互约束模型生成的集成测试用例，为 COTS 更新、替换后整个系统的集成与集成测试提供支持。

后续工作将选择规模更大的系统，实验将生成的集成测试用例用于 COTS 构件更新(或被替换)再集成后的回归测试中，分析交互行为模型和生成的集成测试用例用于回归测试的有效性。当前的实现主要关注顺序程序，进一步研究将本方法扩展到分布式并发程序中。

参 考 文 献

[1] Wu Y, Pan Dai, Chen M. Techniques for Testing Component-based Software [C]// Proceedings of the Seventh International Conference on Engineering of Complex Computer Systems. 2001

[2] Mariani L. A Fault Taxonomy for Component - based Software [C]// Elsevier, International Workshop on Test and Analysis of Component Based Systems (TACOS) Satellite Workshop at the European Joint Conferences on Theory and Practice of Software (ETAPS). 2003

[3] Li K, Groz R, Shahbaz M. Integration Testing of Components Guided by Incremental State Machine Learning [C]// Testing: Academic and Industrial Conference-Practice and Research Techniques. 2006:59-70

[4] Yuan Hai, Xie Tao. Substra: a framework for automatic generation of integration tests [C]// Proceedings of the 2006 International Workshop on Automation of Software Test. 2006

[5] Xie T, Notkin D. Tool-assisted unit-test generation and selection based on operational abstractions [J]. Automated Software Engineering Journal, 2006

[6] Harder M, Mellen J, Ernst M D. Improving test suites via operational abstraction [C]// Proceedings of the International Conference onsoftware Engineering. IEEE, 2003

[7] McCamant S, Ernst M D. Predicting problems caused by component upgrades [C]// Proceedings of the 9th European Software Engineering Conference(held jointly with 11th ACM SIGSOFT International Symposium on Foundations of Software Engineering). ACM Press, 2003:287-296

[8] Ernst M D, Cockrell J, Griswold W G, et al. Dynamically discovering likely program invariants to support program evolution [J]. IEEE Transactions on Software Engineering, 2001, 27(2): 99-123

[9] Biermann A, Feldman J. On the synthesis of finite state machines from samples of their behavior [J]. IEEE Transactions on

Computer,1972,21:592-597
[10] Cook J,Wolf A. Discovering models of software processes from event-based data [J]. ACM Transactions on Software Engineering and Methodology,1998,7(3):215-249
[11] Reiss S P,Renieris M. Encoding program executions [C]// Proceedings of the 23rd International Conference on Software Engineering. 2001:221-230
[12] White L,Cohen E. A domain strategy for computer program testing [J]. IEEE Transactions on Software Engineering,1991,17:703-711
[13] Sourceforge,sourceforge. net/projects/elevatorsim
[14] Ghosh S,Mathur A. Interface mutation to assess the adequacy of tests for components and systems [C]// Proceedings of Technology of Object-oriented Languages and Systems. IEEE Computer Society,2000:37-46
[15] Delamaro M E,Maldonado J C,Mathur A P. Interface mutation: An approach for integration testing [J]. IEEE Transactions on Software Engineering,2001,27(3):228-247

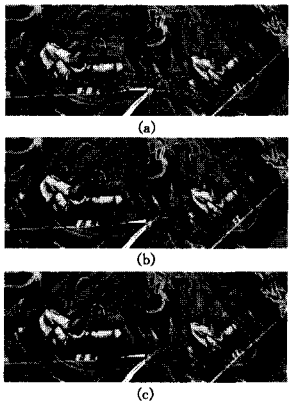
(上接第 225 页)

随着视角和仿射变形的增大,3 种算法的性能均明显下降。由于最后两帧图像的仿射变形太大,导致错误匹配过多,无法鲁棒地估计单应,因而不能准确判断正确匹配的数目,因此仅列出前 4 帧的匹配结果。从表中不难看出,在 3 种方法中,本文算法依旧获得了最多的匹配特征点对,说明本文算法对仿射和视角等宽基线匹配的适应能力有一定的提高,但是仍然不能够抵抗更大的仿射变形。第 3 帧与参考帧之间匹配结果如图 4 所示。

表 1 Graffiti 序列的实验结果

匹配方法	N	P	S	N	P	S	N	P	S
帧		1			2			3	
准确率	0.92	0.88	0.95	0.76	0.37	0.65	0.43	0.08	0.45
正确匹配数	679	406	575	302	64	187	45	5	5

N,P,S 分别代表 N-SVD,P-SVD 和 S-Dist 3 种算法。由于最后两帧的仿射变形太大,导致 3 种方法都失效,表中只给出了前 4 帧的匹配结果。



Graffiti 序列包含较大的视角变化换,这两帧图像之间约有 50°的拍摄视角。从(a)到(c)依次为 N-SDV,S-Dist,P-SVD 算法的匹配结果。

图 4 Graffiti 序列第 0 帧和第 3 帧的匹配结果

结束语 本文设计了一种基于尺度旋转不变特征的 SVD 宽基线自动匹配算法,该算法弥补了原 SVD 方法的不足,利用 SIFT 特征和归一化互相关度量,增强了邻近矩阵的度量能力,实现了算法对旋转、尺度缩放和光照变化等因素的不变性,使得 SVD 算法在宽基线匹配中更加鲁棒。对比实验表明,该算法比基于 SIFT 距离的匹配器和 Pilu 改进的 SVD 算法的性能有较大的提高,对于大的尺度、旋转和光照变化有较强的鲁棒性,即使对较大的仿射变换也有一定的适应能力。同时,本算法避免了参数选择的难题,通过 SVD 分解产生的匹配矩阵能够获得简约的一对一匹配结果,将相似性和排异性匹配原则自然地融为一体,简单有效地实现了宽基线自动匹配。

参 考 文 献

[1] Umeyama S. Least-squares estimation of transformation parameters between two point patterns. IEEE Transactions on Pattern Analysis and Machine Intelligence,1991,13(4):376-380
[2] Chui L,Rangarajan A. A new point matching algorithm for non-rigid registration. Computer Vision and Image Understanding,2002,89(2):114-141
[3] Belongie S,Malik J,Puzicha J. Shape matching and object recognition using shape contexts. IEEE Transactions on Pattern Analysis and Machine Intelligence,2002,24(4):509-522
[4] Shapiro S,Brady M. Feature-based correspondence:an eigenvector approach. Image and Vision Computing,1992,10(5):283-288
[5] Zitova B,Flusser J. Image registration methods:a survey. Image and Vision Computing,2003,21(11):977-1000
[6] Carcassoni M, Hancock R. Spectral correspondence for point pattern matching. Pattern Recognition,2003,36(1):193-204
[7] Lowe D. Distinctive image features from scale - invariant keypoints. International Journal of Computer Vision,2004,60(2):91-110
[8] Mikolajczyk K,Schmid C. A performance evaluation of local descriptors. IEEE Transactions on Pattern Analysis and Machine Intelligence,2005,27(10):1615-1630
[9] Scott G , Longuet - Higgins H . An algorithm for associating the features of two patterns// Proceedings of the Royal Society-Biological Sciences (Series B). London,UK,1991,244(1309):21-26
[10] Pilu M. A direct method for stereo correspondence based on singular value decomposition// Proceedings of IEEE Conference on Computer Vision and Pattern Recognition. San Juan,Puerto Rico,1997:261-266