An Introduction to the Ubitrack Library

Daniel Pustka, Manuel Huber, Peter Keitler

June 28th, 2007

Department of Informatics | Technische Universität München
Purpose of this Introduction

- After the introduction, everybody should
  - have a compiled Ubitrack on his hard drive
  - know how to write and run data flow descriptions in UTQL
  - know how to use Ubitrack in his/her own programs
Outline

- General Introduction
  - Spatial Relationship Graphs and Patterns
  - Library Overview
  - Download and compile session
  - Writing and running data flow descriptions in UTQL
  - Using the API in C++ and Java
  - Outlook
    - Ubitrack Server
Basics: Spatial Relationship Graphs

- Spatial Relationship Graphs describe the tracking situation
- Nodes represent coordinate systems
- Edges represent known transformations
  - Usually directed!

```
Tracker
     /\   /
  6DOF 6DOF
  /\   /\  
HMD  6DOF Object
```
Basics: Spatial Relationship Patterns

- Patterns describe structural properties of tracking algorithms
- Input edges are necessary preconditions
- Output edges describe the result of the algorithm
- A pattern describes how new relationships can be inferred
- Example: Pose concatenation or multiplication
Basics: Patterns and Data Flow Networks

- Data flow networks (DFNs) are formed by connected components that compute new relationships at runtime.
- DFNs only need to change when the topology of the underlying SRG changes.
- DFNs can be constructed by applying patterns.
Basics: Push- and Pull-Style Communication

- **Push-Style Interface**
  - asynchronous
  - all tracking hardware

- **Pull-Style Interface**
  - synchronous
  - request by timestamp
  - senders are continuous
Basics: Combining Multiple Measurements

- **Pull-Pull**

  1. 6.
  2. 3. 4. 5.

  ![Concatenation Diagram](image)

- **Push-Pull**

  1. 2. 3.

  ![Concatenation Diagram](image)

- **Push-Push**

  - not possible in general

  - Special case:
    - Synchronized push

    - If incoming measurements are synchronous

  ![Camera Diagram](image)
Push-To-Pull Conversion

- Buffering
  - Only valid for quasi-static relationships

- Averaging
  - Improves error characteristics
  - Only valid for static relationships

- Interpolation/Extrapolation
  - Linear, quadratic, SLERP

- Kalman Filter
  - Different motion models
  - Improves error characteristics
  - Provides accuracy estimates
Ubitrack: A Layered Approach

- Distribution
- Tracker Registration
- Middleware Layer

- Data Flow Framework
- Data Flow Components
- API
- Data Flow Layer

- Data Structures
- Tracking Algorithms
- SRG Algorithms
- Algorithmic Layer
Outline

- General Introduction
  - Spatial Relationship Graphs and Patterns
  - Library Overview
- Download and compile session
  - Writing and running data flow descriptions in UTQL
  - Using the API in C++ and Java
- Outlook
  - Ubitrack Server
Where to get Ubitrack

- The Ubitrack library is currently available using Subversion only
- If you have a login at the chair:
  - `svn+ssh://<username>@svnnavab.informatik.tu-muenchen.de/svn/ubitrack/trunk`
- Anonymous read-only access
  - `https://svnnavab.informatik.tu-muenchen.de/ubitrack/trunk`
- Checkout either using TortoiseSVN or
  `svn co <repositoryname> ubitrack`
Ubitrack Dependencies

- To build Ubitrack, the following is minimally required
  - A recent C++ compiler (GCC>=3.3 or VC>=7.1)
  - Python (required by the build system)
  - SCons (the build system)
  - Boost (indispensable for modern C++ programming)
- Today’s examples also require
  - OpenCV (for marker tracking, must have HighGui)
  - LAPACK (included for Win32)
  - Glut (for rendering)
- For the Java interface
  - SWIG (interface generator)
  - JDK
Building Ubitrack

- Ubitrack uses the SCons build system
  - platform-independent make replacement
  - provides configure functionality
  - written in python
  - “Makefiles” are python scripts 😊
- To build, simply go to the ubitrack directory and enter
  `scons all`
- Build options can be set using e.g.
  `scons OPENCV_PATH=C:\Code\OpenCV all`
  - options are remembered and need to be specified only once
  - print a list of all options: `scons --help`
Building Ubitrack: Checking the result

- Running `scons` should give the following lines with “yes”

  Checking for C++ header file boost/shared_ptr.hpp... yes
  Checking for dgesvd_() in C library lapack... yes
  Checking for cvGetQuadrangleSubPix( NULL, NULL, NULL ) in C++ library cv... yes
  Checking for cvNamedWindow( "xxx", 0 ) in C++ library highgui... yes
  Checking for glutInitWindowSize( 800, 600 ) in C++ library glut... yes

- If you get a “no” in the lines above, check dependencies and if necessary add build options
Running Unit Tests and Installing

- To check that everything works, enter
  `scons test`
- Finally install the libraries and components using
  `scons install-all`
- Now everything should reside in the `bin` and `lib` directories
Outline

- General Introduction
  - Spatial Relationship Graphs and Patterns
  - Library Overview
- Download and compile session
- Writing and running data flow descriptions in UTQL
- Using the API in C++ and Java
- Outlook
  - Ubitrack Server
The Ubiquitous Tracking Query Language

- Purposes of UTQL
  - Specifying the tracking environment: Spatial Relationship Graphs
  - Specifying capabilities of clients: Spatial Relationship Patterns
  - Specifying application queries

- Today’s focus: Specifying Dataflow Networks
  - normally automatically created by a Ubitrack server
    → still many problems to solve (by us)
  - manual creation is currently more reliable
UTQL Dataflow Specification: Outline

- A UTQL dataflow specifications in XML is enclosed in
  
  ```xml
  <UTQLResponse>
    ...
  </UTQLResponse>
  
  Each single data flow operation is represented as a pattern
  
  ```xml
  <Pattern>
    <Input>
      <Node .../> ... <Edge .../> ...
    </Input>
    <Output>
      <Node .../> ... <Edge .../> ...
    </Output>
    <DataflowConfiguration> ... </DataflowConfiguration>
  </Pattern>
```
Example 1: Specifying a Tracker

A tracker only has <Output> elements

Patterns and nodes:
- **names** are used locally
- **ids** are globally unique

Edges
- refer to node **names**

Nodes and edges have <Attribute>s:
- used for configuration

<DataflowConfiguration>
  <UbitrackLib class="ArtTracker"/>
</DataflowConfiguration>
tells the library how to instantiate the component
Example 2: Inverting the Transformation

```xml
<Pattern name="Inversion" id="Inv1">
  <Input>
    <Node name="A" id="Art1"/>
    <Node name="B" id="ArtBody1"/>
    <Edge name="AB" source="A"
      destination="B" graph-ref="Art1"
      edge-ref="ArtToTarget"/>
  </Input>

  <Output>
    <Edge name="BA" source="B"
      destination="A"/>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib
      class="PosePushInversion"/>
  </DataflowConfiguration>
</Pattern>
```

- **Input edges refer to an output edge of another pattern using**
  - **id** of the pattern: `graph-ref`
  - **name** of the edge: `edge-ref`

- **Note:** Node and edge **names** are specified by the implementation of a pattern and cannot be exchanged!
Assignment 1

- Display a snowman on a flat marker
- Make a UTQL file that connects the required components
- Run it using the utConsole

- Requires the following “SRG” and dataflow:
Hint: Frame Grabber Pattern

```
<Pattern name="FrameGrabber" id="FrameGrabber1">
    <Output>
        <Node name="Camera" id="Camera"/>
        <Node name="ImagePlane" id="ImagePlane"/>
        <Edge name="Output" source="Camera" destination="ImagePlane"/>
    </Output>

    <DataflowConfiguration>
        <UbitrackLib class="HighguiFrameGrabber"/>
    </DataflowConfiguration>
</Pattern>
```
Hint: Marker Tracker Pattern

```xml
<PATTERN name="MarkerTracker" id="MarkerTracker1">
  <INPUT>
    <NODE name="Camera" id="Camera"/>
    <NODE name="ImagePlane" id="ImagePlane"/>
    <EDGE name="Image" source="Camera" destination="ImagePlane"/>
  </INPUT>
  <OUTPUT>
    <NODE name="Marker" id="Marker">
      <ATTRIBUTE name="markerId" value="0x0272"/>
    </NODE>
    <EDGE name="Output" source="Camera" destination="Marker"/>
  </OUTPUT>
  <DATAFLOWCONFIGURATION>
    <UBITRACKLIB class="MarkerTracker"/>
  </DATAFLOWCONFIGURATION>
</PATTERN>
```
Hint: Renderer Pattern

```xml
<Pattern name="Renderer" id="Renderer1">
  <Input>
    <Node name="Camera" id="Camera">
      <Attribute name="virtualCameraFov" value="30"/>
      <Attribute name="virtualCameraNear" value="0.01"/>
      <Attribute name="virtualCameraFar" value="40"/>
      <Attribute name="virtualCameraWidth" value="640"/>
      <Attribute name="virtualCameraHeight" value="480"/>
    </Node>
    <Node name="Object" id="Marker">
      <Attribute name="virtualObjectX3DPath" value="snowman.x3d"/>
    </Node>
    <Edge name="PushInput" source="Camera" destination="Object"/>
    <Node name="ImagePlane" id="ImagePlane"/>
    <Edge name="Image" source="Camera" destination="ImagePlane"/>
  </Input>
  <DataflowConfiguration>
    <UbitrackLib class="RenderModule"/>
  </DataflowConfiguration>
</Pattern>
```
Assignment 2

- Create and run a dataflow that renders an ART-tracked object inside an ART-tracked HMD
Hint: Synchronized Push Multiplication Pattern

```xml
<Pattern name="Multiplication">
  <Input>
    <Node name="A"/>
    <Node name="B"/>
    <Node name="C"/>
    <Edge name="AB" source="A" destination="B"/>
    <Edge name="BC" source="B" destination="C"/>
  </Input>

  <Output>
    <Edge name="AC" source="A" destination="C"/>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib class="PoseSynchronizedPushMultiplication"/>
  </DataflowConfiguration>
</Pattern>
```
Outlook: Other Existing Components

- Push-Pull conversion: Buffer, LinearInterpolation
- Pull-Push conversion: Sampler
- Data type conversion: PoseSplit, FunctionalFusion
- Fixed transformations: StaticMeasurement
- Calibration: AbsoluteOrientation, SPAAM, RotHecKalmanFilter
- Sensor fusion: TimeComplementaryFusion, RotOnlyKalmanFilter
- Debugging: PrintSink, Recorder, Player
- Network transport: NetworkSource/Sink
- Trackers: ART, XSens, MarkerTracker
Outline

- General Introduction
  - Spatial Relationship Graphs and Patterns
  - Library Overview
- Download and compile session
- Writing and running data flow descriptions in UTQL
- Using the API in C++ and Java
- Outlook
  - Ubitrack Server
Getting API Documentation

- The Ubitrack library API documentation can be built using Doxygen and the supplied Doxyfile.
- The documentation is also available online (built nightly):
  http://campar.in.tum.de/personal/fardemo/ubidoc/
    - Username: ubitrack
    - Password: (empty)
- Some information can also be found in the Ubitrack Web:
  http://campar.in.tum.de/UbiTrack/WebHome
Instantiating a Ubitrack Data Flow Network

**C++**
- Get access to the SimpleFacade
  ```
  #include <Ubitrack/Facade/SimpleFacade.h>
  using namespace Ubitrack::Facade;
  ```
- Instantiate a SimpleFacade
  ```
  SimpleFacade f();
  ```
- Load a UTQL dataflow description
  ```
  f.loadDataflow( "test.utql" );
  ```
- Start the dataflow network
  ```
  f.startDataflow();
  ```
- At the end, stop it again
  ```
  f.stopDataflow();
  ```

**Java**
- Get access to the SimpleFacade
  ```
  import ubitrack.*;
  System.loadLibrary("ubitrack_java");
  ```
- Instantiate a SimpleFacade
  ```
  SimpleFacade f = new SimpleFacade();
  ```
- Load a UTQL dataflow description
  ```
  f.loadDataflow( "test.utql" );
  ```
- Start the dataflow network
  ```
  f.startDataflow();
  ```
- At the end, stop it again
  ```
  f.stopDataflow();
  ```
Sending Tracking Data to the Application

- To receive tracking data in an application, the data flow network needs to end in an `ApplicationPushSinkPose` component.

```xml
<Pattern name="ApplicationPushSink" id="PushSink1">
    <Input>
        <Node name="A"/>
        <Node name="B"/>
        <Edge name="Input" source="A" destination="B"/>
    </Input>
    <DataflowConfiguration>
        <UbitrackLib class="ApplicationPushSinkPose"/>
    </DataflowConfiguration>
</Pattern>
```
Getting the Tracking Data

C++
- Derive from SimplePoseReceiver

```cpp
class MyReceiver : public SimplePoseReceiver {
    public:
        void receivePose( const SimplePose& pose ) {
            double tx = pose.trans[0];
            double rw = pose.rot[3];
        }
};
```

- Register callback at the facade

```cpp
f.setPoseCallback( "PushSink1", &myReceiver );
```

Java
- Derive from SimplePoseReceiver

```java
class MyReceiver extends SimplePoseReceiver {
    public void receivePose( SimplePose pose ) {
        double tx = pose.getTrans()[0];
        double rw = pose.getRot()[3];
    }
};
```

- Register callback at the facade

```java
f.setPoseCallback( "PushSink1", myReceiver );
```

- The first parameter to `setPoseCallback` corresponds to the id of the `ApplicationPushSinkPose` component in the data flow network.
Notes

- Most `SimpleFacade` methods return `false` when an error occurred. In this case, `f.getLastErrorMessage()` returns a description.
- If you want the Ubitrack library to print what is going on, enable logging using
  - C++:
    ```cpp
    #include <Ubitrack/Util/Logging.h>
    Ubitrack::Util::initLogging();
    ```
  - Java:
    ```java
    ubitrack.initLogging();
    ```
- For C++ clients, an `AdvancedFacade` is also provided, which provides more flexible access to the data flow network.
- The `SimpleFacade` can also be used from Python 😊
Assignment 3

- Create a C++ or Java application that receives tracking data from the marker tracking.
- Don’t forget to write an appropriate UTQL file.
Outline

- General Introduction
  - Spatial Relationship Graphs and Patterns
  - Library Overview
- Download and compile session
- Writing and running data flow descriptions in UTQL
- Using the API in C++ and Java

Outlook
- Ubitrack Server
The Ubitrack Server

- Goal: Automatic data flow creation
- Automatically share tracking between different clients
- Queries for (unknown) objects with certain attributes
- Automatic reconfiguration of data flow at runtime, when new trackers and/or objects become available or disappear
The Ubitrack Architecture

![The Ubitrack Architecture Diagram]
Server SRG Specification Example

```xml
<UTQLRequest>
  <Pattern name="Art" id="Tracker001">
    <Output>
      <Node name="Art" id="Art">
        <Attribute name="ArtPort" value="5000"/>
      </Node>
    </Output>
  </Pattern>
  <Pattern name="ArtSheep" id="Tracker002">
    <Output>
      <Node name="Art" id="Art"/>
      <Node name="ArtTarget" id="ArtSheep">
        <Attribute name="renderable" value="true"/>
      </Node>
      <Edge name="ArtToTarget" source="Art" destination="ArtTarget">
        <Attribute name="type" value="6D"/>
        <Attribute name="mode" value="push"/>
        <Attribute name="artBodyId" value="1"/>
      </Edge>
    </Output>
    <DataflowConfiguration>
      <UbitrackLib class="ArtTracker"/>
    </DataflowConfiguration>
  </Pattern>
  <Pattern name="ArtSheep" id="Tracker003">
    <Output>
      <Node name="Art" id="Art"/>
      <Node name="ArtTarget" id="ArtHMD">
        <Attribute name="renderable" value="true"/>
      </Node>
      <Edge name="ArtToTarget" source="Art" destination="ArtTarget">
        <Attribute name="type" value="6D"/>
        <Attribute name="mode" value="push"/>
        <Attribute name="artBodyId" value="3"/>
      </Edge>
    </Output>
    <DataflowConfiguration>
      <UbitrackLib class="ArtTracker"/>
    </DataflowConfiguration>
  </Pattern>
</UTQLRequest>
```
Server Query Example

```xml
<PATTERN name="Query">
  <INPUT>
    <NODE name="Camera">
      <PREDICATE>id="ArtHMD"</PREDICATE>
    </NODE>
    <NODE name="Object">
      <PREDICATE>renderable="true"</PREDICATE>
    </NODE>
    <EDGE name="Input" source="Camera" destination="Object">
      <PREDICATE>type="6D"&amp;&amp;mode="push"</PREDICATE>
    </EDGE>
  </INPUT>
  <DATAFLOWCONFIGURATION>
    <UBITRACKLIB class="PosePrintSink"/>
  </DATAFLOWCONFIGURATION>
</PATTERN>
```
Thank you for your attention

- Please use Ubitrack in all your projects!
Replacement for the MarkerTracker

- If OpenCV does not work, use the following:

```xml
<Pattern name="TestSource" id="TestSource1">
  <Output>
    <Node name="A" id="Camera"/>
    <Node name="B" id="Marker0272"/>
    <Edge name="Output" source="A" destination="B"/>
  </Output>

  <DataflowConfiguration>
    <UbitrackLib class="TestSourcePose"/>
    <Attribute name="position" value="0 0 -0.6"/>
    <Attribute name="posnoise" value="0.3"/>
    <Attribute name="rotnoise" value="0.5"/>
  </DataflowConfiguration>
</Pattern>
```