汽车航迹推算研究进展

钱隆

武汉大学测绘遥感信息工程国家重点实验室

2023 年 2 月 28 日

1. 研究内容
2. 研究计划

1. 研究内容
   - Android 时间系统设计
2. 研究计划

1. 研究内容
   - Android 时间系统
   - 设计
2. 研究计划

System 类

```
public static long currentTimeMillis();
```

Returns the current time in milliseconds. Note that while the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger. For example, many operating systems measure time in units of tens of milliseconds.

Returns	
long	the difference, measured in milliseconds, between the current time and midnight, January 1, 1970 UTC.

System 类 I

```
public static long nanoTime ()
```

Returns the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds.
This method can only be used to measure elapsed time and is not related to any other notion of system or wall-clock time. The value returned represents nanoseconds since some fixed but arbitrary origin time (perhaps in the future, so values may be negative). The same origin is used by all invocations of this method in an instance of a Java virtual machine; other virtual machine instances are likely to use a different origin.
This method provides nanosecond precision, but not necessarily nanosecond resolution (that is, how frequently the value changes) - no guarantees are made except that the resolution is at least as good as that of currentTimeMillis().

System 类 II

Differences in successive calls that span greater than approximately 292 years (2^{63} nanoseconds) will not correctly compute elapsed time due to numerical overflow.
The values returned by this method become meaningful only when the difference between two such values, obtained within the same instance of a Java virtual machine, is computed.
The value returned by this method does not account for elapsed time during deep sleep. For timekeeping facilities available on Android see SystemClock.

Returns	
long	the current value of the running Java Virtual Machine's high-resolution time source, in nanoseconds.

SystemClock 类 I

Core timekeeping facilities.
Three different clocks are available, and they should not be confused:

- System.currentTimeMillis() is the standard "wall" clock (time and date) expressing milliseconds since the epoch. The wall clock can be set by the user or the phone network (see setCurrentTimeMillis(long)), so the time may jump backwards or forwards unpredictably. This clock should only be used when correspondence with real-world dates and times is important, such as in a calendar or alarm clock application. Interval or elapsed time measurements should use a different clock. If you are using System.currentTimeMillis(), consider listening to the ACTION_TIME_TICK, ACTION_TIME_CHANGED and

SystemClock 类 II

ACTION_TIMEZONE_CHANGED Intent broadcasts to find out when the time changes.

- uptimeMillis() is counted in milliseconds since the system was booted. This clock stops when the system enters deep sleep (CPU off, display dark, device waiting for external input), but is not affected by clock scaling, idle, or other power saving mechanisms. This is the basis for most interval timing such as Thread.sleep(millls), Object.wait(millis), and System.nanoTime(). This clock is guaranteed to be monotonic, and is suitable for interval timing when the interval does not span device sleep. Most methods that accept a timestamp value currently expect the uptimeMillis() clock.

SystemClock 类 III

- elapsedRealtime() and elapsedRealtimeNanos() return the time since the system was booted, and include deep sleep. This clock is guaranteed to be monotonic, and continues to tick even when the CPU is in power saving modes, so is the recommend basis for general purpose interval timing.

SystemClock 类

```
public static long elapsedRealtimeNanos ()
```

Returns nanoseconds since boot, including time spent in sleep.

Returns	
long	elapsed nanoseconds since boot.

SensorEvent 类

```
public long timestamp
```

The time in nanoseconds at which the event happened. For a given sensor, each new sensor event should be monotonically increasing using the same time base as SystemClock.elapsedRealtimeNanos().

GnssClock 类

```
public long getTimeNanos ()
```

Gets the GNSS receiver internal hardware clock value in nanoseconds.
This value is expected to be monotonically increasing while the hardware clock remains powered on. For the case of a hardware clock that is not continuously on, see the getHardwareClockDiscontinuityCount() field. The GPS time can be derived by subtracting the sum of getFullBiasNanos() and getBiasNanos() (when they are available) from this value.
Sub-nanosecond accuracy can be provided by means of getBiasNanos().
The error estimate for this value (if applicable) is getTimeUncertaintyNanos().

GnssClock 类

```
public long getFullBiasNanos ()
```

Gets the difference between hardware clock (getTimeNanos()) inside GPS receiver and the true GPS time since 0000Z, January 6, 1980, in nanoseconds.
This value is available if the receiver has estimated GPS time. If the computed time is for a non-GPS constellation, the time offset of that constellation to GPS has to be applied to fill this value.
The value is only available if hasFullBiasNanos() is true.
The error estimate for the sum of this field and getBiasNanos() is getBiasUncertaintyNanos().
The sign of the value is defined by the following equation:

$$local\ estimate\ of\ GPS\ time = TimeNanos - (FullBiasNanos + BiasNanos)$$

GnssClock 类

```
public double getBiasNanos ()
```

Gets the clock's sub-nanosecond bias.
See the description of how this field is part of converting from hardware clock time, to GPS time, in getFullBiasNanos().
The error estimate for the sum of this field and getFullBiasNanos() is getBiasUncertaintyNanos().
The value is only available if hasBiasNanos() is true.

GnssClock 类

```
public long getElapsedRealtimeNanos ()
```

Returns the elapsed real-time of this clock since system boot, in nanoseconds.

The value is only available if hasElapsedRealtimeNanos() is true.

1 研究内容
- Android 时间系统
- 设计

2 研究计划

尽可能遍历试验停车场的所有路径。

- 手机型号.
- 手机在车里的位置.
- 手机在车里的姿态.
- 行车速度.
  - 最低速.
  - $5\,\mathrm{km\,h^{-1}}$.
  - $10\,\mathrm{km\,h^{-1}}$.
- 行车路线.
  - 直行.
  - 倒行.
  - 左转.
  - 右转.
  - 上坡.
  - 下坡.

1. 研究内容
2. 研究计划

1. 数据集和模型间的交叉验证
   - 基于 ai-imu-dr 模型开源代码重新训练滤波模型（作者提供的预训练模型无法加载）
   - 基于论文 [1] 训练端到端模型
   - 实采数据以验证基于手机传感器通用性
2. 训练参考真值系统支持
   - 尝试基于手机 GNSS 传感器的 PPS 信号，实现同参考真值系统的无线同步
   - 尝试基于手机 IMU 传感器和参考真值系统传感器动态的初始同步

可能的创新点

1. 基于手机低精度传感器的航迹推算
2. 自采数据包含停车场环境倒车轨迹
3. 模型中包含自适应网络结构识别停车等行为重置参数

参考文献 I

[1] Quentin Arnaud Dugne-Hennequin, Hideaki Uchiyama, and Joao Paulo Silva Do Monte Lima. "Understanding the Behavior of Data-Driven Inertial Odometry with Kinematics-Mimicking Deep Neural Network". In: *IEEE Access* 9 (2021), pp. 36589–36619. ISSN: 21693536. DOI: 10.1109/ACCESS.2021.3062817.

谢谢