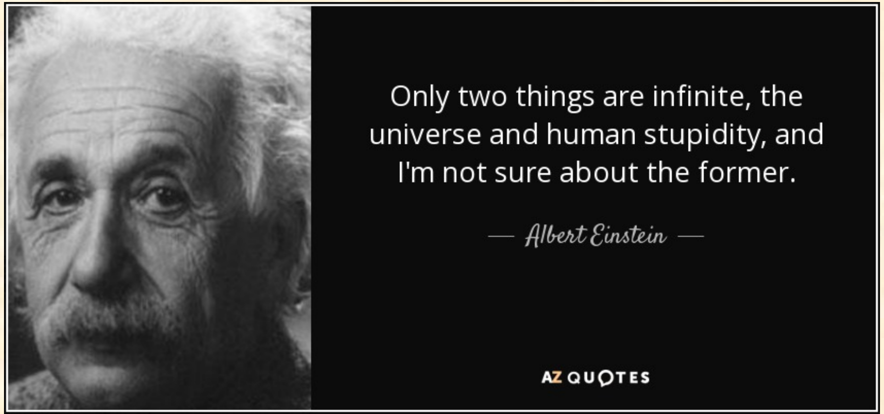
Prof. Philip Koopman

Carnegie Mellon University

Electrical & Computer ENGINEERING

Floating Point



Floating Point Math

Anti-Patterns:
- Not accounting for roundoff errors
  - Tests for floating point equality
- Not handling special values
- Float used if integer does the job
  - Not always good for “big” numbers

Floating Point Math:
- Exponent + Mantissa representation
  - 32-bit, 64-bit, others on some systems
- Roundoff errors due to finite number of mantissa bits
- Special values:
  Infinity, Not A Number (NaN), denorms, signed zero

IEEE Floating Point Format
Single Precision: 32 bits total

1 bit	8 bits	23 bits (with implicit leading 1.)
S	EXPONENT	MANTISSA

Value = (+/-) 1.Mantissa * $2^{(\text{Exponent}-127)}$

Sign: 0=positive; 1=negative

Exponent: 127 bias, radix 2
value is EXPONENT – 127

Mantissa: implicit 1.
value is 1.MANTISSA (binary)

Special zero value:
zero = 0x00000000

Roundoff Errors

- Rounding error due to limited bits
  - Mantissa: 24 bits (implicit leading one)
    - E.g.:   all zero mantissa bits ➔ $1.00000000000000000000000_2$
  - More than 24 bits of value won't fit
    - Converting int to float to int to float in a chain gives:
      0x72345673 ➔ 1916032640.0 ➔ 0x72345680 ➔ 1916032640.0
- Rounding error due to imprecise representation
  - IEEE 754 is radix 2, so decimal fractions can be inexact
    - Repeatedly add 0.1 to a 32-bit float and you get….
      0.1, 0.2, …, 2.799999, … , 49.999809, … , 99.999046
- Floating point comparison pitfall:
  - if (fa == fb) might not match due to rounding error
    - In some cases consider an "approximately equal" test, e.g.:
      if (fabs((fa - fb)/fa) < 0.0001)

```
Iteratively Add 0.1
iteration  1  value=0.100000
iteration  2  value=0.200000
iteration  3  value=0.300000
iteration  4  value=0.400000
iteration  5  value=0.500000
iteration  6  value=0.600000
iteration  7  value=0.700000
iteration  8  value=0.800000
iteration  9  value=0.900000
iteration 10  value=1.000000
iteration 11  value=1.100000
iteration 12  value=1.200000
iteration 13  value=1.300000
iteration 14  value=1.400000
iteration 15  value=1.500000
iteration 16  value=1.600000
iteration 17  value=1.700000
iteration 18  value=1.800000
iteration 19  value=1.900000
iteration 20  value=2.000000
iteration 21  value=2.100000
iteration 22  value=2.200000
iteration 23  value=2.300000
iteration 24  value=2.400000
iteration 25  value=2.500000
iteration 26  value=2.600000
iteration 27  value=2.700000
iteration 28  value=2.799999
iteration 29  value=2.899999
iteration 30  value=2.999999
              e=3.099999
              ue=3.199999
iteration 33  value=3.299999
iteration 34  value=3.399999
iteration 35  value=3.499999
```

Don’t Use Floats for Time!

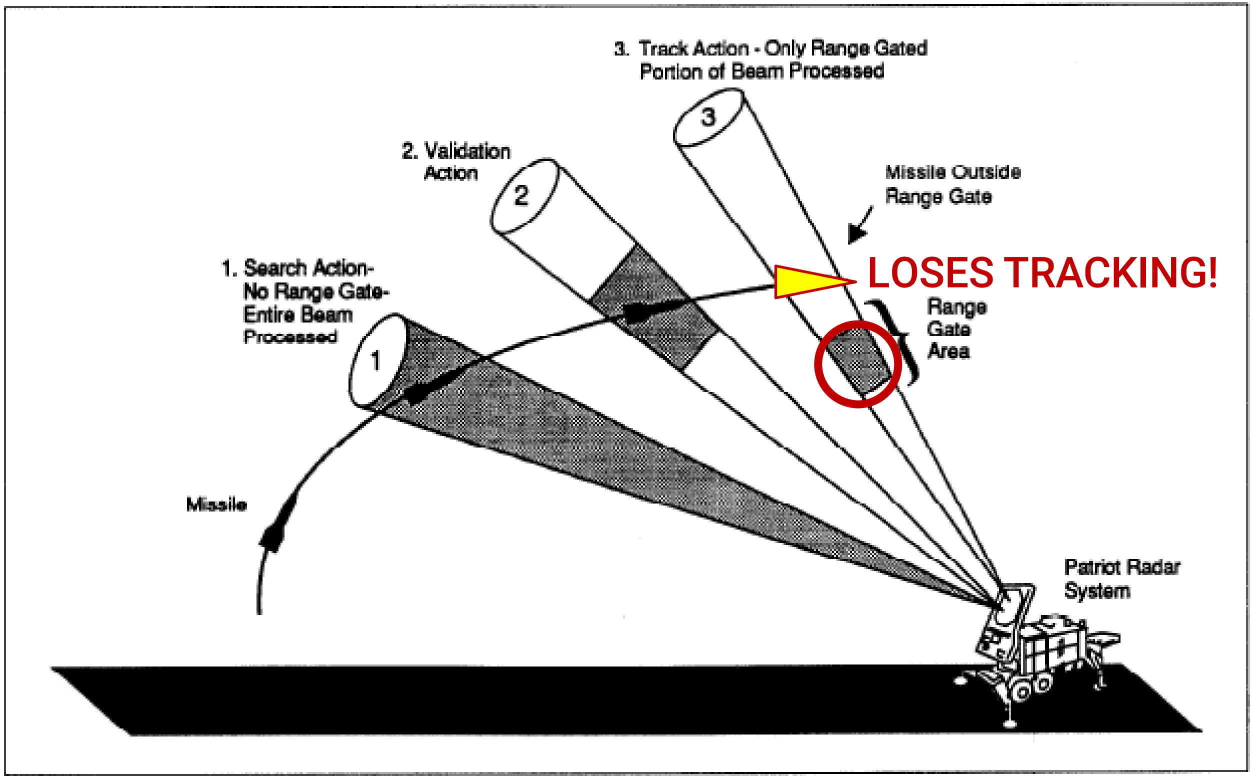
Carnegie Mellon University

- Patriot Missile mishap
  - 1991: Scud kills 28 American (Desert Storm)
  - http://www.fas.org/spp/starwars/gao/im92026.htm
    “after about 20 hours, the inaccurate time calculation becomes sufficiently large to cause the radar to look in the wrong place”
    - “Range gate” used to look where target is predicted to be next
    - Target track is lost if range gate is wrong, resulting in a miss
    - The incident happened 100 hours after the last system reset
- What was the root cause?
  - Patriot designed for aircraft and frequent mobile relocations
    - Scud missiles travel at Mach 5 (3750 mph); Patriot deployed in fixed location
  - Even a small round-off error matters when computing distance = velocity * time
    - Large accumulated base time and high velocity leads to a failure

Patriot Loss of Tracking Mishap

Carnegie Mellon University

Figure 5: Incorrectly Calculated Range Gate



[GAO/IMTEC-92-26]

- Time is integer 10ths of second
  - Converted to 24-bit fractional value for calculation
  - 0.1 seconds is not an "even number" = 0.00011001100110011001100110011001100…
  - At 100 hours, resultant round-off is 0.000000095 decimal [https://goo.gl/5ik1au]
- After 100 hours error was 0.344 seconds = 697 meters error
  (per GAO report)

Special Values

- **Inf: Infinity**
  - E.g., result when dividing by zero, or overflow
- **Denormalized**
  - Number smaller than smallest fraction
    - <$\sim 10^{-45} \ldots \sim -10^{-38}$ No implicit leading 1 in mantissa
- **NaN: "Not a Number"**
  - E.g., square root of negative number
  - Signaling NaN throws exception
  - Default is usually "silent" NaN (no exception)
- **Silent NaN Comparison Pitfall:**
  - Comparison with NaN is always false
  - if (CurrentSpeed > SpeedLimit) {shutdown}
    - Comparison is false for CurrentSpeed of NaN → no shutdown
  - (NaN == NaN) is also false (surprise!); use isnan()

IEEE Floating Point Format
Single Precision: 32 bits total

1 bit	8 bits	23 bits (with implicit leading 1.)
S	EXPONENT	MANTISSA

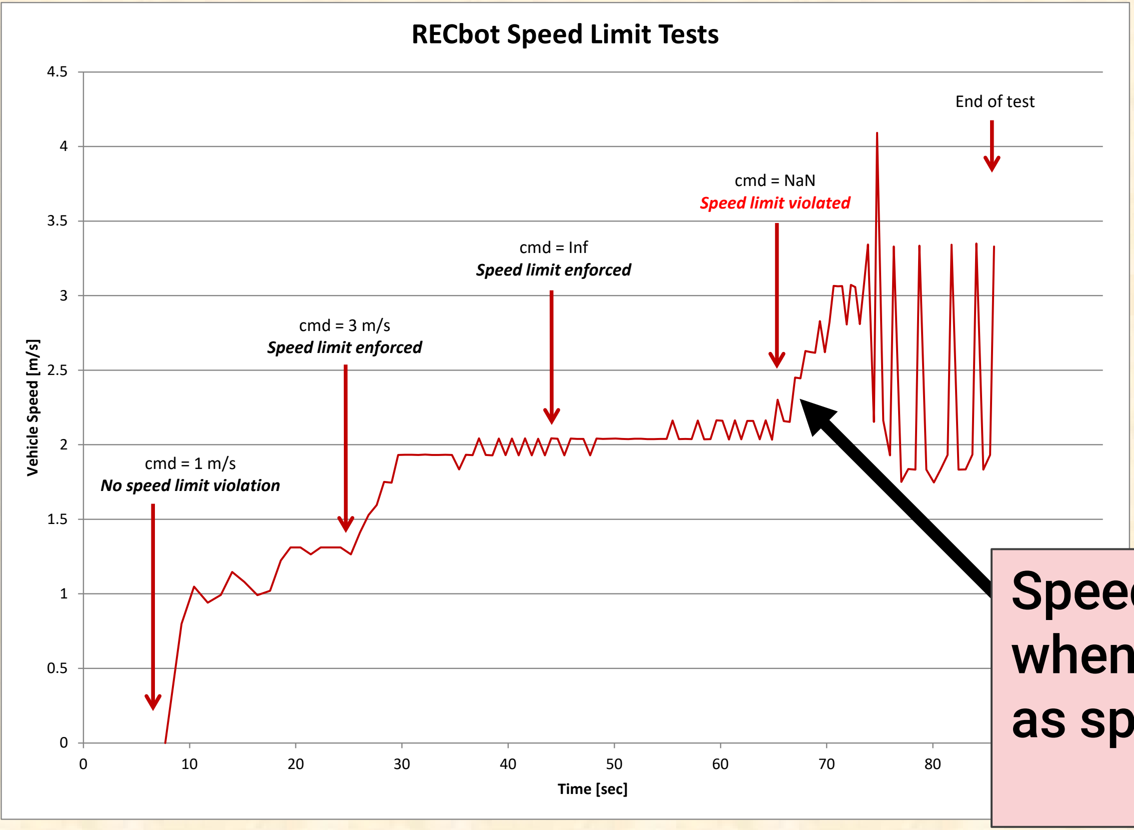
one: 0x3F800000

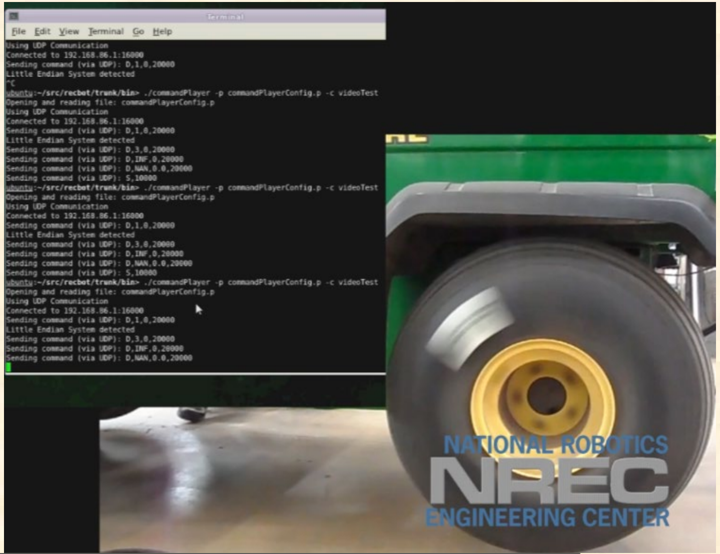
Exponent indicates special values:

-zero: 0x80000000
zero: 0x00000000
+infinity: 0x7F800000
+NaN Signaling : 0x7F800001…
+NAN Quiet/Silent : 0x7FC00000…
…

NaN and the Robot Apocalypse

Carnegie Mellon University





Speed-limit violation occurred when exceptional input sent as speed command

Distribution Statement A - Approved for public release; distribution is unlimited. NAVAIR Public Affairs Office tracking number 2013-74, NREC internal case number STAA-2012-10-23

RoboRace Crash Due To NaN

Carnegie Mellon University

The actual failure happened way before the moment of the crash, on the intialization lap. The initialization lap is there to take the car from boxes to the start/finish line and the car is driven by a human driver during the lap. The intialization lap is a standard procedure by roborace.

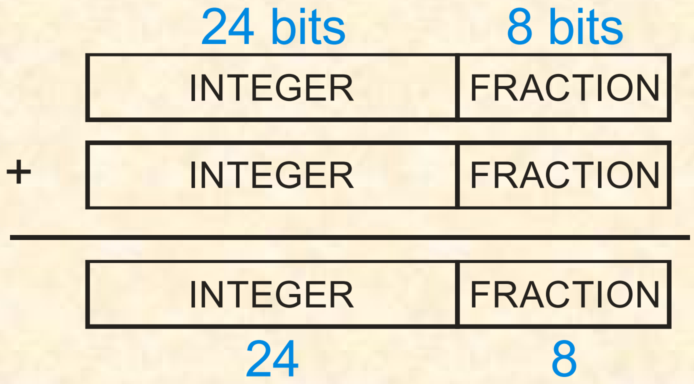
So during this intialization lap something happened which apparently caused the steering control signal to go to NaN and subsequently the steering locked to the maximum value to the right. When our car was given a permission to drive, the acceleration command went as normal but the steering was locked to the right. We are looking at the log values and can see that our controller was trying to steer the car back to the left, but the car did not execute the steering command due to a steering lock. The desired trajectory was also good, the car definitely did not plan to go into the wall.

We are not yet sure what was the actual cause, but it seems that its an extremely rare event during which there was a short spike in the inputs to the controller. Normally, this spike would have been filtered out, but apparently there exists a configuration under which this spike is allowed to propagate through the system and we were "very lucky" to collect it during the competitive run. We had testing days before and had never experienced this.

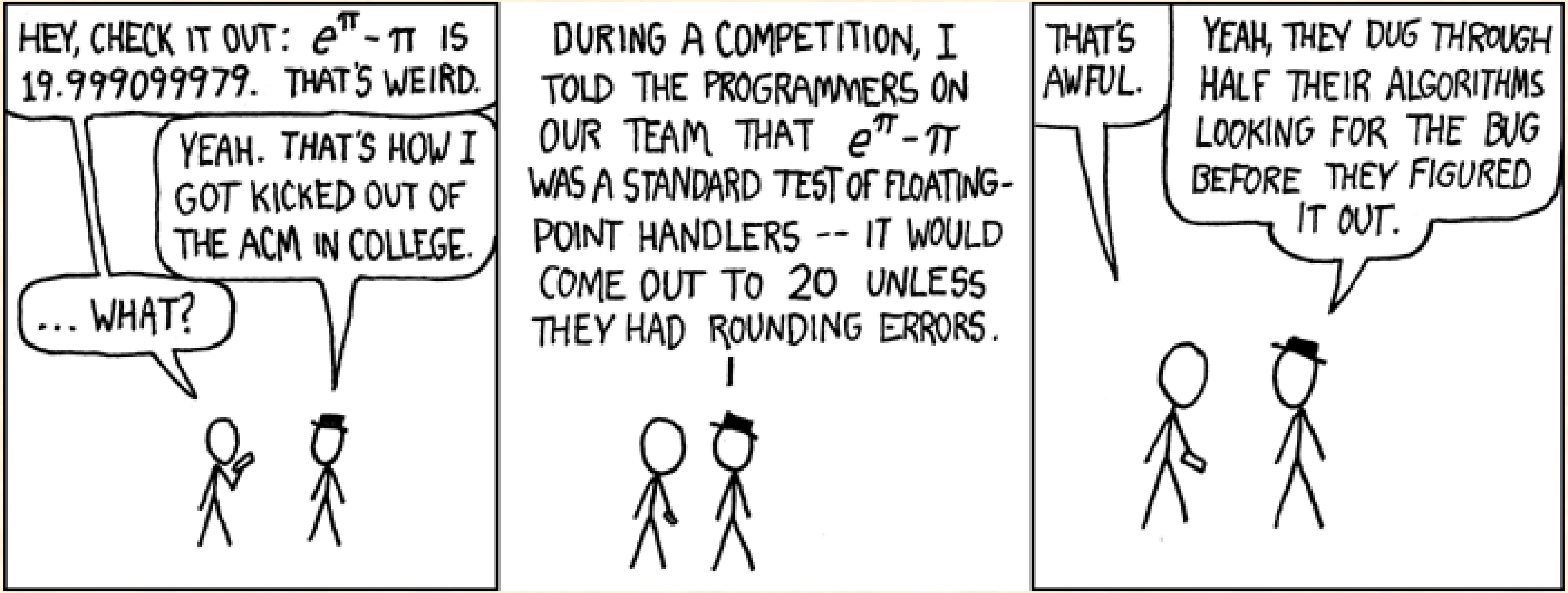
https://www.reddit.com/r/formula1/comments/jk9jrg/ot_roborace_driverless_racecar_drives_straight/
October 2020

Best Practices for Floating Point

- **Use integer math if you can**
  - Scaled integer (e.g., 10ths of a second)
  - Binary Coded Decimal (BCD) + radix point
  - Fixed point (e.g., value *256)
- **Handle special values**
  - NaN is especially tricky to get right
- **Manage and handle roundoff error**
  - Doubles give more bits to work with (53-bit mantissa)
    - But fundamentally, all problems are still there
  - Don’t use floating point as an iterator, including time!
- **Comparisons are especially problematic (NaN, roundoff)**



Fixed Point Addition
(uses normal integer
addition CPU hardware)



https://xkcd.com/217/