A 10 mW Wearable Positioning System
Where Data Filtering and Program Transformation meet.

Henk Muller, Cliff Randell, Andrew Moss

Background

Sensor design:

- Measure data (hardware)
- filter data (use a model of the expected data)
- interpret data (assign meaning to it)

Battery life mostly limited by the second step

Program transformation (John Gallagher):

- Rewrite programs so that they
  - retain meaning
  - run “better” (less power, or faster, or less memory...)

Can you use program transformation techniques to improve battery life?

- Only after you do a lot of work yourself.

Running example: positioning system

Question:

- Can we implement Data Filtering on a PIC?

Answer:

- Yes, we can filter in 100 KHz + 180 bytes.

Ingredients:

- Selection of a suitable filtering algorithm
- Reducing memory requirements of filtering algorithm
- Random number generation.
- Program specialisation in order to speed up code

$\Rightarrow$ Generally applicable; not just on a PIC, not just location.

Location system maths

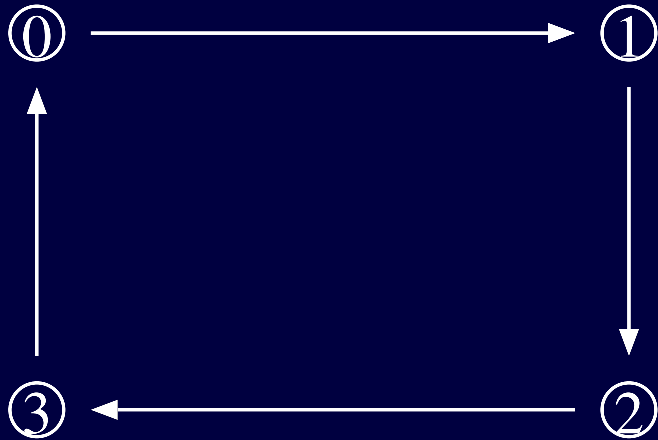
Four transmitters at $(0,0,C)$, $(0,A,C)$, (B,A,C), and $(B,0,C)$;

If we are at a position (x,y,z) in the room

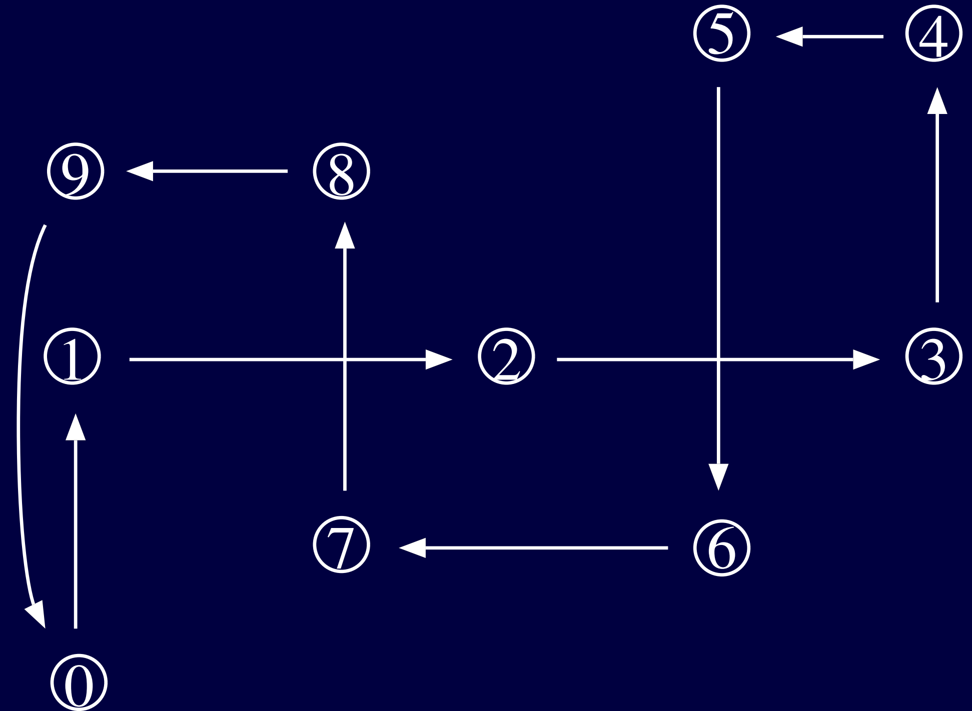
we have measured a distance d_2 to Transmitter 2 and d_3 to Transmitter 3, then:

$$
\begin{aligned}
d_2^2 &= (x-B)^2+(y-A)^2+(z-C)^2 \\
d_3^2 &= (x-B)^2+(y-0)^2+(z-C)^2 \\
d_2^2-d_3^2 &= (x-B)^2-(x-B)^2+ \\
&\quad (y-A)^2-(y-0)^2+ \\
&\quad (z-C)^2-(z-C)^2 \\
d_2^2-d_3^2 &= -2Ay+A^2 \\
y &= \frac{A^2-d_2^2+d_3^2}{2A} \qquad (1)
\end{aligned}
$$

Placement of transmitters



Small system



Larger system

Filtering algorithms

Kalman filter:

- Good at filtering data
- Sensitive to arithmetic rounding and precision
- Requires little memory
- Requires computation of Jacobian and matrix inversion

Particle filter:

- Stochastic approximation or proper filter
- Works on any distribution of errors (as long as it can be modelled)
- Insensitive to rounding errors and low precision

Last bullet is of particular importance on 8-bit micro-controller

Particle filter basics

1. A number of particles, say few hundred, sample the state space
2. Each particle is progressed (like a Kalman filter)
3. For each particle compute probability that measurement supports it
4. Weight of particle is adjusted in order to model PDF of state
5. Particles can be resampled, normalise weights.

$\Rightarrow$ More particles gives a better approximation but costs more

$\Rightarrow$ Mapping required from measurement to state.

$\Rightarrow$ Weights are to be stored. Unless...

Pseudo-code

Process measurement z[j]:

```
For i from 0 to N-1
   progress particle[i]
   weight[i] *= likelihood particle[i] given z[j]
```

Resample:

```
cdf[0] = weight[0];
For i from 1 to N-1
   cdf[i] = cdf[i-1] + weight[i]
For i from 0 to N-1
   newParticle[i] = particle[random based on cdf]
particle = newParticle
```

In situ resampling

Observation:

- Resampling selects good particles in favour of bad ones
- No need to store bad ones, just *probabilistically* ditch them in favour of good ones.
- Do this every step, and you do not need a weight...

Resample while progressing particles sequentially:

- Stochastically replace current particle with previous particle
- Random is biased by probability of previous and this particles.

Good particles will replace bad ones *sequentially*

- Step through in a pseudo-random order in order to prevent bias
- `i*13 mod 64`, `i*7 mod 64`, ..., one addition and conditional subtract.

Pseudo-code

```
step = 1; k = 1
For j from 0
   Obtain z[j]
   step = step + 4; if step > N then step -= N
   For i from 0 to N-1
      progress particle[k]
      likely = compute likelihood particle[k]
      if random * (likely + previous) > likely then
         particle[k] = particle[old]
      else
         previous = likely
      old = k
      k = k + step; if k > N then k -= N
```

Random numbers

Need a fast source of okay random numbers

- 3000 random bytes per second required.
- Numbers need not be cryptographically secure!
- Random numbers cannot have bias - would kill particle filter

Many hardware methods exist: eg Zener diodes, resistors.

- We use a basic Tausworthe generator
- Reseed occasionally by sampling idle RF receiver. (good for 50 bps)

Applying our filter

1. X and Y represented by single bytes, units of 3.43 cm (8.7 x 8.7 m)
2. Chirp every 40 ms, ping every 160 ms
3. 64 X particles, 64 Y particles (two independent filters!)
4. Requires a small number of multiplications with constants (one over 2A) *per particle*
5. Need a fast multiplier

Multiplication

- Hardware without multiplier.
- Almost all multiplications are with compile time constants.

  ⇒ *specialise* a multiplication routine for particular constants.

  (– *specialisation* takes a program and replaces a variable with a constant - like constant propagation but far more aggressive)
- Reduces execution time by factor 3; increase memory foot print!

  ⇒ 69 cycles down to 22

  ⇒ Data independent speed, timing can be by instruction counting.

- Loop is unrolled, if-then-elses are removed, loop is shortened.
- Transmitter configuration encoded in the PICs program

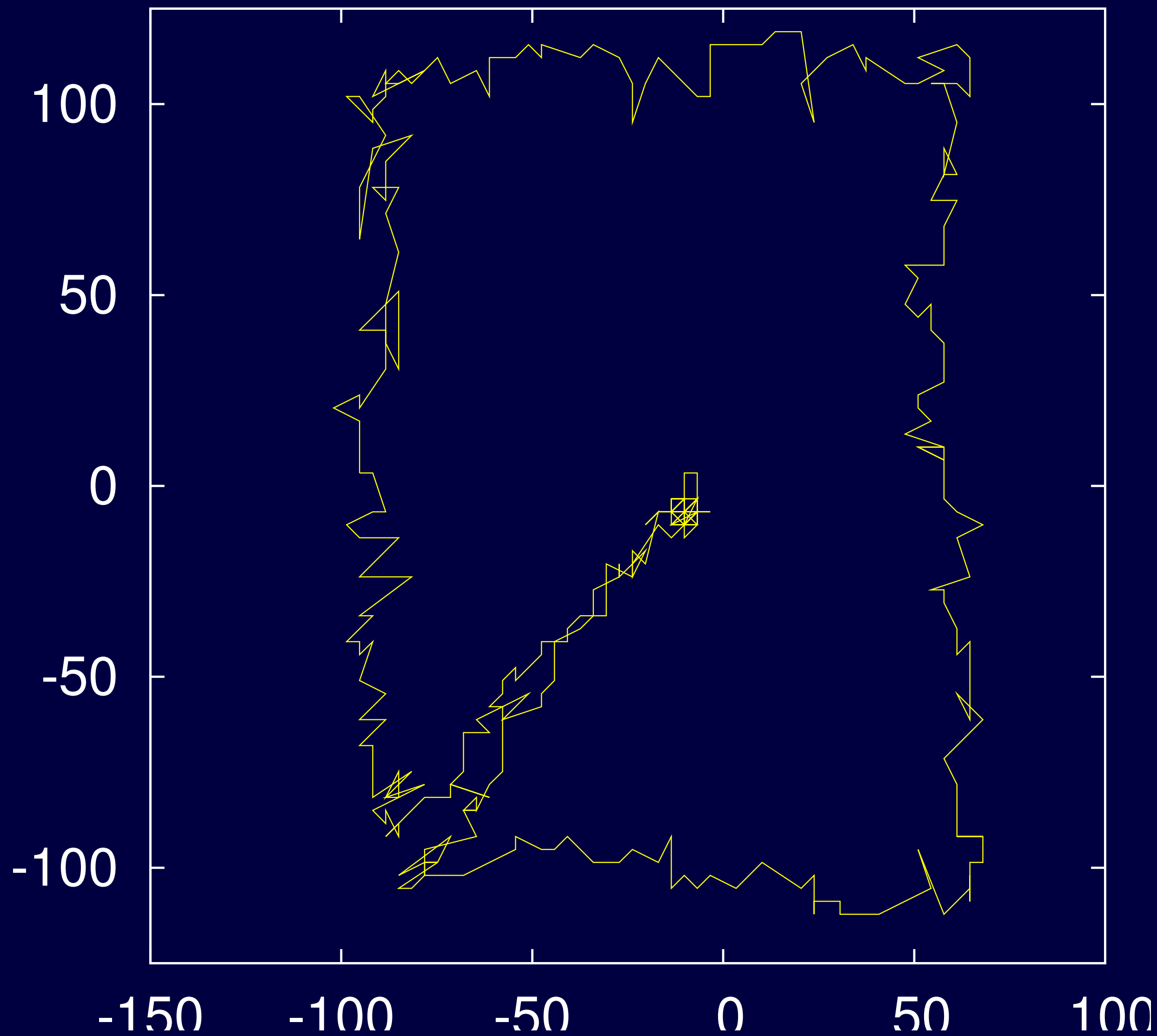
  ⇒ But not encoded in high level code.

Results

Resulting program (all included, RS-232 position output, RF + U/S feed input):

- 1500 bytes program memory (75%) (250 bytes per transmitter)
- 160 bytes RAM (71%) (1 byte per particle, two independent filters)
- 125,000 instructions per second - byte arithmetic.
  - Includes some 16 bit operations coded using byte arithmetic
- PIC takes 2.5mA @ 10 MHz, could run at 125 μA at 1 MHz (50%). Rest of hardware takes 2.5 mA.

Results - II



Tracking a transmitter 1.5m x 2m track.

Conclusion

We show it is feasible to run seemingly complex computations on a micro-controller

- Useful for tracking your shoe, or for integrating other wearable sensors.

Questions?