Team Pi's tips and tricks

TSOPs

These sensor are incredibly simple to use. I'd highly recommend putting 0.1uF caps for decoupling. Hardware filtering is absolutely necessary. We used 2x6mm slits to filter out the IR (I can't remember the depth but I'd guess the slits are ~7mm deep).

Polling at high speeds is the simplest way to read the sensors. It is unecessary to sample more than 127 times per reading. For those that have less experience with interrupts, I don't suggest using them as they can be a nuisance.

Number of sensors	24
Polling period (us)	50
Polling frequency (Hz)	20 000
Sample size	127
Sample frequency (Hz)	157.5

Note: If you're using arduino, the digitalRead function is very slow. Try using digitalReadFast.

Locking

Most would know that TSOPs need to be restarted periodically to prevent "locking." We did this every 50 samples, or 317.5ms.

Filtering

This is a simple filter that adds weighted (K1, K2 etc.) adjacent sensor values to a sensor value.

```
1.  for (i = 0; i < TSOP_COUNT; i++){
2.      /* a rather efficient way to filter data by scoring each data by the tsop by it's
3.      adjacent tsops. Scoring 3 extra pairs is enough (perhaps even 2)
4.      We have to use our own mod function! the % operator is actually remainder in c++. i.e.
5.      doesn't work for negatives! */
6.      uint16_t tempData = K1 *  data[i]
7.                          + K2 * (data[mod(i - 1, TSOP_COUNT)] + data[mod(i - 1, TSOP_COUNT)])
8.                          + K3 * (data[mod(i - 2, TSOP_COUNT)] + data[mod(i - 2, TSOP_COUNT)])
9.                          + K4 * (data[mod(i - 3, TSOP_COUNT)] + data[mod(i - 3, TSOP_COUNT)]);
10.
11.     filteredData[i] = tempData >> 4; // divide by 16 (2^4)
12.
13.     // ... more stuff
14. }
```

I won't tell you the weighting coefficients we used (though you can always find the original code), but they K1 + K2 + K3 + K4 = 16, since we divide by 16 later.

Getting the Angle

Simple enough: find the highest reading and get the corresponding angle. Of course there are other ways. NOTE: It is much more important to make sure the holes/slits you've made for filtering are identical.

In 2014 I tried weighting the angles (for example if we had the following readings).

Sensor 1: 150 Sensor 2: 160 Sensor 3: 130

Clearly the ball angle is between 1 and 2, so the estimated angle would be halfway in between.

The vector approach is fun and works just fine, though it honestly isn't that helpful. Theoretically, it should give you the best results though. It treats every reading as a vector with a direction. It then sums up the vectors and does a simple atan2 (not atan) to get the angle of the resultant vector.

```
1.  void getAngle(uint8_t n){
2.      x = 0;
3.      y = 0;
4.      for (int i = 0; i < n; i++){
5.          // convert vector to cartesian (remember that each bitshift << is *2)
6.          x += filteredDataSorted[i] * scaledCos[indexes[i]];
7.          y += filteredDataSorted[i] * scaledSin[indexes[i]];
8.      }
9.      if (x == 0 && y == 0){
10.         // When vectors sum to (0, 0), we're in trouble. We've got some dodgy data
11.         angle = 0;
12.     }
13.     else{
14.         angle = 90 - (57.3 * atan2(y, x));
15.     }
16.     if (angle < 0) angle += 360;
17.
18.     angleByte = angle * 255 / 360;
19. }
```

Notes.

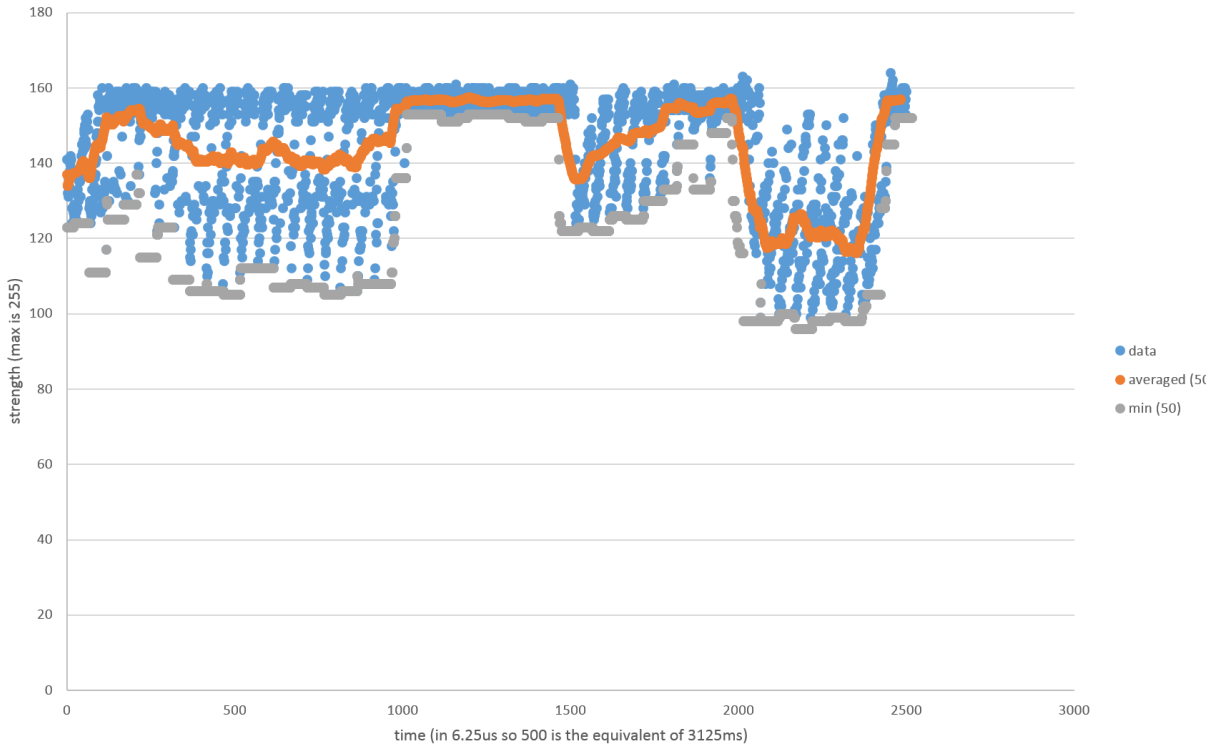
1. scaledCos and scaledSin are arrays of integer trig results generated at the start of the program.
2. We don't need to actually know what the exact value of sine and cosine [-1, 1]. All we need is them to be relative to each other. We do however need a proper atan2 function.
3. Function is passed an integer *n* as the number of readings to use. Readings are selected from filteredDataSorted which is sorted from the highest readings to lowest readings.

Getting the strength of the reading

Strength can give you an approximation of the distance from the ball. As such it is important in determining how the robot should interact with the ball (i.e. how big it should orbit). The piBot used various coefficients/variables and passed these straight into a single line of maths to determine the orbit direction.

The simple way is to take the max reading and use it as the strength. This however can lead to interesting results.

Team Pi’s tips and tricks



Note. At the time of this testing above, the sample size was 255.

The blue dots of the graph are the actual “strength” readings we get if you just took the maximum reading of each. The variation between a few readings can be very large. After trial and error, it was determined that averaging 50 samples was sufficient to get a rather reliable reading.

However, averaging results in delayed response on start and stop, but not during normal gameplay when the ball has already been detected. The main issue we had was that when the ball was removed the robot would still continue to move for a small fraction of a second since the average reading of the previous 50 samples was still above the required threshold.

You can also use the minimum reading of the past 50 readings as this seems to be quite reliable as well. However, never use the highest readings (you can see why from the graph).

Document created on 31/10/2015 by Brian.