Introduction

This document explains a few very basic techniques from signal processing and pattern matching, which can be used for tracking objects in the laser scans. As so often, there are thousand different ways, and it quickly gets very complicated. Nevertheless, there are some simple methods that give sufficiently good results, and are very easy to understand. To keep things simple, the simplest solution to a problem is preferred over a statistically more correct solution. Further reading can be most easily found in literature about computer vision.

Background estimation

Background estimation is a good trick to reduce the amount of data that has to be scanned for objects. The assumption is that there is some kind of static background that doesn’t change (or doesn’t change too quickly), while the objects we are interested in are clearly different from this background. Basically, everything which is not object in a scan or a picture, is background. Since we want to use the background to find the objects, estimating it can be a chicken-and-egg problem.

In the case of our traffic laser scans, we assume that moving objects are sporadic, which means that a particular laser beam at some angle will most of the time measure background, not an object. This means, that if we take a mean over sufficiently many successive scans, this mean will be very close to the correct “background” distance.

The background estimation can either happen at startup, using the first few incoming scans, or it can happen continously throughout the whole run-time of the system.

Calculating mean values

There are different ways to calculate means. Assume we have set of scans $s_1 \ldots s_n$ ($s_i$ are vectors). The most commonly known method is to add and normalise:

$$m = \frac{1}{n} \sum_{i=1..n} s_i$$  \hfill (1.1)

If we want to calculate a running mean throughout the whole runtime of the system, we can either add up everything from the beginning, or we can apply
the sliding window method. Simply keep a buffer of the last \( n \) scans. Whenever a new scan arrives, add it to the buffer, remove the oldest scan from the buffer, and recalculate the mean.

A slightly different, but simpler methods calculates a mean by weighing older scans with an exponentially decaying factor. The sliding mean \( m \) is initialised with the first scan \( s_1 \). (\( m_1 := s_1 \)) With every new scan \( s_t \), the mean \( m_t \) is updated as follows:

\[
m_t = (1 - \alpha) \cdot m_{t-1} + \alpha \cdot s_t
\]

The parameter \( \alpha \) controls the speed of adaptation, and has to be chosen accordingly to the application. (in an Ada program, you would of course only keep one (array-) variable \( mean \), not the whole sequence \( m_1 \ldots m_t \))

This sliding mean is equivalent to the following expression:

\[
m_t = (1 - \alpha)^t s_1 + \sum_{i=2, t} \alpha (1 - \alpha)^{t-i} s_i
\]

You can see that older scans have an exponentially decaying coefficient, which means they contribute less to the mean than newer scans. The advantage of this method is that old scans do not have to be saved.

**Background subtraction**

Now after we know the background, we would like to identify objects that stand out. This is as simple as subtracting the background vector from every new incoming scan:

\[
d_t = s_t - m_t
\]

The resulting difference vector \( d_t \) should have entries close to zero in regions where nothing changed. Distinctly non-zero entries indicate a change.

In our example, “distinctly non-zero” can be difficult to define. There are regions in the scan with very large variations due to moving leaves and trees, which don’t correspond to cars or pedestrians. One method to deal with this, is to calculate a second mean on \( d_t \) for the mean variation of each entry, i.e.

\[
v_t = (1 - \alpha) \cdot v_{t-1} + \alpha \cdot d_t
\]

Only if an entry of \( d_t \) is much larger than \( v_t \) (i.e. \( 2v_t \)), an object is identified.

The result of this stage is a boolean vector, which indicates for each single measurement of a scan (entry of the vector), if it is different from the background model or not.
**Segmentation**

In this stage we would like to identify object candidates, by segmenting the boolean vector into connected *dynamic patches*. The simplest way is to scan through the vector, starting on one side. When the start of a dynamic patch is found, proceed scanning to find the end of it (the first entry without a change from the background). The dynamic patch can then be added to an object buffer.

**Correlation and Tracking**

We have for each new scan a list of dynamic patches. Now we would like to *track* these patches throughout consecutive scans. To be able to track, we first have to be able to identify an object candidate in the following scans. This can be done using a correlation function, i.e. the *normalised sum-of-distances*.

Assume you have a short vector $p = p_1 \ldots p_k$ representing a patch (object), and the scan vector $s = s_1 \ldots s_n$. The correlation function $c$ provides a measure of how well the patch matches the new scan at a certain position $x$:

$$c(x) = \frac{1}{k} \sum_{i=1}^{k} \sqrt{(p_i - s_{x+i})^2}$$

(1.6)

To find the best match, take the $x$ for which $c(x)$ is minimal (scan $x$ across the search range, and record the minimum). Note that the unit of $x$ is half-degrees ($x = 2\alpha$)

Since some values in a scan might be invalid, the difference can’t be calculated for these points. Simply only add the valid differences, and normalise with the number of valid differences:

$$c(x) = \frac{1}{|A|} \sum_{i \in A} \sqrt{(p_i - s_{x+i})^2}$$

(1.7)

(A is the set of all indices $i$ for which $p_i$ and $s_{x+i}$ are valid). You might want to weigh invalid samples differently in order to penalise matches with lots of invalid points.

**Interpolation**

Since the laser scans have a rather low resolution, interpolation can be used to achieve higher accuracy. Instead of using the laser scan directly, use an interpolation function, i.e. linear interpolation:

$$s(x) = (1 - (x - \lfloor x \rfloor)) \cdot s_{\lfloor x \rfloor} + (x - \lfloor x \rfloor) \cdot s_{\lceil x \rceil}$$

(1.8)
Using this function, the correlation function $c(x)$ is now also defined for real numbers $x$. The minimum search can now be performed in smaller steps.

**Tracking**

Object candidates can now be found, and identified in following scans. It should be easy now to set up a tracking system, that has a database of tracked objects, which is updated with every new scan. There are some minor pitfalls, though.

As an object moves through the scene, its appearance on the laser scan (the relative and absolute distances) change continously. Yet, the changes from one scan to the next are small. It is still possible to track the object through the scene, even though it looks completely different at the end of the tracking process. Simply update the object representation, after the new position has been found.

Pattern matching using correlation is computationally expensive. You can reduce the search range by a couple of methods. Dynamic regions give a coarse indication where the object might be in the new scan. Also, while the object is tracked, it is possible to estimate the speed and acceleration, and predict the region in which it will be in the next scan.

Finally, objects might disappear temporarily (occlusion) or completely. Your algorithm will have to decide if an object could still be tracked or not.

**Conclusion**

These few standard techniques should give you a good starting point for your own object tracker. In general, there is no “best” solution to any given problem in object recognition, pattern matching or tracking. Feel free to apply other methods where they might yield better results, like spline interpolation, or Kalman filters for the tracking process. There is no limit to complexity. Nevertheless, don’t get lost in details, focus on the realtime aspects of the assignment.

Good luck!