Real Time Optical Flow Based Range Sensing in Mobile Robots

Simon Perkins * **Gillian Hayes** **

Department of Artificial Intelligence
University of Edinburgh
5 Forrest Hill
Edinburgh EH1 2QL
Scotland

Abstract. The idea of using the optical flow generated by an observer's motion to determine the 3D structure of the local environment has been around for a long time. However, implementing this idea on autonomous mobile robotic platforms working in real time has proved to be quite difficult due to the large amount of processing power needed and the difficulty of getting accurate measurements. In this paper we explain how some simple assumptions can be used to greatly improve the robustness of the results and simplify the processing required. The approach is used to implement a real time optical flow based range sensor on a mobile robot equipped with only modest computing resources. We demonstrate how this sensor can be used as the basis of a simple but effective vision-based navigation behaviour.

1 Introduction

If autonomous mobile robots are to interact in a useful way with their surroundings, it is essential that they have sensory systems that are able to determine the basic 3D structure of the nearby environment around them.

Visual sensing is widely used in nature for this purpose, but artificial vision has traditionally been too computationally complex and unreliable to be useful for real-time autonomous mobile robots. Given the current hardware it seems to be a good idea to look for ways in which we can greatly reduce the computational load of visual sensing while still maintaining sufficient resolution and reliability for it to be of use.

A possible route to simplification is provided by *optical flow*. This is the well known phenomenon whereby relative motion between a visually equipped observer and the surrounding environment gives rise to a 2D motion of the image of that environment across the image surface of the observer. In theory at least the optical flow field provides a *relatively* unambiguous source of information about the motion of the environment, the observer, and the 3D structure of the environment.[1]

Using optical flow is not without its problems however. Right at the start, it is difficult to extract an accurate optical flow field from a sequence of images taken by the robot's camera, due to such effects as the *aperture problem*. Even if this can be done it is then difficult to obtain reliable range readings for the various features in the image, due to the extreme sensitivity of the basic calculations to errors in the measured optical flow field. Finally, traditional approaches still tend to require a lot of processing power.

One of our major aims is to demonstrate that it is possible to build visually guided robots that work reliably in real time and that use only modest processing power. In order to achieve this and also in order to overcome the other problems mentioned above, we had to make some simplifying assumptions as detailed in section 3. Ultimately however, we were able to build a robot capable of navigating around an unstructured environment using optical flow based sensing alone.

2 Optical Flow

The *optical flow field* is a 2D vector field over the imaging surface representing the instantaneous velocity of each point in the image. It is determined by three things: the observer motion, the scene motion, and the 3D structure of the scene.

* simonpe@aisb.ed.ac.uk
** gmh@aifh.ed.ac.uk

[1] Compared to simple intensity information anyway.

The first two of these affect both the direction and magnitude of the optical flow field, but the 3D structure of the scene affects only the magnitude of flow, not the direction. This fact is important in the simplifications that we make later.

A lot of research has looked at ways of extracting all three factors from the flow field. It turns out that, subject to a few minor constraints, the optical flow field *can in theory* be decomposed into the three components (*e.g.* [7], [5]). However, the calculations for deriving the 3D scene structure from the flow field in the general case where nothing is known *a priori* about the three factors, are very sensitive to errors in the measured flow field, which as we will see later is itself very difficult to measure accurately. The instability is largely due to the calculation's dependence upon higher derivatives of the optical flow field velocity, whose calculation amplifies any noise already present.

We cannot measure the optical flow field directly, so instead we must derive the flow from sequences of consecutive images taken with the robot's camera. However, due to the *aperture problem* (*e.g.* [2]), for each small patch of the image it is only possible to calculate locally the component of image motion perpendicular to the local spatial intensity gradient. Because of this, the actual image flow pattern is fundamentally underdetermined by the changing intensity pattern of the image. The famous 'Barber's Pole' illusion provides a good illustration. Assumptions of smoothness (*e.g.* [3]) can be made to resolve the aperture problem, but this introduces further errors.

For all these reasons there have been few successful real-time autonomous mobile robots that make much use of optical flow.

3 Simplifying Assumptions

In order to solve some of the problems mentioned above we make two important simplifying assumptions:

- The observer motion is known.
- The scene is stationary, or largely so.

In many cases these assumptions are not unreasonable. For instance, mobile robots are often equipped with motion encoders that can determine the motion of the robot quite well, at least over short time periods, and as long as the robot is not travelling over too rough terrain. Similarly, for many applications, the assumption that the scene is stationary is valid, although we are looking at ways of relaxing this criterion, so that moving parts of the image can be recognized automatically by the system.

Given these assumptions, several previously mentioned problems are solved. Firstly as we will see, the calculation to obtain 3D structure from the flow becomes relatively straightforward and less sensitive to error in the measured field. Secondly, since the *direction* of the optical flow field at each point is entirely determined by the observer and scene motion, and not at all by the scene structure, and since we *a priori* know both of these motions (the observer motion from the encoders, and the scene motion being zero), we already know the direction of the flow field at all points in the image. This provides the constraint needed to solve the aperture problem entirely. We only need to determine the *magnitude* of the field at each point in order to recover the entire field. Finally, knowing the direction of the flow we can subsample the image as described in section 5 in order to greatly reduce the amount of data that must be processed. This is one of the key elements in achieving a real-time implementation.

4 The Hardware

The task that we set our robot, and the way in which we achieve it, is constrained by the capabilities of the particular robot that we decided to use. That robot is the Department's transputer based mobile robot: **Ben Hope** [1]. Ben is a half metre high cylindrical robot built up on a Real World Interface B12 robot base. It currently carries on board four T800 transputers [4], one of which also acts as a framegrabber. A camera is mounted in a fixed position on top of the robot pointing along the direction of motion. The robot can translate either forwards or backwards, and it can rotate about its vertical axis. The fact that the camera cannot be moved independently of the robot, and that the robot cannot move directly sideways, limits the motion strategies that are available to us. As a starting point we decided to just consider the case of straight ahead translation with no rotation.

5 Obtaining Accurate Flow Fields

It is essential to obtain good estimates of the image motion if they are to be used to produce measurements of the surrounding 3D environment of the robot. We therefore explored a number of methods for deriving the optical flow field from a sequence of images captured by the robot's camera.

5.1 Image Slices

In our current implementation we only consider the case of straight ahead translation of the robot. This produces a *radial* optical flow field. Since we know the directions of all the vectors in this field (given that we know the focus of expansion from the geometry of the robot), we can choose to sub-sample the field and so reduce the amount of data we have to process by just considering 1D radial *image slices*. The basic idea is shown in Figure 1. Since we know that the flow will always be along the slices, it only remains to find the magnitude of the flow at each point along the slice.

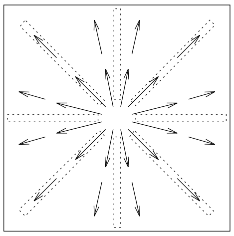

Fig. 1.: The arrows show the radial optical flow field caused by straight ahead motion with the camera pointing in the direction of motion. The dotted rectangles show the positions of typical image slices used by the algorithm.

5.2 The Gradient Approach

The 'traditional' way in which this flow magnitude would be found is to use the *gradient method*. This is based around a continuity equation:

$$\frac{\partial I}{\partial x} u_x + \frac{\partial I}{\partial t} = 0$$

where I is the intensity at a point along the image slice, and u_x is the local image velocity at that point along the slice.

Unfortunately, this equation is only strictly true in the limit in which the time-step between successive frames tends to zero. If the distance that the image moves between frames is comparable to the distance over which the spatial intensity gradient $\frac{\partial I}{\partial x}$ changes, then the equation breaks down. It also does not strictly apply at flow discontinuities, *e.g.* at depth discontinuities in the image. Even working at around 12.5 frames per second, we found this method to be too inaccurate.

5.3 The Tracking Approach

The alternative method which we chose is to track features moving along the image slice to get a direct estimate of their motion. In our case, we use a simple {1, -2, 1} approximate Laplacian of a Gaussian convolution mask (length 15 pixels) in order to locate edge features in the slice. These edge features occur at zero crossings of the output from the convolution. Detected edges are labelled with the slope of the convolution output at that point in order to facilitate matching of edges between successive frames.

Figure 2 shows a plot of the edges detected within a horizontal slice across the image[2] over a sequence of 16 frames. Notice how the edges belonging to closer objects show a greater displacement each frame than the edges belonging to more distant objects. Also note that edges towards the outside of the image move the fastest of all.

Whenever a new frame is captured, roughly 12.5 times a second, the new edges detected in a particular image slice are placed together with previous sets of edges detected from that slice in a variable length pipe (FIFO) structure. At the same time, the robot motion encoders are interrogated to see how far the robot has moved since the previous frame was captured. The displacement of an edge between two successive frames divided by the distance that the robot has moved forward gives a measure of the relative velocity of that edge that we call the *flow-rate*. In order for a new edge to match a previously seen edge, several criteria must be met:

[2] Actually two horizontal slices, since each slice starts at the FOE in the centre and moves outwards.

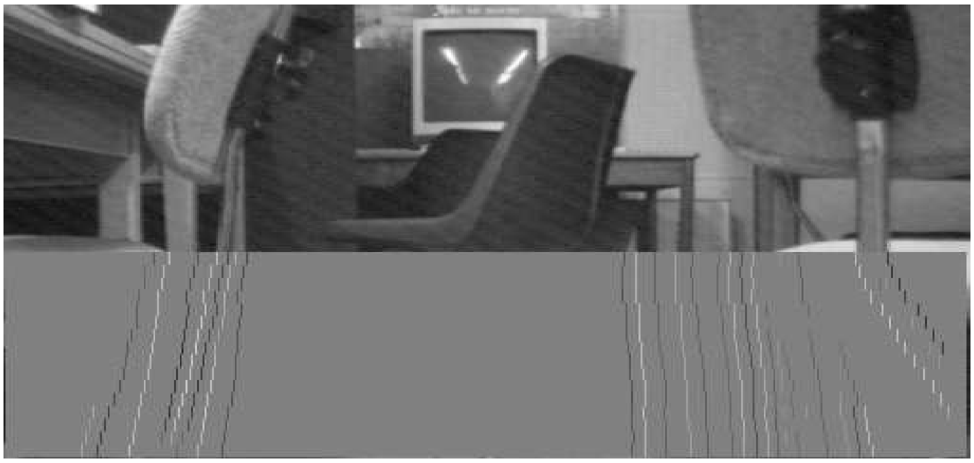

Fig. 2.: Zero crossings detected in a sequence of 16 frames within a horizontal slice. The upper part of the image shows the image just above the location of the slice from frame 1. The lower part shows 16 horizontal bands showing the edges detected within the horizontal slice in each successive frame. The time axis moves downwards.

- Similarity of edge contrast. As measured by the difference in the output of the Laplacian mask either side of the zero crossing. The edges must have similar contrast in order to match.
- Similarity of edge flow-rate. The flow-rate of an edge in an image gradually increases as the feature gets closer to the robot and its image moves further out towards the edge of the picture. It can also change randomly due to random motion of the camera due to the robot moving over an uneven surface. Hence a match must give rise to a flow-rate that is similar to the previously recorded flow-rate for that edge.

For each edge, the locations of several previous sightings of that edge are stored, so that they can be used in a least squares fit when it comes to calculating the 3D location of the edge feature. The number of old edge locations that is kept on varies according to heuristics that decide how likely the tracker is to have got the tracking right for each edge. The better the tracker thinks it is doing on an edge, the more previous sightings of it are kept.

6 Calculating 3D Feature Locations

The output from the edge detection section consists of lists of edges that have been tracked through time. It is now necessary to turn this motion information into 3D locations.

6.1 Basic Calculation

Up till now we have been talking in terms of instantaneous optical flow fields, but at this stage it is better to switch to talking about discrete displacement fields. We consider first the simple case where we just have two measurements of a feature's position, as shown in Figure 3. We know that the image flow will be outwards along the chosen slice, and so the problem is one dimensional.

Now, by similar triangles:

$$\frac{x_1}{f} = \frac{X}{(Z + \Delta Z)}$$

and:

$$\frac{x_2}{f} = \frac{X}{(Z)} \tag{1}$$

Combining these and rearranging we get:

$$Z = \frac{\Delta Z \, x_1}{x_2 - x_1}$$

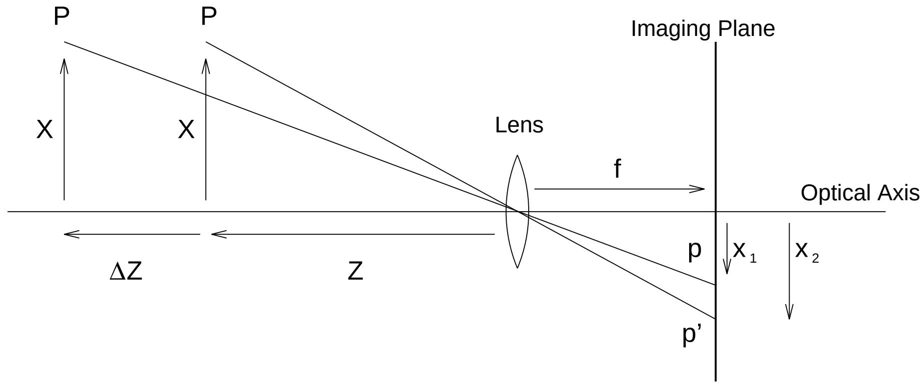

Fig. 3.: The diagram shows the robot's camera imaging a point feature P at two different times. Z is the projection of the position vector of P onto the optical axis and is known as the *depth* of P. X is then the perpendicular distance of P from the optical axis. The robot starts with P at a depth of $Z + \Delta Z$, which it images at p, a distance x_1 from the FOE. A short while later the robot has moved forward and now images the feature at point p' a distance x_2 from the FOE.

which is the equation for the distance of the feature from the robot, projected onto the direction of motion.[3] Once this has been found, the displacement X of the feature away from this axis can be found using equation 1. Note that this calculation is valid for any step size, not just infinitesimal steps. Another nice thing about it is that the depth Z is obtained directly in whatever units the robot's motion encoders use without need for calibration, since the pixel displacements cancel each other out.

The equation contains no noisy derivatives, and is fairly stable to errors in the input values. The error in Z is linearly proportional to the error in ΔZ, and if the error in $x_2 - x_1$ is small compared to the size of the whole thing, then the error in Z also scales approximately linearly with this error. The only problem occurs when the error in $x_2 - x_1$ becomes large and acts so as to reduce that value, in which case Z becomes very large. We try to get round this by ensuring that we don't attempt the calculation when $x_2 - x_1$ is very small, as explained in the next section.

In our current implementation we further reduce the sensitivity of the calculation to errors by performing an approximate least squares fit using all the recorded position measurements for an edge feature.

6.2 Reliability Heuristics

There are many sources of error in the calculation as it stands. For instance, it is easily appreciated that an edge that has only been seen twice and has hardly moved in between is not going to provide a very accurate estimate of its 3D location. Even when an edge has been tracked for some time, it is quite common for the tracker to suddenly confuse it with another edge and then to begin tracking that edge. This happens mainly when one edge is occluded by another. Due to the smoothing imposed by the edge detection there is a tendency for such edges to merge just as they pass each other which fools the tracker. For these reasons we use a variety of heuristics to decide when the data that has been collected for an edge feature is sufficient to calculate a fairly reliable 3D location for that feature. The heuristics we use are:

- Minimum number of sightings. The feature must have been seen a certain number of times, typically 8.
- Minimum total pixel shift. The feature must have moved at least a certain minimum number of pixels in the image, typically 12.
- Maximum variability in 3D location. The 3D location calculated for the feature must not change by more than about 2 cm typically between successive sightings.
- Minimum distance from FOE. Features very close to the FOE tend to give unreliable results so reject any closer than about 30 pixels away from the FOE.

If a feature fails any of the above tests then it continues to be tracked as normal, but no 3D location estimate is put forward to the next stage of processing.

[3] Note, not merely the optical axis. If the camera is tilted then the FOE shifts, but as long as its position is pre-located and the image slices are taken to go through the FOE, then the depth Z can be shown to be the position vector for the edge feature from the robot projected onto the direction of motion.

7 Results

The current implementation of the algorithm on Ben Hope uses all four transputers for processing and works at around 12.5 frames per second. The robot itself moves at a speed of around 30cm a second. The speed has been limited so that if the robot does crash into obstacles, it doesn't smash itself to bits. The implementation makes use of five image slices: two horizontal going left and right from the FOE, one vertical going downwards from the FOE, and two at 45° in between the other slices. Since the camera is mounted on top of the robot, these slices cover most of the area in which the robot can expect to find obstacles.

7.1 Testing the Accuracy of the System

The accuracy of the algorithm at localizing feature points in space was tested using the semi-artificial 'fence' scene shown in Figure 4. This scene contains strong features that the robot is sure to detect so that we can see the ultimate accuracy that the robot can achieve. The scene contains a mixture of occluding and reflectance-change edges at a known distance from the robot.

Fig. 4.: The test scene showing the 'fence'.

Several experiments were run. In each, the robot was started a known distance away from the fence, and was told to move forward one metre and then to stop. It then uploaded the locations of all the features that it had tracked and for which it had obtained range readings.

Figure 5 shows the results obtained for fences at 2 m and 3 m from the robot start point. The circles indicate feature locations, while the dotted lines show a plan view outline of the main obstacles in the room. The dashed 'V-shape' shows the field of view of the robot at the start of its run.

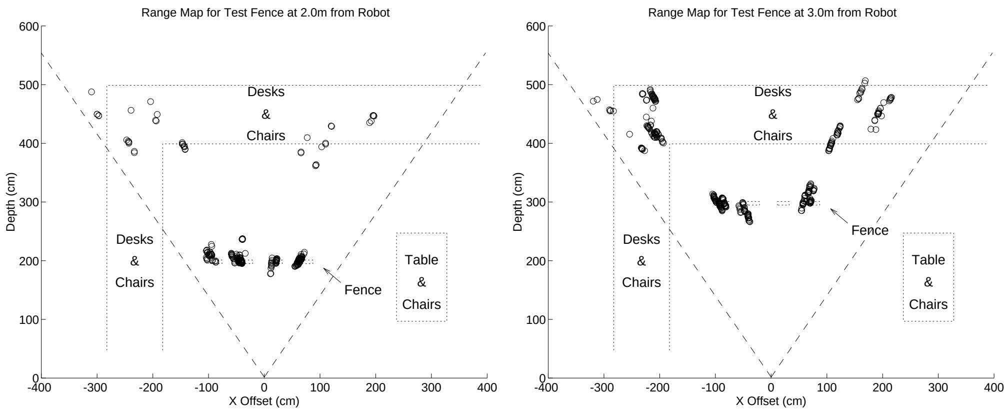

Fig. 5.: Accuracy results with fence at 2 m and at 3 m from robot starting point.

As can be seen, the algorithm is able to localize the strong features of the fence extremely well. As well as the fence, features belonging to the desks and chairs at the back of the room have also been detected and fairly well localized. The localization is typically less than 0.5 m out, and the accuracy improves as the robot approaches the obstacle. This ought to be enough for the robot to be able to navigate around the room without bumping into things. In the 3 m picture the robot's 'blind spot' can clearly be seen. This is due to the fact that near the FOE, the image velocity is low, and hence the reliability heuristics reject quite a lot of feature sightings from that region.

7.2 Map Building and Avoidance

As a demonstration of the usefulness of optical flow based sensing we implemented a simple obstacle avoidance and map building behaviour on board Ben Hope. Exactly the same algorithm described above was used, except that the stream of 3D feature location values was used to update a 2D plan view obstacle map of the robot's environment.

Adding Features to the Map The obstacle map is quantized into 25 cm× 25 cm square cells. Any features that are located in a particular cell and which are at least 20 cm off the ground, cause the value of that cell and the surrounding cells to be increased as described below. The value of a cell represents the likelihood of finding an obstacle within that cell. Only features that are detected higher than 20 cm off the ground are counted in order to avoid the robot being fooled into thinking that slightly incorrectly located marks on the floor are actually obstacles. This does mean that genuine obstacles at heights of less than 20 cm will not be seen.

When a feature is detected within a cell, it increases the value of that cell and the surrounding eight cells according to an approximately Gaussian smoothing mask. This smoothing reflects the fact that only edge features of obstacles are detected and so we propagate outwards from these features in the hope of filling in the gaps in obstacles between edge features.

The map uses a simple world centred representation. Since feature locations produced by the algorithm are always relative to the robot's position, it is necessary to keep track of the robot's position within the map in order to know where to add new features, and this is taken care of by simple odometry using the robot's motion encoders.

Map Decay The algorithm does produce a number of spurious feature locations, despite all the safeguards we use to prevent this happening. If all feature locations were simply added to the map and then left there forever, the value in each cell would continually build up due to this random noise, until the robot believed that there were obstacles everywhere. To counteract this we decay cell values regularly (typically every time the robot crosses a cell boundary) according to a two stage scheme:

$$\mathsf{CellValue} \Leftarrow \begin{cases} \mathsf{FastDecayFactor} \times \mathsf{CellValue}; & \mathsf{CellValue} < \mathsf{BeliefThreshold} \\ \mathsf{SlowDecayFactor} \times \mathsf{CellValue}; & \mathsf{CellValue} \geq \mathsf{BeliefThreshold} \end{cases}$$

The two stage decay means that cells containing just a few features are decayed at a more rapid rate than cells with lots of features. The belief threshold represents the level at which the robot finally believes that there is really an obstacle within a particular cell. In order to reach this threshold, a large number of features must be detected fairly rapidly within a cell, in between the decay cycles. This is characteristic of genuine obstacles, but not of noise. Once this threshold has been passed, the decay rate is low, and so the obstacle will persist in the map for a long time.

The other reason for decaying map values is to correct for errors in the robot's odometry. As time goes on, the robot's sense of its own position becomes more and more inaccurate. Decaying away map features ensures that old sightings of obstacles gradually disappear and are replaced by new sightings which will be more accurate in position relative to where the robot thinks it is.

Avoidance Strategy The robot's avoidance strategy is rather simple. It will keep going in a straight line until it thinks it is within 0.5 m of an obstacle, when it will turn either 90° left or right depending upon which way it thinks is most clear of obstacles. If neither way is sufficiently clear it will turn 180° and go back the way it came from. The robot checks for obstacles using the obstacle map and only avoids obstacles which have a higher value than the belief threshold.

Figure 6 shows the state of the robot's map at two different times during a typical run. The darkness of the squares is proportional to the value of that cell. The map has been overlaid on top of an outline plan of the room in which the experiment was carried out in. The robot did not crash throughout this particular run, but is not always so successful. Typically, the robot crashes once for every ten obstacles that it manages to avoid.

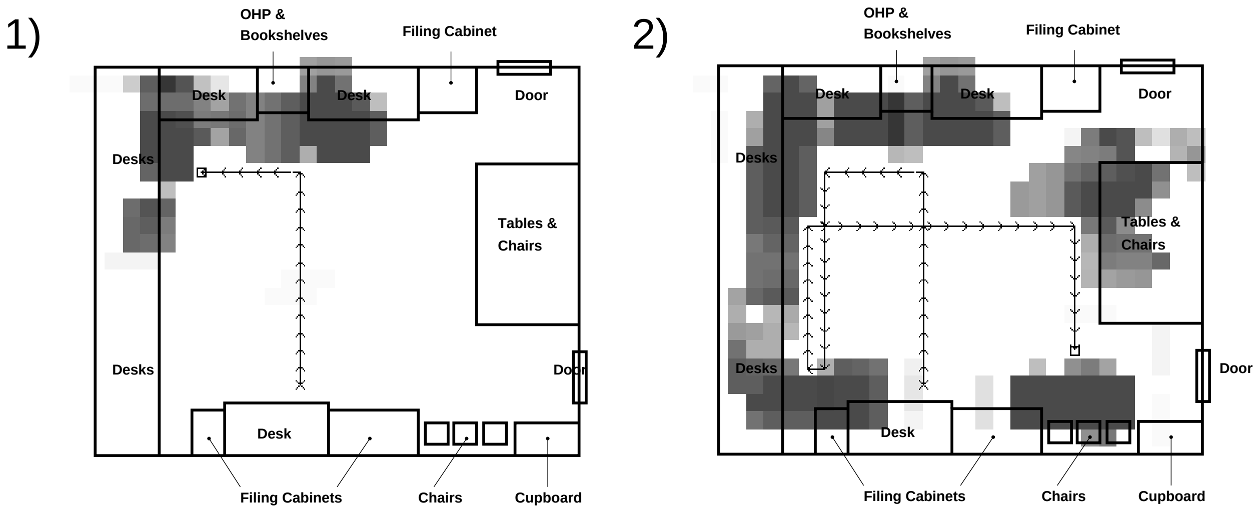

Fig. 6.: Typical map of laboratory built up by robot. The figure shows two different stages along a single run. The robot successfully avoided colliding with any obstacles in the room while building up this map. The robot starts at x

8 Conclusions and Future Work

The main point that emerges from this work is to show that contrary to popular opinion, it *is* possible to build a robot that can successfully navigate around an unstructured unknown environment using purely optical flow and odometry. The system works in real time (at half video frame rate in fact), yet runs on only four T800 transputers, of which only two are working at anything like capacity.

In order to achieve this we make the assumption that the motion of the robot is known, which given the existence of accurate motion encoders is quite reasonable. In return this brings the following advantages:

- Direction of flow is known at all points in image. Therefore the aperture problem is avoided.
- Since we know the direction of flow at all places, we can easily subsample the image using image slices which massively cuts down the amount of data to be processed.
- The calculations to derive 3D locations from the flow are relatively simple and robust.

The main factor currently limiting the accuracy of the system is spurious motion of the camera as the robot moves forward: 'camera jiggle'. Future work will look at ways of compensating for this effect, perhaps using some sort of statistical analysis of the image flow.

The assumption that the scene be stationary is restrictive and we are looking at ways in which moving objects may be detected under the system and dealt with accordingly. This might be done by looking for inconsistencies in the optical flow that we obtain.

Finally, it is currently necessary to locate the FOE by hand before the algorithm can be run. It would be nice to be able to this automatically from frame to frame, perhaps again by some kind of statistical analysis of the image flow. An interesting possibility is reported in [6].

References

1. Peter Forster. A transputer-based autonomous robot. DAI Technical Paper 6, Dept. of AI, University of Edinburgh, 1991.
2. Ellen C. Hildreth. Computations underlying the measurement of visual motion. *Artificial Intelligence*, 23, 1984.
3. Berthold K.P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17, 1981.
4. INMOS Ltd. *Transputer Technical Notes.* Prentice Hall, 1989.
5. H.C. Longuet-Higgins and K. Prazdny. The interpretation of moving retinal images. *Proceedings of the Royal Society of London*, B 208, 1980.
6. Shou-Ling Peng. Temporal slice analysis of image sequences. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 283–288, 1991.
7. J.H. Rieger and D.T. Lawton. Sensor motion and relative depth from difference fields of optic flows. In *Proceedings of the 8th International Joint Conference on AI*, 1983.

This article was processed using the TEX macro package with IRS94 style