RC23029 (W0312-059) December 12, 2003
Computer Science

IBM Research Report

Visual Rotation Detection and Estimation for Mobile Robot Navigation

Matthew E. Albert
Byram Hills High School
Armonk, NY 10504

Jonathan H. Connell
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Visual Rotation Detection and Estimation for Mobile Robot Navigation

Matthew E. Albert
Byram Hills High School
Armonk NY 10504
MEA3581@aol.com

Jonathan H. Connell
IBM T.J. Watson Research
Hawthorne NY 10532
jconnell@us.ibm.com

***Abstract*— There are a number of sensor possibilities for mobile robots. Unfortunately many of these are relatively expensive (e.g., laser scanners) or only provide sparse information (e.g., sonar rings). As an alternative, vision-based navigation is very attractive because cameras are cheap these days and computer power is plentiful. The trick is to figure out how to get valuable information out of at least some fraction of the copious pixel stream. In this paper we demonstrate how environmental landmarks can be visually extracted and tracked in order to estimate the rotation of a mobile robot. This method is superior to odometry (wheel turn counting) because it will work with a wide range of environments and robot configurations. In particular, we have applied this method to a very simple motorized base in order to get it to drive in straight lines. As expected, this works far better than ballistic control. We present quantitative results of several experiments to bolster this conclusion.**

Keywords-robot navigation, vision, landmarks, optical flow

I. INTRODUCTION

Robotic navigation can be divided into two components, a tactical component that handles particular environmental conditions such as obstacle avoidance and doorway traversal, and a strategic component that provides overall route guidance [1]. Our concern here is just the strategic component; there are many well-known, specific solutions to the tactical problems. The question then becomes how does the robot know where it is in relation to something like the floor plan of a building or, at least, how does it know it is correctly following some externally specified path?

For path following, there are a number of solutions that explicitly lay down robot-perceptible guide lines such as tape stripes or RF emitting wires embedded in the floor [2]. Other approaches require the installation of active beacons or at least passive markers (like barcodes) at known locations [3]. The environmental modifications required for these approaches is undesirable due to the cost and labor involved in establishing such an embedded coordinate system. It is also clearly not applicable to novel environments which have not been suitably engineered yet.

For overall positioning there are solutions such as GPS or Glonass. Unfortunately the positional uncertainty is relatively high compared to the size of a typically robot (although this can be decreased using differential GPS) and they rarely works indoors due to signal blockage. While there are starting to be similar indoors systems based on received signal strength of WiFi concentrator nodes [4], the positional uncertainty is still fairly high and the location to signal strengths mapping must be learned (and sometimes relearned if the furniture is rearranged). Another solution is to use an inertial navigation based on gyros and/or accelerometers to integrate incremental displacements from a known starting pose. Such systems are still relatively expensive and most experience drift over time. Also, there are arguably certain environments for which they are inappropriate, such as aboard an aircraft carrier which itself is moving.

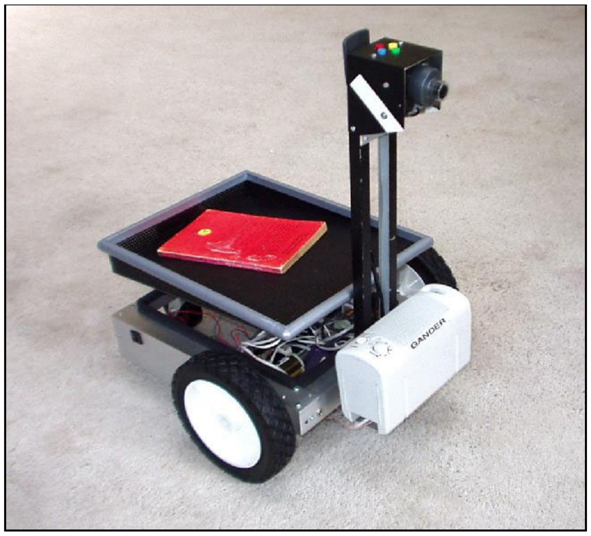
Figure 1. Our robot, “Gander”, has a dual-wheel differential drive and a wide-angle video camera. It is designed to carry items in a basket on its back.

One of the most popular forms of position estimation is odometry. This is typically based on counting wheel rotations (or, equivalently, gear or motor rotations) and then using the geometry of the drive train to infer overall displacement and orientation of the whole robot with respect to the original location. On flat, smooth floors the performance of these systems can be quite good. However, their performance deteriorates rapidly on rough or bumpy surfaces like bricks, slippery surfaces such as sand, rounded or crowned surfaces

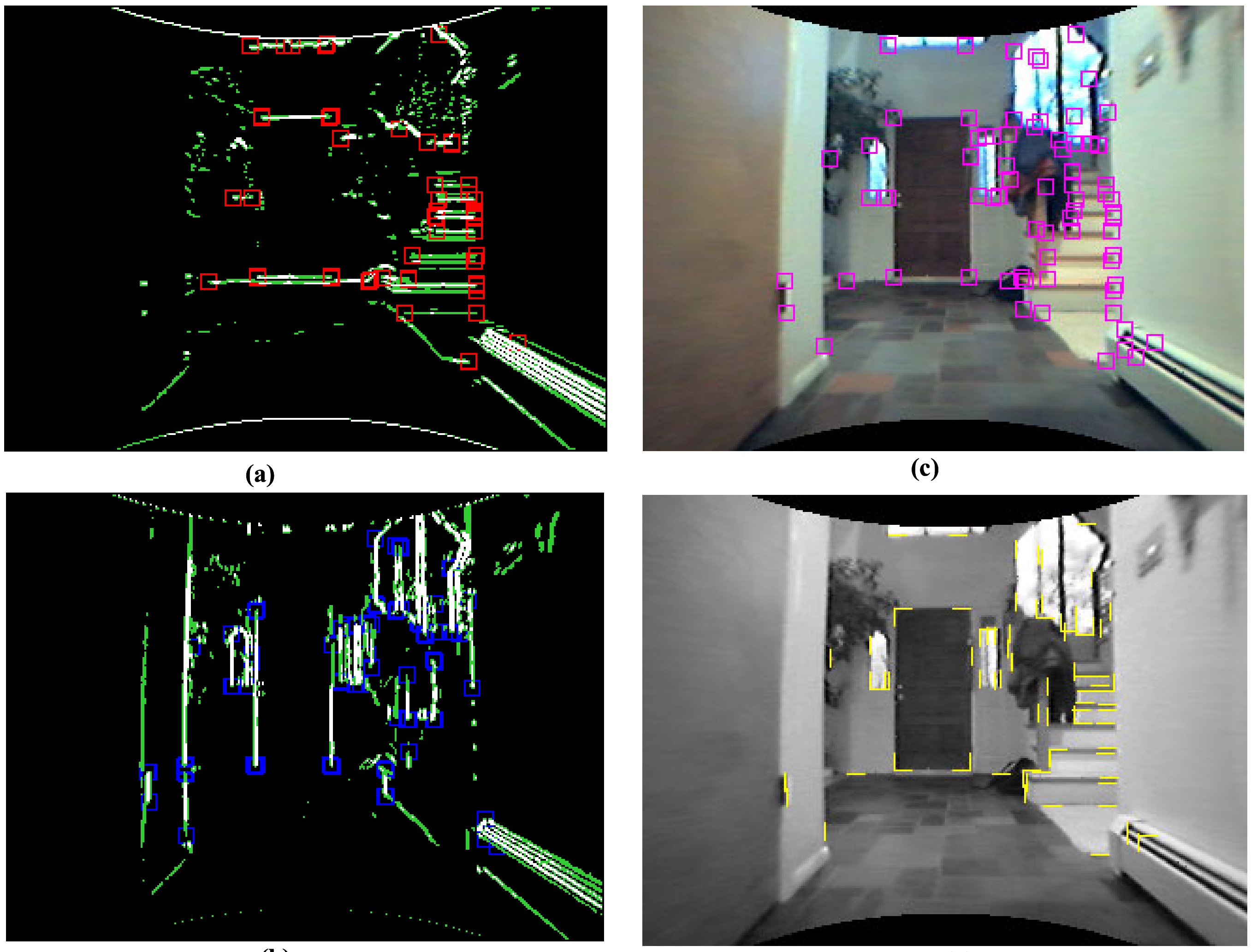
(a) (c) (b) (d)

Figure 2. Landmarks are the end points of horizontal (a) and vertical (b) edges in the image. In (c) the detected landmarks are overlaid on the original image (after lens dewarping). Panel (d) shows the directed edge fragments for each landmark overlaid on the gray scale image from which the edges were extracted.

such as roads, or when they encounter small obstacles like cables or door sills. They are also unsuitable in non-contact environments such as undersea or with aerial vehicles.

Another approach compares current sensor readings to learned or pre-programmed representations of the target environment. There is a long history of systems which compare building floor plans to radial sonar range measurements or, more recently, distance profiles from scanning laser rangefinders [5]. A related approach compares camera snapshots (sometimes very wide-angle) to a series of pre-stored views to find the best match and hence the robot's position within an environment [6]. For all of these approaches, however, not only must the layout of the environment be known -- which paths lead where -- but details of the geometry or appearance of the environment must also be learned (or programmed in).

There are a variety of approaches that exploit various types of landmarks in the environment. The landmarks can be manually installed beacons as described above, specific objects like a particular piece of furniture or overhead lights [7], or pre-existing environmental feature such as posts, convex corners, or doorways [8]. Another class of algorithms relies on landmarks, such as Moravec interest points or contour curvature maxima, which are not directly associated with distinct physical structures [9]. Once the landmarks are found, the robot's positions is determined by triangulation based on the known real-world positions of the landmarks. For such systems to work properly, it is obviously important that the deployment environment actually contains these types of items and that there is some sort of table listing their positions.

Finally, there are other systems that are like odometry or inertial guidance that determine the robot's position based on integrating optical flow [10]. Unfortunately, optical flow can be computationally intensive and small errors can compound quickly over a sequence of video frames. Our system is a combination of the interest point approach, which makes it applicable to a variety of environments, with this flow integration approach, which means no geometric pre-mapping is needed. However, we choose to track the motions of a small numbers of points, which is generally faster than computing optical flow, and we use discrete points matched over long intervals of time, which slows down the drift due to compounded errors.

II. Finding Landmarks

After trying a number of different detection approaches, we decided to use the ending of horizontal and vertical lines as our landmarks. There are many of these in man-made environment and even natural outdoors scenes tend to have a lot of verticals. Since we are interested in straight lines (and later the overall geometry of the scene), we start be undoing the radial lens distortion in our camera. Figure 2c shows the dewarping applied to the original color image. Notice that the edges of the door and wall are fairly straight, and that the floor tiles appear square. A two term correction was necessary because we use a wide angle lens. This was implemented using a fixed sampling table and bilinear interpolation.

$$x' = x + sc_2 * r^2 + sc_4 * r^4$$

We actually convert the image to monochrome (Figure 2d) by averaging the R, G, and B components before dewarping to save time. Separate horizontal and vertical Sobel edge masks are then applied to the monochrome image to yield edge strength maps. These are subjected to a dual thresholding scheme where pixels above the primary threshold are marked as white, while those above a secondary threshold (half the primary) are marked in green. All other pixels are black, as shown in Figure 2a and 2b. The dual thresholding is used to combat the phenomenon of streaking, where a portion of some line temporarily falls below the primary threshold and would otherwise give rise to a spurious landmark.

Next, we scan the horizontal Sobel edges from left to right along the image scan lines. We start a "run" on a white pixel, continue it through other white and green pixels, and terminate it when we encounter a black pixel. This process is depicted in Figure 3. When such a termination is reached, we first check that the edge has been tracked for some minimum length (9 pixels), then we make sure that the pixels directly above and below the ending black pixel are also black (i.e. that the line is truly ending, not bending). If all these conditions are met we generate a new "right ending" landmark. We repeat this procedure scanning backwards, from right-to-left, in the horizontal edge image to find "left ending" landmarks. A similar pair of scans, this time vertically along the image columns, is performed on the vertical edge image to generate "bottom endings" and "top endings". The detected endings are shown as red boxes in Figure 2a, and blue boxes in Figure 2b.

Once all the candidate landmarks have been found, there is a merging and pruning process. Landmarks close to each other with exactly opposite directions are assumed to be either short nuisance segments or small gaps in longer lines. In either case, both candidates are removed. Other landmarks that are close to each other are merged into composite landmarks, like "top left corners", and given slightly adjusted coordinates. Due to the nature of edge detection at corners, during merging the position of horizontal endings is assumed to be flexible in x, and the

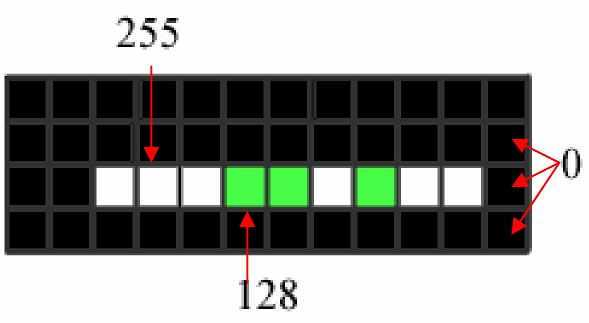

Figure 3. The initial pixel value in a run must be 255., while continuing values must be 128 or 255. A line terminates when the three pixels next to the end have a value of 0. Each run must be at least 9 pixels long.

position of vertical endings is flexible in y. The final landmarks are shown in magenta in Figure 2c overlaid on the original image. Figure 2d shows more clearly the edge fragments (and/or corners) that contributed to each landmark. Note that, because the edges tend to be 2-3 pixels wide and we require a minimum run of 9 pixels, we only mark the ends of lines that are within about +/- 10 degrees of horizontal or vertical in the image.

To make sure there are sufficient landmarks to provide a rotation estimate, we automatically servo the edged detection threshold. Based on the number N of final landmarks found in the current image and a target count (typically 50), we either raise or lower the primary detection threshold for the next image. Generally, we change the threshold some fraction of the way toward a computed target value as detailed below.

$$T' = T + 0.2 * T * (1 - N / 50)$$

For increments with absolute magnitudes of less than one, we force the threshold to change by one in the proper direction. We also impose upper and lower "sanity" bounds on the threshold (such as 150 and 10) to prevent disastrous interpretations and stuck states.

III. Matching Landmarks

To find the correspondence between landmarks in different images, we first determine each landmark's angular position based on its image location and the focal length of the lens. We then try all possible shifts within a certain range of pan and tilt angular offsets. We typically run with a step size of about 0.25 degrees, and a search range of +/- 20 steps horizontally and +/- 10 steps vertically (i.e. +/- 5 degs pan and +/- 2.5 degs tilt). For each potential angular shift combination we determine how many landmarks can be put into correspondence. Landmarks are considered to match when they have at least one direction in common and are within a small distance of each other (after the overall shift). To speed up this calculation, the landmarks are sorted into bins based on their horizontal angle so that only a few neighboring bins have to be search for each landmark comparison. The shift with the most landmarks matched is considered to be the best estimate of rotation. Sometimes, however, there are ties between two different estimates. To resolve these we also compute the residual for each shift as detailed below, and prefer the shift with the lower residual.

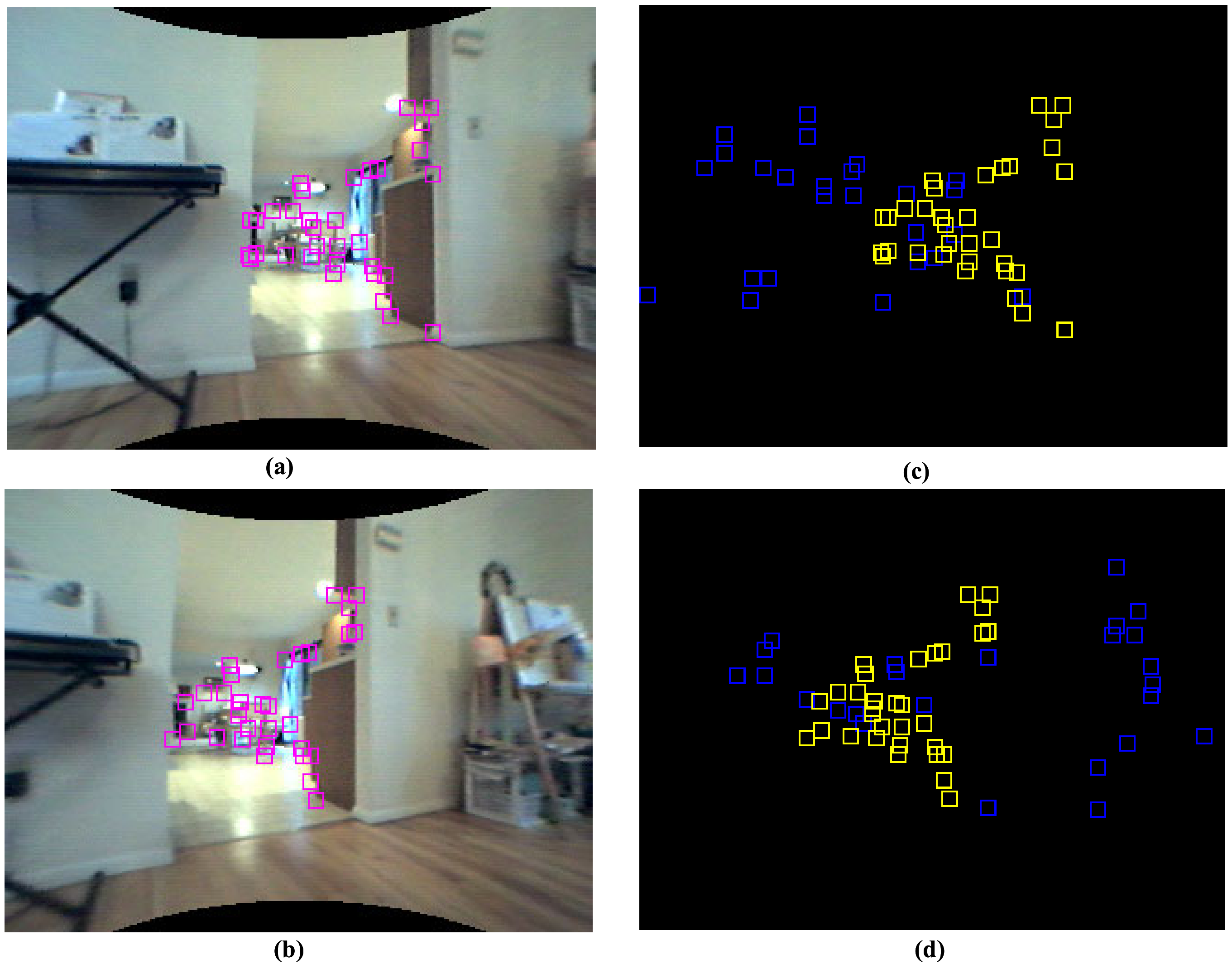

Figure 4. Tracked landmarks. (a) Shows the previous frame while (b) shows the current frame. The landmarks that match between the frames are superimposed in magenta. The landmarks that were found but not matched in each frame can be seen in (c) and (d), where the yellow squares represent tracked ones, and the blue squares represent the untracked. Note the shift of the yellow constellation.

$$R^2 = [\Sigma(x_0 - x_1)]^2 + [\Sigma(y_0 - y_1)]^2$$

Figure 4c and 4d show the landmarks extracted from the images in Figure 4a and 4b, respectively. The matched landmarks for each image are shown in yellow, while the extra, unmatched landmarks are drawn in blue. Just looking at Figure 4c and 4d it is clear that the cluster of yellow (matched) landmarks has shift to the left and hence the robot has turned in that direction. To prove that this makes sense, the matched landmarks are re-plotted in the original images as magenta boxes. As can be seen, the robot is essentially tracking a table and set of chairs in the kitchen without, however, understanding that these are chairs or even individual objects.

We originally matched each frame against the successive one to obtain an instantaneous rotational velocity which was then integrated to obtain orientation. Unfortunately, there is a problem with small estimation errors compounding quickly. Moreover, sometimes a persistent mismatch of landmarks can cause the robots heading to persistently drift in one direction even though the robot is obviously stationary. To ameliorate these problems, we establish an initial "anchor" reference image and compare successive image to this one. We maintain the original anchor frame until we find two frames in a row that match poorly (less than 20 landmarks in correspondence). At this point, we take the current image frame (and associated landmark list) to be the new anchor frame.

We have now explained the complete process involved in generating rotation estimates. These computations can be performed in close to real-time on a modern Pentium 1.2GHz laptop. The image acquisition takes 27ms, the dewarping takes 3ms, landmark finding takes 11ms, and the matching takes 10ms. This comes to a total of 51ms and so allows a frame rate of up to 19 fps. This rate is important since, as part of our experiments, we use the rotation estimate to attempt to servo the robot in a straight path. We employ a simple proportional controller that adjusts the differential power applied to the two wheels. The faster we can compute these estimate, therefore, the tighter the servo loop and the better the control achieved.

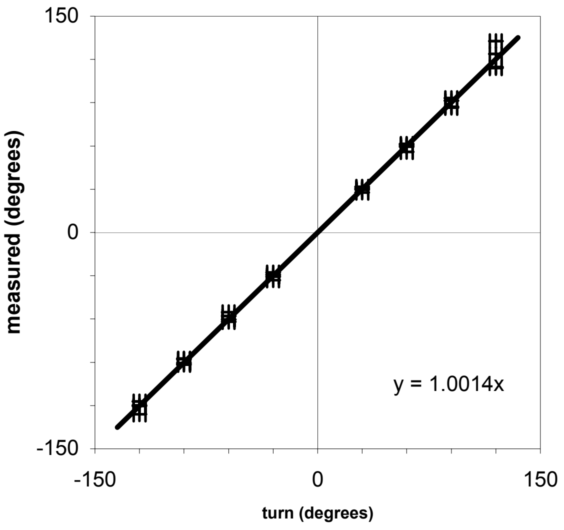

Figure 5. The turn estimates closely match the true turn angles over a wide range of rotations. The data in this graph was collected in four different visual environments.

IV. Experiments

The robot we used (see Figure 1) had two 7 inch diameter wheels driven by DC gearhead motors. The controller was able to set the power level to each motor independently, but there was no feedback loop based on tachometers, back EMF, or current sensing. For sensing, the robot used an X-10 "WideEye" color camera mounted approximately 18 inches above the floor with an 85 degree horizontal field of view. The NTSC analog video signal from the camera was digitized by a Belkin USB 1.1 Videobus adapter at 320 x 240 pixels using 24 bit RGB color. The focal length of the lens was determined to be approximately 172 pixels (for SIF) after dewarping based on the analysis of a number of images with objects at known angular offsets.

In the first experiment we manually rotated the robot a specified amount while recording video at 15 frames per second. This video was then later played back through the analysis system to derive a rotation estimate that could be compared with the known ground truth. The rotation speed was about one degree between frames (15 degs/sec absolute). The heading of the robot was determined by consulting a simple mechanical compass affixed to its head. Care was taken to keep the robot away from large steel objects or under-floor power lines that could skew the readings. For each sequence of rotations the robot was pointed to a specific compass heading, then it was lifted slightly and rotated to a new compass heading. The precision in human alignment of the compass needle was about +/- 2 degrees, so the overall rotation of the robot could have been off by as much as +/- 4 degrees overall since two readings were taken (i.e. a canonical 90 degree rotation might really be from 86 to 94 degrees).

Figure 5 shows the combined results from four different environments: a kitchen, an outdoors deck, a driveway, and a narrow hallway. These environments differed both in the quantity of man-made objects as well as the ratio of man-made to natural objects (which have fewer straight lines in general). For each environment the robot was passively rotated once 30, 60, 90, and 120 degrees to the left, and then once 30, 60, 90, and 120 degrees to the right. This yields a total of 32 data points in the plot, with error bars for the 4 degree uncertainty (the robot's estimates had a much lower quantization of 0.25 degrees). We then performed an overall least squares line fit to this data, forcing the intercept to be zero. As can be seen, the fit is excellent (it overestimates by about 0.14% on average).

TABLE 1: Rotation Estimation Performance

turn (degs)	***+/- 30***	***+/- 60***	***+/- 90***	***+/- 120***
avg. pan error	1.5	2.0	2.2	4.6
avg. tilt error	0.7	1.3	1.7	2.5
new anchors	0.8	2.0	3.8	4.8

Table 1 further explores this data. Here we have taken the absolute error in rotation (pan) for each experiment, and then averaged them across all four environments for each of the forced angles. As can be seen, the error does grow slightly as the rotation angle increases. However, we believe this is due more to the length of the video sequence rather than the angle itself. As a check, we also computed the error of the tilt estimates for each pan angle. Since, in all cases, the robot was on a locally flat surface, the tilt angle should remain zero. As can be seen from the second row of the table, there is similar growth in uncertainty for this estimate, even though the magnitude of the (non) rotation remained constant. Finally, the last row of the table shows the number of new "anchor" frames acquired during each rotation (beyond the initial anchor). As expected, larger rotations causes the system to acquire more anchor frames. On average, a single anchor is valid for a one way offset of about 30 degrees (i.e. +/- 30 degrees from the initial orientation).

In the second experiment we investigated the robot's ability to follow a specific heading using visual information. The robot was positioned at one end of a narrow hallway (see Figure 6) and aimed (optically) at the far end. It was then allowed to travel for a fixed amount of time corresponding to an approximate travel distance of 10, 20, or 30 feet. This was repeated 5 times for each specified distance. At the beginning of each run the robot was centered in the hallway. At the end of each run its lateral offset from the centerline was measured and recorded. The speed of the robot was about 10 inches per second. The laptop computer used to control the robot in real-time was a slower Pentium 450MHz model and hence the visual servo rate was about 7 fps (so about 1.4 inches of travel between processed frames). On average, the robot acquired a new anchor frame every 2 feet.

These results from the plain visual servo tests are plotted to scale on the left half of Figure 7. Note that in no case did the robot crash into a wall; all runs were completed successfully. This is in stark contrast to the ballistic (no visual servo) case where a collision typically occurs within the first 5 feet (since this particular base has a tendency to pull to the left). The average of the absolute offset (sometimes it was to the left, and sometimes to the right) of the robot's position from the hall centerline for each travel distance is listed in the first row of Table 2. Notice that it is fairly constant, largely independent of the travel distance. This is especially good given that the robot was just trying to maintain a constant angular heading; it had no real conception of the geometry of the hallway.

TABLE 2: VISUAL SERVO PERFORMANCE

distance (ft)	*10*	*20*	*30*
normal – avg. offset (in)	5.3	4.3	4.8
torture – avg. offset (in)	3.2	4.5	3.3

We then made the environment more difficult as shown in Figure 6. We constructed a "torture track" by zig-zagging a 1/4 nylon rope back and forth across the hallway. There were 2 traverses in the first 10 feet, 5 in the first 20 feet, and 7 in the complete 30 foot path. The right half of Figure 7 shows the ending lateral offsets for various travel distances under visual servo control. As before, the system was run 5 times for each distance. Just qualitatively, it can be seen that the results are comparable to the case of the "clean" (no rope) hallway. This is borne out by looking at the second row of Table 2. The average lateral offsets are similar to those recorded in the simpler environment. So the system works equally well irrespective of the surface smoothness. Note that the multiple rope traversal would be enough to totally disorient most odometric wheel-revolution counting systems.

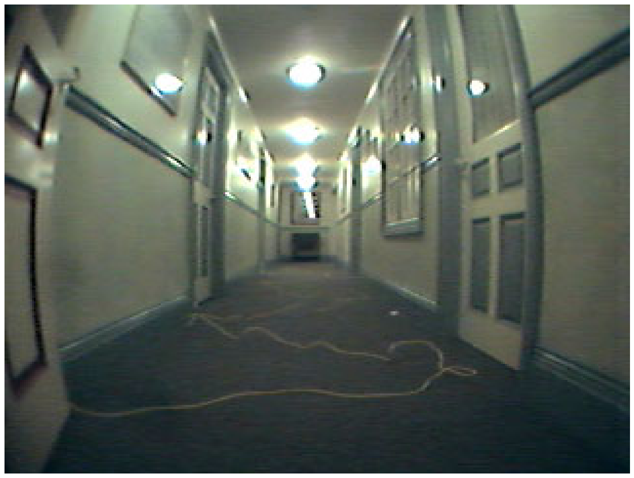

Figure 6. This is the hallway that was used in the experiments. The “torture track” was made by zig-zaging a ¼ inch nylon rope diagonally across the hallway. This would defeat most wheel turn-counting odometry systems.

V. DISCUSSION

We have shown how landmarks can be tracked to give reasonably accurate rotation estimates, and how these estimates can be used to make a robot successfully follow a straight line. Our approach is not as accurate as odometry on a flat surface nor is it completely drift-free. However our approach can be used in odometrically less-friendly environments, and on robots with unknown or changing mechanical properties.

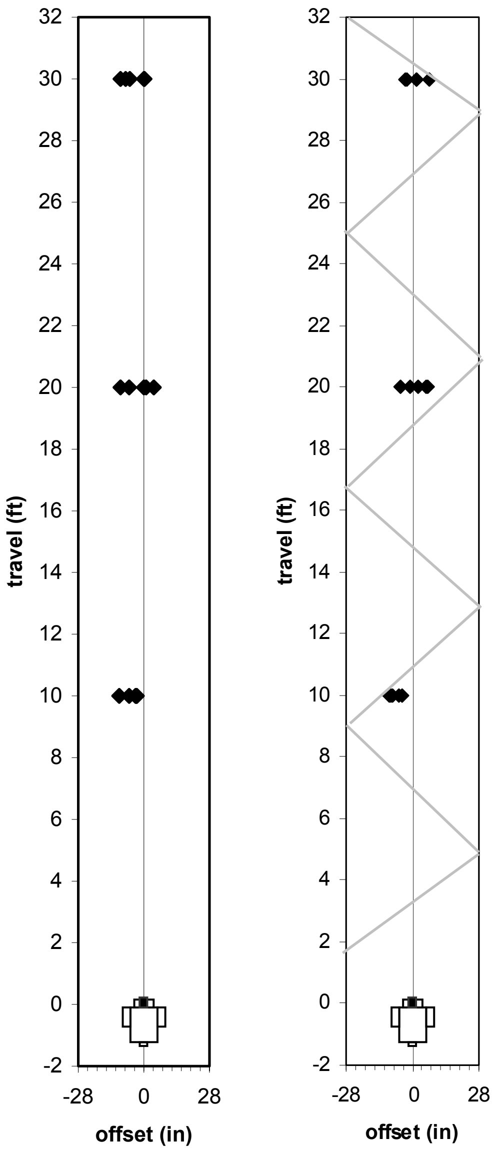

Figure 7. The robot was sent down a narrow hall under visual servo control. The marks show its lateral displacement from the centerline after runs of varying distance. The data on the left was obtained under normal conditions, while the data on the right was obtained from the bumpy “torture track”. As can be seen, surface irregularities make little difference.

The biggest limitation of the current system is the occasional paucity of landmarks in man-made environments. This can happen, for instance, when the robot is turning in a hall with very bare walls. In such cases we have tried to extract subliminal landmarks with a variety of techniques -- histogram equalization, local automatic gain control, relative contrast edge finders, and statistical normalization of texture -- but with little success so far.

We have also been concerned with false matches of landmarks when there are many clustered together. In practice, this turns out not to be much of a problem but we are wary nevertheless. We have tried enforcing minimum separations between landmarks by making sure they only unambiguously match themselves, and have tried suppressing edges whose magnitude, or alternatively ratiometric contrast, are below the average local energy level. None of these variations seem to have much effect.

We have noticed that there are sometimes transient landmarks in an area caused by a fragment of an edge falling below the hysteric threshold, or by slight rotation of the camera causing the minimum length criterion to intermittently fail for oblique edges. To counteract this we have investigated weighted matching where landmarks that have been matched previously are considered more "stable" and given more influence over the shift magnitude when they are matched again. Again, this seems to make little difference and occasionally forces the capture of an extra anchor frame since the old "stable" landmarks bias the system to a less than optimal match (as based on total number of landmarks in correspondence).

There is clearly room for improvement in the hallway traversal system as the trajectories appear underdamped. First, a faster laptop computer could be used to give a more frequent update rate. Second, the gain of the servo loop could be adjusted more carefully. Third, a derivative term could be added to the controller to help damp the oscillations. Fourth, even though we do not have a true velocity estimate, we could assume a roughly constant velocity. This would then let us integrate the small displacement vectors between frames to compute a real 2D offset from the desired path. We could then use a projection pursuit style controller to regain the centerline of the hall.

The obvious next step for this system would be to simultaneously estimate both rotation and translation for real. It is well know that, in general, it is impossible to get a true metric velocity from a visual flow field. However, with additional constraints the scale ambiguity can be resolved. In particular, we know the actual height of the robot's camera above the floor. If we can find landmarks that we believe to actually be on the floor -- like the lowest landmarks in the left and right corners (the center often has reflections of overhead lights), provided that they are below the image center -- we can use the imaging geometry to directly determine the real-world coordinates of these points. This can then be used to set the scaling of the solution and hence to infer the true 2D coordinates of all the other landmarks and, incidentally, the actual translation of the robot base. Note that while this sketch of an approach requires a locally flat floor, it does not require that there be no bumps or that the floor remain flat for large distances.

We also need to integrate this strategic navigation component with various reactive tactical navigation routines. For instance, we could use texture detector or a floor color continuity method for collision avoidance. These should have no impact on the current rotation estimation method. However, if there are dynamic obstacles, like people walking around, this could induce a false sense of rotation in the robot (if they filled a large portion of the field of view). One possible approach to overcoming this problem would be to let the robot do its best rotation estimation, shift temporally adjacent images to negate the perceived ego-motion, then look for residual motion energy (i.e. things not moving with the bulk of the environment). Portions of the image around these areas could then be declared off limits for landmark extraction, leaving more stable areas like the ceiling available. Note that, since we servo on the number of valid landmarks produced, the system will simply lower its threshold to obtain a sufficient number of landmarks in the areas deemed safe.

A similar approach could be used in conjunction with person-following in order to "teach" the robot new paths. We currently have a motion-based following program but are considering changing over to a color region tracker to handle crowds better. Since we know where the target person is in the camera image, we can blank out this whole region for the landmark acquisition system, thus preventing it from relying on landmarks associated with the trainer.

ACKNOWLEDGMENT

The authors would thank Dr. Robert Pavlica of Byram Hills High School for useful comments, and Johuco Ltd. for help in designing the robot.

REFERENCES

[1] J.H. Connell, “SSS: A Hybrid Architecture Applied to Robot Navigation”, ICRA-92, 2719-2724, 1992.

[2] H.R. Everett, *Sensors for Mobile Robots: Theory and Application*, A.K. Peters, 1995.

[3] D. Scharstein and A. Briggs, “Real-time Recognition of Self-similar Landmarks”, *Image and Vision Computing*, 19(11), 763-772, 2001.

[4] "Ekahau Positioning Engine 2.1", http://www.ekahau.com/products/positioningengine, 2003.

[5] S. Thrun et al., “Map Learning and High-Speed Navigation in RHINO”, in *Artificial Intelligence and Mobile Robots*, D. Kortenkamp, R.P. Bonasso, andR. Murphy (eds.), 21-52, MIT Press, 1998.

[6] J. Hong, X. Tan, B. Pinette, R. Weiss, E.M. Riseman, “Image-based Homing”, ICRA-91, 620-625, 1991.

[7] F. Launay, A. Ohya, S. Yuta, “Autonomous Indoor Mobile Robot Navigation by Detecting Fluorescent Tubes”, ICAR-01, 664-668, 2001.

[8] J.H. Lim and J.J Leonard, “Mobile Robot Relocation from Echolocation Constraints”, IEEE *PAMI*, 22(9), 1035-1041, 2000.

[9] S. Se, D. Lowe, and J. Little, “Local and Global Localization for Mobile Robots Using Visual Landmarks”, IROS-01, 414-420, 2001.

[10] N. Ancona and T. Poggio, “Optical Flow from 1D Correlation: Application to a Simple Time-to-Crash Detector”, *Int. Journal of Computer Vision*, 14(2), 209-214, 1995.