Experiments and Thoughts on Visual Navigation

C. Thorpe, L. Matthies, and H. Moravec

Carnegie-Mellon University

Abstract

We describe a second generation system that drives a camera-equipped mobile robot through obstacle courses. The system, which evolved from earlier work by Moravec [6], incorporates a new path planner and has supported experiments with interest operators, motion estimation algorithms, search constraints, and speed-up methods. In this paper we concentrate on the effects of constraint and on speed improvement. We also indicate some of our plans for a follow-on system.

1. Introduction

FIDO is a navigation and vision system for a robot rover. Using only stereo vision, it locates obstacles, plans a path around them, and tracks the motion of the robot as it moves. FIDO's main loop repeatedly:

- picks about 40 points from one member of a stereo image pair
- stereo-ranges those points by a hierarchical correlation technique
- plans a path that avoids those points
- moves forward
- takes two new stereo pictures
- relocates those same points and stereo ranges them again
- deduces vehicle motion from apparent point motion.

This paper describes our experimental investigations and improvements in FIDO's performance. Early versions of FIDO and its predecessor, the Stanford Cart programs, used 9-eyed stereo, took 15 minutes or more per step, and were not always reliable. By using additional geometric constraints, we have been able to increase the reliability while using only 2 stereo images instead of 9. With fewer images and several optimizations, we reduced the run time from 15 minutes to less than a minute per step. We also explored using parallel hardware for further speedups.

Section 2 of this paper discusses the constraints used and their effects on system precision. Section 3 presents optimizations for speed and prospects for parallelism. Finally, section 4 presents some extrapolations on the FIDO experience.

The FIDO system has supported experiments in other aspects of visual navigation, notably *interest operators*, used to pick points to be tracked from image to image, and *path planning*. The results have been presented elsewhere [8, 9]. We found that the simple interest operator used in the original Cart program worked as well as more expensive ones, and it was retained with only slight changes. FIDO does incorporate a new, more flexible, path planner based on a grid combinatorial search and incremental path smoothing.

1.1 Constraints

FIDO uses a variety of constraints to improve the accuracy of its stereo vision and motion solutions. Most reduce the area of the image to be searched by the correlator. A smaller search window reduces the chance of finding a false match and improves system performance in several ways. First, as more points are tracked correctly it becomes easier to identify those incorrectly tracked and delete them. Secondly, more points (and higher precision) improve the accuracy of the motion calculations [10]. Finally, points can be successfully tracked through more images, and over longer distances, for more accurate long term navigation.

Some of the constraints arise from the known relationship between the cameras and the vehicle. Other constraints come from vehicle motion estimates: the image location of an object that has been stereo ranged on a previous step is constrained by approximate knowledge of the vehicle's new position.

We tested FIDO using various combinations of constraints in order to judge their effect. We usually made a live vehicle run with the current best settings, and saved all the images and position predictions in a file. Subsequent runs were done off-line using this stored data, with different constraint settings. Such runs were compared for accuracy of the final calculated position, number of features successfully tracked at each step, and occurrence of any catastrophic failures.

1.2 Imaging Geometry Constraints

These constraints are the simplest to understand and to apply. They depend only on camera and robot geometry, and they are applicable to stereo point matches of both new and previously ranged points.

Near and Far Limits. Point distances are not permitted to be greater than infinity (by the real world) or less than a certain distance (by the nose of the robot). This determines a maximum and minimum stereo disparity of the feature match.

Epipolar Constraint. This is the standard stereo epipolar constraint: if the point of view moves purely sideways the image of a point will also move sideways (in the opposite direction) but not up or down. In the real world of misaligned cameras and distorted vidicons, the image might appear to move a little vertically, so we allow some slop (10% of the image height typical).

1.3 Motion Geometry

The estimated motion of the vehicle from step to step places a strong constraint on point matches. It can be used either *a priori* to limit the search area within an image, or *a posteriori* to gauge the reasonableness of a match. The predicted position of the vehicle can also be combined with the points tracked by vision in the vehicle motion calculation. FIDO uses the motion geometry constraints in the following 4 ways:

Two D Motion. We usually run our robot on locally flat ground, in which case we know it will not pitch, roll, or move vertically. This reduces the problem of determining vehicle motion from 6 degrees of freedom to 3, simplifying the computation and tightening the constraints.

Reacquire Constraint. Given the 3D location of a point relative to a previous vehicle position, and a dead reckoned new position and heading for the vehicle, it is possible to predict where that point should appear in the new stereo pair of images. If this constraint is active FIDO will use the prediction to limit the stereo matcher's search. Three user-settable variables control the error estimates in robot position and orientation, and consequently the size of the search box around the predicted image position.

Prune. When all points from a previous position have been reacquired at a new vehicle location and stereo-ranged, there is a pruning step that looks for points that do not move rigidly with the rest of the points. The points that do not appear to move rigidly have probably been tracked incorrectly, and can be deleted before the least-squares process that solves for vehicle motion. Activating the Prune constraint causes the predicted vehicle position to be included as one of the points in the rigidity test, perhaps weighting the selection to the correctly matched points rather than a coincidentally consistent incorrect set.

Motion Solution. The motion solver determines the motion that minimizes the error between where points have been seen and where they should have been seen given that motion. The predicted vehicle position can be included as one of the points in this least-squares process, weighted more or less depending on the assumed precision of the prediction.

1.4 Results

We made several runs of the FIDO system on Neptune, with fairly consistent results. Data from June 24, 1984 was most extensively analyzed. On that run a single large obstacle was placed a close 2 meters ahead of Neptune's cameras, with the destination set to the far side. It was a tough test for FIDO, since it required the maximum allowed turn (limited by the need to have significant overlap in the views from successive positions) on each step to get around the obstacle and back on course. We ran FIDO with each constraint in what we thought to be its best state, and saved images and dead reckoning information. Then we made a series of off-line runs on the stored data, varying settings and watching the results. Several runs differed in only one parameter from the original, a few others changed two or three. The last group of runs began with one using none of the constraints, followed by a series each with only one constraint on.

Figure 6 summarizes the results. The most important measure of a run's success is the (program's) calculated position at the end of the run: the nearer to the actual (manually) measured position, the better.

Some cautionary notes are in order. The relative success of the run with only the far distance constraint is accidental. During that run, there were two steps where the motion solution was completely wrong but that by coincidence nearly offset each other. Many of the other single constraint runs that appear worse actually had only one wild miscalculation.

Some of the all-but-one constraint runs also appear too good. In many of these cases the dead-reckoning information was sometimes better than the visual tracking. The run with no epipolar constraint has a better final position than the run with no reacquire constraint, because, by luck, it tracked fewer points at the right times and relied on dead reckoning while the latter placed too much reliance on small numbers of tracked points.

Based on our experiences, we make the following observations:

- The epipolar constraint is the single most powerful constraint. Turning it off, and all the others on, significantly decreases the minimum and average number of features tracked and the accuracy of the motion solution. Turning it on, with all others off, significantly increased the number of points tracked. In a sense, this is not surprising, since the epipolar constraint rules out 90% of the image, more than any other constraint.

- No single constraint makes the difference between a successful and a catastrophic outcome.

- In none of the runs was vision as accurate at calculating translation as straight dead reckoning based on motor commands, though in the best runs vision determined the rotation more correctly. It would have been better to use the dead reckoned motion rather than the visually determined one if the number of features tracked dropped below 6 or 7, rather than 4 which was the threshold, at least for the level of ground roughness and mechanical accuracy in the experiments.

- We noticed that even the best runs have about a 20% error in calculated translation, always on the short side. We suspect a small camera calibration error, and possibly systematic errors in representing uncertainty. FIDO calculated a point's 3D location by projecting rays through the centers of the pixels in the stereo images, which gives a location on the near side of the range of uncertainty of distance.

- There is a problem in using all the geometric constraints to cut down the search area since it leaves none for verification and pruning. If we had very accurate motion prediction, we would have to resort to photometry instead of geometry to identify points that had been occluded or otherwise lost.

2. Speed-up Methods

FIDO now takes 30 to 40 seconds per step on a Vax 11/780 under Unix. To run in real time, we would have to reduce that to about 1 second per step. We have looked at several speed-up techniques, including faster processors, dedicated hardware, coding hacks, and parallel processing.

Faster General Purpose Computers

Our VAX is about a one-MIP (Million Instructions Per second) machine. It is technically possible to get the required speedup by simply obtaining a 30-MIP or faster computer. Budget and logistics leave this as a tantalizing future possibility.

Commercial Array Processors

Buying a commercial array processor is more feasible for us than buying a faster computer. About 90 percent of the runtime in FIDO occurs in image array operations and geometric calculations, particularly the convolutions in point matching. These are done by small pieces of code that work on large amounts of data, and are well suited to the pipelined vector arithmetic of available array

processors. We estimate, for instance, that a 100 MIP array processor could give us the desired factor of 30 speedup. We've made several serious attempts to acquire one; so far, this remains another tantalizing possibility.

Coding optimizations

Much effort has been expended on speeding up the Vax implementation. We feel there is little room for left for significant improvements in a time-shared, paged-memory environment. The basic routines, such as the correlator and the interest operator, fit all the criteria for good candidates for optimization [2]: the code is fairly well understood, stable, small, and accounts for a large amount of run time. For instance, the implementation of the correlator uses the following coding techniques:

- The calculations of parameters of the correlating window are done once, outside the main loop.
- Sums and sums of squares for consecutive columns and rows are calculated by Price's technique [7]. The next window total is calculated by adding in the total for the column that just entered the window and subtracting off the total for the column that just left the window.
- Squares are calculated by table lookup. Since the squares are of sums of two pixel values, the table needs only 511 entries.
- Image windows are moved by pointer swapping, rather than by data transfers.
- Loop indices count down to 0, since the VAX hardware has an efficient test-for-not-0-and-branch instruction.
- Formulas are rewritten to eliminate extra calculations. For example,
  $2 * \Sigma(img1 * img2) =$
  $\Sigma((img1 + img2)\uparrow 2) - \Sigma(img1\uparrow 2) - \Sigma(img2\uparrow 2)$
  gives a way of calculating the sum of the products of the pixel values by additions (which are cheap) and squares (which can be done by table lookup) rather than multiplications. The individual sums are also used in other parts of the calculation, so in this case the sum of products comes for free.
- Loop unrolling. The code in the innermost loop is written n times in line, rather than written once inside a loop that counts to n. This saves n increments of the counter and n tests for the end of the loop.
- Register use. The most frequently used variables are located in hardware registers.

These programming techniques reduce the run time of the correlator from 140 ms per call for a straightforward implementation to 4 to 5 ms per call. Similar optimizations have been performed on the other tight loops, such as in the interest operator and the image fine to coarse reduction routine. The user-level routines have been optimized to the point that the single routine that uses the most CPU time is now an image unpacker.

Dedicated hardware

A dedicated microcomputer running FIDO with enough memory to store all the relevant images offered some hope. We tried an implementation of the correlator on a 10-MHz MC68000 system, with all the images held in integer arrays. After eliminating all floating point operations the resulting code still took 29 microseconds per call to the correlator, compared with 4 to 5 on the VAX.

2.1 Parallelism

There are several ways to break FIDO into separate processes that can run in parallel on different machines, including pipelining on macro or micro scales or the use of a master/slave system.

Macro Pipelining

One process might do the reductions, the next could do reacquires, the next the match, another motion-solving, and the last path planning. This organization improves throughput but not the latency. The problem with this method is the sequential nature of FIDO. Since all the image reductions have to be finished before the reacquires can start, all the matches done before the path planning, and so forth, each pipeline stage has to wait for the previous stage. Since each step takes as long as on a serial machine, and since the steps are done sequentially, the time to process any one set of images is the same as on a single processor system.

Micro Pipelining

The processes could be subdivided more finely. For instance, one processor might do the first level of match for one point after another, handing its results to the process that does the next level of match. When matches are finished, the pipeline could be reconfigured for path planning, and so on. This approach requires huge communication bandwidth between processes.

Master/slave

This method has one master process and several identical slave processes. Each slave handles every image processing task: reduction, matching, and interest operator. At any time all the slaves work on the same task with different data. For example, during image reduction, each slave reduces part of the image, and during matching each slave processes its own queue of points. The master process does tasks that require global knowledge such as path-planning or motion-solving, and coordinates the slaves. This more flexible organization avoids several delays inherent in pipelines.

We implemented variants of this idea in our Ethernet-connected multi-Vax environment. Given the existing uniprocessor code, the task was not difficult. The slaves required new code for communication with the master, but the actual work is done by calls to the old image processing routines. The master contains the old path planning and display code, and new communication code and dispatch tables to keep track of each slave's activities. When a slave completes a task the master updates its dispatch table, finds a new task and puts the slave to work again. For instance during point matching each slave is initially given one point to correlate. When a slave finishes its correlation, the master hands it a new point to find. When all the points are handed out the master redundantly hands out points that are still in process on other slaves, and accepts the first answer to be returned, giving some protection against overloaded or crashed processors.

A version of the system that used several VAXes in parallel was swamped, as expected, by the overhead of squeezing images between machines through the Ethernet. Another version that used multiple processes on a single Vax gave us some idea of the performance that might be possible if faster communication, perhaps through shared memory, were available.

The single machine version uses the same decomposition as the multiple machine version, and the same general-purpose interprocess communication package. Because of limitations in the communications package, each slave calculated its own image pyramid.

2.2 Timings for a 28-Step Run

Single Processor	978	
One Slave		
Master	216	
Slave 1		626
Five Slaves		
Master	234	
Slave 1		403
Slave 2		402
Slave 3		403
Slave 4		402
Slave 5		400

Notes:

- The time for the Master varies little with the number of slaves.
- Without image acquisition or communication package overhead the time for a single slave would be about 325 seconds or 12 seconds per step.
- Without image or communication overhead, and with the time for picture reduction shared evenly, the time for each of the five slaves would be 65 seconds, or about 2.5 seconds per step.
- The work spreads very evenly among the slaves. With 5 slaves, the workload is balanced to within the accuracy of our measurements.
- If the master process did not handle images, had zero-cost communication, and didn't have to do image distortion correction, it could run in 75 to 80 seconds, or about 3 seconds per step.
- By comparison, the original uniprocessor system runs in 978 seconds, or 35 seconds per step. With the advantages we assumed above (no image handling overhead) it would still have taken 503 seconds, or 18 seconds per step.

2.3 Remarks

Our experiments suggest that it is possible to decompose FIDO into a 5 to 10 fold parallel set of efficiently cooperating parts running on conventional processors. To realize the run times suggested above we would need the following:

- Shared main memory large enough to hold at least two image pyramids without swapping or data packing. (2 * [256 + 64 + 16 + 4 + 1 + .25] = 700 KiloBytes).
- Fast interprocess communication for small messages.
- At least 5 processors. It takes 5 slave processors to bring the image processing time into the same range as the master process' time.
- A device able to digitize images directly into the shared memory.
- Cameras with less image distortion than our current vidicons, so image warping would not be needed.

3. The Next System

Some simple hardware enhancements could improve FIDO's performance. A pan mechanism for the stereo cameras would permit larger turns while still maintaining continuity of field of view. Motion and heading sensors would improve navigational accuracy and eliminate some catastrophic misperceptions.

Navigational accuracy could also by improved by modifying the motion estimation algorithm. The current algorithm reacquires features in new a image by searching for the features within windows predicted by an *a priori* motion estimate. This makes poor use of the assumption that objects do not move; that is, that they appear to move rigidly from frame to frame. Since all search windows are defined before any search begins, constraint is not propagated from one match to another. A seemly better approach is the iterative registration method [1], [3], [4]. In this method, 3-D feature positions are projected onto a new image using an initial motion estimate, then the motion estimate is refined to optimize some measure of match in the image. We are currently experimenting with the variation proposed by Lucas [4] and plan to report empirical results in the near future.

Two bugbears in our systems to date have been the calibration of camera and motor parameters and the represention of uncertainty in the 3-D locations of perceived objects. We are considering an adaptive approach that calibrates the cameras (semi-)continuously on the fly and adjusts the motor control parameters from observations of past vehicle motions. A simple technique like this was used successfully in an early program that drove the Stanford Cart in straight lines [5]. We are also looking at carrying along uncertainties in feature locations and updating the uncertainty as new measurements are taken. Eventually, we hope to automate the process to the point where calibration simply requires turning on the vehicle and letting it run by itself for a while.

Acknowledgement

This work has been supported by the Office of Naval Research under contract number N00014-81-K-0503.

References

1. H.G. Barrow, J.M. Tenenbaum, R.C. Bolles, H.C. Wolf. Parametric Correspondence and Chamfer Matching. IJCAI-5, 1977.
2. Jon Louis Bentley. *Software Series.* Volume : *Writing Efficient Programs.* Prentice-Hall, 1982.
3. D.G. Lowe. Solving for the Parameters of Object Models from Image Descriptions. Proc. ARPA IUS Workshop, April, 1980.
4. B.D. Lucas, T. Kanade. Optical Navigation by the Method of Differences. Proc. ARPA IUS Workshop, 1984.
5. Moravec, H. P. Towards Automatic Visual Obstacle Avoidance. The 5th International Joint Conference on Artificial Intelligence, MIT, Cambridge, Massachusetts, IJCAI, August, 1977, pp. 584.
6. H.P. Moravec. *Obstacle Avoidance and Navigation in the Real World by a Seeing Robot Rover.* Ph.D. Th., Stanford, April 1980.
7. K.E. Price. *Change Detection and Analysis in Multi-spectral Images.* Ph.D. Th., Carnegie-Mellon University, Department of Computer Science, 1977.
8. C. Thorpe. An Analysis of Interest Operators for FIDO. Proceedings of IEEE Workshop on Computer Vision: Representation and Control, 1984.
9. C. Thorpe. Path Relaxation: Path Planning for a Mobile Robot. Proceedings of AAAI-84, 1984.
10. Roger Tsai and Thomas Huang. Analysis of 3-D Time Varying Scene. Tech. Rept. RC 9479 (# 41904), IBM Watson Research Center, July, 82.

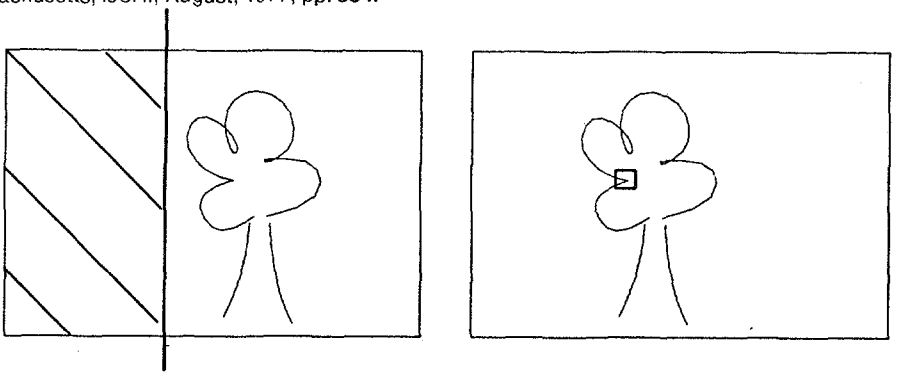

Figure 1: Far Limit

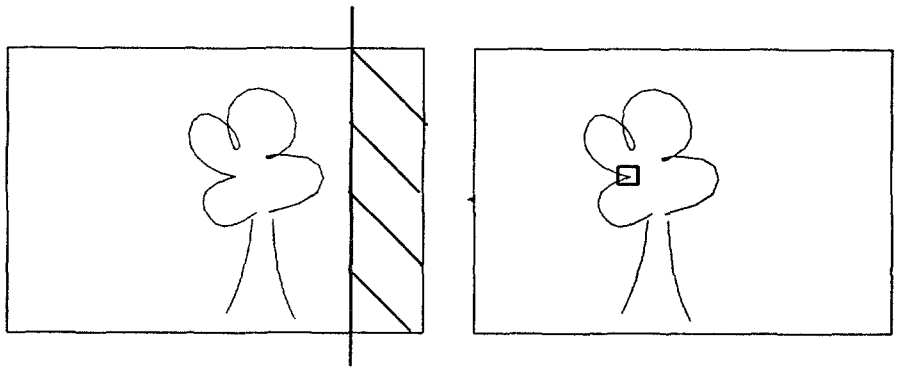

Figure 2: Near Limit

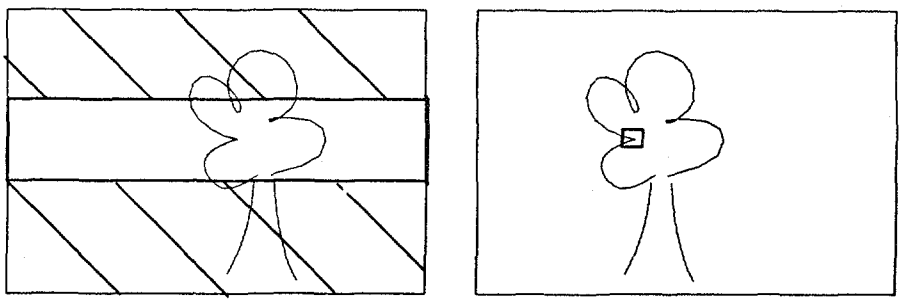

Figure 3: Epipolar Constraint

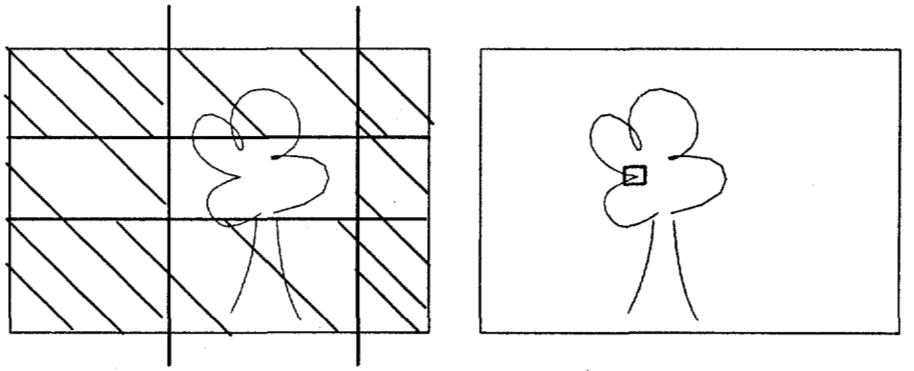

Figure 4: Combined Imaging Constraints

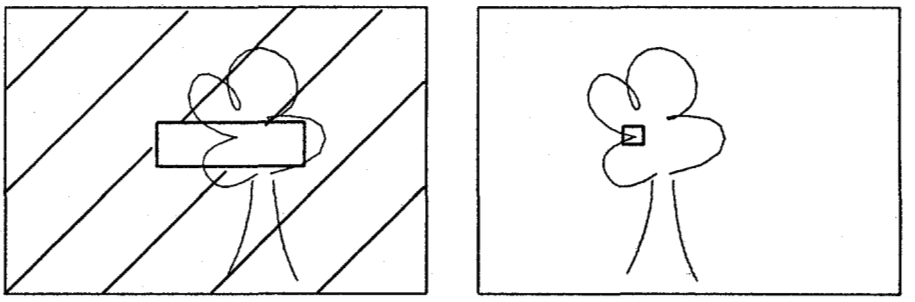

Figure 5: Reacquire Constraint

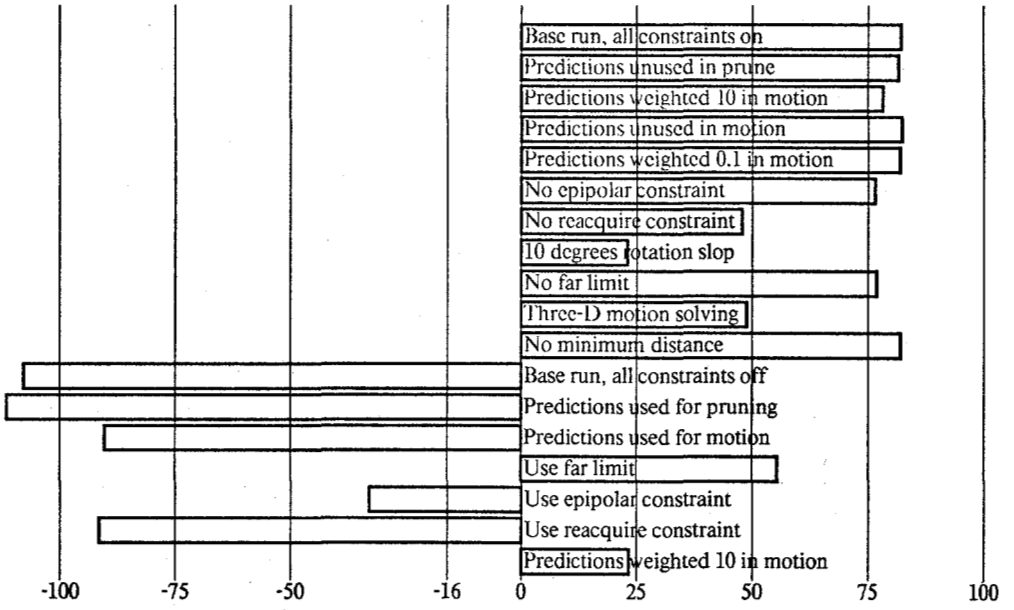

Figure 6: Distance calculated as percentage of actual distance traveled