Controlling a Wheelchair with Image-based Homing

Thomas Röfer
Zentrum für Kognitionswissenschaften, FB 3 Informatik
Universität Bremen
Postfach 330 440, D-28334 Bremen
roefer@informatik.uni-bremen.de

7/4/97

Abstract

This paper presents an application of image based homing for the navigation of a wheelchair. It describes which information can be generated by comparing panoramic images and how this information is used to cope with the kinematic restrictions of the wheelchair in figure 1.

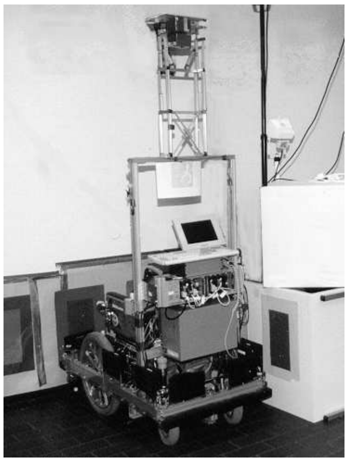

Figure 1: The Bremen wheelchair

1 Motivation

The navigation of autonomous mobile systems (robots) is still an unsolved problem. They operate in a three-dimensional and changing world. In this environment they have to navigate in real time. Therefore, there has to be a suitable technique for navigation. Methods that are based on three-dimensional models are very complex: it takes a lot of time to match the sensor data with three-dimensional objects stored in large databases. Thus, it seems to make sense to search for methods that are less complicated. One possibility is to follow biological examples because even very "simple" animals are able to navigate very precisely.

Cartwright and Collett [1, 2] analyzed the navigation of honeybees. They supposed that bees navigate with the help of "snapshots", i.e. simple images of the environment. The bees remember the exact position of the beehive and their food sources with these images. If they approach such a position, they use the differences between the current image they see and the target image to get back to the stored position. This behavior is also called "homing". Lehrer [6] found that bees use color information for navigation tasks. Cartwright and Collett as well as Wittmann [9] developed some models for image based homing, but they have rarely been used in robotics, for example Hong et al. [4].

Proc. AISB workshop on "Spatial Reasoning in Mobile Robots and Animals", Manchester 1997.
Technical Report Series, Department of Computer Science, Manchester University, ISSN 1361 - 6161.
Report number UMCS-97-4-1. http://www.cs.man.ac.uk/csonly/cstechrep/titles97.html

2 The *Panama* Algorithm

Image-based homing needs a method that obtains the movement instructions from the images taken. The image processing algorithm presented here matches two images taken at different positions and determines the rotation between these positions and the bearing direction from one place to the other. These two values can be used for navigation tasks as shown in the next section. The image processing method uses one-dimensional panoramic color images that are taken by the apparatus shown in figure 2. A spherical mirror is mounted above a video camera. The camera points up to the bottom of the sphere (Figure 2a) and sees a 360° "hemispherical" image (Figure 2b). The horizontal plane is mapped to a circle in the camera's image. The pixels of the circle are used as panoramic images, i.e. a one-dimensional sequence of pixels that wraps round to form a circle. Each of these pixels "looks" in a different direction. Therefore a pixel's position in the sequence corresponds to the angle that it is heading to.

The *Panama*-algorithm (*Pan*oram*a* *ma*tching) presented here is a correlation based image processing method that calculates the optical flow field [3] between two of these images taken at different positions. The flow field is determined by matching the two images using neighborhood preservation for stabilization. This is inspired by Kohonen's [5] self organizing feature map but the neighborhood preservation is more strict. The whole process is performed under the assumption that the relative sequence of objects is exactly the same in both images [7].

Mathematically panoramic pictures can be described as a vector I with n components that are the images' pixels p_i. In the presented implementation the pixels consist of the color components red, green and blue. It is conceivable that the pictures are generated from information totally different , e.g. the surface properties (roughness, temperature) of the objects in a particular direction.

$$\begin{aligned} I &= (\, p_0^I \quad \cdots \quad p_{n-1}^I \,) \\ \bigwedge_{i=0}^{n-1} p_i &= \begin{pmatrix} r_i \\ g_i \\ b_i \end{pmatrix} \end{aligned} \tag{1}$$

Panama determines where the objects of one picture can be found in the other. This problem is solved on the level of pixels. *Panama* determines where each pixel of one picture J is placed in the other image I. If two pictures are taken at positions which have different distances to a particular object, this object is represented by a greater number of pixels in one image than in the other. Therefore, a one-to-one relation between the pixels of the two pictures cannot occur but the points of one picture can be found *between* the pixels of the other. *Panama* calculates the similarity of each pixel in the first picture with each pair of adjacent pixels in the second image. This establishes where between the two neighboring points the similarity to the pixel in the first picture is best.

2.1 Correlation matrix

The similarity between each pixel of the image J and an adjacent pixel pair of the other picture I is stored in the correlation matrix. In addition, the place is stored where the highest similarity occurs between

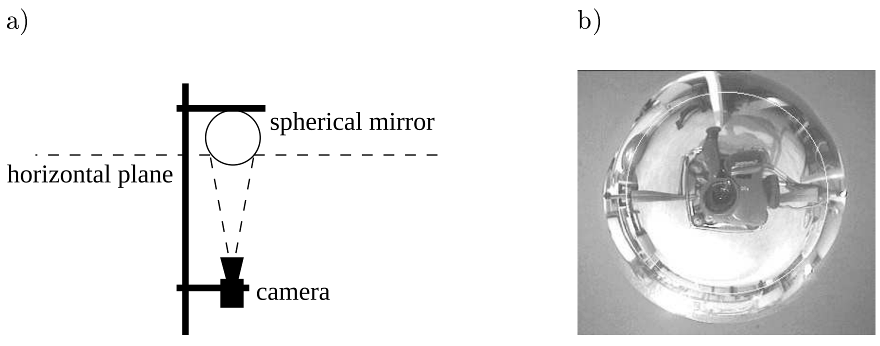

Figure 2: Illustration of a technique to take panoramic images.

the two neighboring pixels and the single pixel. As both images consist of n pixels, the matrix has $n \times n$ entries:

$$C^{I,J} = \begin{pmatrix} c^{I,J}_{0,0} & \cdots & c^{I,J}_{n-1,0} \\ \vdots & \ddots & \vdots \\ c^{I,J}_{0,n-1} & \cdots & c^{I,J}_{n-1,n-1} \end{pmatrix} \tag{2}$$

The similarity between the pixels is not directly determined from the color information but from *weight vectors* which have been previously calculated for every pixel. Weight vectors can consist of many different components, for example of the original color values, their change or smoothed values. In the presented implementation, the weight vectors contain the original color values and their first derivations. The weight vectors of a picture I are defined as

$$W^I = \begin{pmatrix} w^I_0 & w^I_1 & \cdots & w^I_{n-1} \end{pmatrix} \tag{3}$$

A common method to calculate the similarity between two vectors is the Euclidean distance. As the similarity between one vector and two others must be calculated, the method has to be extended. To implement this, a geometric approach is pursued. It connects the two adjacent weight vectors of I by a straight line. The shortest distance between the straight line and the weight vector of a pixel from picture J is used to measure similarity. It is calculated by dropping the perpendicular from the single vector to the straight line. If the perpendicular point is not between the two weight vectors on the straight line, the closest point inside the valid range is used instead.

The similarity $s^{I,J}_{i,j}$ is stored in the correlation matrix for each pixel combination. In addition, it is stored where the similarity has its maximum. This information is represented as a real numbered index $x^{I,J}_{i,j}$. It is calculated by adding the index of the first of the two adjacent pixels with an offset in the range of $[0 \dots 1[$ which states where the matching point is on the straight line between the two weight vectors. Dotted versions of the operators $+$ and $-$ consider the circularity of the images:

$$\begin{aligned} \bigwedge_{i=0}^{n-1} \bigwedge_{j=0}^{n-1} c^{I,J}_{i,j} &= \begin{pmatrix} s^{I,J}_{i,j} \\ x^{I,J}_{i,j} \end{pmatrix} \\ \text{where} \quad & s^{I,J}_{i,j} = |b - af| \\ & x^{I,J}_{i,j} = i + f \\ & a = w^I_{i+1} - w^I_i \\ & b = w^J_j - w^I_i \\ & f = \max\left(\min\left(\tfrac{ab}{a^2}, 1-\varepsilon\right), 0\right) \\ & \varepsilon \approx 0, \varepsilon > 0 \end{aligned} \tag{4}$$

2.2 Matching process

The pixels of the two images I and J are matched in an iterative process under the assumption that the order of objects is the same in both pictures. As a result, pixels are not always assigned to the point with the greatest similarity in the other image. Instead, a compromise is found which preserves the order of pixels. It pairs the points that are as similar as possible. The matching is done by moving the pixels of picture J. This is done by defining the positions of the pixels in this image being variable. These positions are initially identical with the indices of the corresponding pixels. In contrast to the indices, the positions of the points are real numbers and so the pixels can be moved to every place in the range of $[0 \dots n[$.

The matching occurs in t_{max} steps. In every step t of the adaptation process, one pixel of picture J is selected. Then the neighborhood of its current position is searched in a range of $\pm\sigma_t$ pixels for the most similar pixel pair from the other image. It is only necessary to search the corresponding part in the row of the correlation matrix to which it belongs. Once this has been done, this pixel is moved to the position of the most similar point that is as well stored in the matrix. The pixel's vicinity of also $\pm\sigma_t$ pixels is moved in the same direction in a way that does not change the order of the positions. The neighborhood $]-\sigma_t \dots \sigma_t[$ initially contains the whole picture. The adaption process executes until the neighborhood shrinks to only include the pixel. This allows the pixels to move to their globally most similar positions in the beginning of the process. While the process runs, they can select their partners from a continuously shrinking number of positions. With time, fewer pixels move and distances moved in a step decrease. Finally each pixel should reach its optimal position, with respect to the base condition of the neighborhood preservation.

The size of the searched and adapted neighborhood σ_t is determined from the number of the adaptation step t. The decrease of σ_t is not linear, it is greater in the beginning and gets smaller with time:

$$\sigma_t = 1 + \frac{\frac{n}{2} - 2}{t_{max}^2} (t_{max} - t)^2 \tag{5}$$

The positions of the pixels of the image J are stored in vector Y. This vector is initialized with the indices of the points, i.e. the numbers from 0 to $n - 1$:

$$\begin{aligned} Y_{init} &= \begin{pmatrix} y_0 & y_1 & \cdots & y_{n-1} \end{pmatrix} \\ &= \begin{pmatrix} 0 & 1 & \cdots & n-1 \end{pmatrix} \end{aligned} \tag{6}$$

In every step t one pixel $p_{j_t}^J$ is selected. j_t is a function that enumerates the indices with a uniform distribution. The correlation matrix is searched for the most similar pair of pixels in the neighborhood of $\pm\sigma_t$ pixels around the position Y_{j_t}. The result of this search is the position of the best matching point x_{best}.

$$\begin{aligned} s_{i_{best},j_t}^{I,J} &= \min_{i=\lfloor y_{j_t} \dot{-} \sigma_t \rfloor \dot{+} 1}^{\lceil y_{j_t} \dot{+} \sigma_t \rceil \dot{-} 1} s_{i,j_t}^{I,J} \\ x_{best} &= x_{i_{best},j_t}^{I,J} \end{aligned} \tag{7}$$

After the most similar position has been found, the pixel is moved to this place. In addition, all neighboring points with the indices $\left] j_t \dot{-} \sigma_t \ldots j_t \dot{+} \sigma_t \right[$ are moved in the same direction. The greater the distance between the positions of these neighbors and Y_{j_t} is, the less the pixels are moved. This preserves the order of their positions. Figure 3 illustrates the change of the positions during an adaptation step.

$$\bigwedge_{i=0}^{n-1} y_i^{new} = \begin{cases} x_{best} \dot{+} \frac{x_{best} \ddot{-} y_{j_t \dot{-} \sigma_t}}{y_{j_t} \ddot{-} y_{j_t \dot{-} \sigma_t}} \left(y_i \ddot{-} y_{j_t} \right) & \text{if } i \in \left[1 \dot{-} \sigma_t \ldots 0 \right] \\ x_{best} \dot{+} \frac{x_{best} \ddot{-} y_{j_t \dot{+} \sigma_t}}{y_{j_t} \ddot{-} y_{j_t \dot{+} \sigma_t}} \left(y_i \ddot{-} y_{j_t} \right) & \text{if } i \in \left[1 \ldots \sigma_t \dot{-} 1 \right] \\ y_i & \text{otherwise} \end{cases} \tag{8}$$

2.3 Optical flow

After the matching process, the vector Y contains the adapted positions of the pixels in J. The difference between their original positions and their matched positions is the optical flow Y' between the two pictures:

$$\begin{aligned} Y' &= \begin{pmatrix} y_0' & \cdots & y_{n-1}' \end{pmatrix} \\ &\bigwedge_{i=0}^{n-1} y_i' = y_i \ddot{-} i \end{aligned} \tag{9}$$

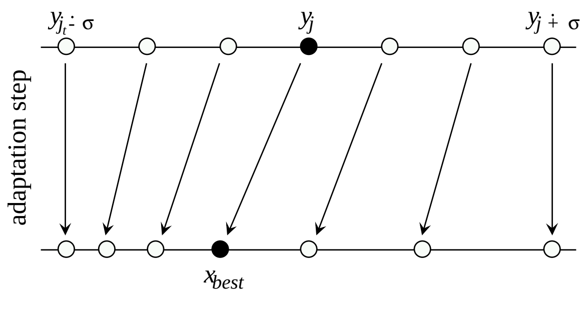

Figure 3: Illustration of the adaptation step in (8).

The calculated flow vectors are in the range of $\left[-\frac{n}{2} \ldots \frac{n}{2}\right[$. As the neighborhood preservation has been used to calculate Y, the flow field Y' is continuous. Therefore, the flow vectors do not cross each other, with one exception: if there are flow vectors up to $\pm\frac{n}{2}$, it is possible that adjacent vectors jump from $-\frac{n}{2}$ to $\frac{n}{2}$ and back (Figure 4c). This may confuse further processing. Therefore, the flow field is converted to a continuous one (Figure 4d) by adding an appropriate offset vector O:

$$Y'' = Y' + O \tag{10}$$

2.4 Rotation

The rotation describes, how much the orientation of the camera has changed between the taking of the two images. It can be determined from the optical flow field that has been calculated in the previous section. Each vector of the flow field consists of two parts: a rotational component and a translational component. This is illustrated in figure 4e. As the pictures are panoramic, the rotational component is equal in all flow field vectors.

Without rotation half of the flow vectors point clockwise and half point counterclockwise. So finding the rotation is equivalent to transforming the flow field in a field which satisfies this criterion. As all flow vectors are only single real numbers, a number must be found which can be subtracted from all vectors, resulting in half of the vectors being positive and half being negative. To find this number all vectors are sorted in increasing order:

$$Y^{sort} = \mathrm{sort}\,(Y'') \tag{11}$$

To make half of this sorted vector field negative and half positive by subtracting a single value from all entries y_i^{sort} it is enough to find a value which makes $y_{\frac{n}{2}-1}^{sort}$ negative and $y_{\frac{n}{2}}^{sort}$ positive. All numbers between $y_{\frac{n}{2}-1}^{sort}$ and $y_{\frac{n}{2}}^{sort}$ satisfy this criterion. Therefore, the center of this range is chosen as the best compromise. The rotation ω is calculated in radians as:

$$\omega = \pi \frac{y_{\frac{n}{2}-1}^{sort} + y_{\frac{n}{2}}^{sort}}{n} \tag{12}$$

2.5 Distance

The *distance* between two images is the size of the translational flow field. It corresponds to the spatial distance between the positions where the two pictures have been taken at, but it is not a metrical measurement. Instead, it depends on the detachment to the surroundings.

First, the rotation has to be eliminated from the flow field in order to calculate the translational components. The rotation ω is simply subtracted from the flow Y'' to get the translational flow field Y''':

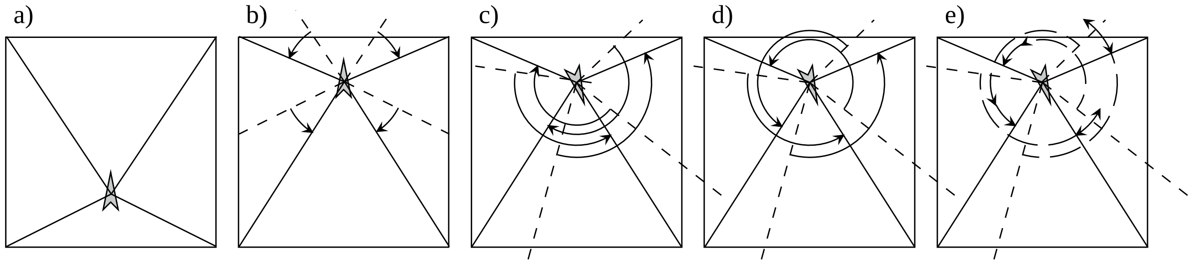

Figure 4: Illustration of the correction of the optical flow. a) The camera's original position with the four bearings of the corners. b) The camera's position after a translation. The arrows indicate the flow Y' between the bearings of the corners in a) and now. c) The camera's position after a translation and a rotation. The arrows indicate the flow Y' without correction. d) Same as c) but the arrows now indicate the corrected flow Y''. e) The solid arrows indicate the translational flow that is identical to the flow in b), the arrows with the broken lines indicate the rotational flow.

$$
\begin{gathered}
Y''' = \begin{pmatrix} y_0''' & \cdots & y_{n-1}''' \end{pmatrix} \\
\bigwedge_{i=0}^{n-1} y_i''' = y_i'' \dot{-} \frac{n}{2\pi}\omega
\end{gathered}
\tag{13}
$$

The distance φ between the two images is calculated by summing up the squares of all flow vectors:

$$
\varphi = \sum_{i=0}^{n-1} (y_i''')^2 \tag{14}
$$

2.6 Bearing direction

The *bearing direction* δ is the direction from the position where picture I has been taken to the position where image J has been taken. δ is relative to picture I or—more exactly—to the direction that the pixel p_0^I is heading to.

The optical flow field consists of n vectors. The direction of the flow vectors is perpendicular to their position in the picture. Their signs determine if they point in clockwise or counterclockwise direction. Each individual flow vector *votes* for a bearing direction. A vector's voting direction is perpendicular to the direction that the corresponding pixel is heading to in the picture. It can be calculated by adding a quarter of the size of the picture n to vector's position. The sign of the added offset depends on the sign of the corresponding vector.

$$
\begin{gathered}
V = \begin{pmatrix} v_0 & \cdots & v_{n-1} \end{pmatrix} \\
\bigwedge_{i=0}^{n-1} v_i = \begin{cases} \begin{pmatrix} \cos(\alpha_i) \\ \sin(\alpha_i) \end{pmatrix} & \text{if } y_i''' \neq 0 \\ \begin{pmatrix} 0 \\ 0 \end{pmatrix} & \text{if } y_i''' = 0 \end{cases} \\
\text{where} \quad \alpha_i = 2\pi \frac{i}{n} + \begin{cases} \frac{\pi}{4} & \text{if } y_i''' < 0 \\ -\frac{\pi}{4} & \text{if } y_i''' > 0 \end{cases}
\end{gathered}
\tag{15}
$$

All the votes are added up as two-dimensional direction vectors. The direction of the resulting vector is used as the searched bearing direction δ.

$$
\delta = \arctan \sum_{i=0}^{n-1} v_i \tag{16}
$$

3 Learning Trajectories

Obviously, the method presented above cannot be used directly to navigate from every position A to every position B. If the two positions are not in the same room for example, it is impossible for the algorithm to calculate a correct flow field or the target direction, respectively.

To enable a robot to cover greater distances, multiple panoramic images have to be stored. This is realized in a teaching method. A teacher controls the robot along a trajectory. Meanwhile, the robot takes images with its camera continuously, e.g. every half second. When the robot reaches the end of the path, it filters the image sequence to remove all images that it does not need to follow the trajectory on its own. The remaining images represent intermediate targets that the robot has to pass on the trained trajectory. These images can be interpreted as a special kind of autonomously chosen landmarks.

The intermediate targets are selected by splitting the whole image sequence into partial sequences. These partial sequences are formed in a way that ensures that rotation and direction can be determined correctly between the last image and every other image of the sequence. Therefore, it is possible to reach the position represented by the last image from all positions represented by other images of a partial sequence. Thus, only the last images of all partial sequences are necessary to describe the trained trajectory. These images are concatenated to a new sequence that is stored permanently.

After the robot has been trained, it is able to move along the path by driving from one intermediate target to the next. To reach a target, it continuously takes images while it drives. Meanwhile, it determines the driving direction by applying the algorithm presented in section 2 to the current image and the image of the next intermediate target. If an intermediate target is reached, the robot continues with the next. It repeats this process until it has reached the last target of the trajectory.

So far, some details have been left open because they depend on how the robot moves:

- How does the robot decide which images are needed to follow the trained path?
- How does the robot optain drive commands from the information generated by the image processing when it is repeating a path?
- How does the robot recognize that it has reached an intermediate target?

4 Controlling a Wheelchair

A wheelchair is used as an experimental robot platform in Bremen. It has four wheels. The front axle drives the wheelchair while the back axle is used for steering. Therefore, the wheelchair moves like a car driving backwards. It is able to drive straight ahead and on elliptical trajectories. In order to be able to describe the wheelchair's trajectories with the two values "rotation" and "direction", the movement possibilities of the wheelchair are further reduced. The wheelchair is only trained with trajectories that consist of straight lines and segments of continuous curvature, i.e. parts of a circle.

4.1 Selection Criteria

Two criteria to separate the image sequence into partial sequences. First, the sequence is separated into partial image sequences that have been taken while the wheelchair has been driving with a constant driving direction (forward/backward/steering angle). To realize this, the drive commands are stored together with the images. The second criterion decides if the image processing algorithm determines the rotation and direction correctly between the last image of a sequence and all other images in this sequence. As the wheelchair drives only straight ahead and on circular paths, direction and rotation must be related in the following way (Figure 5):

$$\delta = \frac{\omega}{2} \tag{17}$$

There is always an error in the determination of rotation and direction that is especially high if the translational flow is small. Therefore the error is weighted by the distance parameter φ. If this weighted error exceeds an predefined threshold ε, the image processing has not produced trustworthy values. In this case, the image sequence is separated in the middle between the last and the referenced image.

$$\varphi \tan \left| \delta - \frac{\omega}{2} \right| > \varepsilon \tag{18}$$

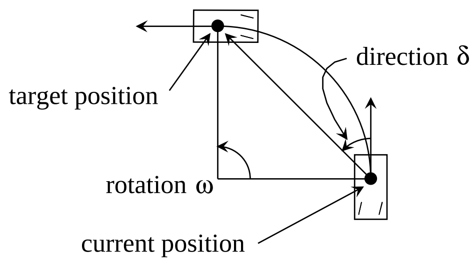

Figure 5: Illustration of the relationship between the wheelchair's rotation and the direction to the target position.

4.2 Drive commands

Equation (17) is used again to control the wheelchair when it is driving along the trained path on its own. The wheelchair chooses its steering angle in a way that compensates for the error between rotation and direction within a certain period of time, e.g. half a second. Thus, it tries to change its rotation to the value that corresponds to the calculated target direction. This strategy does not only work near the trained trajectory, instead it is able to return the wheelchair to the original path. In doing so, the wheelchair returns to the target position from nearly any position if there are no obstacles and the direction/rotation pair is always calculated correctly. Determining the direction is inaccurate if the translational flow—and therefore the distance φ—is small. Therefore, the control strategy is enhanced: if φ gets smaller, the wheelchair directly tries to reduce its rotation ω to zero.

4.3 Stop criterion

The wheelchair has reached an intermediate target as soon as the target position lies behind it, i.e. the direction that the image processing has calculated is in opposition to the wheelchair's driving direction. During the training, the last image is not recorded. Otherwise, the wheelchair would drive too far because it would stop after the position represented by the last image.

5 Results

The method has been implemented on the Bremen wheelchair. Experiments with the wheelchair were performed in a lab environment as well as under presentation conditions with people crowded around the scene. All experiments have been performed with an image resolution of $n = 256$ pixels. As each pixel consists of a red, green and blue component, 768 bytes are needed to store one image. Each matching process has been done in $t_{max} = 4096$ steps. ε has been set to 0.2.

Three different trajectories have been taught to the wheelchair in a room with a size of 7×8 m

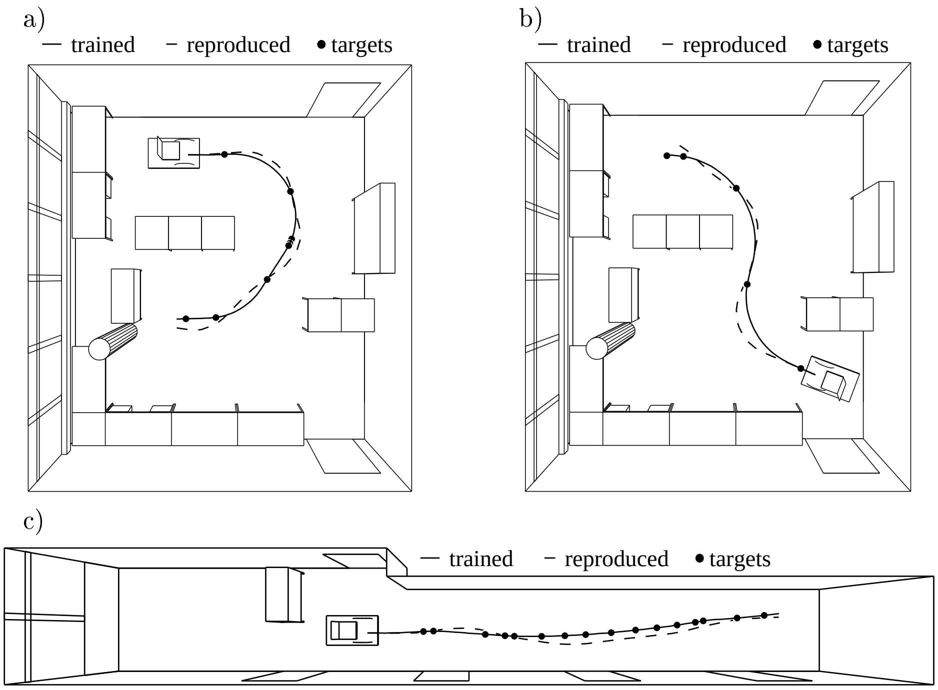

Figure 6: Illustration of the trajectories performed by the wheelchair. The trained trajectories are shown as solid lines. The dots mark the positions where permanently stored images have been taken at. Based on these images, the wheelchair has reproduced the trajectories that are displayed as broken lines.

and in a corridor that is 2 m wide. During the training and the autonomous drive of the wheelchair, its movements have been recorded by the on-board odometers to allow a detailed documentation of the performed experiments. The measured trajectories are visualized in figure 6.

During the autonomous reproduction of the trained trajectories, the wheelchair determines the distance φ, the rotation ω and the direction δ. As shown in figure 5, the nearer the wheelchair gets to the next intermediate target, the smaller the distance φ gets. The direction determination depends on the closeness to intermediate targets, too: the nearer the wheelchair gets to such a target, the bigger the error in the direction δ gets. In contrast, the precision of the determined rotation ω is independent of the closeness to intermediate targets. As the wheelchair ignores the direction if it is close to an intermediate target, a weighted error is calculated, that corresponds to the errors the wheelchair really performs.

In the experiments a) and b) the weighted error is mostly smaller than 0.1, i.e. less than 5°. In the corridor experiment, the error is often bigger. There are two reasons for this: on the one hand the walls are much closer to the wheelchair and this results in a larger optical flow, on the other hand one of the two walls does not provide any visual cues and therefore impedes the matching process. The number of learned images and the average errors are summarized in table 1.

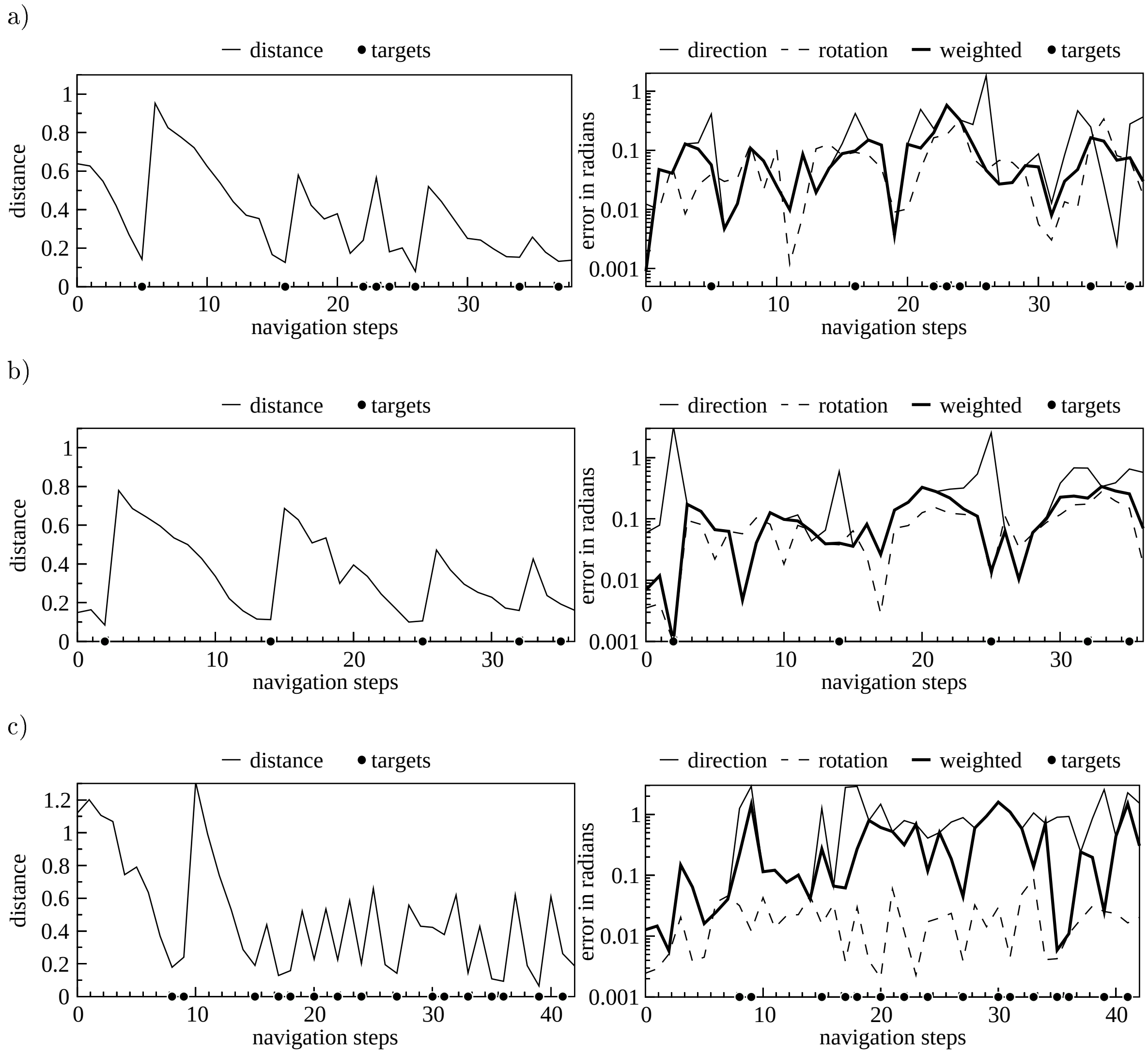

Figure 7: On the left, the determined distance φ during the autonomous drive is presented for each of the three trajectories shown in figure 6. On the right, the errors in the calculation of the rotation, the direction and a weighted combination of both values are displayed.

	learned images	direction error	rotation error	weighted error
trajectory a	8	0.19	0.07	0.08
trajectory b	5	0.36	0.08	0.11
trajectory c	16	0.81	0.02	0.37

Table 1: The number of learned images and the average errors (in radians) in the determination of the direction, the rotation and a weighted combination of both values.

6 Conclusion and Future Work

In this paper, an application of image based homing for the navigation of a wheelchair has been presented. First, the image processing algorithm *Panama* has been introduced, that matches two panoramic images under a strict neighborhood preservation. It extracts the three values *distance*, *direction* and *rotation* which are used to control the wheelchair. Then it has been discussed, how the wheelchair is trained to follow a certain trajectory and how it autonomously decides which images it needs to represent the taught trajectory. Finally, a method has been presented that employs the learned information to enable the wheelchair to repeat the trained trajectories autonomously.

The results show that the presented method is suitable for robot navigation although it is necessary that there are enough visual cues in the surroundings. In the future, the presented algorithm has to be combined with an obstacle avoidance method, as has been done for a simulated holonomic robot [8].

Acknowledgements

The author was supported by the Deutsche Forschungsgemeinschaft and the Freie Hansestadt Bremen through the Graduiertenkolleg “Raumorientierung und Handlungsorganisation autonomer Systeme” Special thanks are due to all members of the student project SAUS, without whose help in constructing the wheelchair this work would not have been possible.

References

[1] Cartwright, B.A. & Collet, T.S. (1983). Landmark Learning in Bees. Journal of Comparative Physiology A 151. pp. 521-543.

[2] Cartwright, B.A. & Collet, T.S. (1987). Landmark Maps for Honeybees. Biological Cybernetics 57. pp. 85-93.

[3] Gibson, J.J. (1950). The Perception of the Visual World. Boston: Houghton Mifflin.

[4] Hong, J., Tan, X., Pinette, B., Weiss, R. & Riseman, E. M. (1991). Image-based Homing. Proc. of the 1991 IEEE International Conference on Robotics and Automation. pp. 620- 625.

[5] Kohonen, T. (1982). Self-organized Formation of Topologically Correct Feature Maps, Biological Cybernetics 43. pp. 59-69.

[6] Lehrer, M. (1993). Spatial Vision in the Honeybee: the Use of Different Cues in Different Tasks. Vision Res. Vol. 34 No. 18. pp. 2363-2385.

[7] Röfer, T. (1995a). Image based homing using a self-organizing feature map. Fogelman-Soulie, F. & Gallinari, P. (Eds.). Proc. Int. Conf. Artificial Neural Networks. EC2 & Cie. Vol. 1. pp. 475-480.

[8] Röfer, T. (1995b). Bildbasierte Navigation mit eindimensionalen 360°-Bildern. Dillmann, R., Rembold, U. & Lüth, T. (Hrsg.). Autonome Mobile Systeme 1995. Springer. pp. 193-202.

[9] Wittmann, T. (1996). Insektennavigation: Modelle und Simulationen. Berichte aus der Biologie. Shaker, Aachen.