Visual Odometry for Tracked Vehicles

Christian Dornhege and Alexander Kleiner
Institut für Informatik
University of Freiburg
79110 Freiburg, Germany
{*dornhege, kleiner*}*@informatik.uni-freiburg.de*

***Abstract*— Localization and mapping on autonomous robots typically requires a good pose estimate, which is hard to acquire if the vehicle is tracked. In this paper we describe a solution to the pose estimation problem by utilizing a consumer-quality camera and an Inertial Measurement Unit (IMU). The basic idea is to continuously track salient features with the KLT feature tracker [12] over multiple images taken by the camera and to extract from the tracked features image vectors resulting from the robot's motion. Each image vector is taken for a voting that best explains the robot's motion. Image vectors vote according to a previously trained *tile coding* classificator that assigns to each possible image vector a translation probability. Our results show that the proposed single camera solution leads to sufficiently accurate pose estimates of the tracked vehicle.**

I. INTRODUCTION

Methods in the field of Simultaneous Localization and Mapping (SLAM) are usually limited to be applied to wheeled robots operating within a 2D or close to 2D environment. This is due to the fact that these methods require a good estimate of the robot's pose, which is typically estimated from shaft encoders mounted on the robot's wheels or from a scanmatching algorithm applied to range measurements of a Laser Range Finder (LRF). However, the latter can only be applied within a 3D environment if the robot is surrounded by continuous walls, which is, for example, the case within buildings [11] and mines [9]. After a disaster, as for example an earth quake, the structure of the environment can be in any shape, and hence the robot is exposed to the full 3D problem, e.g. with strong structural variation in the vertical direction.

Figure 1 depicts typical 3D structures, such as stairs (Figure 1(a)) and random stepfields (Figure 1(b)) as found within the test arenas of the National Institute for Standards and Technology (NIST). On such obstacles, tracks are even more likely to slip during locomotion, and the measurement of their revolutions might not be reliable, and hence pose tracking via shaft encoders nearly impossible. Also pose tracking from LRF data is much harder since 2D LRFs are insufficiently reflecting the environmental structure, e.g. minor variations in the vehicle's roll might lead to major variations in the range measurements obtained from the LRF.

In this paper we describe a solution to the pose estimation problem of a tracked robot by utilizing a consumer-quality camera and an Inertial Measurement Unit (IMU). Our final goal is to utilize this pose estimate for making autonomous and semi-autonomous behavior on complex obstacles, as shown in Figure 1, possible, and to perform SLAM on tracked robots driving on the ground.

(a) (b)

Fig. 1. The *Lurker* robot during the RoboCup Rescue competition in Osaka. (a) climbing stairs and (b) searching for victims in a large random stepfield.

We assume that the robot most likely moves according to its heading and the underlying surface, i.e. it does not translate sidewards, downwards, and upwards. We also assume that the IMU provides sufficiently accurate estimates of the three Euler angles *yaw*, *roll*, and *pitch*. Furthermore, revolutions of the tracks are limited to constant velocities, either *forward*, *backward*, or *none*.

The basic idea is to track salient features continuously with the KLT feature tracker [12] over multiple images taken by the camera and to calculate from the tracked features difference vectors that indicate the robot's motion. Since we estimate rotations by the IMU and thus are only interested in determining translations from the images, vectors that are affected by rotations are filtered out in advance. From the filtered set of vectors the true translation of the robot is determined based on the voting of all single translation vectors. Each vector votes for one of the possible translations according to a previously trained *tile coding* classificator.

The remainder of this paper is structured as follows. In Section II we discuss related work. In Section III we introduce the sensors and experimental platform utilized for the evaluation of the introduced method. In Section IV we describe the tracking and filter algorithm, and in Section V the classification and voting approach. Finally, we provide results from real world experiments in Section VI and conclude in Section VII.

II. Related Work

Corke and his colleagues introduced a solution for visual odometry in the context of a planar rover equipped with an omni-directional vision system [2]. In contrast to our work, which also aims at indoor application, they assume that the robot operates in a large plane, as usually the case on planetary analog environments. Nister and colleagues present a system for visual odometry that works with single and stereo vision, respectively [8]. Their results generally show that the data processing of a stereo system leads to a highly accurate estimate of the robot's pose, which has also been confirmed by other researchers' work [4], [7]. The usage of a stereo system has generally the advantage that the correspondence between features moving within images and the real movement of the robot is directly provided by the vision system. This is particularly important if the robot moves with different velocities. However, results proposed in this paper show that with the simplified kinematics of tracked robots, a single but lightweight camera solution can also lead to sufficiently accurate pose estimates.

III. Experimental platform

(a) (b)

Fig. 2. The hardware utilized for the visual odometry system: (a) an Inertial Measurement Unit (IMU) from *Intersense*, (b) the widely used *Logitech QuickCam Pro 4000*.

In order to determine the robot's orientation, we utilize a 3-DOF IMU of the type *IntertiaCube2* [5] (see Figure 2(a)). One advantage of this sensor is that, due to its internal compass, measurements are drift free, i.e. errors do not accumulate over time. For feature tracking we use a common web cam, known as *Logitech QuickCam Pro 4000* [6] (see Figure 2(b)). This device has a particularly low power consumption, is lightweight ($< 300g$), and can be connected via USB to a wide range of computers.

Experiments were carried out on a tracked robot (shown in Figure 1) and on a wheeled robot (not shown) that was additionally equipped with four shaft encoders and a *Hokuyo URG-X003* LRF. The latter system has been used for generating the ground truth of the robot's pose while operating in 2D environments.

IV. Feature Tracking

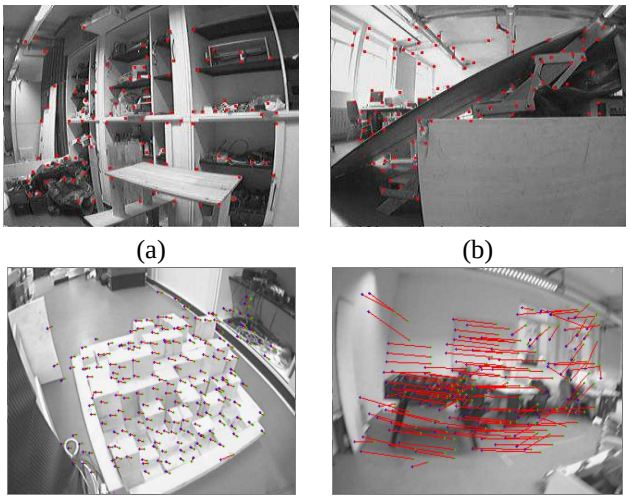
In general, an image sequence can be described by a discrete valued function $I(x, y, t)$, where x, y describe the pixel position and t describes the time. We assume that features detected in an image also appear in the subsequent image, however translated by $\mathbf{d} = (\xi, \eta)^T$:

$$I(x, y, t + \tau) = I(x - \xi, y - \eta, t) \tag{1}$$

Usually, a feature tracker tries to determine this translation by minimizing the squared error ϵ over a tracking window. For brevity we define $I(x, y, t+\tau)$ as $J(\mathbf{x})$ and $I(x-\xi, y-\eta, t)$ as $I(\mathbf{x} - \mathbf{d})$, leading to the following error measure with a weighting function w. [12]

$$\epsilon = \int_{\mathcal{W}} [I(\mathbf{x} - \mathbf{d}) - J(\mathbf{x})]^2 w d\mathbf{x} \tag{2}$$

To facilitate the process of feature tracking, the selection of appropriate features, i.e. features that are easily to distinguish from noise, is necessary. Hence, features that show light-dark changes, e.g. edges, corners, and crossings, are selected with high probability by the KLT feature tracker. In Figure 3, examples of KLT's adaptive feature selection and the tracking over a series of images are shown. For our purpose we use an implementation of the KLT tracker by Stan Birchfeld. [1]



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

Fig. 3. KLT feature tracking: (a), (b) Features shown by red dots are adaptively selected within images. (c), (d) Feature tracking over two subsequent images. The vectors between two corresponding features, shown by red lines, indicate the movement of the camera. (d) The tracking over a series of five images.

A. Tracking over a series of images

Our main goal is to determine the robot's forward or backward movement. However, when traversing obstacles, the robot's motion is not a exclusively forward or backward motion. It is overlaid with noise that originates from slippage of the tracks and shaking of the robot's body due to rough terrain, leading to additional up- or downward movements.

Since the above described effects usually do not accumulate over time, and hence can be reduced, our method generates trackings over multiple frames, rather than to perform single frame trackings only. This is achieved by

saving single trackings between two subsequent images, up to a maximal amount, in a buffer. If trackings of the same feature coexist over more than two images, their corresponding translation vectors $\mathbf{d_i}, \mathbf{d_{i+1}}, ..., \mathbf{d_k}$ are replaced by a single translation vector $\mathbf{d_{ik}}$, consisting of the vector sum of all trackings between $\mathbf{d_i}$ and $\mathbf{d_k}$.

The summed translation vector is more robust compared to single trackings, since it averages out irregular jitter effects. We assign to each tracking a weight $w_{ik} = |k - i|$ in order to reward trackings over multiple frames [1].

B. Filtering of Rotations

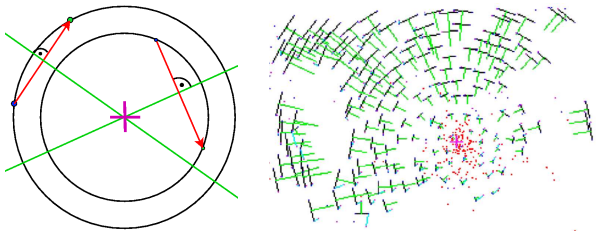
Since we focus on the translation estimation from image sequences, rotations have to be filtered out in advance. However, due to the high latency time of the employed camera system (a web cam connected via USB 1.1), this can only be accurately achieved on the image data directly, which will be described in this section.

Given a feature tracking between two images of the form $(x_i, y_i) \rightarrow (x_j, y_j)$, which includes a rotation around the point r_x, r_y with angle α, one can derive with given rotation matrix $R(.)$, a corresponding rotation free tracking $(x_i, y_i) \rightarrow (x'_j, y'_j)$ after the following equation:

$$(x'_j, y'_j) = (r_x, r_y) + R(-\alpha) \cdot (x_j - r_x, y_j - r_y) \quad (3)$$

Therefore, in order to perform the filtering of rotations, one has to determine the rotation center (r_x, r_y) and rotation angle α.

Rotating points of different radii describe concentric circles around the rotation center. If considering two feature trackings whose features are lying on a circle, one can see that the perpendicular bisectors of the two lines, respectively connecting start- and endpoint of each feature tracking, subtend in the rotation center, as shown by Figure 4(a).



(a) (b)

Fig. 4. (a) The perpendicular bisectors of the side (green) of the tracking vectors (red) subtend at the center of the circle (magenta). (b) Example of the Monte Carlo algorithm: The perpendicular bisectors of the side (green) point to the center of rotation (magenta). Red dots depict the sampled intersection points.

[1]Note that these weights are used during the voting process, which will be described in Section V.

Algorithm 1: Sample up to n possible centers of rotation

Input: A set of feature trackings: T
Output: A set of calculated intersection points: C

```
C = ∅;
for i = 0; i < n; i++ do
    t1 ← selectRandomFeatureTracking(T);
    t2 ← selectRandomFeatureTracking(T);
    s1 ← calculatePerpendicularBisector(t1);
    s2 ← calculatePerpendicularBisector(t2);
    (cut, det) ← calculateIntersectionPoint(s1,
    s2);
    if det < minDeterminant then
        continue ;
    end
    C ← C ∪ cut;
end
```

We exploit this property with a Monte Carlo algorithm for estimating the true center of rotation (see Figure 4(b)). First, up to n possible centers of rotation are sampled from the set of feature trackings T by algorithm 1. Second, all sampled centers of rotation are put into a histogram, whereas the final center is determined by the histogram's maximum.

Furthermore, one has to determine the rotation angle, which can be done by calculating the vector cross product. Given a feature tracking $(x_i, y_i) \rightarrow (x_j, y_j)$ rotated around (r_x, r_y) by α, one can calculate the cross product considering the start- and endpoint of the feature trackings as endpoints of vectors starting at the rotation center. Suppose $v_i = (x_i - r_x, y_i - r_y)^T$ and $v_j = (x_j - r_x, y_j - r_y)^T$ are vectors derived from tracking images I and J, respectively. Then, the angle between these vectors $\alpha = \angle(v_i, v_j)$ can be calculated from the cross product as follows.

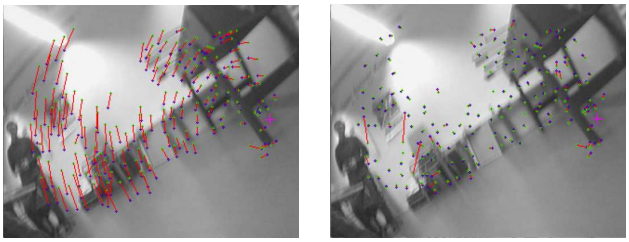
$$v_i \times v_j = ||v_i|| \cdot ||v_j|| \cdot \sin(\alpha) \quad (4)$$

Given the rotation center (r_x, r_y) from the previous estimation, one can determine the true rotation angle α by averaging over rotation angles from all single feature trackings.

Finally, it is necessary to prevent the algorithm to be executed on rotation free sequences. This is achieved by only adding a center of rotation to the histogram, if it is located within the bounding box of the image. Center of rotations that are far from the bounding box are most likely due to quasi-parallel feature translations, which in turn indicate a rotation free movement. If the number of centers of rotation is bellow a threshold λ, the transformation of Equation 3 is not applied. We determined experimentally $\lambda = 10$.

V. Classification

From the set of filtered translation vectors, one can determine the robot's translation. However, the projection from translation vectors of the vision system to the robot's translation depends on the intrinsic parameters of

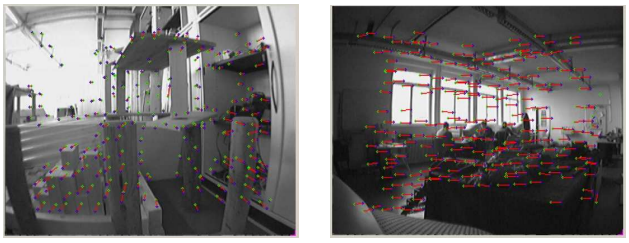


(a) (b)

Fig. 5. Example of the rotation correction while the robot changes the angles of its front flippers. The feature vectors before (a) and after (b) the correction.

the camera, e.g. focal length and lens distortion, and on the extrinsic parameters of the camera, e.g. the translation and rotation relative to the robot's center. This projection can either be determined analytically or by a mapping function. Due to the assumption of a simplified kinematic model, we decided to learn this mapping with a function appproximator described in this section.

Figure 6 depicts the effects from vehicle translation if the camera is mounted towards the driving direction and perpendicular to it. As can clearly be seen, forward and backward translations are easier to detect if the camera is mounted perpendicular, hence we decided, without loss of generality, to mount the camera in this way.



(a) (b)

Fig. 6. The effect of the robot's forward translation regarding two different ways of mounting the camera on the vehicle: (a) towards the driving direction and (b) perpendicular to it. As can be seen by the vector length, the latter arrangements allow an easier detection of translation.

A. Learning of classification probabilities

The learning is based on data collected and automatically labeled during teleoperation runs under mild conditions, i.e. without heavy slippage. During a second phase, the data labeling has been verified on a frame to frame basis. This procedure allows the efficient labeling of thousands of trackings since single images contain several features. Each labeled tracking is described by the class assignment $c \in C$ and the vector $v = (x, y, l, \alpha)^T$, where x, y denotes the origin in the image, l the vector length and α the vector heading. As already mentioned, class assignments are regarding the robot's translation, e.g. $C = \{forward, backward, ...\}$.

Given the labeled data, tile coding [10] function approximation is used for learning the probability distribution

$$P(c \mid x, y, l, \alpha). \tag{5}$$

Tile coding is based on tilings which discretize the input space in each dimension. Shape and granularity of these discretizations can be adjusted according to the task, for example, the discretizations regarding the origin x, y has been chosen coarse due to minor local differences regarding the correlation with the class assignment. Furthermore, tilings are overlaid with a randomized offset in order to facilitate generalization. During learning each tile is updated after

$$w_{i+1} = w_i + \alpha \cdot (p_{i+1} - w_i), \tag{6}$$

where w_i is the weight stored in the tile, $p_i \in \{0, 1\}$ the class probability, and α the learning rate, which is set to $\alpha = \frac{1}{m}$, where m is the number of total update steps, in order to ensure normalized probabilities.

B. Classification by voting

Based on the probability distribution in equation 5, each vector v_i votes individually for a class assignment c_i with respect to its location, length and heading:

$$c_i = \underset{c \in C}{\operatorname{argmax}} P(c \mid x_i, y_i, l_i, \alpha_i) \tag{7}$$

Let $c_i^k = I(c_i = k)$ be the class indicator function, which returns 1 if $c_i = k$ and otherwise 0. Then, the final classification a can be decided based on the maximal sum of weighted individual votes from each vector:

$$a = \underset{k \in C}{\operatorname{argmax}} \sum_{i=1}^{N} c_i^k \cdot w_i \tag{8}$$

Note that w_i increases according to the number of times the underlying feature has successfully been tracked by the feature tracker described in Section IV.

C. Generating odometry using a IMU

In order to determine the distance d traveled between two images I and J, we assume a constant translational velocity v_T of the robot [2]. Given time stamp t_j and t_i of image I and J, respectively, d can be calculated by:

$$d = \begin{cases} v_T \cdot (t_j - t_i) & \text{if class = forward} \\ -v_T \cdot (t_j - t_i) & \text{if class = backward} \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Finally, we calculate from the yaw angle θ of the IMU and the robot's last pose $(x_{old}, y_{old}, \theta_{old})^T$ the new pose of the robot:

$$(x_{new}, y_{new}, \theta_{new})^T = (x_{old} + d \cdot \cos(\theta), y_{old} + d \cdot \sin(\theta), \theta)^T \tag{10}$$

[2]Note that this value could also be automatically be adjusted according to the vehicles current set-velocity.

Run	Trav. dist. [m]	Vis. odo. [cm/m]	Wh. odo. [cm/m]
lab_1 (2D)	91.53	7.82 ± 1.84	6.17 ± 1.54
lab_2 (2D)	73.72	8.25 ± 2.46	7.59 ± 1.94
cellar (2D)	98.40	10.72 ± 4.68	11.77 ± 4.42
ramp (3D)	6.36	13.28 ± 9.2	-
palette (3D)	2.37	22.08 ± 8.87	-

TABLE I

RELATIVE ERROR OF THE VISUAL ODOMETRY AND WHEELED ODOMETRY COMPARED TO GROUND TRUTH DATA (EITHER MANUALLY MEASURED FOR 3D RUNS OR ESTIMATED BY SCANMATCHING FOR 2D RUNS).

VI. EXPERIMENTAL RESULTS

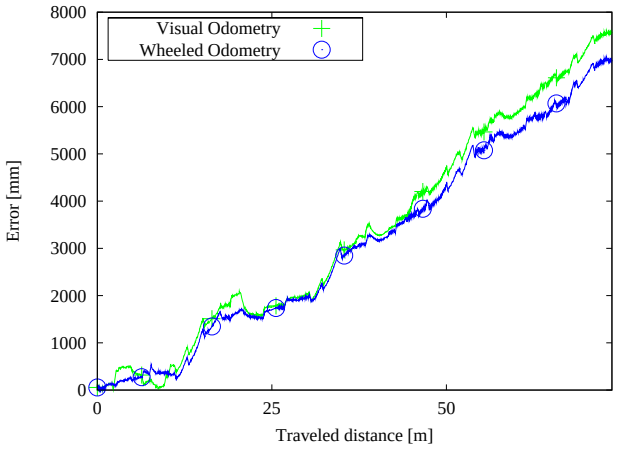
Extensive experiments have been carried out on both the tracked robot *Lurker*, operating on 3D obstacles, and the wheeled robot *Zerg*, operating on flat surfaces (see Section III for a description of these robot platforms). The 2D setting has the advantage that results from the visual odometry system can directly be compared with accurate ground truth data. We determine position ground truth from shaft encoder and IMU measurements, which are processed by dead reckoning, and ranges measured by a Hokuyo URG-X003 Laser Range Finder (LRF), which are processed by a scanmatching method [3]. Ground truth on 3D obstacles was measured manually.

Table I gives an overview on the measured mean and standard deviation of the relative distance error from visual odometry and wheeled odometry on both robot platforms. Since we are mainly interested in tracked vehicles, the *Zerg* kinematics has been modified in order to be similar to that of tracked vehicles, i.e. to allow only a subset of possible velocities, which are in case of the *Lurker* robot: *stop*, *forward*, and *backward*.

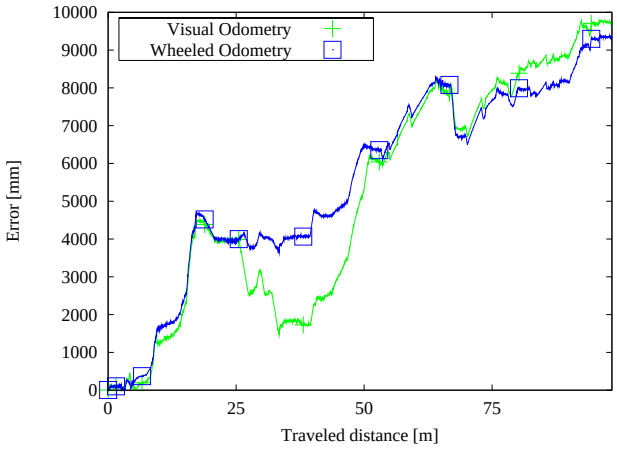
The results clearly show that on the *Zerg* platform the visual odometry reaches an accuracy comparable to the conventional odometry and thus could possibly replace or support it. In the cellar environment, the visual odometry turned out to be even superior, which can be explained by the higher degree of wheel slippage that we noticed within this harsh environment.

Figures 7 (a) and (b) depict the accumulation of the distance error of both the visual odometry and wheeled odometry, within the lab and cellar environment, respectively. The robot's visual odometry-based trajectories from the robotic lab and cellar are shown in Figure 8.

The real advantage of the visual odometry, however, reveals if the robot operates on 3D obstacles. The results in Table I indicate that the introduced method, if applied while operating on 3D obstacles, provides reasonable measurements of the robot's motion. We are confident that these results are sufficient for controlling the execution of behaviors, and probably also allow first steps towards SLAM on 3D obstacles. Figures 9 (a) and (b) depict the accumulation of the distance error during locomotion over 3D obstacles.



(a)

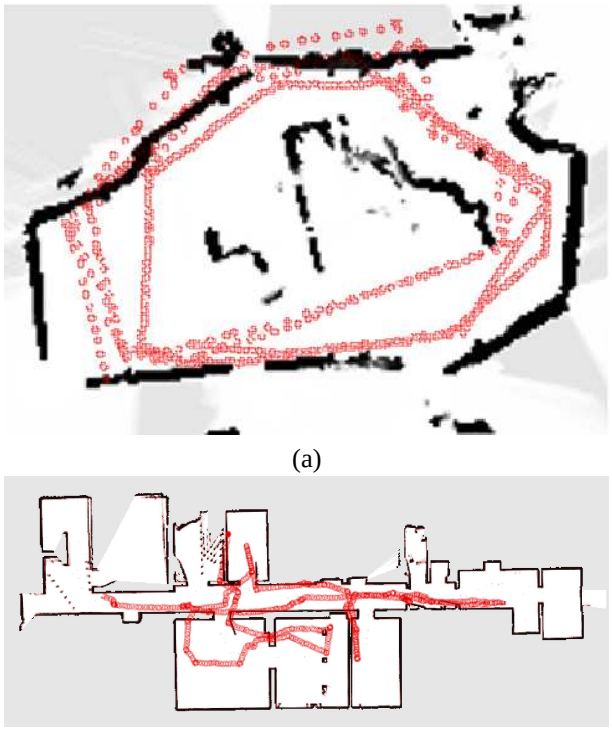


(b)

Fig. 7. The accumulating distance error of the visual odometry method compared to conventional shaft encoder odometry: (a) measured in the robotic lab, a $5m \times 5m$ squared area. (b) measured in a cellar of $15m \times 50m$ on the campus of the University of Freiburg.

The results indicate that, in case of tracked robots, the *tile coding* classification and voting applied to a simple kinematic model lead to sufficiently accurate results. From log files it has been determined that during the *cellar* run 87% (96%), the *ramp* run 81% (93%), and the *palette* run 94% (99%) of the classifications detected the correct motion of the robot, whereas numbers in brackets denote the voting-based improvement.

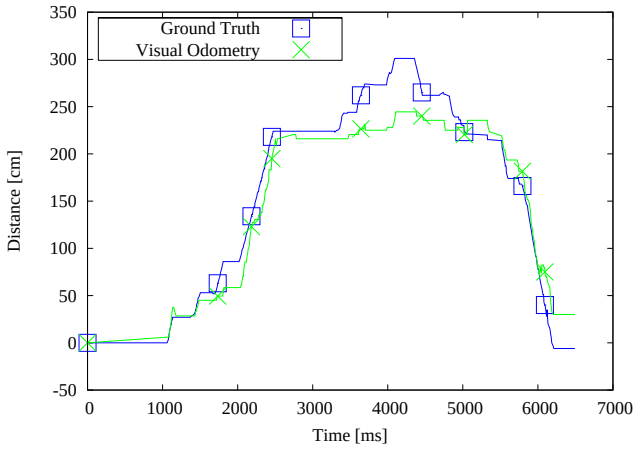
While processing an image resolution of 320×240 on a $IntelPentiumM, 1.20GHz$, we measured for the complete processing without KLT feature tracking an average processing time of $24.08 \pm 0.64ms$. This leads, together with the feature tracker, to a maximal frame rate of $5.34 \pm 1.17Hz$. If processing an image resolution of 160×120, the complete processing without KLT feature tracking needs $8.68 \pm 0.3ms$ and allows a total frame rate of $17.27 \pm 1.81Hz$. Experiments proposed in this paper were carried out with the higher resolution. However, experiments with the lower resolution showed, that the results lead to a comparable accuracy.



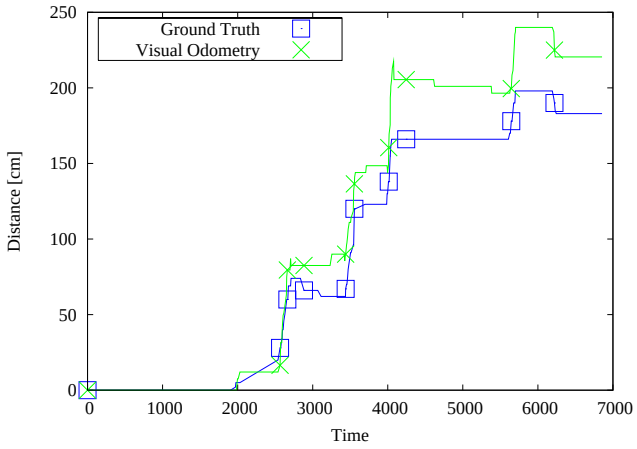
(a)

(b)

Fig. 8. Trajectories, generated by the visual odometry, projected onto the maps of (a) the robotic lab and (b) the cellar of building 52 on the campus of the University of Freiburg, respectively.



(a)



(b)

Fig. 9. Results from driving over (a) a ramp and (b) a wooden palette. The blue curve indicates the manually measured ground truth, and the green curve indicates the distance estimated by visual odometry, respectively.

VII. Conclusion

We proposed an efficient method for visual odometry that can be applied in the context of tracked vehicles. The method has been evaluated with a cheap camera system (below 50 US$) that can be connected to nearly every computer. As our results show, the accuracy of the method, if applied while driving on flat surfaces, is comparable to that one found on wheeled robots, and, if applied while climbing 3D obstacles, leads to reasonable pose estimates, which for example can be taken as a basis for the autonomous control of the vehicle.

In future work we will evaluate the method further for the execution of autonomous behaviors on obstacles and for SLAM and Monte Carlo Localization (MCL) while driving in the plane. Furthermore, we would like to decrease the latency of the system in order to allow even higher frame rates, and also to allow the pose estimation by more than one camera, i.e. to capture also translations into the other two dimensions. We are confident that the performance of the system can easily be increased by reducing the resolution of the camera, and also by decreasing the number of features by selecting them more specific to the environment.

References

[1] Stan Birchfeld. Derivation of kanade-lucas-tomasi tracking equation. `http://www.ces.clemson.edu/~stb/klt/`, 1996.

[2] P.I. Corke, D. Strelow, and S. Singh. Omnidirectional visual odometry for a planetary rover. In *Proceedings of IROS 2004*, 2004.

[3] D. Hähnel. *Mapping with Mobile Robots*. Dissertation, Universität Freiburg, Freiburg, Deutschland, 2005.

[4] D.M. Helmick, Y. Chang, S.I. Roumeliotis, D. Clouse, and L. Matthies. Path following using visual odometry for a mars rover in high-slip environments. In *IEEE Aerospace Conference*, 2004.

[5] Intersense. *Intersense IntertiaCube2*, 2005. `http://intersense.com/products/prec/ic2/InertiaCube2.pdf`.

[6] Logitech. *Logitech QuickCam Pro 4000*, 2006. `http://www.logitech.com/index.cfm/products/details/US/EN,CRID=2204,CONT%ENTID=5042`.

[7] A. Milella and R. Siegwart. Stereo-based ego-motion estimation using pixel tracking and iterative closest point. In *IEEE International Conference on Computer Vision Systems ICVS '06*, pages 21– 21, 2006.

[8] D. Nister, O. Naroditsky, and J. Bergen. Visual odometry. In *Proc. IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR 2004)*, volume 1, pages 652–659, 2004.

[9] A. Nuechter, H. Surmann, K. Lingemann, J. Hertzberg, and S. Thrun. 6d slam with an application in autonomous mine mapping. In *In Proc. of the IEEE International Conference on Robotics and Automation (ICRA)*, 2004.

[10] R.Sutton and G.Barto. *Reinforcement Learning*, chapter Linear Methods. The MIT Press, 1998.

[11] C. Stachniss, G. Grisetti, and W. Burgard. Recovering particle diversity in a rao-blackwellized particle filter for slam after actively closing loops. pages 667–672, 2005.

[12] Carlo Tomasi and Takeo Kanade. Detection and tracking of point features. Technical Report CMU-CS-91-132, Carnegie Mellon University, April 1991.