Team MuCAR-3 at C-ELROB 2009

M. Himmelsbach, F. v. Hundelshausen, T. Lüttel, M. Manz, A. Müller, S. Schneider and H.-J. Wünsche

***Abstract*— This paper briefly describes the hard- and software system of TAS/UniBw's entry for the 2009 C-ELROB robotic trials, participating at the autonomous navigation scenario. At the core of the system is the so-called tentacles approach to reactive robot navigation, that we already demonstrated successfully at the 2007 C-ELROB trials. Since then, the basic tentacles approach has been extended in many ways. We outline an efficient method for accumulating 3D LIDAR data into multi-layered occupancy grids to be used for tentacle evaluation. We then show how to combine obstacle-avoidance and path following behavior within the reactive tentacles approach. Finally, we show how visual information can be used for tentacle evaluation and ERI-Card detection.**

I. Introduction

After 2007, our robot MuCAR-3 will participate at this years C-ELROB 09 autonomous navigation scenario for the second time. The ELROB autonomous navigation scenario is characterized by the hard access path that has to be taken by the vehicle in order to follow the given sparse way points. Two years ago, we first used our tentacle approach to reactive robot navigation to cope with the challenges posed by the rough terrain, with great success. However, the robot had difficulties in choosing the right of possibly many driving options at crossings, and some manual intervention of a safety driver was necessary to lead the vehicle into the right direction.

Since then, the tentacles approach has gone some way and many improvements have been made, some of which will be presented in this paper. Most notably, the tentacle approach now combines local obstacle avoidance with global path following behavior. In addition, obstacles are detected more reliably by accumulation of LIDAR data into a wrapped occupancy grid. Finally, evaluating a tentacle now also considers the visual appearance of the environment.

Before we explain these improvements in more detail, the next sections will first introduce our robot car MuCAR-3 and its hard- and software architecture.

II. MuCAR-3 Hardware Architecture

A. Vehicle

The present autonomous robot car of UniBw is named MuCAR-3 (Munich Cognitive Autonomous Robot Car 3^{rd} Generation), a VW Touareg vehicle equipped with a automatic gearbox and a very powerful generator. The vehicle has full drive-by-wire modifications and appropriate sensors to allow for autonomous driving. An survey of MuCAR-3's hardware is given in Fig. 1.

All authors are with department of Aerospace Engineering, Autonomous Systems Technology (TAS), University of the Bundeswehr Munich, Neubiberg, Germany.

Contact author email: joe.wuensche@unibw.de

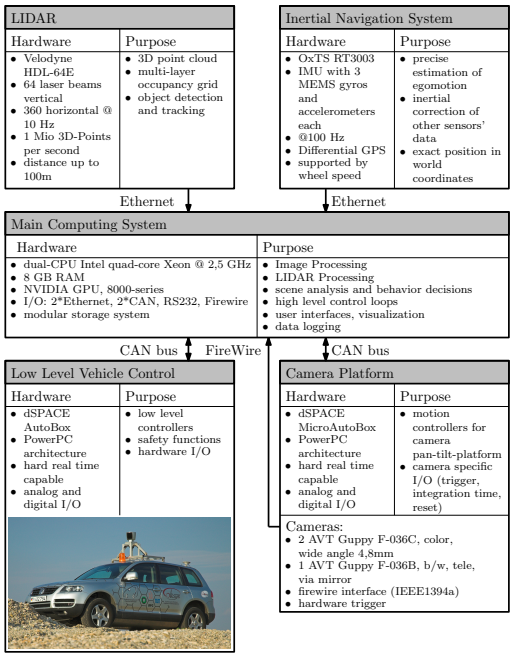


Fig. 1. Overview of MuCAR-3's Hardware Architecture

B. Actuators

As described above, MuCAR-3 has actuators for steering, throttle, brake, the parking brake and also the gearshift of the automatic gearbox. All can be controlled from the low level computers with electrical signals.

The *throttle actuator* utilizes the "e-gas" system of the Touareg, an analog voltage comparable to the throttle pedal position.

For *braking* we use the Touareg's standard electric brake booster which is controlled by a PWM signal. This booster is normally used by the ESP system to boost the brake pressure and is not designed for continuous operation. Hence we have a spindle drive with shaft joint mounted onto the *parking brake* pedal, allowing for safe braking during prolonged standstill even in slopy terrain.

Steering actuator is realized by an electric motor with planetary drive, connected by a chain linkage to the original steering shaft. The maximum torque of the motor is 4.3 Nm. It is supported by the servo-steering, which is modified to give full assistance while autonomous driving).

For *gearshift operation* a spindle drive with shaft joint is mounted to the gearshift lever.

C. Sensors

In addition to the sensor signals which are provided by the stock vehicle's CAN bus such as velocity or steering angles,

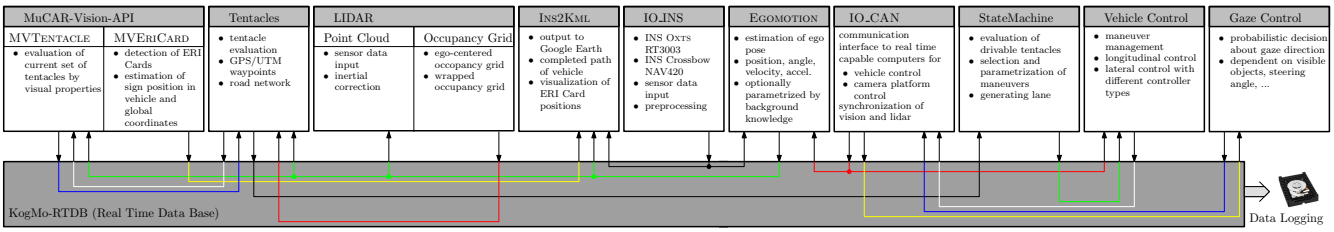


Fig. 2. Software Architecture during of MuCAR-3, showing components used in C-ELROB 09 only. All inter-process communication is done using the RTDB, the different lines emblematize the transferred data objects.

MuCAR-3 is equipped with additional sensors.

Our main sensor in C-ELROB 09 scenario is the *Velodyne LIDAR system* (Light detection and ranging), a state-of-the-art high-resolution laser range scanner mounted at the vehicle's roof (Fig. 3(a)). It generates a 3D point cloud with 10Hz (64 lasers, distance range of 120m, vertical range of 26.5°, 360°, angular resolution 0.1°, 10^6 meas./s). Each point consists of range, bearing and reflectivity measurements.

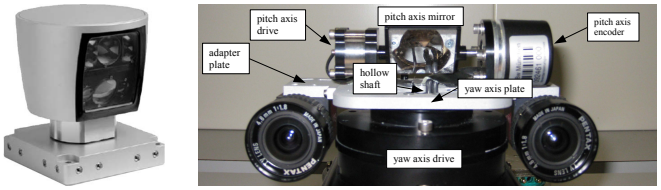
Fig. 3(b) shows the *Camera Platform MarVEye 7* (Multi-focal active / reactive Vehicle Eye), our second main sensor system. It is build up similar to the human eye, with two wide angle cameras (4.8mm lenses), and a high resolution tele camera (50mm lense, 0.01° resolution, "Fovea"). The tele camera image is inertially stabilized by an additional gyro moving the mirror. We use Firewire cameras AVT Guppy, equipped with b/w or rgb sensors of HDRC CMOS type. The image recording of the three cameras is synchronized by a hardware trigger signal, which is either in free-run or synchronized to special events.

We use an *INS* (Inertial Navigation System) from Oxford Technical Solutions, the RT3003. The INS consists of a 6-dof IMU (Inertial Measurement Unit) coupled to a differential GPS. Our dual antenna model allows for determination of heading after power-up without moving the vehicle. The used MEMS gyroscopes have a higher drift than fibre optic or laser ones, but this is compensated by a well tuned Kalman filter.

D. Computing System

Main computing power is provided by a dual-CPU Intel quad-core Xeon L5420 system, with CPUs specially designed for low energy consumption. An additional high performance NVidia GPU (8000 series) allowes for highly parallel data processing and can be user-programmed in CUDA, a C-similar language. The system is equipped with 8 GB RAM, Ethernet, Firewire, CAN and RS-232 interfaces and a modular storage system for data logging. Electrical power is provided by a 450W DC-DC power supply from 12V vehicle power.

Two hard real-time capable dSPACE computers run the low-level controllers. The AutoBox accesses the vehicle I/O (throttle, brake, steering, gearshift actuator, ...) and provides a steering rate control, a velocity controller and a stop-distance controller. Last ones utilize a detailed inverse engine and brake model for setting the desired acceleration. The



(a) (b)

Fig. 3. Velodyne LIDAR (a) and Camera Platform MarVEye 7 (b)

MicroAutoBox runs the platform controllers and accesses the motor controllers (pan and tilt axes). Also hardware trigger generation and integration time analysis of the cameras is located here.

III. MuCAR-3 Software Architecture

We use a Debian Linux operating system with a modified kernel for lower latency; hard real-time performance can't be guaranteed at present. Inter-process communication and data handling in the background is done by KogniMobil Real Time Data Base (RTDB) [1], developed within the TCRC28 "Cognitive Automobiles" [2]. All data is time-stamped. Data logging mechanisms are provided by the RTDB also.

Fig. 2 is an overview of the processes running in C-ELROB 09. In the oncoming sections LIDAR-related work (Sec. IV), Tentacles (Sec. V) and MuCAR-Vision (Sec. VI and Sec. VII) are discussed.

I/O processes communicate with the sensors and the low-level real-time computers.

The *Egomotion Estimation* collects all available data which is provided by the several sensors and uses the time-stamped measurements for improvement of the egomotion estimation. Currently we include measurements from the INS, vehicle's odometry and air suspension. Estimation itself is done by Extended Kalman Filter techniques, the state vector consists of 23 states. Different system models are used for prediction in the substates, e.g. a bicycle model for lateral motion.

The *Vehicle Control* (VC) provides high level vehicle control operations. They can be accessed grouped to maneuvers, for example driving along a lane, driving displacements for lane-change or overtaking, driving forwards and backwards with given velocities or with a velocity dependent distance to another object (ACC). Additionally, VC has security instances which check maneuver parameters considering the vehicle's capabilities.

In C-ELROB 09, maneuvers are selected by a *Statemachine* utilizing a scene analysis.

Gaze Control application decides about where to look at with the camera platform. Normally, the gaze direction depends in a probabilistic way on the objects wanted to track; alternatively, if only drivable terrain is to be checked, gaze direction is selected according to the current steering angle and the Ackerman model.

Ins2Kml provides a data output to KML-Files, which can be visualized with Google Earth. Here the driven tracks and detected objects along the track are recorded.

IV. Occupancy Grid Mapping

We use a $2\frac{1}{2}$-D ego-centered occupancy grid of dimension $200\text{m}\times200\text{m}$, each cell covering a small ground patch of $0.15\text{m}\times0.15\text{m}$. Each cell stores a single value expressing the degree of how occupied that cell is by an obstacle given the current scan only. For calculating the occupancy values, we first inertially correct the LIDAR scan, taking the vehicle's motion into account (exploiting IMU and odometric information). This is done by simultaneously moving the coordinate system of the vehicle while transforming the local LIDAR measurements to global 3D space. After a frame is completed, all points are transformed back into the last local coordinate system of the vehicle, simulating a scan as if all measurements were taken at a single point of time instead of the 0.1s time period of one LIDAR revolution.

Similar to Thrun *et.al.* [3], each cell's occupancy value c_{occ} is then calculated to be the maximum absolute difference in z-coordinates of all points falling into the respective grid cell, $c_{occ} = |c_{z_{max}} - c_{z_{min}}|$.

A. Accumulation of Obstacles

For accumulating sensor information from multiple scans in the occupancy grid, we simply let the vehicle freely move over the grid while keeping the grid at a fixed position. Unfortunately, it turns out that keeping records of extreme z-coordinate measurements over multiple scans introduces many spurious obstacle cells, i.e. ones with large absolute z-coordinate differences, into the grid. The reason is that accumulating raw 3D point cloud data requires very high precision in ego-motion estimation that the inertial navigation system of the robot cannot provide, especially in harsh environments causing violent sensor pitch motions. Resorting to ICP-style algorithms [4] is also no solution due to the additional computational load involved.

A solution to this problem is to accumulate the obstacles detected at a cell instead of the raw point measurements. We thus introduce two additional counters c_o and c_f to each cell for counting scans where an obstacle was detected at the cell and scans where the cell was believed to be free of obstacles, resp. With both counters initialized to zero, let c_{n_t} denote the number of points from scan at time t that got mapped to cell c and let $c_{z_{max_t}}$ and $c_{z_{min_t}}$ denote the z-coordinate extremes of all those c_{n_t} points. The two counters of the cell are then updated as follows, with T being a threshold

Fig. 4. MuCAR-3 freely moving over the occupancy grid and accumulating obstacles. Black indicates low obstacle probability, light yellow indicating high obstacle probability. Note the large ochre area not hit by any beam, thus having obstacle probability of 0.5.

suitable for obstacle detection (e.g. $T = 0.1m$):

$$c_{o_t} = \begin{cases} c_{o_{t-1}} + 1, & c_{n_t} \geq 2 \wedge c_{occ_t} > T \\ c_{o_{t-1}} - 1, & (c_{n_t} \geq 2 \wedge c_{occ_t} \leq T) \vee c_{n_t} < 2 \end{cases} \tag{1}$$

$$c_{f_t} = \begin{cases} c_{f_{t-1}} + 1, & c_{n_t} \geq 2 \wedge c_{occ_t} \leq T \\ c_{f_{t-1}} - 1, & (c_{n_t} \geq 2 \wedge c_{occ_t} > T) \vee c_{n_t} < 2 \end{cases} \tag{2}$$

In other words, whenever we measure an obstacle at a cell, we increment its obstacle counter and decrement the free counter. Contrary, if the measurements at a cell do not indicate an obstacle, we increment the free counter and decrement the obstacle counter. If the measurements do not tell us anything about there being an obstacle at the cell or not, we decrement both counters (note both counters will only be decremented if their values are positive). Apart from the occupancy of a cell based on the current scan only, we can now tell the probability of a cell being occupied at any time t based on numerous successive scans as

$$c_{P_{occ_t}} = \begin{cases} \frac{c_{o_t}}{c_{o_t}+c_{f_t}} & c_{o_t} + c_{f_t} > 0 \\ 0.5 & otherwise \end{cases} \tag{3}$$

Fig. 4 shows a snapshot of the vehicle moving over the occupancy grid while accumulating obstacles.

B. Wrappable Occupancy Grid

Obviously, we also need to handle the case of the vehicle or any sensor measurement leaving the physical boundaries of the grid, shown red in Fig. 4. Two classical solutions, scrolling grids and wrappable maps, are described in [5]. In scrolling grids, the grid is extended at the borders the vehicle crosses and reduced at the opposite borders, involving costly memory operations. Wrappable maps get rid of the memory operations by mapping all 2D space using modulo arithmetics into a finite grid map. This way however, an infinity of points in global coordinates correspond to a single

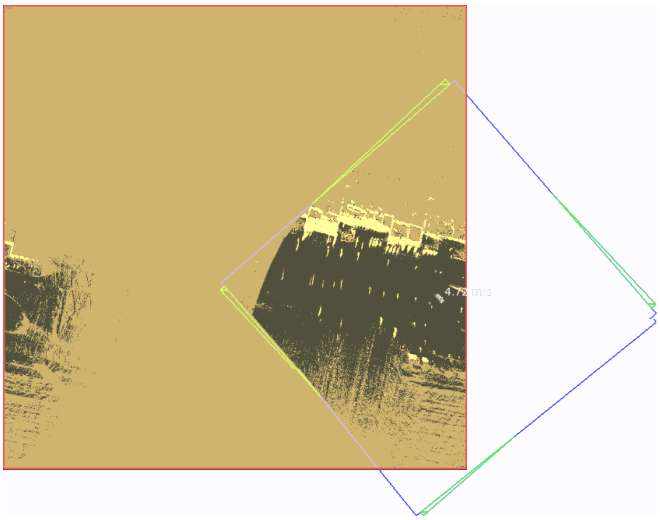


Fig. 5. Disambiguation of data in wrapped occupancy grids by resetting the cells (*green*) that are no longer part of the FOV (*blue*) after updating the grid on vehicle motion.

cell in the map, so remnants of obstacles of arbitrary age may remain in the map indefinitely.

To remove this ambiguity, we define a square region around the vehicle as the current field-of-view (FOV) and assure uniqueness of cell coordinates within this region only. All we need is two assumptions for FOVs: First, no new sensor measurement is allowed to fall outside the current FOV. Second, pairs of different coordinates lying within the FOV must not get mapped to the same grid coordinate. Both assumptions are easily met by choosing the dimensions of the grid and FOV appropriately. If the vehicle moves, then resetting the information of cells part of the last FOV but no longer contained in the new FOV disambiguates new from old cell data.

For refresh rates of 10Hz, it turns out that the number of cells that actually need to be reset after vehicle motion is quite small. However, it remains to show how they can be efficiently found. We do this by representing the last and the current FOV as polygons and solving the set difference operation on these polygons with methods from computational geometry. The resulting polygons are then split into triangles and scan conversion is applied to visit all grid cells that need to be reset. This is illustrated in Fig. 5.

V. Driving With Tentacles

While quite complex approaches to mobile robot navigation exist (eg., solving the SLAM problem [6], methods based on trajectory planning [7]), our research was driven by the search for simplicity: *What is the simplest approach that lets a robot safely drive in an unknown environment?*

A. Basic Idea

Our basic intention underlying the tentacles approach is to let our robot move within an unknown environment similarly to how a beetle would crawl around and uses its antennae to avoid obstacles. Indeed, in our approach the analogue to

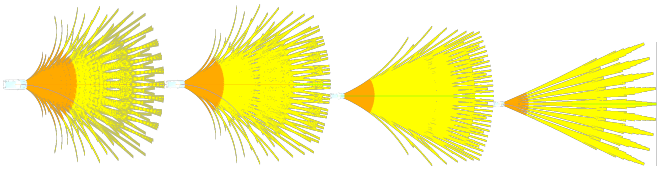


Fig. 6. The tentacles we use at the C-ELROB 09. Every image shows one set of tentacles to be selected according to robot velocity, with velocity and tentacle length increasing from left to right. Every set contains as much as 1000 tentacles, discretizing a subspace of curvatures, lateral offsets and headings.

an insects antennae are target trajectories representing the basic driving options of the vehicle given the current vehicle state. These driving options that we named "tentacles" then probe the occupancy grid to determine drivability and other properties of the corresponding target trajectory. The main loop of the approach is as simple as follows: In each cycle, first select a subset of tentacles according to the vehicle state. Then, by inspecting the occupancy grid, assign properties to each tentacle and select one of the tentacles as the target trajectory for the vehicle to drive.

B. The Details of a Tentacle

As said, a tentacle basically represents a target trajectory defined in the reference frame of the vehicle. These are represented as point samples p_i on arbitrarily shaped line sequences and denoted by $t_{lsq} = \{p_1, \cdots, p_N\}$. Fig. 6 shows the tentacle geometries we use at the C-ELROB 09.

To make evaluation of tentacles based on the occupancy grid as efficient as possible, we make the assumption that both the geometric relation between the vehicle and tentacles, and also the one between the occupancy grid and the vehicle always remain static. If so, the memory offsets to all grid cells influencing the evaluation of a tentacle can be precalculated offline. We make use of this fact by defining, for each tentacle, two sets of grid cells. The first of these sets, t_{narrow}, contains cells within a lateral distance d_{narrow} to the tentacle trajectory t_{lsq}, $t_{narrow} = \{c_1, \cdots, c_M\}$. The second set t_{wide} contains all cells within a larger distance d_{wide}, resp., and additional cell weights according to the corresponding cell's lateral distance to the trajectory.

Finally, to associate each cell index with a longitudinal length on the tentacle trajectory, we longitudinally divide the trajectory into equally spaced bins and record for each cell the index of the bin it was mapped to. Thus, the information collected offline for every cell mapped to a tentacle can be summarized as $c_i = \{c_{i_{memindex}}, c_{i_{weight}}, c_{i_{bin}}, \}$. This is illustrated in Fig. 7 for one tentacle.

When using wrapped occupancy grids as described in Sec. IV, the assumption made is not valid as the relation between the vehicle and the grid changes with the vehicle moving. However, the FOV stays fixed to the vehicle. Thus we can easily build a new vehicle-centered grid by transforming the coordinates of FOV cells defined in vehicle coordinates to the fixed grid coordinate system and look up the current cell data there.

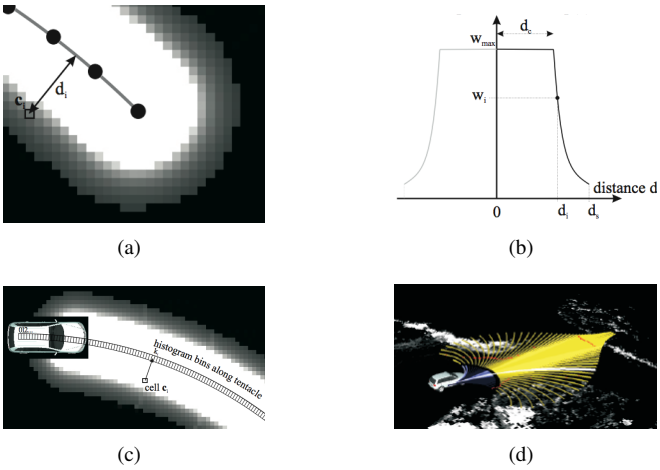


Fig. 7. One tentacle trajectory and mapping of cells based on lateral distance (a), assignments of weights to the mapped cells according to their lateral distance (b) and mapping of cells to longitudinal histogram bins (c). The big picture: one set of tentacles selected according to current vehicle state and the occupancy grid they relate to (d).

C. Tentacle Evaluation and Selection

We will now give a brief description of the properties we evaluate for a tentacle and how a tentacle gets selected for execution based on these properties. The interested reader is referred to [8] for more details on this topic.

1) Drivability: The most important of all tentacle properties for sure is its drivability, $t_{drivable}$. Indeed, all other properties of a tentacle are only evaluated if the tentacle was considered drivable. For evaluating this property, we only have a look at the cells in t_{narrow} of a tentacle, which we now refer to as the tentacle's classification area. For all cells in this set, we increment the corresponding histogram bin if the cell itself is considered an obstacle, i.e. if the probability of it being occupied exceeds 0.5. We then move along the bins up to the crash distance, which is simply the distance the vehicle needs to stop. If any of the bins values exceeds a threshold $T_{obstacle}$, an obstacle is detected and the tentacle is considered undrivable. If no such bin exists within the crash distance, the tentacle is considered drivable.

2) Distance to Obstacles: This property is closely related to the drivability of a tentacle. However, of all drivable tentacles, we would prefer to select the one which can be driven for longer time, keeping the vehicle away from threatening obstacles. This is reflected by the property $t_{clearness}$, which is just the longitudinal length to the bin where an obstacle was detected (or infinity if there is no obstacle along the tentacle).

3) Flatness: Especially in offroad scenarios, a tentacle should be assigned a value reflecting the flatness of the ground it would lead the vehicle to. Unfortunately, this is hard to achieve with the obstacle probabilities reported by the occupancy grid as this involves thresholding z-coordinate differences and flatness might just be "thresholded away". Apart from the obstacle probabilities, the grid thus provides a second layer comprised of the z-coordinate extremes measured at a cell by points of the most recent scan hitting the cell at least twice. Assigning a flatness $t_{flatness}$ to a tentacle is then a simple matter of computing the weighted sum of absolute z-coordinate differences of all cells within t_{wide}, making use of the weights $c_{i_{weight}}$ assigned to cells during tentacle construction.

With all drivable tentacles being assigned their properties, it now remains to select one of the drivable tentacles for execution. To be able to fine-tune the influence of individual properties, we decided to combine them all in a weighted sum, choosing the one tentacle minimizing this sum. As a prerequisite, we require all properties be normalized to uniform range. Again, we refer to [8] for details on how the properties are best normalized. Formally, selecting a tentacle is described by

$$t_{selected} = \underset{\{t|t_{drivable}=1\}}{\operatorname{argmin}} \; a_1(1-t_{clearness})+a_2 t_{flatness} \quad (4)$$

where the a_i denote the weights allowing fine-tuning the resulting behavior.

The benefit of precalculated tentacle structures gets apparent in online execution of the algorithm, where we are able to evaluate as much as 1000 different tentacles within the cycle time of the system of 0.1s. In the following of the paper, we will show how high-level knowledge and properties derived from other sensor modalities can be incorporated via the introduction of new summands into this last equation.

D. Incorporation of High-level Knowledge

From what has been told so far, the robot would most probably still not find its way from A to B without incorporation of some high-level knowledge and reasoning, such as deliberately planning a route. Luckily, there is hardly any robotic competition not providing any route information beforehand. This is true for both the DARPA challenges and the annual ELROB robotic competitions, albeit both represent extremes in terms of way point coverage. We will now show how this kind of information is integrated into the tentacles approach that is otherwise reactive by nature.

Suppose we are given a collection of GPS way points, that we connect in sequence to form a sequence of lines similar to the one describing a tentacle trajectory. Two opportunities immediately arise on how to consider this trajectory for tentacle selection. The first one would be to use the distance of a tentacle evaluated at some look-ahead length to this target trajectory as another tentacle property $t_{targetdistance}$. This would obviously mean selecting a tentacle leading the vehicle right onto track. However, this is not the desired behavior in most of the cases. It happens that buildings or dense vegetation prevent the GPS signals from reaching the sensors on board the vehicle at all. Even worse, GPS measurements might deteriorate due to signals being reflected. In any case, blindly following a given GPS target trajectory is no reasonable choice. This is especially true if two successive way points as given cover far distances and the resulting line could describe anything but the road, country track or street really in-between.

A more reasonable strategy would thus be to go into the same direction as the target trajectory without trying to

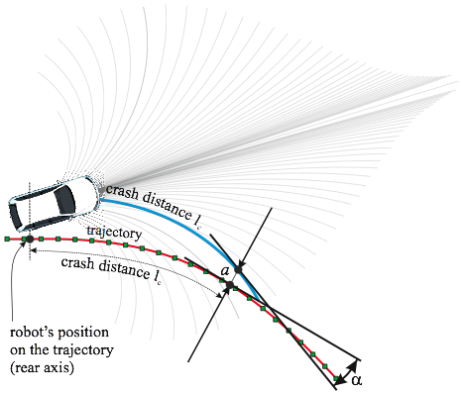


Fig. 8. Evaluating how well a tentacle (*blue*) corresponds to a human-provided target trajectory (*red*, with GPS way point samples shown *green*).

actually approach it. The according property of a tentacle $t_{targetdirection}$ would then be calculated as the angular difference between the tangents at both trajectories at a given look-ahead distance and involves matching the vehicle onto the target trajectory.

In formulating an appropriate tentacle property, we want to be flexible in trading-off tentacles leading to the track and tentacles going into the same direction as the given track. This is again formulated by a weighted sum of both properties as

$$t_{target} = b_1 t_{targetdistance} + b_2 t_{targetdirection} \quad (5)$$

contributing another summand to Eq. 4. In both cases, the look-ahead distance is taken to be the distance the vehicle needs to halt. The evaluation of both tentacle properties - leading to the path and following its direction - is illustrated in Fig. 8.

VI. Evaluating Tentacles By Visual Appearances

Up to now the road following capabilities of the tentacles approach without using dense GPS-Points are limited to scenarios with significant obstacles or non-flat surfaces along what humans would otherwise easily recognize as roads. In order to overcome these limitations we extended the tentacles approach to also incorporate visual features. There is a vast number of works on road detection and following algorithms. Basically, all these works try to identify the road according to visual features special to roads, like road boundaries [9], [10], texture and color [11]. The visual feature we propose goes the other way around: Instead of trying to recognize a road in an image, we try to segment non-drivable pixels in an image.

A. Visual Cue

In numerous video streams of non-urban scenarios, we discovered that one of the the best and efficient ways to segment non-drivable terrain is to utilize the saturation channel of the HSI (hue saturation intensity) color space. The feature we propose makes use of the knowledge we gained - that the road surface may show all kinds of colors but there will be only a vanishing amount of brightly colored pixels on

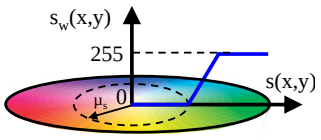


Fig. 9. Weighting function for the saturation channel

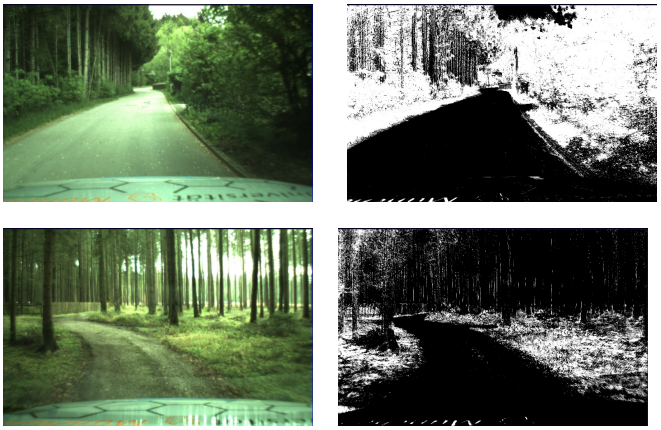


Fig. 10. Comparison between RGB-images on the left side and weighted saturation images on the right side (white encodes no drivable terrain)

a road's surface. In the opposite, this means that almost all brightly colored areas will most likely correspond to non-drivable area. To further enhance this effect we transform the saturation value of each pixel $s(x, y)$ according to the dynamic weighting function Eq. 6,

$$s_w(x,y) = \begin{cases} 0, & s(x,y) \leq \mu_s \\ \frac{255(s(x,y)-\mu_s)}{s_{off}}, & \mu_s < s(x,y) < (\mu_s + s_{off}) \\ 255, & s(x,y) \geq (\mu_s + s_{off}) \end{cases} \quad (6)$$

whereas μ_s represents the temporal low-pass filtered mean saturation value in the lower image parts excluding the engine hood and s_{off} is a parameter to adapt the weighting transition. The equation is pictured in Fig. 9.

Fig. 10 shows the performance of the feature in an easy scenario (top) and a more challenging scenario (bottom).

One additional advantage of our HSI feature is that it does not depend on any earlier measurements of the road in front of the car. For example, the famous histogram back-projection algorithm used in [11] for road detection relies on the vague assumption that if the vehicle was well positioned on the road at one step of time t, then, at time $t+1$, exactly those pixels will belong to the road that show the same color statistics as those believed to belong to the road one time step earlier. Obviously, such approaches can easily fail if the vehicle leaves the road without the algorithm taking notice of it or if the visual appearance of the road suddenly changes.

This is important because even if there are a thousand tentacles no tentacle might actually fit precisely into the road boundaries. This is due to the fact that tentacles only represent a limited set of driving probabilities and will never cover all real-world road geometries. As a consequence, it is most likely that none of the pixel sets associated with any

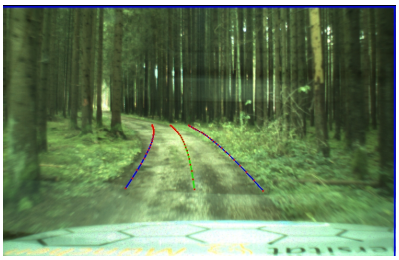


Fig. 11. Image of a single tentacle's skeleton line and support area

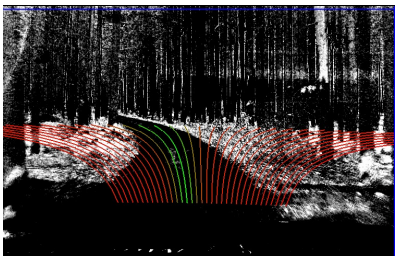


Fig. 12. Rating a sparse tentacle set according to the introduced visual cue (green skeleton line encodes drivable tentacles)

tentacle will fit the true pixel value distribution of the road. Therefore, knowledge about which pixels really belong to the road is even more incomplete compared to approaches based on more fine-grained road geometry hypotheses, such as the 4D approach for road tracking [9]. Hence, no serious statistics can be done.

Summarizing, compared to all approaches incorporating some kind of history, our feature for road segmentation allows the vehicle to recover from any wrong decision previously taken. Of course there may be more non-drivable terrain which is not recognized by the proposed visual cue (see lower part of Fig. 10), but in combination with a geometric road hypotheses and an obstacle avoidance system such as the tentacles approach it becomes a powerful additional feature.

B. Tentacle Evaluation

As described in Subsec. V-B the tentacles are defined in the vehicle coordinate system. In order to do the perspective projection of the tentacles into the camera frame coordinates we use homogeneous transformation matrices which include the vehicle roll and pitch angles measured by an IMU as well as the gaze direction of the camera platform MarV-Eye7 derived from the steering angle of MuCAR-3 via the ackermann steering geometry and a simple pinhole camera model.

First of all the sample points of the tentacles' skeleton lines $t_{lsq} = \{p_1, \cdots, p_N\}$ are transformed into the camera frame coordinates. If 70% from the skeleton points p_i of a single tentacle i lie within the frame boundaries the tentacle is declared as visible otherwise the tentacle is declared as not visible and the visual quality t_{sat} is set to a fix value of 0.8. Meaning, that even if a tentacle is not visible in the camera frame it can still be drivable. For visible tentacles the 3D classification area boundaries are sampled and projected into the camera frame presented in Fig. 11 for a single tentacle. Subsequently we sum all weighted saturation pixel values between the left and the right sampled border lines (blue lines in Fig. 11) symbolised by $s_w(x_t, y_t)$ and normalize the sum by the number of considered pixels N_t.

$$w_{sat} = \frac{\sum_j s_w(x_{tj}, y_{tj})}{N_t} \tag{7}$$

To speed up the summing process we use line-oriented integral images like [11] to evaluate the weighted sum w_{sat} for each tentacles.

To make the visual quality values t_{sat} comparable to other features, they are normalized to the range [0,...,1], where a value of 0 designates a preference for such a tentacle. The value is calculated by the sigmoid-like function

$$t_{sat}(w_{sat}) = \frac{2.0}{1.0 + e^{-c_{sat} \cdot w_{sat}}} - 1 \tag{8}$$

where the constant c_{sat} is calculated by Eq. 9 to yield $t_{sat}(w_{sat0.5}) = 0.5$ at value $w_{sat0.5} = 70$ in our implementation:

$$c_{sat} = \frac{ln1/3}{-w_{sat0.5}} \tag{9}$$

Fig. 12 shows the value t_{sat} of the HSI feature for a sparse tentacle set.

Like in Subsec. V-D we incorporate the visual quality t_{visual} of a tentacle into the tentacle evaluation via a weighted sum

$$t_{visual} = c_1 t_{saturation} \tag{10}$$

contributing another summand to Eq. 4.

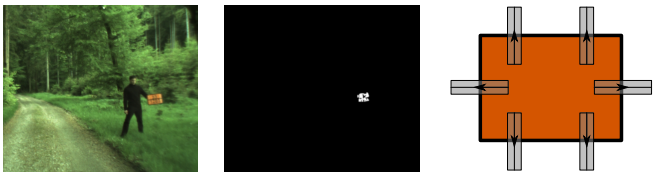
VII. ERI-Card Detection

Part of the challenge in the autonomous navigation scenario is the detection and localization of ERI-Cards (Emergency Response Intervention Cards). Our approach for the detection of these cards is split up into two stages. First a pure visual detection step recognizes the cards in a camera image, then a visually guided tracking using non-linear filter technology allows us to rediscover the card in the following images.

A. Visual Detection

Besides the black letters, the most salient feature of an ERI-Card is its orange color. This suggests to start the detection process with a color-driven preprocessing step. In particular for monochromatic objects *histogram backprojection (HB)* [12] has proven to be a suitable choice for the extraction of colored regions based on color histograms. We use HB together with a manually created hue-channel histogram to generate a binary image showing image regions with orange color, as illustrated in Fig. 13.

Applying the *morphological operation opening* removes the noise in that binary image and – due to the morphologic kernel involved – eliminates all structures that are smaller



(a) (b) (c)

Fig. 13. Conversion of a color-image (a) to a binary image (b), that only shows orange-colored regions, using *histogram backprojection*. (c) shows how the *Cronos-operators* are applied to measure the ERI-Card's edges.

than an ERI-Card as it would appear in appr. 20m of distance. This allows us to address the black letters in the later processing which would, if the ERI-Card was too far away, become too blurred due to the camera resolution.

Assuming that there is at most one ERI-Card visible at a time, we use *continuously adaptive meanshift* [13] to find the center of the most dominant orange region (in case there is one). In order to decide whether or not this region corresponds to an ERI-Card, we have trained a *boosted cascade of haar-like features* [14]. This classifier is evaluated at different scales at the position of the *camshift's* maximum.

If the *boosted classifier* identifies an image region as ERI-Card, we use multiple *Cronos-operators* [15] to take measurements of the ERI-Card's outer edges (see Fig. 13c). The responses of these operators include edges not only from the card's edges but also from the card's inside (e.g. letters) and the outside (e.g. vegetation). Because of this the *RANSAC-algorithm* [16] is used to find a card-model by combining edge-strength and aspect-ratio of the card.

B. Filtering and Tracking

The card-model containing the ERI-Card's position, width and height in the image in combination with the known 3D geometry is used to generate a first hypothesis for the card's real world position relative to the camera. Using the camera's extrinsic parameters this card position can be transferred into coordinates referenced to our vehicle's reference coordinate system. Note that we describe the card's position using polar coordinates to account for the potentially faulty distance compared to the more precise angle measurements.

Based on this first hypothesis we initialize a particle filter which integrates the data of our IMU into its process model. This enables the filter to compensate for our vehicle's movement in rough terrain. Furthermore, in the measurement stage we rely on two distinct measurements. The first measurement evaluates how likely a particle (projected into a camera image) represents a possible ERI-Card with respect to responses of *Cronos-operators*, taken the same way as in the detection stage. The second measurement addresses the amount of orange-colored pixels occupied by a particle's image location. Both of these measurements are combined into one weight for each particle.

Continuous tracking and filtering of an ERI-Card improves its position estimate over time. As soon as an ERI-Card becomes invisible (due to occlusion or because it has left the image) the position estimate is converted into a global GPS location.

VIII. Conclusion

In this paper we described our vehicle MuCAR-3 and its configuration for the C-ELROB 2009. Our system has been enhanced in many ways, such that our tentacle approach has been adapted to the challenging autonomous navigation task. This improvement includes a combination of visual data with accumulated LIDAR data, making GPS information only necessary at crossings.

With the described system we hope to successfully participate at C-ELROB 2009.

IX. Acknowledgements

The authors gratefully acknowledge funding by DFG excellence initiative research cluster *Cognition for Technical Systems* - CoTeSys, see www.cotesys.org, as well as generous funding by the Federal Office of Defense Technology and Procurement (BWB).

References

[1] M. Goebl and G. Färber, "A Real-Time-capable Hard- and Software Architecture for Joint Image and Knowledge Processing in Cognitive Automobiles," in *Proceedings of the 2007 IEEE Intelligent Vehicles Symposium*, Istanbul, Turkey, June 13-15 2007.

[2] "Transregional Collaborative Research Center 28 – Cognitive Automobiles." [Online]. Available: http://www.kognimobil.org

[3] S. Thrun, M. Montemerlo, and A. Aron, "Probabilistic terrain analysis for high-speed desert driving," in *Proceedings of Robotics: Science and Systems*, Philadelphia, USA, August 2006.

[4] P. J. Besl and N. D. McKay, "A method for registration of 3-D shapes," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 14, no. 2, pp. 239–256, 1992.

[5] A. Kelly and A. Stentz, "Rough terrain autonomous mobility - part 2: An active vision, predictive control approach," *Autonomous Robots*, vol. 5, pp. 163–198, 1998.

[6] M. Dissanayake, P. Newman, S. Clark, H. Durrant-Whyte, and M. Csorba, "A solution to the simultaneous localization and map building (slam) problem," *Robotics and Automation, IEEE Transactions on*, vol. 17, no. 3, pp. 229–241, June 2001.

[7] N. Sariff, N.; Buniyamin, "An overview of autonomous mobile robot path planning algorithms," in *4th Student Conference on Research and Development, 2006. SCOReD 2006.*, 27-28 June 2006, pp. 183–188.

[8] F. von Hundelshausen, M. Himmelsbach, A. Müller, and H.-J. Wünsche, "Driving with Tentacles - integral structures of sensing and motion," *International Journal of Field Robotics Research*, 2008.

[9] E. D. Dickmanns and B. D. Mysliwetz, "Recursive 3-d road and relative ego-state recognition," *IEEE Transaction on Pattern Analysis and Machine Intelligence*, vol. 14, pp. 199–213, February 1992.

[10] B. Southall and C. Taylor, "Stochastic road shape estimation," *ICCV*, vol. 1, pp. 205–212, 2001.

[11] C. K. U.Franke, H.Loose, "Lane recognition on country roads," *Intelligent Vehicles Symposium*, pp. 99–104, 2007.

[12] M. J. Swain and D. H. Ballard, "Color indexing," *International Journal of Computer Vision*, vol. 7, 1991.

[13] G. R. Bradski, "Computer vision face tracking for use in a perceptual user interface," *Intel Technology Journal*, 1998.

[14] P. A. Viola and M. J. Jones, "Rapid object detection using a boosted cascade of simple features," in *IEEE Computer Vision and Pattern Recognition*, 2001, pp. I:511–518.

[15] S. Fürst, "Handbuch für Cronos v4.4.2.145," Neubiberg, 2001.

[16] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.