Path Planning, Replanning, and Execution for Autonomous Driving in Urban and Offroad Environments

Roland Philippsen, Sascha Kolski, Kristijan Maček, and Roland Siegwart
Autonomous Systems Lab, Swiss Federal Institute of Technology (ETHZ)
Zürich, Switzerland
roland.philippsen@mavt.ethz.ch

***Abstract*— We present an autonomous driving system that is capable of planning, replanning, and executing paths for driving in urban and offroad environments. For planning, we rely on the E* algorithm which computes a smooth navigation function that takes into account traversibility information extracted from laser scans. The path execution algorithm is centered around a kinodynamic controller which follows the gradient of the navigation function. This work is based on prior experience with the SmartTer vehicle, which we are in the process of updating, and the focus is on integration.**

I. Introduction

Every year, thousands of people are killed in road accidents, with millions more injured. The vast majority of these accidents are due to human error, with roughly 5% caused by vehicle defects [1]. Such staggering findings motivate the use of driver assistant systems and fully automated vehicles to increase driver and passenger safety.

Driver assistant systems can help drivers to identify dangerous vehicle states and traffic scenarios and reduce the risk of accidents. These driver assistant systems are widespread in all categories of vehicles and range from anti-lock brakes to radar based adaptive cruise control. The development of these systems has been accelerated by integrated drive-by-wire components such as electronic gas pedals, brakes, and steering systems.

The development of such components has also hastened the arrival of autonomous passenger vehicles. In 1997, the NavLab vehicles travelled 'no hands' across the United States, requiring only accelerator and brake pedal interaction from the driver [2]. In 2005, 23 autonomous vehicles started a race across the Nevada desert in the DARPA Grand Challenge race [3], with 5 of them finishing the 211.1 Km distance.

Most of these systems depend on environmental structure like driving lanes or dense sets of GPS points. However, in many common driving scenarios neither of these sources of information will be available, for example, when leaving a road and entering a parking lot.

Autonomous navigation in unstructured environments is an active research area in field robotics, and a number of effective approaches have been developed that address this task [4], [5], [6], [7]. A common technique is to maintain a map of the environment and use this to plan safe paths to a desired goal location. As the vehicle traverses the environment, it updates its map and path based on its observations.

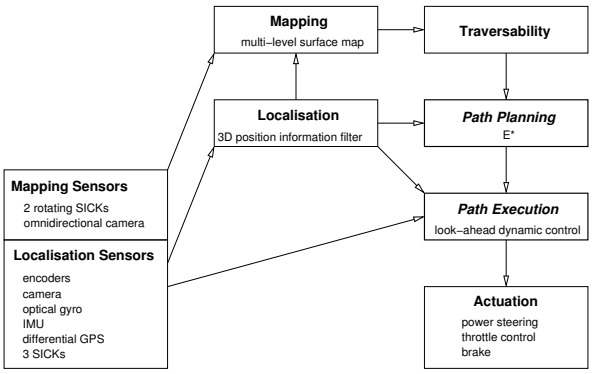


Fig. 1. System Overview of the SmartTer autonomous vehicle. The sensors provide information for localization, mapping, and navigation. During localization of the robot using sensor-fusion based on a 3D information filter, a 3D map is built and traversability data extracted. The planner uses this information to plan a smooth navigation function that trades off collision risk versus detours. Finally, path execution is based on a kinodynamic model of the vehicle and uses a look-ahead of the navigation function in order to choose motion commands that follow the gradient toward the goal while respecting safe driving conditions and dynamic limits.

Our previous work with the *SmarTer* autonomous vehicle [8] demonstrates reliable localization and allows us to assemble 3D maps of an outdoor environment. However, we also identified potential areas of improvement and are currently updating to new sensor technology, optimizing scanner mount points, and integrating another planning algorithm and improving its link with path execution. The medium term goal is a robust experimental platform with stable basic navigation on which further research can rely.

This paper concentrates on the integration of the E* planner and a novel algorithm for path tracking that takes into account kinodynamic constraints. Fig. 1 shows the overall system components. In section II we describe the computation of smooth navigation functions using E*, section III provides details about the interpolation method that allows to achieve this, section IV presents our path execution approach, and in section V we explain how these components have been integrated.

II. Interpolated Path Replanning with E*

Mobile robot path planning approaches can be divided into five classes [9]. Road maps, exact and approximate cell decompositions, potential fields [10], and navigation

functions. E* is a graph-based method for computing smooth navigation functions.

A. The E* Approach

In order to resolve the problem of discrete path choices encountered in traditional graph based methods, the concept underlying E* is formulated in the continuous domain independently of any specific $\mathcal{C}$-space representation. The algorithm computes $\mathcal{C}$-space samples of a crossing-time map defined by the monotonic expansion of a continuous closed surface or a contour from the goal through the environment. By modulating the propagation speed in function of environment characteristics, i.e. the effort of traversing certain regions, the crossing-time map becomes a navigation function that reflects the influence of the environment on the optimal-path of the robot, and it suffices to follow its steepest gradient from any point to drive the robot to the goal. The continuous formulation allows us to interpolate "between" edges, which resolves the issue with movement choices at a fundamental level.

The monotonicity of the E* navigation function stems from the upwind property: The wavefront can be interpreted as the envelope of all possible locations the robot can reach if it starts on the goal and always moves with the maximum allowed speed. This envelope is ever expanding, which allows us to uniquely determine the region of influence of each location by tracing the propagation direction. If a location is subsequently modified, it is thus possible to determine which portions of an existing crossing-time map can be kept intact, and which portions need re-computation. This is the key to efficient replanning by reducing the effort of incorporating new environmental information.

B. The E* Algorithm

The environment in which the robot evolves is represented as an undirected graph G embedded in configuration space $\mathcal{C}$. Note that grids, which are used for the traversability information of the system presented in this papers, fall into this category. Several properties are attached to the nodes $c \in G$, two of which are relevant for a high-level understanding:

- The *value* $v(c) \geq 0$ represents the sample of the continuous crossing-time map at the node c, or the "height" of the navigation function.
- The difficulty or cost of traversing a given configuration is encoded in the traversal *effort* or risk $r(c) \in [0, 1]$. A lower r implies a higher wavefront propagation speed. Note that the risk is actually stored as *meta-information* $m(c)$ to keep E* independent of the interpolation method.

Nodes that await *expansion*, which is the elementary propagation step from a given node to its neighbors, are queued in the *wavefront*. Planning proceeds until the wavefront is empty or the node containing the robot has been expanded. The wavefront is ordered by ascending *key*, which is designed to result in a strictly upwind propagation order. Similarly to D*-Lite [11] and Field-D* [12], a one-step lookahead of the crossing time called rhs-value is used in conjunction with the estimated $v(c)$ to calculate the queue key as $\min(v(c), \mathrm{rhs}(c))$. When the rhs and value of a node equal each other, the node is called (locally) consistent. Wavefront propagation drives the algorithm towards a state where all nodes are locally consistent.

Graph-based planners often use a spanning tree to trace the path from the robot to the goal. E* extends this to a directed *upwind graph* U with unique edges $(c_1, c_2) \in U \Rightarrow (c_2, c_1) \notin U$. It allows to retrieve the upwind set $U(c)$ of nodes that were involved in computing $v(c)$ as well as the downwind set $D(c)$ of nodes that were influenced by $v(c)$. All descendants of a node can be determined if its environmental information changes.

E* is independent of the interpolation method, as long as it respects some constraints and the *kernel* form given in (1). The details of kernels are beyond the scope of this paper, and we only present k_{LSM} which is a good trade-off between smoothness and computational effort. An interpolation kernel k is a function that estimates the crossing-time value of a node, based on the risk (or effort) of traversing it, in conjunction with the values of its neighbors.

$$(u, B) = k(c, Q) \tag{1}$$

where u is the new value for node c, $B \subseteq Q(c)$ is the set of neighbors used in the computation of u, and $Q(c)$ is the propagator of c at the time of expansion (2), which ensures that only valid candidate neighbors are provided to the kernel.

$$Q(c) = \{n \in N(c) \mid \ v(n) < \infty\} \tag{2}$$

where $N(c)$ is the set of neighbors (adjacent nodes) of c. Nodes with infinite value are either obstacles or have not been expanded yet, such as after initialization or following an increase in $r(c)$. Q is ordered by ascending v of the contained nodes: $v(Q_i) \leq v(Q_j)\ \forall\ i < j$. It is possible for Q to be empty ($Q = \{\}$) or contain a single node ($Q = \{Q_1\}$). Handling these cases is part of the required properties for a kernel. B must accurately reflect the information used for the computation of u. This is required such that modifications to the environment model can be consistently propagated to all concerned nodes.

Listing 1 shows pseudo-code for E*. $\{g_i\}$ is the set of goal nodes, W denotes the wavefront queue, and key(c) is the key with which c has been inserted into W. Note that $D(c)$ in line 23 has to be copied, because the call to UpdateVertex(d) changes the upwind graph U. Procedure ComputePropagator(c) applies equation (2), Pop(W) removes and returns the top node from W, and TopKey(W) returns the key of the top node or ∞.

III. Interpolation Approach

In this section we present the interpolation kernel used for the SmartTer planning system. It is denoted k_{LSM} because it is based on the *Level Set Method* [13], which provides a robust grid-based algorithm for calculating the time-dependent position of an evolving curve. It is applicable

Listing 1 Pseudo code of the core procedures in E*.

```
procedure Requeue(c)
01  if v(c) = rhs(c)
02    if c ∈ W then remove c from W
03  else
04    if c ∉ W
05      insert c with key = min(v(c), rhs(c)) into W
06    else if key(c) ≠ min(v(c), rhs(c))
07      remove c from W
08      insert c with key = min(v(c), rhs(c)) into W
procedure UpdateVertex(c)
09  if c ∉ {g_i}
10    Q ← ComputePropagator(c)
11    (rhs(c), B) ← k(c, Q)
12    for all u ∈ U(c) remove (u, c) from U
13    for all b ∈ B
14      if (c, b) ∈ U then remove (c, b) from U
15      add (b, c) to U
16    Requeue(c)
procedure Propagate()
17  c ← Pop(W)
18  if v(c) > rhs(c)
19    v(c) ← rhs(c)
20    for all n ∈ N(c) UpdateVertex(n)
21  else
22    v(c) ← ∞
23    for all d ∈ D(c) UpdateVertex(d)
24    UpdateVertex(c)
procedure main()
25  initialize rhs(c) = v(c) = ∞ ∀c ∈ G
26  initialize goal rhs(g) =true distance ∀g ∈ {g_i}
27  initialize W with {g_i}
28  while (rhs(c_robot) ≠ v(c_robot)) or TopKey(W) < v(c_robot)
29    if W = {} then the goal is unreachable
30    Propagate()
```

when the $\mathcal{C}$ graph G is a regular grid. For E*, where the propagation speed is always positive and depends on position only, we can use the more efficient special case called *Fast Marching* [14]. It is calculated on a N-dimensional grid:

$$\sum_{i=1}^{N} \left(\max(D^{-x_i}T_\iota, 0)^2 + \min(D^{+x_i}T_\iota, 0)^2\right) = \frac{1}{F_\iota^2} \quad (3)$$

where T_ι is the crossing time of the wavefront for a grid cell, with ι an N-dimensional index, F_ι is the propagation speed, and $D^{\pm x_i}$ is the finite difference that replaces the continuous ∇, with x_i an axis of the grid.

The two-dimensional implementation of k_{LSM} is a first-order upwind interpolation scheme for (3). The finite difference expressions are of the form $D^{-x_1}T_{ij} = (T_{ij} - T_{i-1,j})/h$, where we substituted $\iota = (i, j)$ and h is the grid resolution (distance between two nodes).

Developing $D^{\pm x_{i,j}}T_{ij}$ leads to a quadratic equation with coefficients that take values based on a switch on the sign and magnitude of the finite difference operators.

It is possible to determine the terms that will yield the optimal solution beforehand. Interpolation implies using up to two neighbors, which need to lie on different axes. Without loss of generality, it can be assumed that the two neighbors leading to the best interpolation are **A** and **C**, and that $T_{\text{A}} \leq T_{\text{C}}$. The final expression for the crossing-time interpolation is (4), see [15] for a detailed development.

$$\begin{aligned} T &= \begin{cases} T_{\text{A}} + h/F & \Leftarrow\ T_{\text{C}} - T_{\text{A}} \geq h/F \\ \frac{1}{2}\left(-\beta + \sqrt{\beta^2 - 4\gamma}\right) & \text{otherwise} \end{cases} \\ \beta &= -\left(T_{\text{A}} + T_{\text{C}}\right) \\ \gamma &= \tfrac{1}{2}\left(T_{\text{A}}^2 + T_{\text{C}}^2 - h^2/F^2\right) \end{aligned} \quad (4)$$

where T_{A} and T_{C} are the values of the best neighbors, h is the distance between two neighbors, and F is the propagation speed at (i, j). Recall that $T_{\text{A}} \leq T_{\text{C}}$ and note that $F \to 0 \Rightarrow T \to \infty$. The final equations for k_{LSM} are given in (5).

$$\begin{aligned} (u, B) &= k_{\text{LSM}}(c, Q) \\ u &= T \\ B &= \begin{cases} \{Q_1\} & \Leftarrow\ T_{\text{C}} - T_{\text{A}} \geq h/F \\ \{Q_1, Q_2\} & \text{otherwise} \end{cases} \\ T_{\text{A}} &= v(Q_1) \\ T_{\text{C}} &= \begin{cases} \infty & \Leftarrow\ Q = \{Q_1\} \\ v(Q_2) & \text{otherwise} \end{cases} \\ F &= m(c) \end{aligned} \quad (5)$$

Nodes which do not have neighbors of type **A** and **C**, such as neighbors of obstacles or cells on the grid border, use the fallback solution $T_{\text{A}} + h/F$.

IV. Path following - execution

Having computed a navigation function with E*, the question arises of how to control the vehicle's motion such that it tracks the potential's gradient until the goal is reached. The novelty presented here is combining arc based low-level trajectory generation with global path following. The reference point feedback loop to vehicle controls, based on local path configuration, ensures smooth and stable path following. It avoids oscillations often caused by replanning or the holonomicity of gradient descent. Furthermore, the low-level arcs which are kinodynamically feasible are tested to ensures secure vehicle commands in the presence of obstacles, based on traversability information. The algorithm is called "Traversability-Anchored Dynamic Path Following".

A. Reference point dynamics

Following the negative gradient of the global navigation function from the current vehicle position $\{x, y\}$ to the goal $\{g_x, g_y\}$, the reference path is a set of points $\left\{c_{r,d} = c_k; k = 1, \ldots, N_g, c_1 = \{x, y\}, c_{N_g} = \{g_x, g_y\}\right\}$. This path is further smoothed to give a reference path $\mathcal{C}_r(s)$ which is described with the curvilinear parameter s and curve gradient $\|\mathcal{C}_r\,'(s)\| = \sqrt{p'^2(s) + q'^2(s)} \neq 0, \forall s \in [0, s_f]$.

The kinematic level control objective is to find a longitudinal velocity v_l and steering angle ϕ of the vehicle to follow the reference path $\mathcal{C}_r$ given by the global planner. In particular, a desired reference point is defined on $\mathcal{C}_r(s_d)$ as:

$$\begin{aligned} x_d &= p(s_d) \\ y_d &= q(s_d), (0 \leq s_d \leq s_f)\,. \end{aligned} \quad (6)$$

In contrast to the commonly used path following approach using orthogonal projection of the vehicle position to the curve $\mathcal{C}_r$ with a fixed lookahead, here the position of the reference point is defined by dynamics of the curvilinear parameter $\dot{s}_d$ which includes feedback on the vehicle pose and velocity as in [16]:

$$\dot{s}_d = \frac{c_o e^{-\alpha\rho} v_{max,l}}{\sqrt{p'^2(s) + q'^2(s)}} \left[1 - \frac{2}{\pi} \arctan(\zeta\chi(\rho))\right] , \quad (7)$$

where $\rho = \sqrt{\Delta x^2 + \Delta y^2}$ is the current lookahead distance with $\Delta x = x_d - x$, $\Delta y = y_d - y$ and $v_{max,l}$ the vehicle's max longitudinal velocity. The factor $c_o = e^{\alpha d_{\rho_o}}$ normalizes the curvilinear speed, where the ρ_o represents the nominal lookahead distance and α is the convergence rate towards it. Essentially, in nominal condition when the reference path is straight and no obstacles are near, as described further in Sec. IV-C, both vehicle and reference point travel at $v_{l,max}$ and the lookahead distance is the nominal ρ_o.

However, when significant curvature changes are encountered, the vehicle must slow down to follow the path at a safe error margin. In order to quantitatively describe the necessary vehicle steering activity along $\mathcal{C}_r$ a curvature effort term was introduced in [16]:

$$\chi(\rho) = \frac{\sum_{i=1}^{N_\rho} \|\Delta\kappa_i\| \Delta s_i}{\rho}, \ \forall \rho > 0 \, . \quad (8)$$

The smooth reference path is described as a collection of dense waypoints $\left\{(x_d(s), y_d(s)) : s = (s_1, s_2 \ldots, s_{N_\rho})\right\}$ and N_ρ denotes the number of points up to the reference point ρ. Here, $\Delta\kappa_i = \kappa_i - \kappa_{i-1}$ is the incremental change in curvature and $\Delta s_i = s_i - s_{i-1}$ is the incremental change in parameter *s* that may be unequally spaced. Eq. (8) takes into account all curvature (thus also direction) changes along the reference path up to the current reference point. If the curvature effort is $\chi(\rho) = 0$ then the vehicle is moving along a path with no curvature change, i.e. a straight or circular path segment, and it may proceed at maximum speed provided other constraints are satisfied. The factor ζ essentially determines how fast the reference point will slow down in the presence of obstacles, thus increasing the safety margin for larger ζ but also rendering the vehicle drive less speed-efficient. The $\arctan$ is a saturation funtion for large curvature efforts where both reference point and the vehicle slow down significantly.

B. Kinodynamically feasible vehicle trajectories

The description of the car motion is based on the Ackermann kinematic model:

$$\dot{x} = \cos\theta \, v_l \, , \qquad \dot{y} = \sin\theta v_l \, , \qquad \dot{\theta} = \frac{v_l}{L} \tan\phi \, , \quad (9)$$

with $\{x, y, \theta\}$ being the robot pose and $\{v_l, \phi\}$ the longitudinal velocity and steering angle as control inputs and L the axes distance of the front and rear wheels.

According to the Ackermann kinematics, the vehicle follows a circular path for a given kinematic level control input $\{v_l, \phi\}$. Therefore a set of arc vehicle trajectories can be defined as:

$$\mathcal{A} = \{a_{i,j} = \{x_{i,j}, y_{i,j}\} ; i = 1 \ldots N_v, j = 1 \ldots N_\kappa\} \, , \quad (10)$$

where N_{v_l} denotes the number of arc sets due to longitudinal velocity $v_{l,i}$ discretization and N_c the number of arcs due to curvature κ_j discretization, which corresponds to a steering angle ϕ_j.

At each control cycle a trajectory $a_{i,j}$ is chosen, corresponding to a control input $(v_{l,i}, \phi_j)$ that is feasible with respect to the environment constraints, e.g. obstacles and goal directedness, but also according to the limitations on the vehicle motion itself. The kinematic limitations on the vehicle motion are the maximum longitudinal velocity $v_{l,max}$, the minimum allowed[1] vehicle speed $v_{l,min}$, and the maximum steering angle ϕ_{max}. The dynamic limitations are the maximum longitudinal acceleration $\dot{v}_{l,max}$ and the maximum steering rate $\dot{\phi}_{max}$. The aim here is to define a minimum set of arcs neccessary to take into account the dynamic limitations of the vehicle at each time instant. From Eq. 9 it follows that:

$$\ddot{\theta} = \frac{v_l}{L \cos^2\phi} \dot{\phi} + \frac{\dot{v}_l}{L} \tan\phi \, . \quad (11)$$

Typically, the low-level steering control loop (e.g. power steering) is faster than the longitudinal velocity control loop, thus for small time increments the longitudinal velocity can be considered constant with respect to the angular rate of the vehicle. Therefore, the second term can be neglected in Eq.11. Given that the vehicle is currently on a trajectory defined by $\{v_l, \phi\}$ and the steering rate is at its maximum $\dot{\phi} = \dot{\phi}_{max}$, the curvature change within the kinematic level control sample time-step T_s can be expressed as:

$$\Delta\kappa\left(\phi\right) = \frac{v_l}{L \cos^2\phi} \dot{\phi}_{max} T_s \, . \quad (12)$$

Taking the smallest curvature change within a control cycle T_s such that switching between neighboring arcs is feasbile according to steering limitations leads to the a-priori number of arcs due to the curvature discretization:

$$N_\kappa = 2 \cdot \lceil \frac{\tan\phi_{max}}{L \, \Delta\kappa\left(\phi_{max}\right)} \rceil + 1 \, , \quad (13)$$

taking into account the central $\kappa\left(\phi = 0\right) = 0$ separately. Moreover, assuming the vehicle drives at its full longitudinal acceleration capability, the number of arc sets due to longitudial velocity discretization is obtained as:

$$N_{v_l} = \lceil \frac{v_{l,max} - v_{l,min}}{\dot{v}_{l,max} T_s} \rceil \, . \quad (14)$$

This minimum a-priori number of arcs can be further refined for smoother kinematic control if computational resources for testing each arc are available on-line. Although the total number of arcs can be large, at each kinematic control cycle only a small subset is dynamically feasible. This idea is similar to the Dynamic Window approach [17] which was first derived for a differential drive robot.

[1]unless the goal is reached or during an emergency brake

C. Configuration space feasible vehicle trajectories

In order to chose an optimal vehicle trajectory-arc at each cycle T_s, each arc that is dynamically feasible is checked for potential collision with obstacles. The global navigation function of Sec. II provides the configuration space obstacle regions. If a prohibited node is encountered along an arc $a_{i,j}$ that is less then time $T_{b,i} = \frac{v_{l,i}}{\dot{v}_{l,max}}$ away from the starting vehicle position, the arc is banned. For a prediction horizion T_h of vehicle motion along an arc $a_{i,j}$, the traversability cost $\Gamma_t^{(i,j)}$, cost to goal $\Gamma_g^{(i,j)}$ and orientation alignement to the current reference point on the path $\Gamma_o^{(i,j)}$ are given as:

$$\Gamma_t^{(i,j)} = \sum_{\iota=1}^{N_\iota} r\left(a_{i,j}^{(\iota)}\right), \tag{15}$$

$$\Gamma_g^{(i,j)} = v\left(a_{i,j}^{(N_\iota)}\right), \tag{16}$$

$$\Gamma_o^{(i,j)} = 1 - \frac{\|\Delta\theta_{i,j}(T_s)\|}{\pi}, \tag{17}$$

where $a_{i,j}^{\iota}$ corresponds to a sampled point on the arc $a_{i,j}$, $r(\cdot)$ the risk of traversing the closest node on the graph G, $v(\cdot)$ the value of the navigation function at the end of the arc with N_ι being the number of discrete points. The angle difference between the reference point position and vehicle center to the actual vehicle orientation is $\Delta\theta_{i,j}(T_s) = \theta_d(T_s) - \theta_{i,j}(T_s)$, for the simulated reference point and vehicle position at time T_s. This forward simulation ensures stable transition between arcs with respect to orientation change.

According to the feedback scheme of the Eq. 7, the longitudinal velocity taken at each cycle is the one that minimizes the difference between the feasible longitudinal velocities from the available arc set and the reference velocity proportional to the current distance between the vehicle and the reference point ρ:

$$v_l^\star = \underset{v_{l,i}}{\operatorname{argmin}} \left\{ \|v_{l,i} - \mu\rho\cos\Delta\theta_{i,j}(T_s)\| \right\}, \tag{18}$$

where $\mu = \frac{v_{max}}{\rho_o}$. This ensures that the kinodynamic constraints are satisfied with established feedback loop from the reference point.

The optimal steering commands $\phi^\star$ chosen at each control cycle minimizes the total weighted sum cost:

$$\phi^\star = \underset{\phi_{i^\star,j}}{\operatorname{argmin}} \left\{ \Gamma^{(i^\star,j)} = \gamma_t\Gamma_t^{(i^\star,j)} + \gamma_g\Gamma_g^{(i^\star,j)} + \gamma\Gamma_o^{(i^\star,j)} \right\}. \tag{19}$$

V. Integration

Our vehicle is a Smart fortwo passenger car that has been modified for autonomous operation. SmartTer localization is based on an information filter that fuses data from encoders, an optical gyro for precise measurement of the heading angle, an IMU for 6DOF motion estimation, and a differential GPS. Three fixed laser scanners and two rotating laser scanners acquire information for mapping, yielding traversability information and a 3D representation of the environment [8].

Fig. 2. Our autonomous Smart car platform. There are three fixed laser range finders mounted on the front of the vehicle and on the sides of the roof, and two spinning laser range finders mounted together on the center of the roof.

To test our current integration efforts at an early stage, the results presented below are based on runs simulated in an environment created from logged real-world data. This permits relatively fast and lightweight experimentation without the risk of hardware failure and removes the complexity of localization and mapping which would distract from developing planning and execution. The simulation interacts with the vehicle code through the same APIs as on the real vehicle, so the effort involved in porting to the SmartTer is reduced.

To generate this world representation we play back the log files from test runs performed on our vehicle into a high resolution traversabiltiy map which the simulator uses to emulate the laser scanner on the vehicle. Based on this artificial sensor data, cells that contain obstacles are extended to configuration space to reflect the actual size of the vehicle. As the Smart is not of circular shape and does not have a differential drive, here a worst case estimation is done taking into account the car's size and steering system. As the Smart is probably the most circle-like car in the market the risk of potential lock because of too conservative $\mathcal{C}$-space extension stays low.

While the car moves, this $\mathcal{C}$-space extended map is updated from the laser readings and fed to the E* motion planner. E* calculates an optimal path on this map and extracts a local goal from that path that is handed to the local planner described in section IV that controls the vehicle toward this local goal.

VI. Results

The first simulation illustrates the replanning behavior with the help of a simplified example shown in fig. 3. The robot discovers the traversability for each cell the front laser scanner sweeps (unkown cells are supposed to be free). As it progresses toward the goal, it adapts its plan as it incorporates this new knowledge. In the top right zoom of fig. 3 it is about to discover that the traversability of the region it enters is very low. It then attempts to go around below the wall to the right, as it has not yet discovered that there is no passage

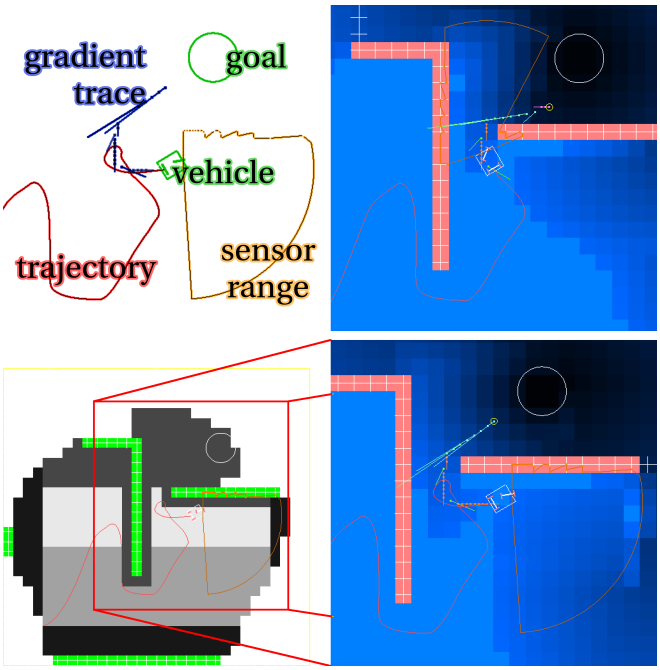
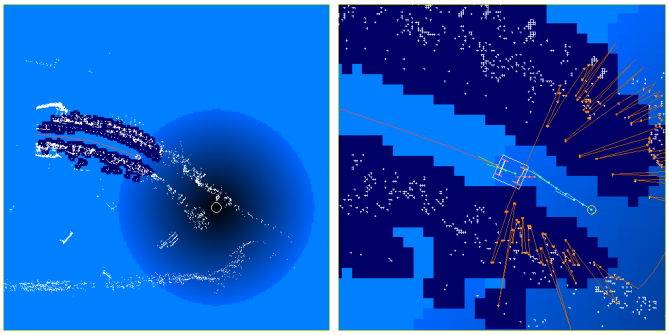


Fig. 3. This exploration example illustrates replanning. In the lower left you can see which parts of the environment have been discovered by the robot (darker regions signify lower traversability, obstacles are green). On the right you see two zooms to the robot position, with the navigation function in blue (darker means closer to the goal) and the obstacles in pink.



(a) global view (b) zoom

Fig. 4. These reruns based on real data illustrate how the system would perform on SmartTer. Note how obstacle are grown by the radius of the vehicle.

there. Once the robot discovers the even less traversable region to the right, the plan is adapted to turn back once more and go through the previously more expensive passage which now has a lower accumulated traversability cost (see the loop-back trajectory in the lower right).

In fig. 4, real data acquired from a run with SmartTer on our campus has been fed into the simulator in order to study real-world cases. Here, instead of directly discovering traversibility as in the simplified example above, we use simulated laser scans to feed the $\mathcal{C}$-space traversibility map. Changes are accumulated until a threshold is reached and then the navigation function is updated by propagating out these changes until the wavefront reaches the robot.

VII. Conclusion and Outlook

We have presented the autonomous driving system we are developing on the SmartTer vehicle. Traversibility-map based planning and kinodynamic path tracking have been described in detail. With E* we directly receive smooth navigation functions, which avoids post-processing – the path can be directly fed into our controller, which has proven its performance in outdoor tests on the SmartTer in urban and offroad environments.

As mentioned in the introduction, the system presented here is an improvement over a previous version, and we focussed on the integration of E* and high-performance dynamical path tracking. It is still work in progress and the results we have presented are based on simulations to show that the algorithms work properly together. Next will come the integration on the updated vehicle, with the goal of testing autonomous driving and 3D mapping approaches.

References

[1] M. Shell, "Final report of the european esafety working group on road safety, online available," 2003. [Online]. Available: http://europa.eu.int/informationsociety/activities/esafety/indexen.htm
[2] C. Thorpe, T. Jochem, and D. Pomerleau, "The 1997 automated highway demonstration," in *1997 International Symposium on Robotics Research*, 1997.
[3] "Darpa grand challenge race website." [Online]. Available: http://www.darpa.mil/grandchallenge
[4] A. Kelly, "An intelligent predictive control approach to the high speed cross country autonomous navigation problem," Ph.D. dissertation, Carnegie Mellon University, 1995.
[5] A. Stentz and M. Hebert, "A complete navigation system for goal acquisition in unknown environments," *Autonomous Robots*, vol. 2, no. 2, pp. 127–145, 1995.
[6] O. Brock and O. Khatib, "High-speed navigation using the global dynamic window approach," in *IEEE International Conference on Robotics and Automation (ICRA)*, 1999.
[7] S. Singh, R. Simmons, T. Smith, A. Stentz, V. Verma, A. Yahja, and K. Schwehr, "Recent progress in local and global traversability for planetary rovers," in *IEEE International Conference on Robotics and Automation (ICRA)*, 2000.
[8] P. Lamon, S. Kolski, and R. Siegwart, "The SmartTer - a vehicle for fully autonomous navigation and mapping in outdoor environments," in *Proceedings of the CLAWAR*, Brussels, Belgium, 2006.
[9] J.-C. Latombe, *Robot motion planning*. Dordrecht, Netherlands: Kluwer Academic Publishers, 1991.
[10] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," *International Journal of Robotics Research*, vol. 5, no. 1, 1986.
[11] S. Koenig and M. Likhachev, "D* lite," in *Proceedings of the National Conference on Artificial Intelligence (AAAI)*, 2002.
[12] D. Ferguson and A. Stentz, "Using interpolation to improve path planning: The Field D* algorithm," *Journal of Field Robotics*, vol. 23, no. 2, pp. 79–101, Feb. 2006.
[13] J. Sethian, *Level Set Methods – Evolving interfaces in geometry, fluid mechanics, computer vision, and materials science*. Cambridge University Press, 1996.
[14] R. Kimmel and J. Sethian, "Computing geodesic paths on manifolds," *Proc. Natl. Acad. Sci. USA*, vol. 95, no. 15, pp. 8431–8435, July 1998.
[15] R. Philippsen and R. Siegwart, "An interpolated dynamic navigation function," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2005.
[16] K. Macek, I. Petrovic, and R. Siegwart, "A control method for stable and smooth path following of mobile robots," in *Proceedings of the European Conference on Mobile Robots*, 2005.
[17] D. Fox, W. Burgard, and S. Thrun, "The dynamic window approach to collision avoidance," *IEEE Robotics & Automation Magazine*, vol. 4, no. 1, pp. 23–33, March 1997.