HUNGARIAN JOURNAL OF
INDUSTRY AND CHEMISTRY
Vol. 48(1) pp. 11–17 (2020)
hjic.mk.uni-pannon.hu
DOI: 10.33927/hjic-2020-03

THEORETICAL BACKGROUND AND APPLICATION OF MULTIPLE GOAL PURSUIT TRAJECTORY FOLLOWER

ERNŐ HORVÁTH*[1,2], CLAUDIU POZNA[3], PÉTER KŐRÖS[2], CSABA HAJDU[2], AND ÁRON BALLAGI[2]

[1]Department of Computer Engineering, Széchenyi István University, Egyetem tér 1, H-9026 Győr, HUNGARY
[2]Research Center of Vehicle Industry, Széchenyi István University, Egyetem tér 1, H-9026 Győr, HUNGARY
[3]Faculty of Electrical Engineering and Computer Science, Transilvania University of Brasov, B-dul Eroilor nr. 29, 500036 Brasov, ROMANIA

Autonomous and self-driving technology is a rapidly emerging field among automotive-related companies and academic research institutes. The main challenges include sensory perception, prediction, trajectory planning and trajectory execution. The current paper introduces a design strategy and the mathematical background of the optimization problem with regard to the multiple goal pure pursuit algorithm. The aim of the algorithm is to provide a low degree of computational complexity and, therefore, a fast trajectory-tracking approach. Finally, in terms of our approach, not only the theoretical questions but the application challenges will be described as well.

Keywords: autonomous, self-driving, algorithm, trajectory-tracking

1. Introduction

The current paper deals with the design strategies, theoretical background and application of trajectory tracking. The trajectory-follower approach that is discussed is called the multiple goal pure pursuit algorithm, in particular the windowed version. In mobile robots or vehicles, the trajectory design strategy seeks to identify the best possible trajectory which approximates a desired trajectory based on waypoints. This also assumes that the trajectory design phase has already created the desired trajectory.

This paper is organized as follows. The first section introduces the general problem and summarizes the state-of-the art and most relevant results. The second section deals with the design strategy of trajectory tracking and defines the problem precisely. The third section defines the optimization problem and possible solutions, while the fourth section summarizes the results. Finally, the last section includes a summary and conclusions.

The current state-of-the-art results The essence of pure pursuit is to choose a look-ahead point from the desired trajectory in front of the vehicle and steer according to it. This simple idea can be realized in many different ways, e.g., the classical pure pursuit algorithm uses a fixed look-ahead distance. This means that on the given track, a look-ahead point can be determined as a subset of waypoints, which are a fixed distance from the vehicle. Choosing a look-ahead distance is a tradeoff between control overshoots and precision. If the look-ahead distance is small, the overshoot will appear because the chosen point is so close that the vehicle cannot react instantaneously, thus continuously exceeds its target. In this case, the absolute distance (error) is smaller but the vehicle will oscillate. In contrast, if the chosen point is too far away, the vehicle will follow the point after the following corner so cuts off the bend. This behavior can be eliminated if the look-ahead distance is adjustable or changeable. One of the most obvious ways to modify the look-ahead distance is to scale the speed of the vehicle accordingly [1]. The other method of modifying the look-ahead distance involves a fuzzy controller that automatically adjusts the look-ahead distance based on path characteristics, velocity and tracking errors [2, 3]. Similarly to the fuzzy approach [4], it was shown that an autonomous vehicle can drive at a velocity of approximately 80 km/h along explicit paths using differential GPS data. To improve the classical pure pursuit algorithm and eliminate steering latency, CF-Pursuit [3] replaced the circles employed in pure pursuit with a clothoid curve to reduce fitting errors.

Although many papers have examined this domain, the available source code is rather rare. A few of the most notable versions are Autoware [5], Python [6] and Raptor Unmanned Ground Vehicle (UGV) [7]. Our research group also introduced a pure pursuit method based on multiple look-ahead points [8], in this paper its updated

*Correspondence: herno@ga.sze.hu

version will be presented.

2. The design strategy

The motivation of our research was to construct a simple but flexible kinematic trajectory tracking approach. In addition, the implementation of a *more human-like thinking* was sought, in other words, to track and follow more goal points on a road simultaneously. Drivers also take multiple considerations into account, e.g., not only is lane-keeping an important task but a cyclist in the same lane also influences planning. In our previous work the multiple goal pursuit algorithm can be viewed as a supplement to the pure pursuit algorithm. The most significant change was the calculation of curvature. Assuming the reference trajectory is given with a set of geometric points T, position P, steering angle γ and orientation θ of the vehicle, $G \subseteq T$ is defined as the set of selected goal points with a preset length $N = |G|$:

$$G = \{G_1, G_2, \dots, G_N | G_k, (l+o) \leq k < (l+o+N)\} \tag{1}$$

An angle α_f from a domain of possible angles $[\alpha_{\min}, \alpha_{\max}]$ is assumed (presumably the wheel angle limits). A sequence of curvatures can be calculated each with a radius $p_f = -2/\alpha$ and a centre point C_f. A goal point G_k from set G to any C_f determines a line segment: the normalized difference between the length of this segment and radius p_f is the distance $d \in \mathbb{R}^+$ from the curvature. The metric for selecting a good angle is the sum of this difference for each goal point.

$$d_{\text{sum}} = \sum_{k=0}^{N} \left| \left\| \overrightarrow{C_f G_k} \right\| - \rho_f \right|. \tag{2}$$

Here d_{sum} denotes the total distance for goal point set G. The aim is to minimize this distance.

This paper introduces a new approach to the multiple goal pure pursuit algorithm, which is based on *windowing*. This algorithm is not regarded as a classical trajectory tracker since it slightly redesigns the original trajectory. With this method, a trajectory similar to the original one is created, but slight modifications render it more suitable for our vehicle to track. The design strategy consists of identifying the optimum trajectory which approximates a desired trajectory defined by a set of chosen points. This means that the aforementioned set is initial data and a minimum problem must be defined.

From the multitude of possibilities, the minimum problem will be defined starting with the following hypothesis:

1. During the approximation, the steering angle γ is constant;
2. Changing the steering angle γ is an instantaneous process.

If the previous hypotheses are correlated, it can be concluded that the initial set of points is approximated by a trajectory composed from arcs of a circle. The transition from one arc to another is instantaneous. For a particular arc of the circle, a subset (referred to here as the *working set*) is selected from the initial set.

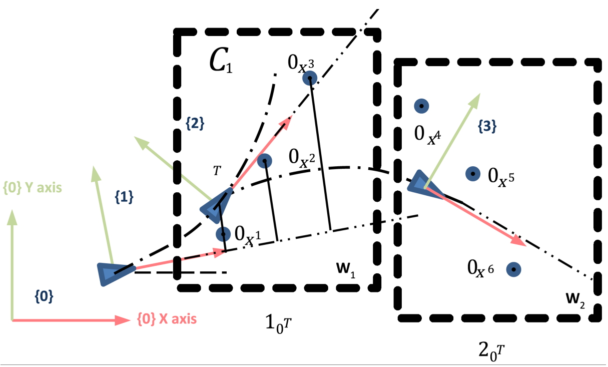

Figure 1: The windowed goal pursuit strategy

The following decisions are subjects of debate:

- The number of points included in the working set;
- The number of commune points which belong to different working sets;
- The position of the car when the next working set is defined.

The aforementioned observations are shown in Fig. 1, where a possible strategy is illustrated. The figure contains two referential systems. The first, 0, is the global (stationary) referential system; the second, $1, 2, 3$, is the mobile referential system attached to the vehicle. The blue triangles denote the abstraction of the vehicle. Two windows are labelled as $\mathbf{w}_{1,2}$. Each of them contains a *working set* of three points.

The trajectory is a composition of two arcs of a circle, C_1 and C_2. The first (C_1) is defined in referential frame 1 using the window $\mathbf{w}_1$, i.e., the points $0_{x^1}, 0_{x^2}$ and 0_{x^3}. The second (C_2) is defined in referential frame 2 using the window $\mathbf{w}_2$, i.e. the points $0_{x^4}, 0_{x^5}$ and 0_{x^6}. Each of the mentioned points is defined in referential frame 0.

3. The optimization problem

Using m windows and n points for each window, the mathematical definition of the problem is

$$R_j = \arg \left(\min \sum_{i=1}^{m} e_i \right), \tag{3}$$

where R_j denotes the radius of arc C_j, e_i stands for the distance from point i to arc C_j, j represents the current window number $j = 1, \dots, m$, and i is the current point number which belongs to window j.

The distance from point x_i to C_j is also debatable. Fig. 2 illustrates two possibilities: the first (Fig. 2a) only considers one coordinate of the current point. In contrast, in Fig. 2b the shortest distance is considered.

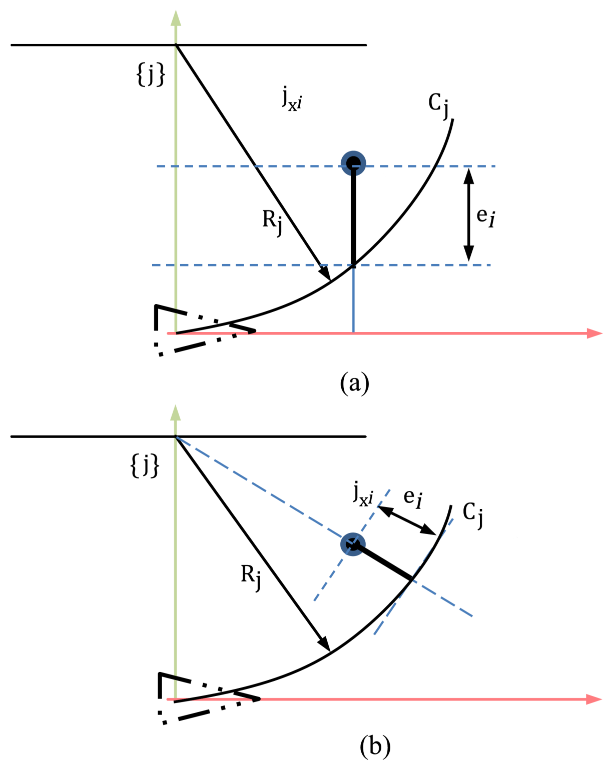

Figure 2: Possible ways of computing the distances

For the first choice:

$$e_i = y_i - C_j\left(x_i\right), \tag{4}$$

where

$$\begin{bmatrix} {}^j x_i \\ {}^j y_i \\ 1 \end{bmatrix} = {}^j_0 T\, {}^0\mathbf{x_i} = \begin{bmatrix} c\theta & -s\theta & 0 \\ s\theta & c\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} {}^0\mathbf{x_i}; \tag{5}$$

θ denotes the orientation of the referential system j; and x_j, y_j represent the coordinates of the origin of referential system j.

If this definition is assumed, it is evident that this is a quadratic regression problem with the analytical solution

$$\frac{1}{R} = \kappa = \left(\mathbf{X^t X}\right)^{-1}\left(\mathbf{X^t Y}\right), \tag{6}$$

where

$$\mathbf{X} = \begin{bmatrix} x_1^2 + y_1^2 \\ \ldots \\ x_n^2 + y_n^2 \end{bmatrix} \tag{7}$$

$$\mathbf{Y} = \begin{bmatrix} 2y_1 \\ \ldots \\ 2y_n \end{bmatrix} \tag{8}$$

For the second choice:

$$e_i = \left| R - \left\| \begin{bmatrix} x_i \\ y_i - R \end{bmatrix} \right\| \right| = \left| R - \sqrt{x_i^2 + \left(y_i - R\right)^2} \right| \tag{9}$$

which has a numerical solution.

3.1 Algorithm flowchart

In Fig. 3, a flowchart is presented in order to understand the approach more comprehensively.

The algorithm uses the vehicle parameters, pose, trajectory points and input parameters as inputs. The vehicle parameters are wheelbase b, steering angle γ and its position. Also, the points: ${}^0\mathbf{x}_k$ are part of the initial data. The algorithm parameters are: working set n, skyline d and curvature limit. This limit decides whether the locomotion is based on a linear or circular trajectory window. For every iteration, a new window is calculated.

4. Discussion

4.1 Simulation

The simulation uses the previous algorithm for a kinematic model which is in accordance with the starting hy-

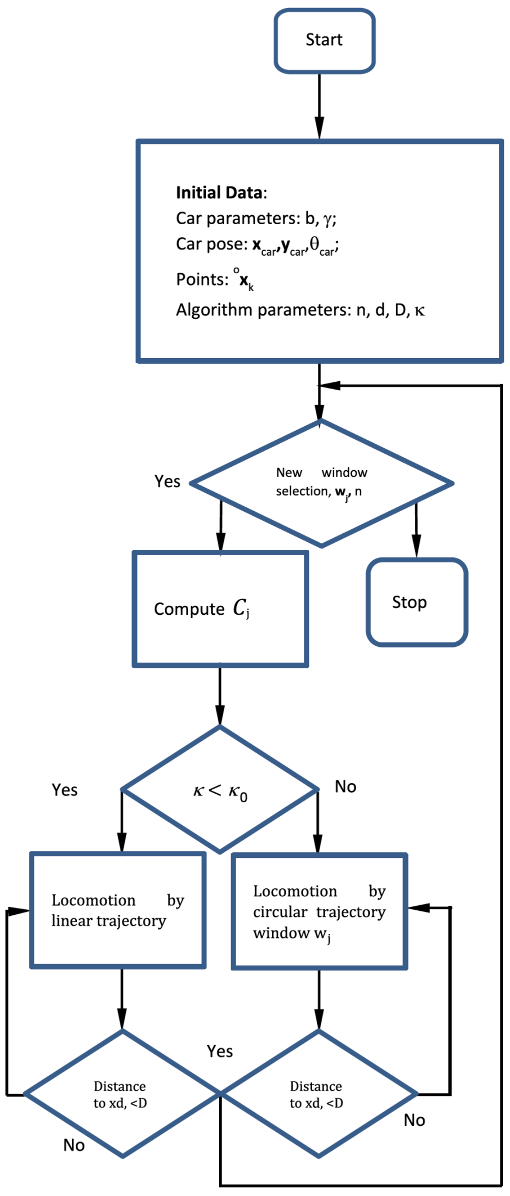
Start
Initial Data:
Car parameters: b, γ;
Car pose: $\mathbf{x}_{car}$, $\mathbf{y}_{car}$, θ_{car};
Points: ${}^o\mathbf{x}_k$
Algorithm parameters: n, d, D, κ
Yes
New window selection, $\mathbf{w}_j$, n
Compute C_j
Stop
Yes
$\kappa < \kappa_0$
No
Locomotion by linear trajectory
Locomotion by circular trajectory window w_j
Yes
Distance to xd, <D
Distance to xd, <D
No
No

Figure 3: Algorithm flowchart

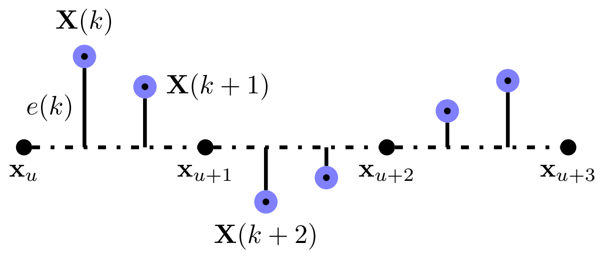

Figure 4: The definition of lateral deviation

pothesis, i.e., the steering is instantaneous. In the first simulation, the following kinematic bicycle model

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \frac{v}{L}\tan(\gamma) \\ \gamma = ct \end{cases} \tag{10}$$

was used. To analyze the results, the lateral approximation error was defined:

- The lateral deviation error of the current position $e(k)$ is the distance between this position and the current segment of the desired trajectory (Fig. 4);
- The total deviation error e_T is the sum of the error throughout the simulation from 1 to N;
- The relative deviation error e_r is the ratio of the total error to the total number of simulated points.

$$e(k) = \text{distance}\left(\mathbf{x}(k), \overline{{}^0\mathbf{x}_u{}^0\mathbf{x}_{u+1}}\right) =$$
$$= \left|\overline{\mathbf{x}(k)\mathbf{x}_{u+1}} - \widehat{\mathbf{x}_u\mathbf{x}_{u+1}}\left(\overline{\mathbf{x}(k)\mathbf{x}_{u+1}} \cdot \overline{\mathbf{x}_u\mathbf{x}_{u+1}}\right)\right| \tag{11}$$

$$e_T = \sum_{k=1}^{N} e(k) \tag{12}$$

$$e_r = \frac{e_T}{N} \tag{13}$$

These errors are dependent on the number of points included in the working set (n) referred to as the *skyline* and the position of the decision point (d). For the desired trajectory, a study relative to this dependency was conducted and the relative errors for each case were obtained (n, d).

The non-instantaneous steering was also simulated (Fig. 5). In this case, the kinematic bicycle model can be modified as follows:

$$\begin{cases} \dot{x} = v\cos(\theta) \\ \dot{y} = v\sin(\theta) \\ \dot{\theta} = \frac{v}{L}\tan(\gamma) \\ \gamma = \gamma_0 + \dot{\gamma}t \\ \dot{\gamma} = ct \end{cases} \tag{14}$$

The steering angle is a linear function where the steering velocity is constant during the steering process. The

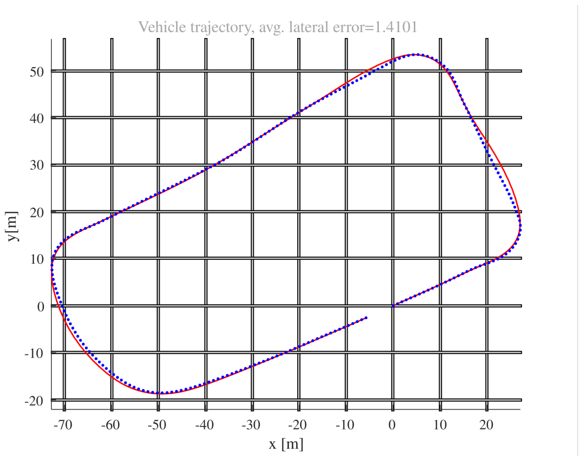

Figure 5: Simulation result: the blue dots denote the waypoints, the red line represents the trajectory generated by our model

trajectory of the robot is no longer an arc of a circle but a combination of two curves. The first of which is a clothoid (also referred to as an Euler spiral) when $\gamma = \gamma_0 + \dot{\gamma}t$ and an arc of a circle when $\gamma = ct$. In this case, the simulation result shows a lower quality of approximation. Given this phenomenon, the robot orientation angle error is defined as the difference between the orientation angle in the hypothesis where the steering is instantaneously modified to the final value and the orientation angle in the hypothesis where the steering angle is continuously modified from the initial to the final value.

$$e_\theta = \frac{v}{L}\left(\tan(\gamma_{\text{f}})\,\Delta t - \int_0^{\Delta t} \tan(\gamma_0 + \dot{\gamma}\tau)d\tau\right) =$$
$$= \frac{v}{\dot{\gamma}L}\left(\tan(\gamma_{\text{f}})\,\Delta\gamma - \log\left(\frac{\cos(\gamma_0)}{\cos(\gamma_{\text{f}})}\right)\right) \tag{15}$$

where γ_0 denotes the initial steering angle, $\gamma_{\text{f}} = \arctan(L\kappa$ represents the final steering angle, $\dot{\gamma}$ stands for the steering angle velocity, and Δt is the steering time.

It is observed that Eq. 12 is in accordance with our intuitions: if the steering angle velocity increases the orientation error angle will decrease; in contrast, the velocity is proportional to the error.

A strategy where a maximum orientation error $e_{\theta_{\max}}$ is allowed can now be contemplated and since the maximum steering velocity $\dot{\gamma}$ is an actuator parameter (which is constant), the following function for the vehicle velocity is proposed:

$$v_{\text{new}} = \frac{e_{\theta_{\max}}}{e_\theta} v_{\text{old}}. \tag{16}$$

In conclusion, the algorithm must be improved in order to yield better results. The following possibilities are suggested (Fig. 6):

- Refine the input set of points: interpolate new points;
- Change the vehicle velocity during locomotion (Eq. 16);

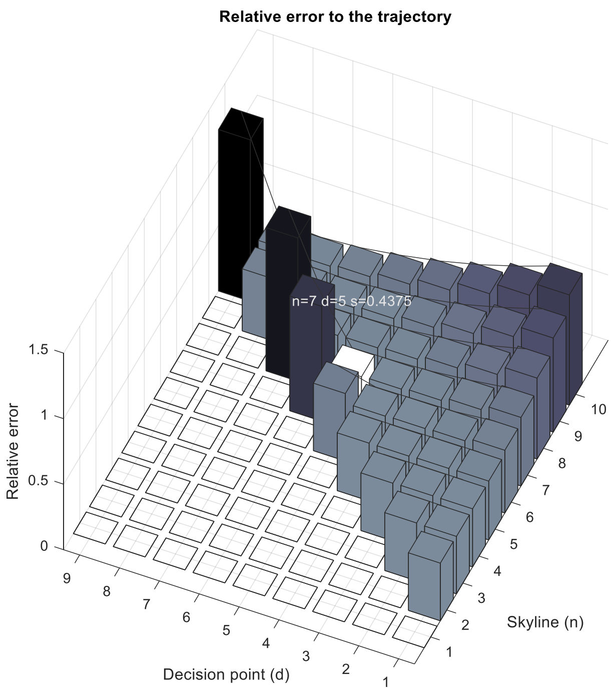

Figure 6: The proposed new strategy

- Define the new windows by including the points which succeed the decision point in the new working set.

With regard to an examined sample trajectory, a 3D graph (Fig. 7) is provided which shows the relative dependency obtained in terms of the relative error value in each case (n, d). In Fig. 7, the x axis denotes the skyline (n), the y axis represents the decision point (d), and the z axis stands for the relative error.

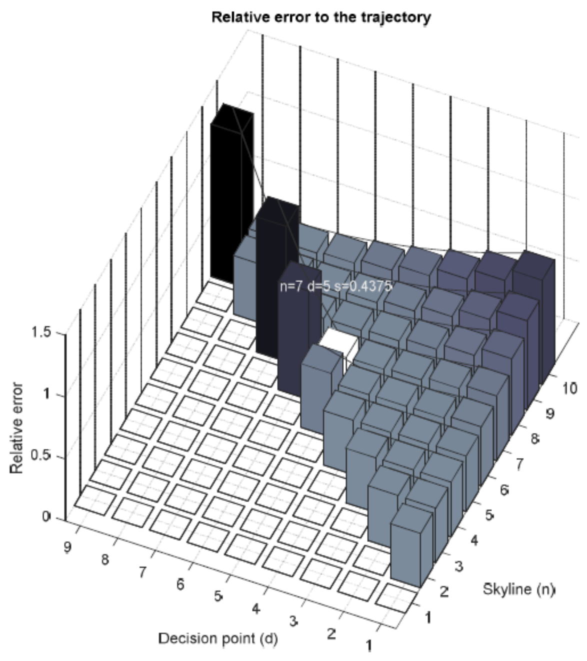

Figure 7: The relative error depends on the skyline (n) and decision point (d)

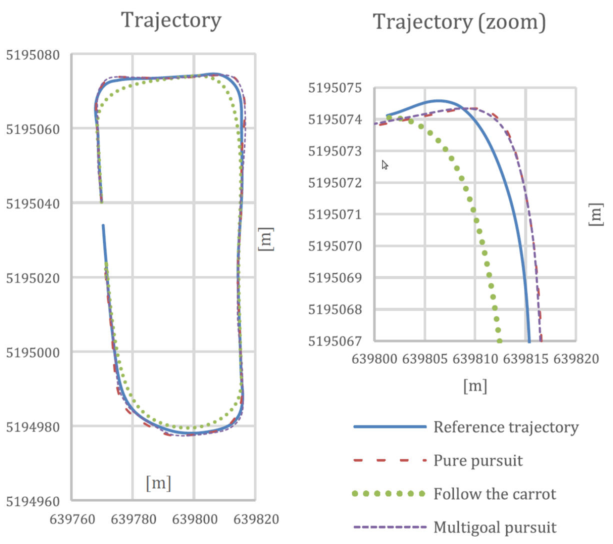

Figure 8: Typical behaviors of the examined approaches

4.2 Experiments

Firstly, the behavior of the algorithm in its own simulation environments was tested; the two presented approaches (the plain multiple goal pursuit and the windowed multiple goal version) were developed in parallel in Python and MATLAB. These simulations also used real-world trajectories and the vehicle model was first a kinematic, and later a dynamic one. Subsequently, it became necessary to even test the algorithms in Open Source Robotics Foundation's (OSRF) Gazebo for two reasons. Firstly, implementation of the algorithm as a Robot Operating System (ROS) node was sought and secondly, a more precise dynamic model of the vehicle already existed in Gazebo. Gazebo also facilitated the integration of the useful software as well as agile migration between development and target.

For these purposes, the algorithm had to be recreated in C++ not only because of the ROS compatibility but computational resources as well. Simulation-based execution yielded no significant deviation from the simulated tests. In Fig. 8, some of our results are shown regarding the examined trajectories. The x and y axes are in meters. This simulation is based on real-world measurements taken at the ZalaZone Automotive Test Track in Zalaegerszeg and the GPS coordinates are assigned using Universal Transverse Mercator (UTM) because of the minimal degree of distortion involved. The Follow the Carrot and speed ratio-based pure pursuit algorithms had to be implemented in order to experiment with them in the same environment.

In another experiment, autonomous functionalities were installed in an electric vehicle (Nissan Leaf). Fig. 9 represents the vehicle in operation. The operational domain of this vehicle was limited to the ZalaZone Auto-

Figure 9: The vehicle used in our experiments

motive Test Track and the university campus, moreover, it was required to reach a relatively slow speed of 25 km/h. The vehicle has front-wheel drive, with a wheelbase of 2.70 meters and a track width of 1.77 meters. To ensure instantaneous localization of the vehicle, two highly accurate Real Time Kinematic (RTK)-capable GPSs were used. One was KVH's Fiber Optic Gyro 3D Inertial Navigation System (GEO-FOG 3D INS), the other was Swift Navigation's Duro Inertial RTK. These sensors were also used to capture reference trajectories.

Our software was based on OSRF's ROS [9]. Our computing platform first features the NVIDIA Jetson TX2, later the NVIDIA Jetson AGX Xavier. Low-level control is realized by using a National Instruments' CompactRIO Controller, namely cRIO-9039, as the Real-time and FPGA modules - the NI-9853 CAN, NI-9403 DI, NI-9205 Voltage Input and NI-9264 Voltage Output Modules to be exact.

After the algorithm exhibited satisfactory behavior in the simulation environment, it was tested in a real-world scenario. During the experiment, the NVIDIA Jetson TX2 was used in the Nissan Leaf. The NVIDIA Jetson TX2 is a 7.5-watt embedded controller in which the Ubuntu 18.04 LTS Bionic Beaver along with the ROS Melodic Morenia were used. The embedded controller had a memory capacity of 8 GB and memory bandwidth of 59.7 GB/s as well as NVIDIA's Denver2, quad-core ARM Cortex-A57 CPU and integrated 256-core Pascal GPU installed. During our experiments, only the KVH's GEO-FOG 3D INS sensor was used as the source of localization, the wheel angle reference signal in addition to the speed reference via CAN were provided, and the wheel angle and speed were measured back in the same protocol. Nevertheless, during acquisition of the data, all the sensory information was logged by two of Velodyne's 16-channel LIDARs, Sick's 1-channel Laser Measurement Sensor LMS111 and the camera stream into ROS bag files. The following chart (Fig. 10) shows an example of how the algorithm generated reference trajectories for two bends on the routine track, moreover, the measured wheel angle is shown.

The waypoints for the algorithm were provided by driving through a predetermined path then saving as well as filtering the position and speed data via the waypoint_saver node.

In this experiment, only the multiple goal pursuit al-

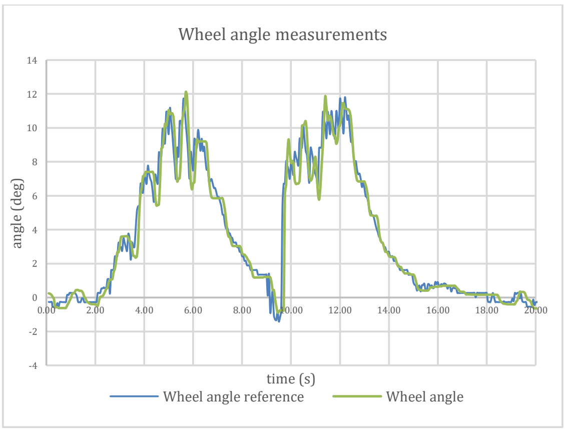

Figure 10: Changes in the wheel angle around two bends

gorithm with one parameter set was used. This set is better compared to Autoware's pure pursuit algorithm in terms of lateral deviation, but the speed ratio-based version worked even better. As a result, the multiple goal pursuit algorithm was able to control the vehicle with the aforementioned embedded controller at slow speeds of approximately 25 km/h. The algorithm is able to perform smooth and continuous movements along a predefined trajectory, even maneuvering around bends and along edges.

5. Conclusion

The current paper described the development of a trajectory-tracking approach, namely the multiple goal pursuit, and summarized the mathematical and theoretical background needed to understand its working principles. Enhancements to and variations in the basic approach are also described with regard to their benefits and weaknesses. Furthermore, a brief insight into the development process is given. Firstly, an initial version was developed in our own simulation environment, later the code became an ROS node and the simulation environment was replaced by Gazebo. Finally, real-world tests showed the viability of the new algorithm, which was tested by an autonomously guided vehicle in a closed but real-world traffic environment.

One of the benefits of this algorithm is that it can be tuned more precisely around bends, moreover, *it involves more human-like thinking*, that is, tracks and follows more waypoints on the road simultaneously. Furthermore, this approach provides quick and reliable reference signals for car-like robot kinematics, thus can follow the trajectory with relatively small errors.

The discussed algorithm was implemented in MATLAB and C++ as a Robot Operating System node and our measurements were based on debug data generated by this node. This implementation is currently publicly available on GitHub in addition to other measurements, videos, results and source codes: `https://github.com/szenergy/szenergy-public-resources`.

Acknowledgements

The research was carried out as part of the "Autonomous Vehicle Systems Research related to the Autonomous Vehicle Proving Ground of Zalaegerszeg (EFOP-3.6.2-16-2017-00002)" project in the framework of the New Széchenyi Plan. The completion of this project is funded by the European Union and co-financed by the European Social Fund.

REFERENCES

[1] Paden, B.; Čáp, M.; Yong, S. Z.; Yershov, D.; Frazzoli, E.: A survey of motion planning and control techniques for self-driving urban vehicles, *IEEE Trans. Intell. Veh.*, 2016, **1**(1), 33–35 DOI: 10.1109/TIV.2016.2578706

[2] Ollero, A.; García-Cerezo, A.; Martinez, J.: Fuzzy supervisory path tracking of mobile reports, *Control Eng. Practice*, 1994, **2**(2), 313–319 DOI: 10.1016/0967-0661(94)90213-5

[3] Samuel, M.; Hussein, M.; Mohamad, M. B.: A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle, *Int. J. Computer Applications* 2016, **135**(1) 35–38 DOI: 10.5120/ijca2016908314

[4] Rodríguez-Castaño, A.; Heredia, G.; Ollero, A.: Analysis of a GPS-based fuzzy supervised path tracking system for large unmanned vehicles, *IFAC Proceedings Volumes*, 2000, **33**(25) 125–130 DOI: 10.1016/S1474-6670(17)39327-8

[5] Kato, S.; Tokunaga, S.; Maruyama, Y.; Maeda, S.; Hirabayashi, M.; Kitsukawa, Y.; Monrroy, A.; Ando, T.; Fujii, Y.; Azumi, T.: Autoware on board: Enabling autonomous vehicles with embedded systems, in Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems, 2018, 287–296 DOI: 10.1109/ICCPS.2018.00035

[6] Sakai, A.; Ingram, D.; Dinius, J.; Chawla, K.; Raffin, A.; Paques, A.: PythonRobotics: a Python code collection of robotics algorithms, 2018, arXiv: 1808.10703

[7] Giesbrecht, J.; Mackay, D.; Collier, J.; Verret, S.: Path tracking for unmanned ground vehicle navigation: Implementation and adaptation of the pure pursuit algorithm, Defence Research and Development Canada (DRDC), Suffield, Alberta, Canada, 2005. `https://apps.dtic.mil/sti/pdfs/ADA599492.pdf`

[8] Horváth, E.; Hajdu, Cs.; Kőrös, P.: Novel pure-pursuit trajectory following approaches and their practical, in 10th IEEE International Conference on Infocommunications (CogInfoCom), Naples, Italy, 2019.

[9] Quigley, M.; Conley, K.; Gerkey, B.; Faust, J.; Foote, T.; Leibs, J.; Wheeler, R.; Ng, A. Y.: ROS: an open-source Robot Operating System, ICRA Workshop on Open Source Software, 2009 **3**(2)