2015 Pattern Recognition Association of South Africa and Robotics and Mechatronics
International Conference (PRASA-RobMech)
Port Elizabeth, South Africa, November 26-27, 2015

A Local Planner for Ackermann-Driven Vehicles in ROS SBPL

Nicolas Limpert and Stefan Schiffer and Alexander Ferrein
Mobile Autonomous Systems & Cognitive Robotics Institute
FH Aachen University of Applied Sciences, Aachen, Germany
Email: nicolas.limpert@alumni.fh-aachen.de, s.schiffer@fh-aachen.de, ferrein@fh-aachen.de

***Abstract*— For a mobile service robot, safe navigation is essential. To do so, the robot needs to be equipped with collision avoidance and global path planning capabilities. The target platforms considered in this paper are Ackermann-driven vehicles. Hence, a planning approach which directly takes the kinematic and dynamic constraints of the vehicle into account is required. The Search-based Planning Library (SBPL) package of the Robot Operating System (ROS) provides global path planning which takes these constraints into account. However, it misses a local planner that can also make use of the Ackermann kinematic constraints for collision avoidance. A local planner is useful to take dynamic obstacles into account as early as possible. In this paper, we extend the SBPL included in ROS by a local planner, which makes use of motion primitives. We extend the ROS package and show first experimental results on Ackermann-driven vehicles.**

I. Introduction

For a mobile service robot, safe navigation is essential. For this, the robot needs to be equipped with collision avoidance as well as with global path planning capabilities. The former is usually referred to as local path planning while the latter problem is called global path planning. The problem of path planning can be defined as in [1]:

Path planning is the problem of finding a path in the free configuration space $\mathcal{CS}_{free}$ between an initial configuration and a final configuration. If one could compute $\mathcal{CS}_{free}$ explicitly, then path planning would be a search for a path in this n-dimensional continuous space. However, the explicit definition of $\mathcal{CS}_{free}$ is a computationally difficult problem, theoretically (it is exponential in the dimension of $\mathcal{CS}$) and practically. [...] Fortunately, very efficient probabilistic techniques have been designed that solve path planning problems even for highly complex robots and environments. They rely on the two following operations.

1) Collision checking checks whether a configuration $q \in \mathcal{CS}_{free}$ or whether a path between two configurations in $\mathcal{CS}$ is collision free, i.e., whether it lies entirely in $\mathcal{CS}_{free}$.
2) Kinematics steering finds a path between two configurations q and q' in $\mathcal{CS}$ that meets the kinematic constraints, without taking into account obstacles.

Many service robots meet holonomic kinematic constraints (omni-directional drives), or can at least turn on the spot. Then, following a path in 2D space can, for instance, be done by decoupling the global path planning problem (finding a path between two points in the world) from the local path planning or navigation problem (collision avoidance). For instance, [2], [3] describe how to find a path in the geometric space of the robot, and then search for a realisation of that path in the configuration space of the robot. However, when it comes to car-like vehicles, one has to take the non-holonomic constraints into account. Then, decoupling the search for a path in Euclidean space from the search for a path in the configuration space cannot easily be done any more. Instead of planning a path in a discretized Euclidean space (grid), one can, for example, use lattice graphs based on motion primitives [4]. Motion primitives based on the possible configurations of the robot describe reachable states in Euclidean space. For a car-like drive as shown in Fig. 1(a), the parameters of the configuration space are the velocity and the steering angle. The robot moves on an arc which depends on both parameters. Different sets of parameters describe different arcs. To reach a goal in the global map, we need to find arcs consisting of different feasible parameters. Motion primitives encode different sets of these parameters and hence different arcs. The task is to find a sequence of feasible motion primitives that lead the robot to the goal.

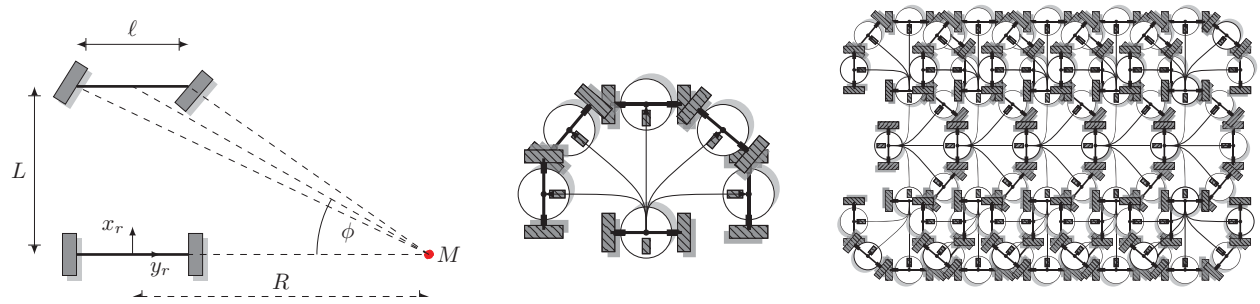
In this paper we describe an extension of the SBPL planner, a lattice graph motion planning algorithm for car-like vehicles implemented in the Robot Operating System (ROS) [5]. The available implementation provides only a global planner, no local planner for Ackermann steering is available. This means, for one, that no local collision avoidance for car-like vehicle is available with SBPL. For another, if the robot fails to execute the pre-planned path due to slight imperfections (slip or drift), a new global plan has to be computed. In many cases, the global plan would still be valid, but a bridge plan to meet the global plan by making small adjustments would be required. Our contribution in this paper is:

1) a local planning algorithm based on motion primitives for car-like vehicles;
2) an integration of the local planner into the ROS Move Base package together with the SBPL lattice planner used for global path planning.

We report on first experimental results of the SBPL global and local planner on a 1/10 model cart with Ackermann steering.

The remainder of this paper is organised as follows. In the next section, we review related approaches for robot motion planning. In Section III we introduce the SBPL planning algorithm and its implementation in ROS, before we describe

978-1-4673-7450-7/15/$31.00 ©2015 IEEE



(a) Ackermann steering

(b) Motion primitive example for 5 orientations/speed-pairs

(c) (Incomplete) Lattice graph based on the 5-motion primitive. The graph is expanded up to depth 5 at 0°, and depth 1 for $\pm 90^\circ$ are expanded; for reasons of clarity, no nodes at 45° are expanded.

Fig. 1. The basic idea of lattice-based navigation planning based on motion primitives.

our extension to the planning system in Section IV. It is followed by first experimental results in Section V. We conclude with a discussion and an outlook to future work.

II. Related Work

As described in the introduction, there are approaches that decouple the path planning and the motion planning problem. One such example is Stachniss and Burgard [2]. In their approach, first a Euclidean path is found in the global map. To follow this path with the robot, valid motion commands in an area around the Euclidean path are sought. Our approach follows a similar idea. It first renders a feasible path in the Euclidean space around the robot. It then tries to follow this path as close as possible. Compared to their planning approach, ours rather follows the idea of Pivotraiko and Kelly [6] which made use of pre-computed motion primitives in order to plan with feasible motions and to speed up planning. Their work is not restricted to Ackermann-driven robots but rather provides a general method to plan smooth trajectories. Our approach can also be used for other robotic platforms—one only has to provide the several motion capabilities in the form of motion primitives and tune the appropriate executing component (e.g. one has also to consider sideways movement for holonomic robots).

The idea of planning motions with motion primitives can also be transferred to Bezier curves which let the user plan smooth motions for non-holonomic robots. In [7], Liang et al. present an approach to plan motions for an automatic parking assistant based on Bezier curves and a PID controller. The approach is able to plan smooth trajectories for the parallel parking tasks for Ackermann-driven robots. The first instance of our planner makes a similar use of a PID approach in order to get the current difference from the robot's position to the next way-point on the path. Rather than using motion primitives, Bezier curves can probably serve motion plans feasible for the particular robot in narrow situations. They can lead to better motions than the combination of motion primitives. The downside is that every motion combination has to be calculated from scratch rather than making use of pre-computed motion primitives.

Choi et al. [8] also had the idea of making use of piece-wise Bezier curves to combine a set of these curves as a set of feasible motions in an appropriate order. The usage of piece-wise Bezier-curves is much closer to the idea of a set of motion primitives which are used by SBPL. The Bezier curves are parameterised according to the motion constraints of the robot. It is therefore an interesting work for the SBPL Lattice Planner as it takes the idea of motion primitives further by defining the different dynamic constraints not as motion primitives, but mathematically as Bezier curves.

An even more complex motion planning problem is tackled in the work of Laumond [9]. This work incorporates motion plans for articulated vehicles. The approach is to generate vector fields in order to achieve feasible motions. This idea is more complex than the generation of motion primitives as the planning approach has to take other body shapes into account. However, it makes an interesting alternative as it lets the robot plan in much more dynamic environments just like the Bezier curves do. Our approach is founded on the work of Likhachev et al. [10]. They make use of lattice graphs based on motion primitives for self-driving cars. They implemented their approach in the ROS SBPL Lattice Planner which is presented in detail in the next section. Our work is founded on this approach.

Making use of motion primitives is very similar to the work of F. von Hundelshausen et al. in [11]. Their approach is to incorporate a 360 degree laser range scanner in order to evaluate a set of "tentacles". These "tentacles" form a set of feasible motions based on the current robotic vehicle's velocity and surrounding obstacles. This makes them a very similar idea to the motion primitives used in our approach. Their benefit is that they can incorporate the current robot's velocity. Another work that takes Ackermann geometries into account is [12] which makes use of Dubin's paths [13]. The idea is to perform motion commands that only consist of a combination of the plain motions left, forward and right. Reeds and Shepp [14] extend this work and add backward movements. The benefit of this approach over ours is that it always delivers the optimal path under the assumption that there are no obstacles in the way. Unfortunately how-

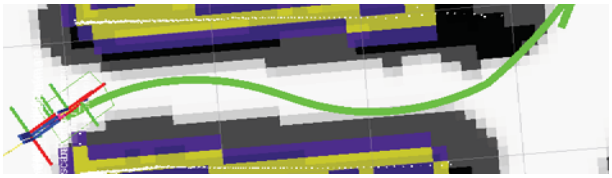


Fig. 2. SBPL Lattice Planner planning through a corridor with too sharp a turn for the local planner.

ever, in a real world scenario there are obstacles in the way. In the ROS environment there exists a local planner called `ackermann_local_planner` which makes use of Dubin's Paths. This work has not been considered in our experimental setup for a comparison as it does not provide any collision avoidance at all.

In ROS, a navigation package called the `move_base` which makes use of a global and a local planning aspect is provided. The `move_base` also delivers a so-called `base_local_planner` which provides implementations of the Trajectory Rollout Planner and Dynamic Window Approach. They both deliver the idea of sampling a set of motions and choose the appropriate motion in order to get a plan to reach a navigational goal. However, the `base_local_planner` is used for robots that are at least differentially driven and can rotate in place. This makes it difficult to be used on Ackermann-driven vehicles as they cannot rotate in place.

III. The SBPL Package for ROS

In [10], Likhachev and Ferguson describe a global planner for generating complex dynamically-feasible manoeuvres for autonomous vehicles travelling at high speeds over large distances. They use an approach by Pivotraiko and Kelly [15] to generate lattice graphs based on motion primitives. Their approach makes use of the anytime A* variants ARA* and AD* [16] to come up with a dynamically feasible motion plan at any time given. When travelling at high speeds, it is important to have an immediate answer what (dynamically feasible) action the robot should perform next in order to reach the next target point. The SBPL approach was used by the Tartan Racing Team [17], the winning team of the DARPA Urban Challenge (DUC). For a concise overview of the DUC, see [18].

The basic principle of lattice graphs based on motion primitives is shown in Fig. 1. Fig. 1(b) shows motion primitives for a robot with 5 different successor states. Based on the kinematic and dynamic constraints of the robot platform and the current state, robot platforms with non-holonomic motion constraints only have certain feasible successor states. For instance, as a non-holonomic robot cannot turn on the spot, it has to perform quite some manoeuvre (driving forwards, pushing backwards, driving forwards again) to come to a stand-still at the same position but with a different orientation depending on the turning angle of the vehicle. The parameters for the Ackermann drive (Fig. 1(a)) are steering angle and translational speed. Based on these two parameters and the current state, a feasible successor state of the vehicle which is reachable from the current state can be computed.

Now, when trying to navigate to a point in the global frame of the robot, one has to derive the feasible drive commands to reach that very point. Unlike road-map approaches which first compute a path in the 3D space and then generate the motion commands, lattice graph planning approaches already take the feasible commands into account while planning a path to the goal. Fig. 1(c) shows an example for our 5-motion primitive. In each state, feasible motion primitives can be applied to come up with the successor state in the search graph. The nodes of the graph represent the pose of the robot in 3D space, the arcs denote feasible motion commands.

The SBPL Lattice Planner is an implementation of a global planner using motion primitives in ROS.[1] It can be used as a global planner within the `move_base` package.[2] The ROS `move_base` package is a navigation package for global and local path planning algorithms. It takes standard information such as odometry information, a global map or a localisation pose into account to come up with a `cmd_vel` message that is directly sent to the motors. The `cmd_vel` message gives the translational and rotational velocity the robot has to execute in the next time step. In addition, this package implements a state machine for recovery behaviours and a local *cost map*. The cost map is a local map around the robot which takes dynamical obstacles (which are not represented in the global map) into account. The local planner uses this cost map for collision avoidance while following way points computed on the global map. The *cost map* can also incorporate different dynamical layers which can include environment information gathered from the robot's sensors.

Figure 2 shows an example of a global plan derived by SBPL for a narrow passage, where SBPL tries to find a solution through this passage. The robot is located at the left end of the corridor, the target point is around the opposite corner. The black cells represent the global cost map, i.e. the obstacles that could be perceived by the sensors of the robot. The small white dots represent the sensor readings from a laser range finder showing the walls of the corridor. One can observe that the obstacles and walls are extended in the cost map to guarantee a collision-free path. One can further observe the application of the motion primitives leading to a left-right-left combination of turns to navigate around the corner at the end of the corridor. The final left, however, touches an obstacle in the global cost map. The SBPL Package for ROS is also used for other planning tasks, for instance, for the manipulator planning package MoveIt!.[3]

IV. A Local SBPL Planner for Car-like Vehicles

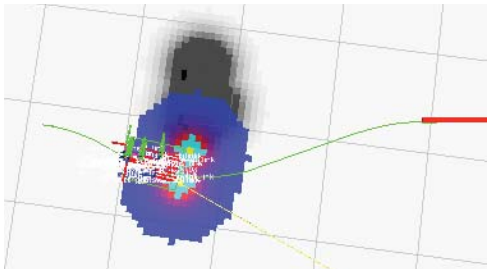
In order to plan feasible motions for our Ackermann-driven robots (Fig. 3) we deployed the ROS SBPL planning package as a local planner and merged it with the ROS `pose_follower` package for local navigation. Addition-

[1] `http://wiki.ros.org/sbpl_lattice_planner`
[2] `http://wiki.ros.org/move_base`
[3] `http://moveit.ros.org/`

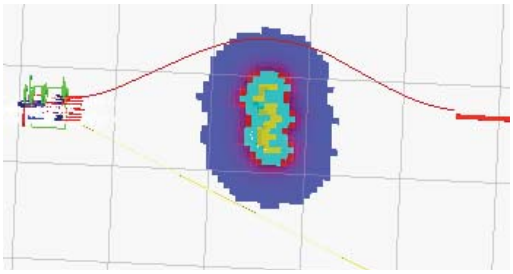
(a) FHAC Rover is a 1/10 car model.

(b) Rugged-terrain vehicle MQOne with Ackermann steering [19].

Fig. 3. Ackermann-driven robots used in the experiments.



(a) Re-planning in global frame



(b) Re-planning in local frame

Fig. 4. Comparison of global and local re-planning on sudden obstacle

ally, we use the `carrot_planner`[4] to get very basic global goals (Euclidean description of start and goal).

As we mentioned in the previous section, the SBPL planner works fine for global path planning. It takes the dynamic constraints of the robot platform into account to come up with a plan consisting of feasible motion patterns of the used vehicle. One drawback, however, is that the SBPL is only available as a global planner. This means that dynamic obstacles that appear right in front of the robot and hence are not represented in the global map of the robot, are not taken into account. For the original use-case of the package, this was not a problem.

Our use-cases, however, are a bit different, since our vehicles drive at much lower speeds and in more narrow environments. We want to use the SBPL planner for our 1/10 FH Aachen Rover (Fig. 3(a)) as well as for our rugged-terrain mining vehicle MQOne (Fig. 3(b)). The former vehicle is a 1/10 model of a car-like vehicle which is used for teaching activities such as the International ROS Summer School [20]. The second vehicle is the research robot MQOne [19] which is a rugged-terrain robot design used for mapping tasks in underground mines. In our target scenarios we need to take dynamic obstacles into account and we should take such obstacles into account as early as possible. This is especially useful for Ackermann-driven platforms to avoid unnecessary manoeuvring.

In our approach, we use SBPL for local planning. Once the global planner (`carrot_planner`) sends the goal, the local planner receives a list of way-points. The local planner then considers the current deviation from the robot's position to the next global way-point and plans a path to that respective way-point by making use of the SBPL Lattice Planner. As shown in Fig. 1, SBPL combines the possible motion primitives in order to reach the desired goal position as closely as possible. When the local planner generated a path to the next way-point, the `pose_follower` (basically a P controller for linear and angular velocities) generates motion commands to drive to the particular way-point. If an obstacle is detected along the path, local re-planning is initiated. This way, we consider dynamic obstacles as soon as possible and only re-plan to the next global way-point instead of re-planning globally. In Fig. 4(a), the result of a global re-planning of the path is shown.

1) The robot realises that it is about to collide with a dynamic obstacle.
2) It starts some emergency stop behaviour.
3) It tries to find a global path around the obstacle.

In Fig. 4(b) our novel SBPL-based local planner can be seen in action.

1) The ROS cost map is updated with the dynamic

[4] `http://wiki.ros.org/carrot_planner`

obstacle.
2) The local planner initiates the planning procedure to find motion commands that guide the robot around the obstacle back on the globally planned path.
3) The advantage is that the global plan is still valid after surrounding the obstacle.

Hence, no new global planning needs to be initiated when collision avoidance kicks in. Further, using motion primitives for trying to find a way around the obstacle leads to smoother overall behaviour of the robot. One should note that an emergency stop needs to be in the portfolio for the local planner as well to ensure that the robot can stop in front of the obstacle in case no local plan around the obstacle could be found. The test scenario shown in Fig. 4 was simulated using the ROS simulation environment Gazebo.

V. First Empirical Results

In this section we show first experimental results. We compare our novel local SBPL-based planner with SBPL as a global planner in combination with the aforementioned `pose_follower` for local navigation. We compare both approaches in terms of planning time, path length, and number of re-planning steps in different simulation scenarios. For simulating the 1/10 model car, we use the Gazebo simulator.[5] The tasks in each of the scenarios were to advance about 10 meters around different obstacles. These obstacles were unknown in the initial planning phase. The first test was to try and avoid a pylon. In the second test scenario, the robot drove directly towards a wall standing in the environment. In the third test scenario, the robot had to drive around a slalom course where several blocks were obstructing the direct path to the goal.

Results are shown in Tab. I. The numbers result from averaging over twenty runs for every scenario. The planning time refers to the time that the SBPL component in each configuration needs for planning. In the scenario, where the robot was to drive around a pylon, both approaches are nearly equal in terms of the total travel distance and planning time. The local planner has a slightly lower path execution time despite the fact that it re-planned two times more often than the global planner.

In the second scenario (Tab. I(b)), where the robot was heading towards a wall and had to plan a path around it, our novel local SBPL planner was able to execute the planned path much faster. The `pose_follower` needed 50 seconds more despite a similar path length and planning time. The advantages of using a local planner in combination with a global planner over using just a global planner can clearly be seen in our third test scenario. The results are shown in Tab. I(c). This was a test setup with several blocks standing in front of the rover forming a slalom course. The robot had to plan a way through the blocks to reach the target point. Our approach yielded a smaller travel distance and a much shorter execution time. The execution time of the our approach is only half of the global planner plus the pose follower. The local planner re-planned about 4 times more often than the global approach, yielding nonetheless a shorter overall planning time.

These results suggest that using a local SBPL approach together with a global one is beneficial in general. One of the reasons is that with the global planner plus pose follower re-planning is only initiated when the robot decides that the global path cannot be followed as it is leading through an obstacle.

Initial testing was in simulation, secondary testing was with real hardware. Fig. 5 shows a city-like environment that we used during our International ROS Summer School which we held between August 10–24 in Aachen. The challenge for the students from over 20 nations was to program a pizza delivery robot. They made use of a number of different ROS packages such as Hector SLAM for mapping and localisation [21]. Fig. 5(c) shows the cost map of the environment consisting of four blocks surrounding drive-ways. Fig. 5(d) shows a planned path in this environment. While some more tweaking of the motion primitives would be required to increase the planner's performance, we demonstrated successful navigation of Ackermann-driven vehicles through narrow passages.

VI. Discussion

In this paper we presented a method for navigation on Ackermann-driven mobile robots where we use SBPL as a local planner. This alleviates incorporating the kinematic constraints while handling dynamic obstacles. SBPL uses motion primitives and hence considers the kinematic constraints. This is especially useful for Ackermann-driven vehicles as the constraints might result in quite extensive manoeuvres. Using SBPL as a local planner improves handling of dynamic obstacles as compared to only using SBPL as a global planner. This is because the global planner considers non-static obstacles only when collision is immanent, whereas the local planner can modify the path as soon as an obstacle comes into the field of view. In our experimental evaluation we show that using our local planner is successful in both, a Gazebo simulation environment and in a real scenario with the FHAC Rover.

Planning with motion primitives has the benefit of being able to define the particular kinematic constraints in order to plan feasible motions to reach a navigational goal. Providing a component for local navigation that uses motion primitives, is improving the handling of dynamic obstacles. This is of particular importance if the kinematic constraints cause cumbersome manoeuvring in order to react to local changes like a nearby obstacle. In our evaluation we could show that in such situation we can cut the time needed to complete a path in half, using SBPL as a local planner.

The downside of motion primitives is that one always has to evaluate and define the capabilities of the particular robot in the form of motion primitives manually. If one could let the robot find out its very own physical constraints and define those as motion primitives it would enable the robot to perform planning based on the particular motion constraints.

[5] `http://gazebosim.org/`

TABLE I

RESULTS FROM DIFFERENT TEST SCENARIOS

(a) The pylon scenario.

Data	global SBPL + pose_follower	carrot + local SBPL
travel dist [m]	11.342	9.342
execution [s]	40.839	35.493
planning [s]	0.196	0.189
avg. # re-plans	2.9	5.65

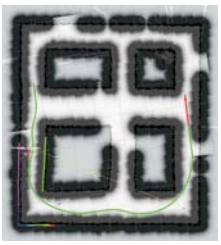
(b) The wall scenario.

Data	global SBPL + pose_follower	carrot + local SBPL
travel dist [m]	16.837	10.698
execution [s]	90.8	41.6
planning [s]	0.365	0.331
avg. # re-plans	3.9	4.6

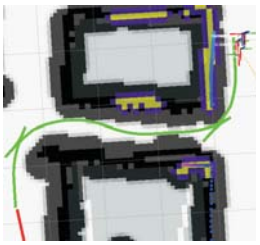
(c) The slalom course scenario.

Data	global SBPL + pose_follower	carrot + local SBPL
travel dist [m]	18.986	12.945
execution [s]	90.4	48.5
planning [s]	0.343	0.314
avg. # re-plans	6	17.7

(a) City-like environment (streets and blocks) used in the ROS Summer School 2015.



(c) Map of the environment



(d) Screenshot of path in execution

Fig. 5. Planner in action at the ROS Summer School 2015

ACKNOWLEDGEMENTS

We thank the ROS Summer School team for their support in this work. We would further like to thank the anonymous reviewers for their valuable comments and suggestions to improve the quality of the paper.

REFERENCES

[1] M. Ghallab, D. Nau, and P. Traverso, *Automated Planning: Theory & Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[2] C. Stachniss and W. Burgard, "An integrated approach to goal-directed obstacle avoidance under dynamic constraints for dynamic environments," in *IEEE/RSJ International Conference on Intelligent Robots and Systems, Lausanne, Switzerland, September 30 - October 4, 2002*, 2002, pp. 508–513.

[3] S. Jacobs, A. Ferrein, S. Schiffer, D. Beck, and G. Lakemeyer, "Robust collision avoidance in unknown domestic environments," in *RoboCup 2009: Robot Soccer World Cup XIII*, ser. LNCS, vol. 5949. Springer, 2009, pp. 116–127.

[4] M. Pivtoraiko and A. Kelly, "Efficient constrained path planning via search in state lattices," in *International Symposium on Artificial Intelligence, Robotics, and Automation in Space*, 2005.

[5] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "ROS: an open-source Robot Operating System," in *ICRA Workshop on Open Source Software*, 2009.

[6] M. Pivtoraiko and A. Kelly, "Kinodynamic motion planning with state lattice motion primitives," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2011, pp. 2172–2179.

[7] Z. Liang, G. Zheng, and J. Li, "Automatic parking path optimization based on bezier curve fitting," in *IEEE International Conference on Automation and Logistics (ICAL)*. IEEE, 2012, pp. 583–587.

[8] J.-W. Choi, R. Curry, and G. Elkaim, "Piecewise bezier curves path planning with continuous curvature constraint for autonomous driving," in *Machine Learning and Systems Engineering*. Springer, 2010, pp. 31–45.

[9] J.-P. Laumond, "Nonholonomic motion planning versus controllability via the multibody car system example," DTIC Document, Tech. Rep., 1990.

[10] M. Likhachev and D. Ferguson, "Planning long dynamically feasible maneuvers for autonomous vehicles," *The International Journal of Robotics Research*, vol. 28, no. 8, pp. 933–945, 2009.

[11] F. Von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles: Integral structures for sensing and motion," *Journal of Field Robotics*, vol. 25, no. 9, pp. 640–673, 2008.

[12] X.-N. Bui, J.-D. Boissonnat, P. Soueres, and J.-P. Laumond, "Shortest path synthesis for dubins non-holonomic robot," in *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 1994, pp. 2–7.

[13] L. E. Dubins, "On curves of minimal length with a constraint on average curvature, and with prescribed initial and terminal positions and tangents," *American Journal of mathematics*, pp. 497–516, 1957.

[14] J. Reeds and L. Shepp, "Optimal paths for a car that goes both forwards and backwards," *Pacific journal of mathematics*, vol. 145, no. 2, pp. 367–393, 1990.

[15] M. Pivtoraiko and A. Kelly, "Generating near minimal spanning control sets for constrained motion planning in discrete state spaces," in *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2005, pp. 3231–3237.

[16] M. Likhachev, D. Ferguson, G. Gordon, A. Stentz, and S. Thrun, "Anytime search in dynamic graphs," *Artificial Intelligence*, vol. 172, no. 14, pp. 1613–1643, 2008.

[17] D. Ferguson, T. M. Howard, and M. Likhachev, "Motion planning in urban environments," *Journal of Field Robotics*, vol. 25, no. 11-12, pp. 939–960, 2008.

[18] M. Buehler, K. Iagnemma, and S. Singh, *The DARPA Urban Challenge: Autonomous Vehicles in City Traffic*, 1st ed. Springer Publishing Company, Incorporated, 2009.

[19] S. Rebel, F. Hüning, I. Scholl, and A. Ferrein, "Mqone: Low-cost design for a rugged-terrain robot platform," in *Intelligent Robotics and Applications - 8th International Conference, ICIRA 2015, Portsmouth, UK, August 24-27, 2015, Proceedings, Part II*, ser. LNCS, vol. 9245, 2015, pp. 209–221.

[20] A. Ferrein, S. Kallweit, I. Scholl, and W. Reichert, "Learning to program mobile robots in the ros summer school series," 2015.

[21] S. Kohlbrecher, J. Meyer, T. Graber, K. Petersen, U. Klingauf, and O. von Stryk, "Hector open source modules for autonomous mapping and navigation with rescue robots," in *RoboCup 2013: Robot World Cup XVII*, ser. LNCS. Springer Berlin Heidelberg, 2014, vol. 8371, pp. 624–631.