Master Thesis, 2008
Supervisor: Christian Balkenius
Department of Cognitive Science
Lund University, Sweden.

AUTONOMOUS PARALLEL PARKING

-A study based on the implementation of a car-like mobile robot

Johan Jernström
johan.jernstrom@gmail.com

May, 2008

Abstract:
The area of autonomous parking attracts a great deal of attention from the research community and the automobile industry. Research has showed that many robotic navigation systems fail to deliver the performance and flexibility of animal navigation techniques, even though they are both mathematically and technologically complex. Biomimetic robotic research, i.e., research that tries to mimic biological solutions, has inspired me to take on a simplified approach to the task of autonomous parallel parking. In this study, three ways to reduce the complexity of the task are shown and demonstrated by the implementation of a car-like mobile robot. First, a distinction between to autonomously find *a free parking place and to autonomously* park *a vehicle is made, where only the latter is of relevance. Second, nothing but the goal, i.e., the free parking place, needs to be plotted on the internal map. Finally, the solution should only use cheap and commonly available technologies. The practicability of the prototype is measured in terms of performance, accuracy and reliability, with satisfactory results.*

Keywords: autonomous parking, car-like vehicle, path planning, map-learning, robotic navigation

1. Introduction

Parking in a tight spot can be a very hard task, even for experienced drivers. It requires precise maneuvers and distance estimations – tasks that humans are not specialized to accomplish. For this very specific part of driving, computers are much better fit to contend with the nature of the task. Therefore, a computer system to aid or even replace the driver during this maneuver would be ideal.

Indeed, this topic attracts a great deal of attention from the research community as well as the automobile industry (Chao et al., 2005; Holve & Protzel, 1996; Lo et al., 2003). One of the commercially available versions was released in Japan by Toyota in 2003 (Dawson, 2004). The system, available as an extra option of the Prius model, has however some drawbacks. For example, it takes two minutes to park the car, a relatively long time compared to human drivers, at least to some of us. In addition, it is not able to stop for any unforeseen obstacles in its path while parking. Perhaps these issues are part of the reason to why it was not released in America.

Biomimetic robotic research, i.e., research that tries to mimic biological solutions, has inspired me to further investigate the problem of robotic navigation in general and the task of autonomous parallel parking in particular. The research has showed that the classic view on navigation seems to be of little help. Many robotic navigation systems fail to show the performance and most of all the flexibility of animals, even of insects like ants or bees (Franz & Mallot, 2000). Moreover, many of the research prototypes developed are based on rather complex methods and technology, which makes the technique both expensive and difficult to implement.

In this study, three ways to reduce the complexity of the task are shown and demonstrated by the implementation of a prototype called APRIL – Autonomous Parking Robot Implemented in Lego.

In the following, the basic concepts of classic robotic navigation, like methods for localization, map-learning and path-planning, are presented. I also present a short review of the existing research related to autonomous parking. Finally I give a further description of the general aim and idea behind this study. In the next chapter the materials and method used in this study are presented. The technology and design of the prototype, APRIL, are also described. Chapter three gives a presentation of the measured performance and capabilities of APRIL. In the last and fourth chapter, the results and conclusions from this study are discussed.

1.1. Robotic Navigation

The task of autonomous parking of a car-like vehicle is a special case of robot navigation. Navigation has been defined as a process able to answer the following questions: "Where am I?", "Where are other places with respect to me?" and "How do I get to other places from here?" (Levitt & Lawton, 1990). This view of navigation has become the most commonly used in robotic navigation (Franz & Mallot, 2000).

1.1.1. Localization Strategies

One of the most important yet complicated task in the effort to develop fully autonomous robots, is to know the position of the robot, to answer the question "Where am I?" (Borenstein et al., 1997). Furthermore, localization is closely coupled to map-learning; you need to have some sort of spatial representation to localize, and, conversely, you need to know your position to create such a representation (Meyer & Filliat, 2003a).

To self-localize, a robot needs to combine the two available sources of information, *idiothetic* and *allothetic* (Meyer & Filliat, 2003a). The idiothetic source provides internal information about the robot's movements. It consists of measurements of speed, acceleration, leg movement or wheel rotation (*odometry*). This data can be used in a very straightforward way to roughly estimate the robot's position. Allothetic information consists of input from the environment such as vision, smell, touch, range measurements, etc. This information can be used to either directly recognize a place by associating it to the sensory input, or by converting it to a spatial (possibly metric) map of the environment (Meyer & Filliat, 2003a).

These methods are often used in conjunction. The reason is that they both have serious, but complementary, drawbacks. Since idiothetic data is constantly summed it is subject to cumulative error, growing over time. On the contrary, the allothetic information is constant over time but suffers from perceptual aliasing, i.e., two distinct locations are possibly registered as the same. Therefore, one common method to compensate for these problems is to roughly estimate the position using odometry and then use landmark detection to compute the precise position. Landmarks are either natural or artificial and have a fixed and known location. (Meyer & Filliat, 2003a; Borenstein et al., 1997).

Generally you can divide idiothetic techniques for self-localizing into odometry (wheel rotations) and inertial navigation (gyroscopes and accelerators) (Borenstein et al., 1997). Examples of allothetic techniques are triangulation and trilateration (such as GPS and other active beacon systems) and map-based positioning. The latter matches allothetic information with a stored map. This method is the most difficult and demanding one since it requires high accuracy of the sensors and enough distinguishable features in the environment.

1.1.2. Map-Learning

To answer the question "Where are other places with respect to me?" the robot needs to use its sensory inputs to obtain and maintain a single global representation of the environment (Franz & Mallot, 2000). The difficulties that arise when the robot does not already have an available, or only a partial, map of its environment can be divided into two areas (Meyer & Filliat, 2003b).

First, when the robot has no a priori knowledge of its surroundings, it has to build a map and localize itself simultaneously, also referred to as *SLAM* - Simultaneously Localization And Mapping (Meyer & Filliat, 2003a). This is problematic, since every error made in the localization during map-learning gets incorporated in the resulting map. Some methods deal with this by allowing the map to be corrected afterwards if inconsistencies are found, others do not.

Secondly, when no or only a partial map exists, the robot has to take into account that the place could either be an already known place, or a totally new one. This evaluation step is one of the most crucial for map-learning. One way to solve this problem is to treat every conclusion that the robot makes about its surrounding and position as a hypothesis, allowing it to be reevaluated as additional features of the environment are found. Methods that use only one or multiple concurrent hypotheses exist (Meyer & Filliat, 2003a; Meyer & Filliat, 2003b).

There is a distinction between two types of representations: metric and topological maps (Meyer & Filliat, 2003a, 2003b). In metric maps, coordinations of objects in the environment, mainly obstacles, are stored in a common reference frame. Topological maps, on the other hand, store (allothetic) definitions of different places along with information on how to get from one defined place to another. These places do not necessarily correspond to the real spatial layout of the environment.

The simplest way of building a metric map is called an *incremental scheme*. It estimates the position of the robot and subsequently adds features to the map around its position using allothetic sensor data. The occupancy grid is an example of this procedure (Meyer and Filliat, 2003b). This map consists of a grid where each square is either filled, meaning an obstacle has been detected there, or empty.

The procedure of building topological maps seems easier. It consists of recording the actions that the robot performs to get to one place, and then store a unique definition of that place. Whenever a new place has been reached it is added to the map. However, to determine whether a place is new, or already plotted in the map, is, as mentioned, the difficult part (Meyer and Filliat, 2003b).

Topological maps are especially beneficential when it comes to path-planning, since it can be used directly to find different paths to a certain goal. Despite the fact that metric maps can be more complicated to build and require a metric sensor model, they have the advantage of being transferable between different robots and are

comprehendible by humans (Franz & Mallot, 2000; Meyer and Filliat, 2003a).

For the prototype in this study, however, a metric map was chosen. A metric map is best suited in this case since the sensors will provide measures in a metric model and because it is going to be used to estimate a path into regions where the vehicle has not been before. Also, in this context it is likely that the human driver wants to be able to see and verify the internal map that the system is using to locate the parking place.

1.1.3. Path-Planning

Path-planning is, in contrast to localization and map-learning, a rather independent task that takes place once the position of the robot on a map has been estimated. It answers the question "How do I get to other places from here?", i.e., it consists of a sequence of actions to be taken to reach a certain goal (Meyer & Filliat, 2003b).

One critical aspect of path planning is to integrate high-level reasoning with low-level execution and control. Furthermore, both localization and map-learning often have a lot of inherent uncertainty (Saffiotti, 2001). To deal with these difficulties Saffiotti (2001) propose the use of fuzzy logic. This technique has the intrinsic property of combining numeric, low-level, and symbolic, high-level, reasoning. Since it also provides the ability to represent and distinguish between different types of uncertainty, such as vagueness and inaccuracy, it makes a favorable addition to traditional tools (Saffiotti, 2001).

Franz and Mallot (2000) present a seven level hierarchy of navigation behaviors according to its complexity. The first four levels are so called local navigation behaviors which means that only one place needs to be recognized, namely the goal. It is shown that many biologically inspired navigation techniques do not require the existence of a map.

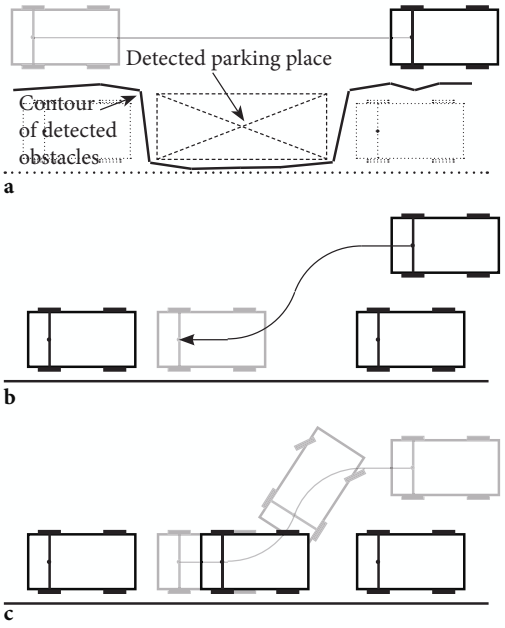


Figure 1.1. The three succeeding steps of parallel parking:
a) Step 1: Localize a free parking place
b) Step 2: Plan an appropriate path
c) Step 3: Execute the parking maneuver

The problem of autonomous parking constitutes a good example of what Franz and Mallot (2000) call local navigation, since only the goal, i.e., the parking place, is of relevance to the path-planner.

1.2. The Task of Autonomous Parking

The three questions of navigation also relate to the task of autonomous parking. The vehicle has to know where it is, where other vehicles are located in relation to itself and how to get to a free parking place from its current position. Autonomous parking is generally divided into three succeeding steps (Jiang & Seneviratn, 1999; Laugier et al., 1999; Lo et al., 2003), as seen in figure 1.1.

The first step, to localize a free parking place, involves a simple version of SLAM, i.e., simultaneous localization and map-learning. During the next step, a path-planning procedure takes place where a trajectory from a starting point to the final position, i.e., in the parking place, is computed. The third step simply executes the maneuver according to this plan.

The task of autonomous parking is usually performed by a combination of planned and reactive actions (Gomez-Bravo et al., 2004, Holve & Protzel, 1996, Laugier et al., 1999).

1.2.1. Step 1: Localize a Free Parking Place

During this phase, the vehicle acquires its knowledge of the local environment. The robot rarely has a complete map of the environment beforehand, and in this case it does not need one. All it has to know is the shape of the contour formed by other parked cars and its own position in relation to a free parking place (Holve & Protzel, 1996), see figure 1.1 a.

This is usually accomplished by driving straight forward, under the presumption that a possibly free parking place is somewhere on the right or left side further on. At the same time the distance covered and the distances to the obstacles along one of the sides are measured (Holve & Protzel, 1996; Laugier et al., 1999; Jiang & Seneviratn, 1999). Odometry and other range-measuring techniques such as ultrasonic or laser sensors are common but more sophisticated visual systems are also used (Chao et al., 2005).

This step is in most cases performed by a combination of simple reactive behaviors such as wall-following and obstacle avoidance (Gomez-Bravo et al., 2004).

1.2.2. Step 2: Plan an Appropriate Path

Perhaps the most complicated task in the problem of autonomous parking is the path-planning. This is due to the fact that a car-like vehicle (i.e., with two rear wheels and two directional front wheels) is affected by three types of constraints (Laugier et al., 1999):

- Kinematic constraints: Every car-like vehicle is subject to kinematic constraints called nonholonomic, i.e., the geometric shape of its motion is restricted. The motion can only be performed in one direction perpendicular to its rear wheel axis and its turning radius is lower-bounded.
- Dynamic constraints: The acceleration and velocity of the vehicle are restricted depending on the capabilities of its actuators (engine power, steering-servo, breaking force, etc.).
- No collision constraints: It is, of course, strictly forbidden to collide with any of the already parked cars or other, possibly moving, objects in the surrounding.

There exist different, more or less elaborate, approaches to the path-planning process of non-holonomic vehicles. The dynamic constraints are often omitted in the motion planner, and all the mathematical models mentioned here presume that no slippage occurs. A common approach to calculate the parking maneuver is to start from the final position (i.e., the parked vehicle) and then create a feasible path to get to the starting point. This maneuver then can be reversed in order to park the vehicle (Gomez-Bravo et al., 2001; Lo et al., 2003).

Lo *et al.* (2003) use the simple principle of forming circular arcs and straight lines to determine the parking maneuver, see figure 1.1 b. The position where two segments are connected is the turning point when the vehicle changes its steering angle. Since this method requires that the vehicle stops every time the steering angle should be changed, it is not appropriate when a continuous curvature profile is desired (Laugier et al., 1999). Laumound *et al.* (1994) present a motion planner that solves the problem of nonholonomic motion by recursively plan subdivisions of the path, disregarding the constraints. This technique is claimed to be both fast and to result in near-minimal length trajectories. Many of the path-planners based on geometry are influenced by the work of Murray and Sastry (1993) which presents a nonholonomic motion planner based on sinusoids.

Since every trajectory is dependent on its starting position, this is the first variable that has to be determined. In the solution suggested by Gomez-Bravo *et al.* (2004), the lateral displacement of the vehicle during the parking maneuver is determined by the starting point. Therefore, this point is first elected depending on environmental data by the use of fuzzy logic. Once the starting point is reached, it uses a mathematical model similar to the ones above to calculate the maneuver.

In order to guarantee a strictly collision free trajectory, Jiang and Seneviratne (1999) present a motion planning algorithm based on the concept of a forbidden area. The forbidden area is defined by the areas where the appearance of a certain reference point on the vehicle would result in collisions. By first identifying this area the planning of a guaranteed collision free path is facilitated.

Alternatively, the maneuver of the vehicle can be performed using only reactive behavior based on heuristics (Gomez-Bravo et al., 2004; Holve & Protzel, 1996). These behaviors are often implemented by fuzzy logic that consists of rules created out of knowledge from human drivers (Gomez-Bravo et al., 2001; Holve & Protzel, 1996).

1.2.3. Step 3: Execute the Parking Maneuver

At this point, all left to do is to execute the parking maneuver. This is done by following the trajectory given from the last step, see figure 1.1c.

In some cases, reactive obstacle avoidance is carried out during this phase as well. In the implementation presented by Laugier *et al.* (1999) the vehicle is even able to notice when the car in front of the parking place moves. If the new size of the parking place is still big enough, the parking maneuver is adjusted and carried on, otherwise it aborts the parking.

Since the planned trajectory in some cases are very exact, and the data from both odometry and ultrasonic sensors are known to be very imprecise, the behavior during this step is often aided by fuzzy-logic (Chao et al., 2005; Holve & Protzel, 1996; Gomez-Bravo et al., 2001).

Some of the models support multiple iterations of the parking maneuver to place the vehicle deeper inside the parking place (Holve & Protzel, 1996; Jiang & Seneviratne, 1999; Laugier et al., 1999). This allows the vehicle to park in smaller parking places.

1.3. Aim of the Project

Biological research shows us that animals, including humans, do not answer all the three questions related to navigation. To navigate, one really has to know only one thing: "How do I reach the goal?" (Franz & Mallot, 2000). This shows that you actually can achieve better results by knowing less, the thing that matters the most is that you know the *right* things.

My belief is that the task of autonomous parking is simple enough to be performed without the need for complex planning and precise computations that requires very accurate and therefore often complicated and expensive technology. Instead, a system consisting of a few robust and well designed behaviors implemented with simple and cheap technology should be able to perform the tasks just as good as more complex solutions. In other words, I want to try to solve the problem by keeping the simple things simple, by knowing less but knowing only the right things.

To achieve this, the presented solution will try to reduce the complexity of the task in at least three ways. The first way to reduce the complexity is to avoid the navigation problems that arise if you strive to build a vehicle that is able to navigate autonomously along the street. This is often done during the first step of the task when the vehicle localizes the parking place. However, the problem of finding a free parking place is not part of

this task, that is a completely different problem. The field of robotic navigation has yet to mature enough to be able to safely handle the complex situation of driving along a trafficked street.

Solutions that claim to find the free parking place autonomously also have to handle all the difficulties involved when driving on a trafficked street. This implies the capability to navigate in an unknown and dynamic environment with the presence of other moving obstacles such as cars, bikes and even people. In many cases, the first phase of the task, to locate the free parking place, is not very well described and seems to be achieved merely by simple wall-following and obstacle avoiding behaviors (Lo et al., 2003; Holve & Protzel, 1996; Gomez et al., 2004). These behaviors are probably good enough in a laboratory environment but would be both dangerous and frustrating in a real traffic situation.

Therefore, this solution will not aim to autonomously *find* a parking place, only to autonomously *park* the vehicle once a free parking place has been found. This is the part that human drivers find difficult and where computers can be of help.

The second way to reduce the complexity of the task is not to depend on a constantly updated metric map. The only time the vehicle will need an internal metric map is when the path-planning occurs, i.e., just before the actual parking maneuver begins. Also, the map does not even have to hold information about anything else but the goal, which, in this case, is the position of the selected parking place. Therefore, the only time a map will be generated and used is after the parking place has been found and when the path is being planned. After that, the map can be forgotten.

The third way to reduce the complexity is to use sensors that are simple, cheap and easy to make us of. Basic sensors for range-measurements should provide information enough to create the necessary knowledge of the surroundings. These sensors can be made with well known technologies, such as infrared light or ultrasonic sound, and are even already available on todays cars with back-sensors.

More advanced systems, like camera based vision for example, require additional image processing which enhance the overall complexity and cost of the solution. The only advantage with this kind of technology is that the parking place can be located without scanning it during passing, which, in most cases, are necessary anyway if you do reverse parking (Chao et al., 2005). Therefore, all knowledge and implemented behaviors of the robotic vehicle will depend on raw range measurement data.

2. Materials and Method

2.1. Technology

The hardware used in the project is mainly the Lego Mindstorms robotic kit complemented with high precision distance sensors from Sharp. The kit includes all the necessary hardware for developing robots and provides an easy and fast way to do quick prototyping.

Apart from the Lego bricks used to construct the vehicle itself, the hardware used in this study is:

- 2 x Lego servo motors with
  built in rotation sensor
  Accuracy: +/- 1 degree
  Power: 9 V
  Torque: 16.7 Ncm
  Speed: 146 rpm

- 2 x Sharp infrared distance sensor
  Range: 4-30 cm
  Accuracy: +/- 1 mm
- 1 x Sharp infrared distance sensor
  Range: 10-80 cm
  Accuracy: +/- 1 mm

- 1 x Lego NXT-Brick onboard computer
  Powered by 6 AA batteries
  32 bit CPU, 256 Kb Flash 64 Kb Ram
  3 Motor ports
  4 Sensor ports
  100 x 64 pixel LCD
  Loudspeaker
  Bluetooth 2.0
  USB

The assembling of the hardware is, thanks to the Lego technology, very easily done. The Lego bricks makes it possible to construct and design the vehicle pretty much how you want, and the electronic parts can be connected without any configuration. No further development of the hardware has been made, except for the substitution of the ultrasonic sensors included in the kit by infrared sensors from Sharp. The infrared sensors provides about 30 times higher precision and can be read twice as often compared to the ultrasonic sensors.

The main part of the control software is run on a laptop computer and communicates with the NXT-Brick using a wireless bluetooth connection. The software is developed with Microsoft´s .Net 2.0 framework in C#. Programming was done in Visual Studio 2005.

The on-board software is developed in Lego's own programming language called NXT-G and runs on the virtual machine on the original firmware.

2.2. Solution Design

In this study the hypothesis is that it is possible to solve the task of autonomous parking in a simplified and therefore more robust way compared to other implementations. To solve the task, the vehicle still has to perform the three subtasks: localization, planning and execution, see figure 1.1.

2.2.1. The Localization Step

The vehicle should not be able to *autonomously find* a free parking place. It must, however, be able to *recognize* one once it is found by the driver.

When the driver knows that he or she is about to approach a suitable parking place, the Autonomous Parking Control System (APCS) is activated by the press of a button. The vehicle then starts "looking" for a free parking while the driver keeps driving. Thus, the localization step is performed by totally passive behaviors that simply register the movement of the vehicle and the readings from the sensors.

A found parking place is defined by the vehicle as a sustained range measure above a certain threshold on the right side of the car (in countries where the traffic is on the right side) for a specific length. The vehicle does not make any other assumptions than that a parking place is expected somewhere on the right side and that the driver keeps the vehicle somewhat parallel to the road and the parking place.

Once the above criteria has been met, the vehicle notifies the driver that a free parking place has been located. From that point on it is possible for the driver to pass over the control to the vehicle with a second push on the APCS-activation button. Then the driver can let go of the wheel and sit back, while the vehicle performs the parking maneuver, see figure 1.1 b.

If that particular parking place was not the intended one, the driver just keeps on driving. The found parking place is "forgotten" after a certain distance and the vehicle continues to search for a parking place. If a new parking place is found the system notifies the driver again.

2.2.2. The Planning Step

Once a parking place has been found and the driver has commanded the vehicle to park, the path-planning starts. This step can be divided in three successive steps.

The first step is the making of a simplified metric map. The data recorded during the localization step is recollected from the log. Only the most recent data, beginning just before the last found parking place, is needed. The data is then processed by an algorithm that creates four points connected by lines that constitute the contour of the parking place, see figure 2.6.

In the second step the created map is used to estimate an appropriate path from the current location to a position where the vehicle is parked. The path should consist of a series of segments that are either straight lines or curved arcs, see figure 2.7a and 2.7b.

The third step consists of translating the computed path to a series of motor commands, an execution sequence.

2.2.3. The Execution Step

Once the path is planned, and the execution sequence is computed, the parking maneuver can begin, see figure 1.1 c. The movement is constantly monitored and controlled to avoid collisions. If unexpected obstacles are met, the execution is stopped. When finished, the car is fully parked and the driver can leave the vehicle.

2.3. The Vehicle Model

The kinematics of the vehicle built in this study can be described by the graph in figure 2.1. See table 2.1 for a description of the parameters and used measures.

The reference point $\mathbf{p}_0$, located in the centre of the rear axis, describes the position of the vehicle. Every time the vehicle has moved a distance, s, the coordinates x and y and the direction Λ, of $\mathbf{p}_0$ are updated according to the equations below. The steering angle, α, denotes the effective steering angle, i.e., the angle calculated from the measured turning radius.

One should also note that because of the way the steering is constructed on APRIL, the steering angle slightly affects the overall direction of the vehicle, as shown in figure 2.2. When the front wheels are steered in one direction, the front axis centre point gets pushed in the opposite direction. The offset was experimentally fitted to a linear function of the steering angle and was taken into account when the vehicles position was updated, as seen in equation (1).

$$\Lambda_{diff} \approx 0.0332\alpha \tag{1}$$

$$r = \frac{l}{\tan(\alpha)} \tag{2}$$

$$\Delta\Lambda = \frac{s}{r} = \frac{s\tan(\alpha)}{l} \tag{3}$$

$$\Delta x = s\cos(\Lambda + \Lambda_{diff}) \tag{4}$$

$$\Delta y = s\sin(\Lambda + \Lambda_{diff}) \tag{5}$$

Since the accuracy of the equations are dependent on that s is kept small in comparison to r, the update of the

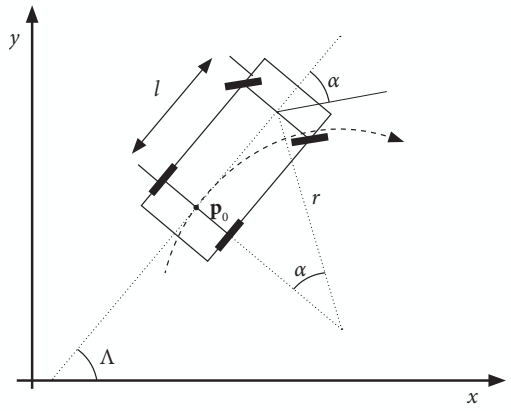


Figure 2.1. The vehicle model

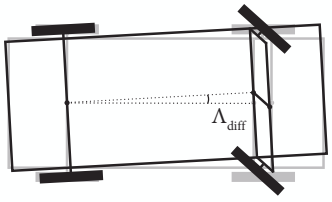


Figure 2.2. The construction of the vehicle's steering and its effect on the vehicle's direction

Parameter	Symbol	Value
Width of the car	W	165 mm
Length of the car	L	420 mm
Wheelbase	l	248 mm
Track width	w	145 mm
Maximum steering angle	α_{max}	30°
Minimum steering angle	α_{min}	–30°
Minimum turning radius	r_{min}	430 mm
Minimum parking width	p	180 mm
Minimum parking length	d	520 mm

Table 2.1. Parameter specifications

vehicles position is done continuously whenever the vehicle is moving. The equations are valid for slow movement with no significant slippage.

It is also worth mentioning that since steering to the right denotes a positive steering angle, and vice versa, the y-axis of the coordinate plane is inverted. This primarily affects the sign of the coordinates in the Cartesian plane that constitutes the vehicle's internal map, and can be seen on some of the printouts in the experimental results in chapter 3.

To conform with real cars, the prototype has been built in the scale 1:11 of the typical car shown in figure 2.3. A model world consisting of a street with parking places on the right side of the road has been built in the same scale. The specifications of the typical parking situation that the world model is based on is shown in figure 2.4, with measures shown in the scale 1:1.

For a technical drawing of the prototype, see Appendix I.

2.4. The Software Architecture

The software is divided in three components: The Autonomous Parking Component, the Vehicle Component and the Onboard Component. See the UML model in Appendix II.

In addition to these components, the software also contains a graphical user interface, not included in the UML diagram. The interface allows the user to see an illustration of the vehicle's internal map with the contingent parking place and planned parking maneuver marked down on it. It also contains controls to manually maneuver the vehicle, either in the precise way, where the exact distance or steering angle is sent to the vehicle, or by simply pressing a button, making it move in one direction until one releases the button. A screenshot of the user interface can be seen in Appendix III.

The Autonomous Parking Component contains the highest level of logic to solve the task of autonomous parking. It uses the functionality implemented in the Vehicle Component to control the vehicle. The Vehicle Component, in turn, communicates with the OnBoard Component which implements the most basic level of control of the vehicle.

Length	4.9 m
Width	1.8 m
Height	1.5 m
Wheelbase	2.8 m
Turning radius	5.5 m

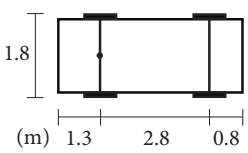


Figure 2.3. Specifications of a typical car (Vägverket, 2004a)

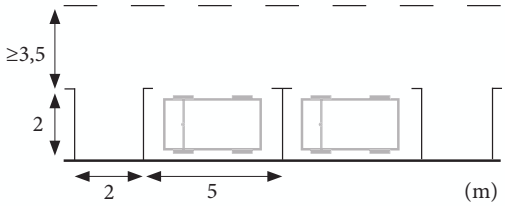


Figure 2.4. Specifications of a typical parallel parking situation (Vägverket, 2004b)

2.4.1. Autonomous Parking Component

The autonomous parking component implements the high level control of the vehicle that provides all the functionality needed to solve the task of autonomous parking. It consists of several subcomponents that collaborate and bring different parts to the solution.

Maybe the most important subcomponent, called 'Slammer' after the term Simultaneous Localization And Mapping, is constantly registering any movement made by the vehicle and updating its position accordingly. It also stores the sensor data associated with every new position of the vehicle. This information is maintained and stored in a data structure called 'Map'. The map is also capable of providing a graphical representation of the stored information used by the graphical user interface.

The three components 'ParkingPlaceFinder', 'Planner' and 'Executer' implement the mechanisms that perform the three distinguished steps of autonomous parking described in the solution design (section 2.2).

The 'ParkingPlaceFinder' is, when activated, constantly looping through the map, looking for a free parking place. Every time a new parking place has been found it notifies the rest of the system. 'ParkingPlace-Finder' produces a data structure called 'ParkingPlace' which in turn consists of four points and some metadata about the parking place.

This information, along with the current position of the vehicle, provided by the 'Slammer', is the only information needed by the 'Planner'. The 'Planner' produces, like its name implies, a plan for the vehicle to reach a parked state inside the parking place. The produced output is stored in a data structure called 'Plan' which is a list of path-segments. Every path-segment has information about its length, direction and steering angle.

The 'Executor' then uses this list of path-segments and executes them one by one. During the execution it monitors the process and warns the rest of the system if anything goes wrong.

The 'CAS', Collision Avoidance System, monitors the movement of the vehicle and the readings of the range sensors. If a collision is about to occur, this component alerts the rest of the application and puts the brake on the engine.

The algorithms used in the 'ParkingPlaceFinder' and the 'Planner' are further described in section 2.5 and 2.6.

2.4.2. Vehicle Component

The vehicle component contains a model of the hardware used by the prototype along with the associated functionality to control it. The most important class in this component, the 'Vehicle' class, is the main hub for controlling the prototype. All subcomponents are attached to this class. It also initiates the bluetooth connection with the vehicle and communicates with the OnBoard Component.

Since even the motor has built in rotation sensors, called 'TachoMeter', all parts in APRIL can be derived from the same 'Sensor' class. This class provides the basic logic of providing a single value that is constantly updated. It also keeps track of whether this value is changing and if so, if it is increasing or decreasing. APRIL supports two types of 'Range-Sensor' namely IRSensors, infrared, or USSensors, ultrasonic, although it only makes use of the infrared ones.

Additionally there are two types of motors, 'Engine', which makes the vehicle go forward or backwards, and the 'Steering' which controls the steering of the vehicle.

A wrapper class called 'NxtCommunicator' provides the low level logic to send the actual commands to the NXT-Brick over the bluetooth protocol. This is an implementation of the Lego Bluetooth API in C#, called NXT# (Fokke, 2008).

The internal communication in the Vehicle Component is primarily event-driven. The 'Vehicle' class listens to data sent from the NXT-Brick (called state reports) and announces, i.e., triggers an event, whenever new information is available (which is about once every 50 ms). The different sensors then update their own value and, if any changes was made, in turn announce other subcomponents that their value has been changed. The 'Slammer', for example, listens for the event 'Moved' sent by the 'Vehicle' class, and updates the map whenever this happens. The 'Moved' event, triggered by the 'Motor' is in turn inherited by the 'ValueChanged' event in the 'Sensor' class.

The 'Motor' can control the motor in two ways. Either by sending a single command to the NXT-Brick simply telling it to start or stop one of the motors. This method also provides the possibility to control the speed of the motor but has the disadvantage that the motor keeps moving until another command stops it. Since there is a delay in the information sent to and from the vehicle, this method results in imprecise movements and is only used when the vehicle is manually controlled.

The other way to control the motors is to send a message to the mechanism implemented by the OnBoard Component to regulate movements described below.

2.4.3. OnBoard Component

Because of the small delay caused by the bluetooth link, about 50 ms per call, some processes that need to be executed in close interaction with the hardware has to be run on the onboard computer. The 'Vehicle Component' run on the laptop therefore interacts with a smaller, low-level program run on the NXT-Brick, called the 'OnBoard Component'.

This component is mainly responsible for regularly collect and report information about the state of the vehicle (i.e., information from the engine, steering and range sensors), so called state reports. This makes it possible to have the information pushed to the main software on the laptop actively, rather than passively being pulled. By not first having to ask the vehicle questions about each of the motors and sensors individually, where each question takes time, a huge amount of time is saved. In this way information about the state of all the vehicle's sensors and motors can be sent to the control software in half the time it takes to ask for one of these. The temporal resolution of the system, i.e., how often information from the sensors is updated, is essential to the accuracy of the map and the decisions made by the program.

Since on-board execution also provides readings from the rotation sensors in the motors in real time, precise regulation of motor movement can be achieved. This function accepts commands from the laptop where an exact length of a movement is provided. The movement can then be executed with a precision of about +/- 1 degree, where 360 degrees represent one full rotation of the motor axis. The actual precision of the motion of the vehicle is, of course, affected by numerous factors making it less precise.

2.5. The Parking Place Finder Algorithm

This algorithm is responsible for ending up with a result similar to the one in figure 2.5. It has to work in four different situations. First, it should of course be able to locate an empty space beside the vehicle when no other cars are parked along the street. In the second and third situation, it should be able identify a free parking place located right before or after another parked car. The fourth situation, where the system probably is of most help, is when the parking place is located in between two other parked cars.

Generally it consists of looping through all the range measurements stored in the map in the same order they were stored. The algorithm compares all range measurement with a minimum value, usually the width

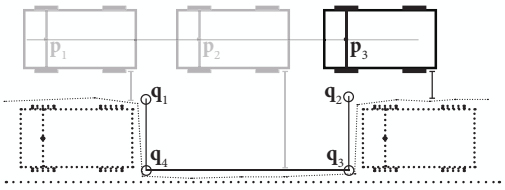


Figure 2.5. Simplified metric map

of a parking place. If a measure greater than that is found, a temporary, zero-length, parking place is created. The parking place is then gradually extended as long as the next range measurement also exceeds the minimum allowed measure ($\mathbf{p}_2$ in figure 2.5). Eventually, if the parking place reaches the minimum allowed length, the real parking place is found. The parking place keeps being extended further and further as long as the minimum allowed distance is measured. When a distance that is too short is measured, the parking place stops being updated ($\mathbf{p}_3$ in figure 2.5).

To judge whether the parking place is located right next to the vehicle or behind the outer edge of a line of parked cars (as in figure 2.5), the minimum value to be found by the range sensor is adjusted. If obstacles in a certain, relatively close range are detected, they are treated as other parked cars ($\mathbf{p}_1$ in figure 2.5). The distance to such objects are taken into account when the minimum value that has to be found is calculated. For example, at position $\mathbf{p}_2$ in figure 2.5, the distance to an obstacle must exceed the distance measured in $\mathbf{p}_1$ plus the width of a parking place.

Otherwise, if no obstacles are detected, the minimum value is the width of a parking place.

Since the vehicle always strives to park along the outer edge of the parking places, the found parking places always have the width of the narrowest allowed parking place.

To increase the accuracy of the algorithm, different kinds of filtering are applied to the sensor readings. For example, when the distance to the line of other parked cars are estimated, a basic median filter is used, and to better identify the edges of a parking place, big angles are treated as corners. Another technique used to better fit the position of a found parking place to the real parking place is to, if applicable, position it exactly between the edges of a gap in the contour of other parked cars.

2.6. The Path-Planning Algorithm

APRIL makes use of two different strategies when it is planning the parking maneuver. Depending on the length of the found parking place, it chooses the strategy best suited for the current situation. The goal position of the vehicle, in both algorithms, is in the center of the parking place.

Strategy A, the most intuitive, is simple to compute but only suitable for long parking places. Strategy B, on the other hand, is especially good for shorter parking places.

Both algorithms assume that the vehicle is somewhat parallel with the parking place. Therefore, if the angle relative to the parking place exceeds a threshold of three degrees, the first step is to adjust the direction of the vehicle. This is done by reversing while turning in the appropriate direction. It is not likely to exist any obstacles behind the vehicle since it is the way the vehicle just came.

Both algorithms then center the parking place in the coordinate plane to simplify some of the equations. In the transformed coordinate plane, $\mathbf{q}_4$ in figure 2.6 serves as origo.

2.6.1. Parking Strategy A

This algorithm, illustrated in figure 2.6a, first calculates the total length the vehicle has to be moved in the x- and y-plane, Δx and Δy respectively, i.e., between $\mathbf{p}0$ and $\mathbf{p}1$.

It makes use of two circles, $\mathbf{C}_1$ and $\mathbf{C}_2$ with radius r equal to the minimum turning radius, to form two arcs (segment 2 and 3). The height of the arcs are fitted to $\frac{1}{2}\Delta y$ each. Since the height of one arc can never be more than r, a straight segment between the two arcs is added to make up the difference if Δy is greater than $2r$ (not shown in figure).

The summed width of the arcs are then calculated. The difference between this length and Δx is added as a straight segment in the beginning of the path (segment 1).

Last a final segment is added to centre the vehicle in the parking place (segment 4).

This strategy is not iterative and has always at least 3, and at most 5, segments, depending on the initial position of the vehicle.

2.6.2. Parking Strategy B

The second parking strategy was developed to deal with tighter spots. While the algorithm for strategy A is fairly static, with a known range of segments, strategy B has an unlimited, and in beforehand unknown, number of segments. An example of a planned parking maneuver using strategy B is shown in figure 2.6b.

The algorithm actually starts from the end, with the vehicle in its goal position, perfectly parked inside the parking place. From there it begins by, in theory, moving the vehicle backwards while turning maximum to the right, to make the vehicle aim out of the parking place

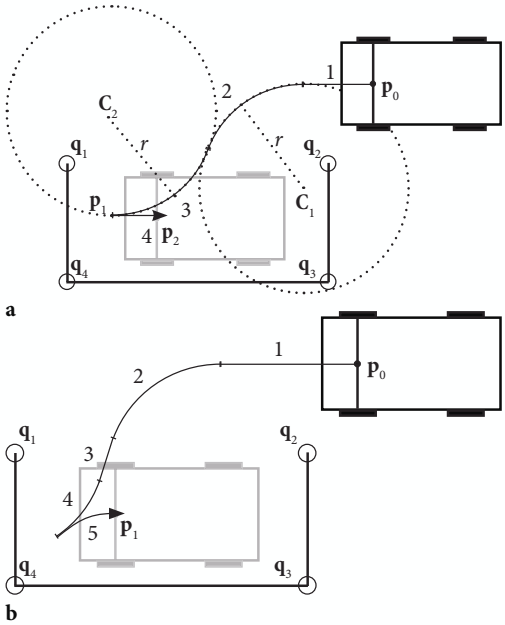


Figure 2.6. Illustration of parking strategy A (a) and parking strategy B (b)

(segment 5). When one of the corners of the vehicle gets too close to one of the edges of the parking place, the motion is reversed and the steering angle turned in the other direction (segment 4). This procedure is repeated until the vehicle is able to drive straight out of the parking place without hitting any of the corners of the parking place (the path in figure 2.6b consists of two iterations). The critical corner in this case is the corner named $\mathbf{q}_2$ in figure 2.6b. Each of these iterations constitutes individual, curved segments (segments 4 and 5). The shorter parking place, the more segments need to be made.

When this, first (or last) part of the algorithm is completed, the procedure is similar to the one in strategy A. First, the total length the vehicle has to be moved in the x- and y-plane, Δx and Δy respectively, to get to the starting point, is calculated. Then, the width and height of the arc that is necessary to turn the vehicle in the same direction as the road again, is calculated. The next segment consists of the remaining height that is left after you subtract the total height, Δy, with the height of the arc (segment 3). After that you add the curved segment that constitutes the arc itself (segment 2). The length of the last segment (segment 1) is calculated by subtracting the sum of the movement in the x-plane during the last two segments (segments 2 and 3) from Δx.

Finally, the list of these segments is reversed to form the resulting plan of the parking maneuver.

2.7. Experimental design

To test the success of the proposed solution, a number of experiments to measure the performance of APRIL and its parking algorithms, were constructed. Five different tests were made to test different aspects of the solution. Three of these tests are conducted empirically, and two of them are simulated.

First, simulated tests are made to measure the efficiency of the path-planning algorithms. The test are performed by letting the algorithms compute parking maneuvers in different parking situations. The situations are varied by the distance from the vehicle to the contour of other parked cars, and by the length of the parking place. The length varies between 420 mm, which is the length of the vehicle and therefore the theoretical limit of shortest possible parking place, to 1000 mm.

The tests measure the numbers of segments in every possible path, as well as the total length of the proposed parking maneuver. The two algorithms are tested individually, i.e., the vehicle uses one of the algorithms, regardless of length, to allow them to be compared.

Then the actual prototype itself is tested in three ways. First, the time it takes for APRIL to execute a parking maneuver in relation to the length of the parking place is measured for eleven parking places with varying lengths. The time it takes to manually park the vehicle in the same parking places is also measured for reference.

Secondly, the accuracy of APRIL's internal map and its movements are measured by a practical test. The test is performed by starting at a specific position (0,0), go

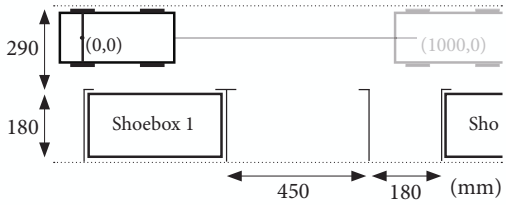


Figure 2.7. Parking situation in the experimental setup

forward a certain distance while looking for a parking place, perform a parking maneuver, return to traffic and finally reverse the same distance as in the beginning. After this procedure is done, the vehicle should have returned to its exact starting position, and the deviation in x- and y-plane can be measured. The test is conducted ten times to get an average measure of the vehicles accuracy.

Finally, a test measuring the reliability of the prototype is performed. This is simply achieved by letting APRIL find and park in ten different parking places, with varying lengths, and count the number of times it hits anything.

The practical tests are all made on the model street illustrated in figure 2.7. White shoe boxes are placed inside the other parking places to represent parked cars. The white color is chosen because it improves the accuracy of the infrared sensors. The starting point (at coordinate 0,0) and the ending point (at coordinate 1000,0) of the search path, as well as the first shoebox (shoebox 1), has fixed positions and remains the same during all experiments. The position of the second shoebox (shoebox 2) is moved from -50 to +50 millimeters, ten millimeters at a time, in relation to the reference position depicted in the figure. The measures are chosen so that strategy A is chosen by the planner whenever the found parking place has a length equal to, or longer, than the one in the figure, i.e., 630 millimeters. If the found parking place is conceived as shorter, strategy B is used. In this way, the vehicle is supposed to use each strategy about the same number of times during the experiments.

3. Experimental Results

See appendix IV for printouts of the internal map generated during the eleven trials.

3.1. The Path-Planning Algorithms

The graph in figure 3.1 shows the number of segments produced by the two parking path-planning algorithms. Only collision free paths are included in the graph. A low number of segments are desired since it results in maneuvers with few stops. Each time the vehicle has to change the steering angle or direction, i.e., for each segment of a plan, it stops.

The graph is used to determine the minimum allowed length of a parking place, as well as to determine when to use the one strategy instead of the other.

As one can see, strategy A has a flat curve, with only four segments regardless the length of the parking place.

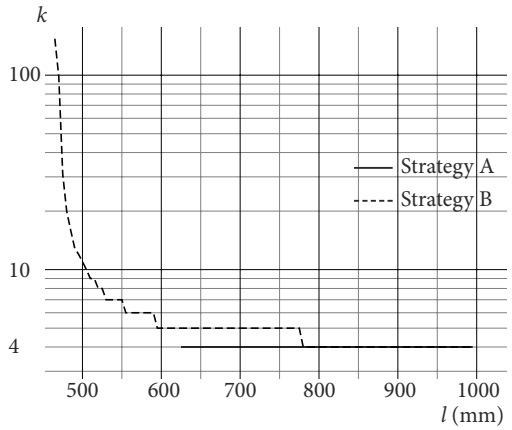


Figure 3.1. The number of segments for the planned parking maneuver k as a function of the length of the parking place l (mm). Only collision free paths are included in the graph.

Its disadvantage though, is that it only provides feasible paths for parking places that are at least 625 millimeter long.

The number of segments produced by strategy B, on the other hand, are very high for short parking places, and then drastically decreased at around 450–500 millimeters. Even though it is theoretically possible to use strategy B to park in parking places as short as 465 millimeters, the high number of segments make the plans not practicable until the length is about at least 500 millimeters. For a 500 millimeters long parking place the number of planned segments are eleven, which is quite many, but still considered acceptable. This length is therefore suitable as the shortest allowed parking place. At about 600 millimeters the curve evens out at about the same level as strategy A with 6-4 segments.

The graph in figure 3.2 shows the length of the paths produced by the two path-planning algorithms. Only paths that are collision free are included in the graph. The path were computed immediately after the vehicle found the parking places.

The graph shows that the curves for the two different strategies are divergent. While strategy A is producing shorter paths as the length of the parking place is

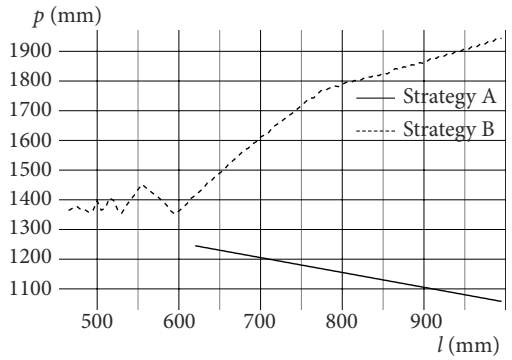


Figure 3.2. The length of the planned parking maneuver p (mm) as a function of the length of the parking place l (mm). Only collision free paths are included in the graph.

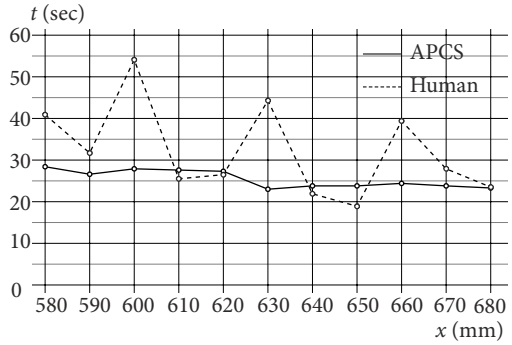


Figure 3.3. The time t (seconds) it takes to execute the parking maneuver as a function of the length of the parking place l (mm).

increasing, strategy B is doing the opposite. This can be explained by the fact that when strategy A is used, the vehicle often has to go a bit past the parking place to make room for the parking maneuver. When the parking place is long enough this is not needed, which explains why longer parking places are beneficial for strategy A but not for strategy B.

In sum, the two graphs in figure 3.1 and 3.2 show that a suitable threshold value for the shortest length allowed for a valid parking place is about 500 millimeters, where the number of segments produced by strategy B is acceptable. Strategy A, however, produces shorter paths and is therefore more suitable whenever it is possible. Therefore, strategy A is used for parking places longer than 625 millimeters.

3.2. Performance

The performance of the implemented solution was measured by the time it took the actual prototype to autonomously park in various parking places. For reference, the time it took by a human controller to park the vehicle was also measured. The results are shown in figure 3.3.

As one can see, the time it takes to perform the parking maneuver remains quite constant at 20–30 seconds for the APCS, while it varies quite a lot for the human driver. The curve of the APCS has one, noticeable change in the middle, where the shift between strategy A and B occurs. Strategy A, which was used for parking places longer than 625 millimeter, is slightly faster, about 4 seconds, than strategy B.

The time it took the human controller to park the vehicle was highly affected by whether the maneuver succeeded at the first try or not, i.e., if the position of the vehicle had to be readjusted after entering the parking place. When the final position was reached in one go, the time it took the human driver could even be less than for the APCS. Compared to the APCS, the human has the advantage of being able to adjust the steering angle without stopping, which saves some time. However, this is made up by the APCS's ability to compute and execute a correct maneuver in one try which makes its average performance faster. Although it is not very distinct, the

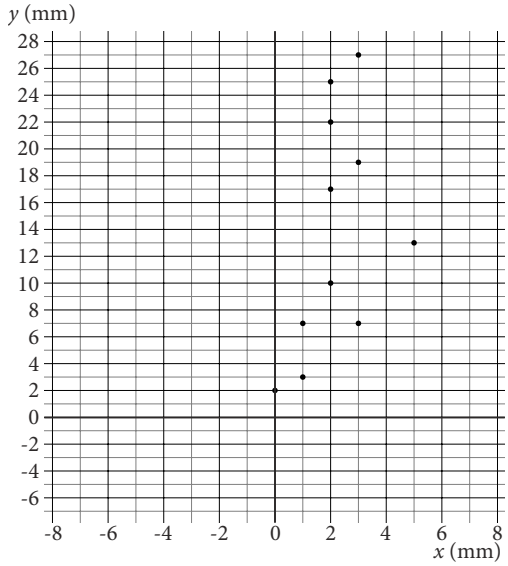


Figure 3.4. The points show the deviation from the estimated and actual position of the vehicle after it has performed a parking maneuver and returned to its initial position for eleven subsequent trials.

human also tends to park in a shorter time for longer parking places.

3.3. Accuracy

The accuracy of the vehicle was measured by letting the vehicle find and park in a parking place and then return to its initial position in eleven different parking situations. The difference between the known deviation from the initial position, i.e., the deviation that the vehicle has estimated and is able to compensate for, and the actual deviation, was measured.

The accuracy of the position estimation, in the *x*- and *y*-axis, of the vehicle is illustrated in figure 3.4. The points in the graph denote the deviation from the known position. The standard deviation is 13.8 millimeters to the right, 2.2 millimeters forward and 0.5 degrees to the right.

As one can see in the graph, the vehicle is considerably less accurate in the *y*-axis, than in the *x*-axis where the deviation never exceeded five millimeters.

3.4. Reliability

To measure the reliability of the prototype, APRIL was observed during the eleven subsequent parking maneuvers performed during the other experiments.

During the experiments, the vehicle never collided with the other parked cars, i.e., the shoeboxes, when controlled by the APCS software. However, when the human controller parked the car, the vehicle collided three times. All three collisions occurred during the last part of the maneuver, when the vehicle was positioned between the two shoeboxes, in the three shortest parking places; 580, 590 and 600 millimeters long.

4. Discussion

4.1. Results

The experimental testings of APRIL show that the parking algorithms are able to compute feasible paths for parking places that are at least 500 millimeters long. Since the vehicle is 420 millimeters long, this means that the prototype is able to park in parking places with a marginal of only 40 millimeters on both ends of the vehicle. In the scale 1:1 this corresponds to a parking place that is 5.5 meters long, and with a margin of 0.3 meters. In the typical parking situation used for reference, the parking place is 7 meters long, with a margin of 1.05 meters on both side of the vehicle (the typical vehicle is 4.9 meters long).

They also show that the two strategies supplement each other and provide an optimal solution when strategy A is used for long parking places and strategy B for shorter ones.

The practical tests of APRIL show that the performance of the system, measured in the time it takes to perform a parking maneuver, is as good as, or better, than a human driver. The main performance benefit comes from the fact that the system is able to compute and execute the exact parking maneuver in one try, whereas human drivers often need to correct the position of the vehicle during the parking maneuver.

During the experiments, the system never caused a collision with any of the obstacles, which gives a rough indication of the systems reliability. The human did, however, cause three collisions during the eleven trials. In a real implementation, however, the human would still need to supervise the parking maneuver. The collision avoidance system in APRIL, called CAS, is only able to detect obstacles in certain angles because of the limited numbers of sensors (three) and does not react quick enough to be reliable if the obstacles are moving.

Finally, the measuring of the accuracy of the vehicle shows that the position estimation of the vehicle is still good even after it has found a parking pace, parked in it and then returned to its initial position. The accuracy of the estimation of the position of the vehicle in the *x*-axis is considerably better than in the *y*-axis. The deviation in the *x*-axis were at the most five millimeters. The deviations in the *y*-axis on the other hand could be as much as 27 millimeters. This is explained by the fact that the precision of the steering is the least accurate, which results in less precise estimations when the vehicle is moving along the *y*-axis, i.e., making turns.

4.2. Discussion

Localization and mapping in the context of autonomous parking is special compared to other types of tasks that involve robotic navigation in two ways. First, you have to get it right immediately. Since the driver does not want to sit in the car waiting while driving back and forth, scanning the surroundings over and over, you have only one chance to get it right. After the initial localization

step you depend on the map you have built and have no possibility to verify it in some way.

Many localization and mapping techniques depend on the fact that you will go back to the same place more than once and thus have the possibility to compare the sensor and odometry measurements (Meyer & Filliat, 2003b). In this way you often have the ability to update and further develop the internal map and estimated position. This also applies to the evaluation and correction of the often quite noisy sensor data. Filtering techniques, like particle filtering, are dependent on visiting the same places more than once, which make them hard to use in the first step of the task where the localization and mapping occurs.

The other thing that makes localization and mapping special in the case of autonomous parking is that it is a really limited area that you need to map and that you only move a short length during the whole task. This makes the odometry less sensible to errors. In general, the most difficult thing about odometry is the cumulative addition of errors. In this case, however, you only move for a certain, rather short, length during the whole maneuver so the errors do not grow large.

To some extent, this outweighs the fact that you only have one chance to map the environment, because this makes the errors made during mapping and localization less important.

In related research, the parking strategies used are often some kind of variation of strategy A. In fact, all the solutions in the related research mentioned here use a path-planning strategy similar to the one in strategy A. None of the reviewed research articles concerning the problem of autonomous parallel parking uses more than one general design of the planned maneuver. Furthermore, I have not found anyone that compares different strategies to each other. In this study, only two strategies have been compared, but it has still proven to be very beneficial to use different strategies for different parking situations.

Strategy A has proven to be more efficient than strategy B for parking places longer than 625 millimeters, both in terms of time and length, but it does not provide collision free paths for shorter ones. Although 625 millimeters correspond to about the same length of a typical parking place in the scale 1:11 (Vägverket, 2004b), there are many cases where the ability to park in shorter parking places are needed. Obviously, the other cars are not always parked exactly where they should be (since they are parked by humans, and humans do make mistakes sometimes, as shown in the experiment described in section 3.4).

The solution used in some implementations (Holve & Protzel, 1996; Jiang & Seneviratne, 1999; Laugier et al., 1999), is to make strategy A iterative, i.e., able to get deeper inside a parking place in more than one iteration. When the parking place is too short, the two curved segments (segment 2 and 3 in figure 2.6a) are repeated in altering directions, until the vehicle is placed deep enough inside the parking place.

Since the key principle behind strategy A is that each iteration should move the vehicle sideways without changing its direction, every iteration has to consist of two curved segments. The change of direction during the first segment, is cancelled out by the second. This is not the case in strategy B. The principle behind strategy B is quite the opposite. It begins by changing the direction of the vehicle in order to move "sideways", and then change the direction back again by a number of iterations, see figure 2.7. Only the last step needs to be repeated.

Therefore, iterations of strategy A will always result in more segments than iterations of strategy B. Notice how steep the curve indicating the number of segments of the planned path for strategy B in figure 3.1 is. For an iterative version of strategy A, this curve would be even steeper. In the implementation in this study, where the vehicle is stopped between every segment, this is not an acceptable solution. Therefore, the principle of combining the benefits from two different strategies was used by APRIL.

4.3. Conclusions

It is shown in the experiments, that APRIL presents a satisfactory solution to the problem of autonomous parallel parking. The parking maneuvers it performs are sufficiently fast, accurate and reliable.

The main contribution of this study is the practical implementation and evaluation of the three ways to simplify the problem of autonomous parking set up as the aim for this study.

The strategy, inspired by biomimetic robotic research, to construct a map that only contains information about the goal, has proven to be quite successful. Even though the map only contains the positions of four points, constituting the free parking place, it still provides enough information to the path-planning algorithms to compute collision free paths.

It is also shown that even though the technology and sensors used by APRIL are very basic, they still provide adequate data to the map, and that their accuracy is sufficient for the task. In fact, only one range sensor and the two rotation sensors in the motors were needed to construct the map.

Finally, the distinction between to *autonomously find* a parking place and to *autonomously park* in one, greatly reduced the complexity of the problem. By not having to handle the difficulties of autonomously driving along a trafficked street, something I don´t think the science of robotics is ready for yet, many problems were avoided. In addition, the usability of the system is enhanced by letting the human drive the vehicle manually until he or she has found a desired parking place. The driver is still much better at judging in what parking place to park and how to drive along the street. The proposed system only tries to do what it does best, and in a way that cooperates with the human capabilities rather than trying to replace them.

4.4. Future work

The next step would be to equip the solution with behaviors that react to live sensor data during the execution of the parking maneuver. In addition, by using the sensor data recorded during execution of the parking maneuver, one could update and correct the existing map, and adjust the planned maneuver accordingly. This would enhance the solution and make it even more flexible and accurate.

5. Acknowledgements

First of all, I wish to thank Björn Samuelsson who has helped me a lot with the mathematical aspects of the problem and has been a great support in discussions about the general solution.

I also would like to thank my supervisor Christian Balkenius who gave me the confidence of taking on this project in the first place, although I had never before worked with anything close to robotics. It has been great work, but great fun!

6. References

Blais, F., Rioux, M. & Domey, J. (1991). Optical range image acquisition for the navigation of a mobile robot. Robotics and Automation, 3, 2574–2580.

Borenstein, J., Everett, H. R., Feng, L. & Wehe, D. (1997). Mobile robot positioning sensors and techniques. Mobile Robots, 4, 231–249

C., Ho, C., Lin, S. & Li, T. (2005). Omni-Directional Vision-Based Parallel-Parking Control Design for Car-Like Mobile Robot. *Proceedings of the 2005 IEEE International Conference on Mechatronics*, 562–567.

Dawson, C. (2004). In a Tight Spot – and Loving It. *Business Week*, 3903, 156.

Fokke, B. (2008). NXT# - Lego Mindstorms Bluetooth API Implementation in C#.
Source code available at *http://nxtsharp.fokke.net/* [2008-05-05]

Franz, M. O. & Mallot, H. A. (2000). Biomimetic robot navigation. *Robotics and Autonomous Systems*, 30, 133–153.

Gómez-Bravo, F., Cuesta, F. & Ollero, A. (2001). Parallel and diagonal parking in nonholonomic autonomous vehicles. *Engineering Applications of Artificial Intelligence,* 4, 419–434.

Gómez-Bravo, F., Cuesta, F. & Ollero, A. (2004). Autonomous parking and navigation by using soft-computing techniques. *Proceedings of the World Automation Congress*, 15, 173–178.

Holve, R. & Protzel, P. (1996). Reverse Parking of a Model Car with Fuzzy Control. *Proceedings of the 4th European Congress on Intelligent Techniques and Soft Computing - EUFIT'96*, 2171-2175.

Jiang, K. & Seneviratne, L. (1999). A sensor guided autonomous parking system for nonholonomic mobile robots. *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, 1, 311–316.

Laugier, C., Fraichard, T., Garnierm P., Paromtchik, I. & Scheuer, A. (1999). Sensor-Based Control Architecture for a Car-Like Vehicle. *Autonomous Robot,* 6, 1–18.

Laumond, J., Jacobs, P., Taix M. & Murray, R. (1994). A motion planner for nonholonomic mobile robots. *IEEE Transactions on Robotics and Automation,* 5, 577–593.

Levitt, T. & Lawton, D. (1990). Qualitative navigation for mobile robots. *Artificial Intelligence*, 44, 305–360.

Lo, Y., Rad, A., Wong, C. & Ho, M. (2003). Automatic Parallel Parking. *Proceedings of the 2003 IEEE Intelligent Transportation Systems*, 2, 1190–1193.

Meyer, J. A. & Filliat, D. (2003a). Map-based navigation in mobile robots - I. A review of localization strategies. *Cognitive Systems Research*, 4, 243–282.

Meyer, J. A. & Filliat, D. (2003b). Map-based navigation in mobile robots - II. A review of map-learning and path-planning strategies. *Cognitive Systems Research*, 4, 283–317.

Murray R. & Sastry S. (1993). Nonholonomic motion planning: steering using sinusoids. *IEEE Transactions on Automatic Control*, 5, 700–716.

Paromtchik, I. E. & Nassal, U. M. (1995). Reactive motion control for an omnidirectional mobile robot. *Proceedings of the Third European Control Conference*, Rome, Italy, September 5-8, 1995, 3074-3079.

Provine, R., Schlenoff, C., Balakirsky, S., Smith, S. & Uschold, M. (2004). Ontology-based methods for enhancing autonomous vehicle path planning. *Robotics and Autonomous Systems*, 1-2, 123–133.

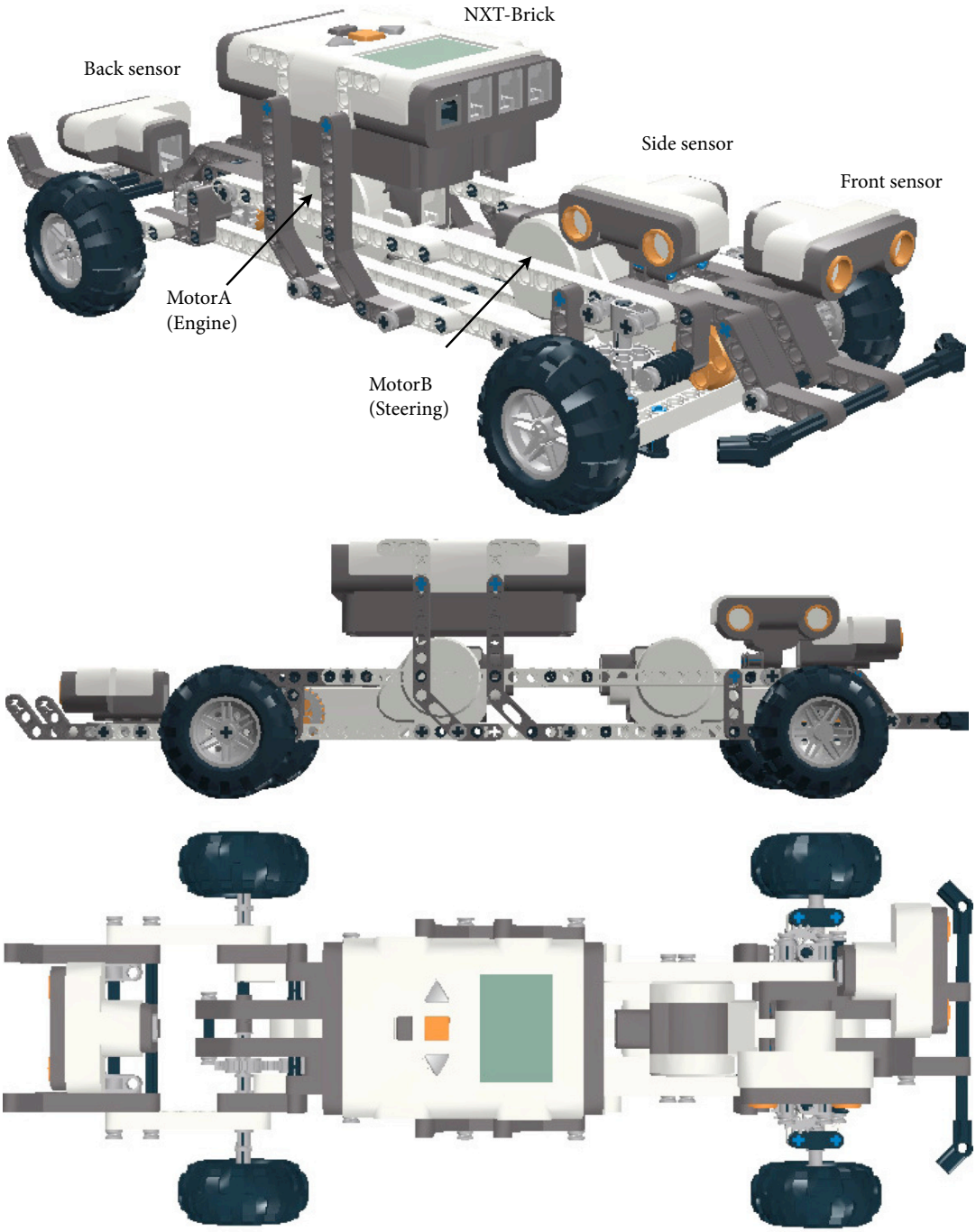
Saffiotti, A. (2001). The uses of fuzzy logic in autonomous robot navigation: a catalogue rasionné. *Soft Computing*, 1, 180–197

Thornton, C. (1994). Adaptation of obstacle-avoidance in the face of emerging environmental dynamics. *Proceedings of AIRTC-94*.

Vägverket (2004a). Vägar och gators utformning, Grundvärden. Excerpt from *VV Publikation 2004:80*.
Available at *http://www.vv.se/filer/publikationer/Grundvarden.pdf* [2008-05-05]

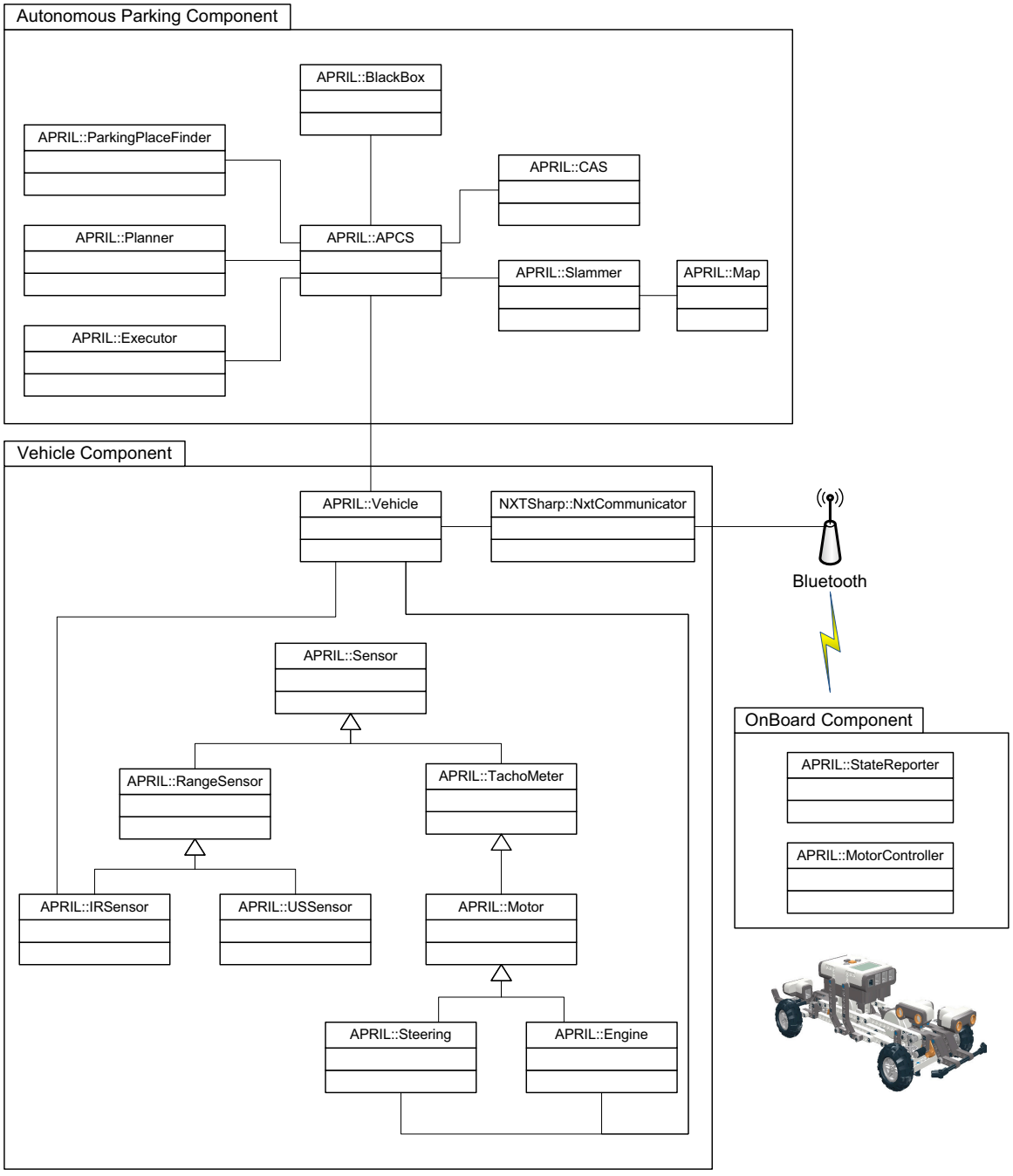
Vägverket (2004b). Vägar och gators utformning, Sido-anläggningar. Excerpt from *VV Publikation 2004:80*.
Available at *http://www.vv.se/Filer/Publikationer/Sidoanläggningar.pdf* [2008-05-05]

APPENDIX I: TECHNICAL DRAWINGS OF APRIL



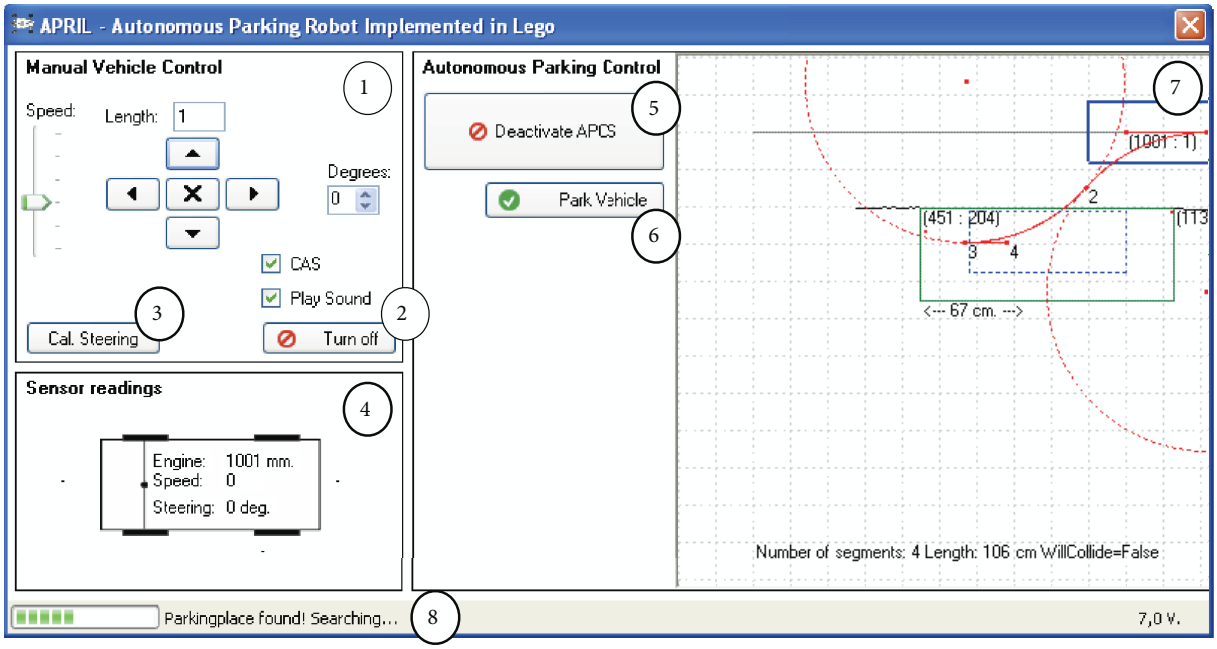
NB: For technical reasons the sensors shown on these drawings are represented as the ultrasonic sensors that came with the Lego Mindstorms Robotic kit. The infrared sensors used on APRIL is about the same size and placed on the same positions, but look different. Also, the cables from the sensors and motors to the NXT-Brick are left out for graphicness.

APPENDIX II: SIMPLIFIED UML MODEL



NB: For simplicity, all members, properties and methods are left out in this UML-diagram, as well as some of the data structures and helper classes. Also, the graphical user interface component is left out.

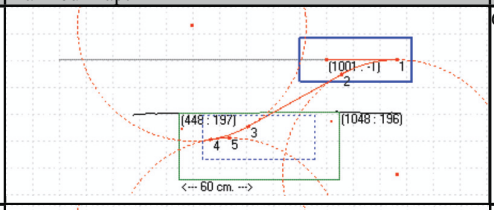
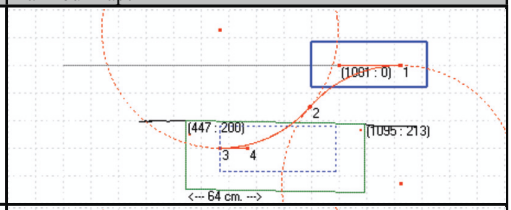
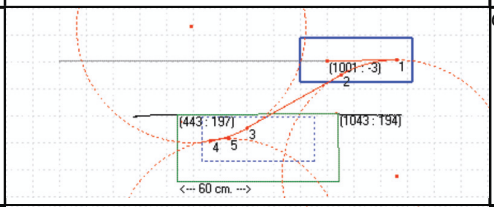
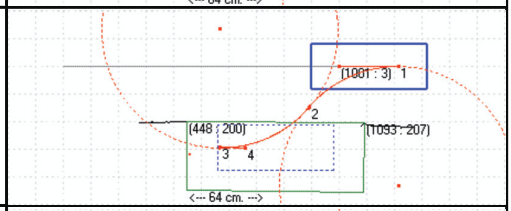
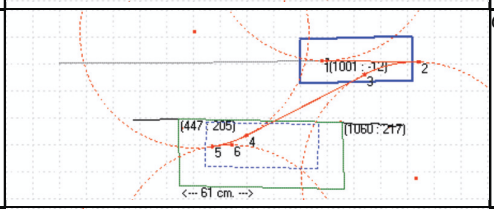
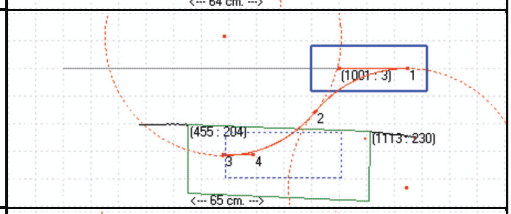
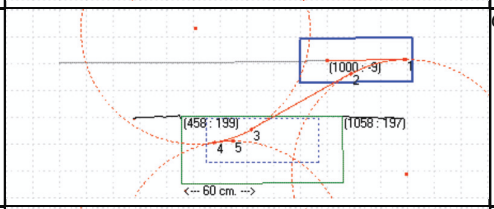
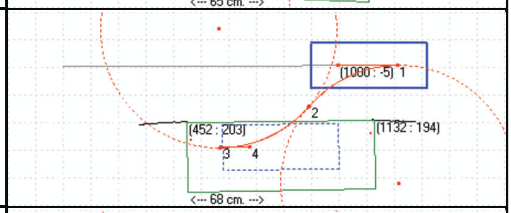
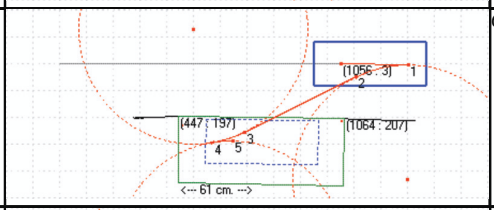
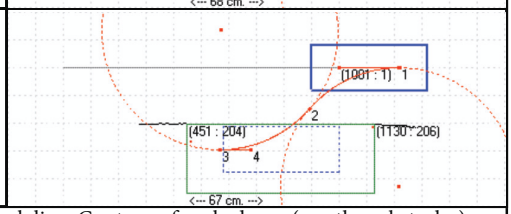
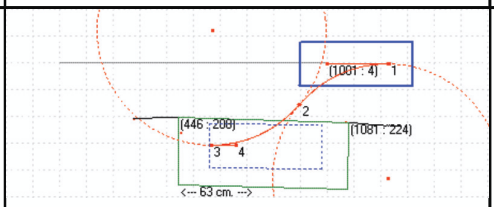
APPENDIX III: SCREENSHOTS FROM THE APCS-SOFTWARE



A screenshot from the graphical user interface. The screenshot shows the state of the interface directly after a parking place has been found and the parking maneuver has been computed. For feedback and debugging purposes, the internal map, and the planned maneuver, is graphically plotted in the user interface.

1. Manual Vehicle Control:
   This is where you control the vehicle when it is not performing an autonomous parking maneuver. You can either specify the length of the movement (in millimeters), or the steering angle (in degrees) and press the directional buttons, or you can use the keyboard to go forward/backward (W/S) or steer left/right (A/D). The vertical bar determines the speed of the engine (only applicable when using the keyboard to control the vehicle).
2. This button initiates the bluetooth connection to the vehicle and starts the OnBoard Component, or, as in this picture, stops it and disconnect.
   The 'CAS' checkbox turns the collision avoidance system on or off.
   The 'Play Sound' checkbox enables or disables the auditive feedback.
3. This button opens a wizard that helps the user to calibrate the prototypes steering, i.e., adjust the steering until it is straight.
4. This panel shows the raw data that the sensors are receiving.
5. This button activates or deactivates the APCS. When it is activated the vehicle is continuously searching for free parking places.
6. When a free parking place has been found, this button gets enabled. Pressing it will initiate the autonomous parking maneuver.
7. This is a graphically translated version of the generated internal map. It is updated every 1/10th of a second.
8. The status bar indicates the current status of the APCS. It also has a battery meter for the vehicle.

APPENDIX IV: EXPERIMENTAL PRINTOUTS

Length:	Planned map:	Length:	Planned map:
580 mm		640 mm	
590 mm		650 mm	
600 mm		660 mm	
610 mm		670 mm	
620 mm		680 mm	
630 mm		• Black line: Contour of parked cars (or other obstacles) • Thin red dots: Skipped sensor readings (filtered) • Gray line: Driven path • Blue solid line: Current vehicle position • Blue dashed line: Goal position of the vehicle • Green solid line: Parking place • Red solid line: Path segments • Thick red dots: turning centre of a curved segment • Red dashed line: Only drawn for graphicness • Gray dashed line: Scale indicators (10 mm*10 mm)	