AutoTruck
Automated docking with internal sensors

OSCAR ANDERSSON
LUCAS MOLIN
AutoTruck

Automated docking with internal sensors

OSCAR ANDERSSON
LUCAS MOLIN

Batchelor’s Thesis at ITM
Examiner: Nihad Subasic

TRITA-ITM-EX 2018:69
Abstract

The purpose of this bachelor thesis was to discover how an articulated vehicle can park itself using a pre-defined parking path with a combination of ultrasonic sensors as well as a rotary angle sensor.

The project was divided into two parts: constructing a small scale demonstrator and the software controlling the demonstrator. The demonstrator was constructed from off-the-shelf components and custom parts. The truck was designed based on a rear wheel driven truck with Ackermann steering. The localization of a parking spot and measuring other distances was done with ultrasonic sensors and the hitch angle was measured by a rotary angle sensor.

The performance of the demonstrator was evaluated by measuring the trailers angle difference from the center line of the parking spot.

The performance was deemed to be reasonably good with successful parkings in 8 out of 10 attempts.

Mechatronics, Trailer, Truck, Articulated vehicle, Self driving
Referat

Självparkerande lastbil

Kandidatarbetet syftar till att undersöka hur ett ledat fordon kan parkera sig självt efter en förbestämd parkeringsruttn med en kombination av flera ultraljudssensorer samt en vinkelgivare.

Projektet består av två delar; konstruktion av ett miniaturfordon samt mjukvaran som styr fordonet. Fordonet tillverkades från butiksköpta komponenter och skräddarsydda delar. Lastbilens design var baserad på en bakhjulsdriven Ackermannstyrad lastbil. Identifieringen av en parkeringsplats samt avståndsmätning hanterades av ultraljudssensorer och hitch vinkeln mättes av en vinkelgivare.

Miniaturfordonets prestanda utvärderades genom att mäta släpets vinkelskillnad från centerlinjen av parkeringsplatsen.

Prestandan ansågs att vara tillräckligt god med lyckade parkeringar i 8 av 10 tester.

Mekatronik, Trailer, Lastbil, Ledat fordon, Självkörande
Acknowledgements

We would like to thank Nihad Subasic for rewarding discussions and helpful tips in mainly the problem formulation, but also throughout the project. We would also like to thank the assistants Johan Ehrenfors, Martin Gylling and Stanislav Minko for invaluable help with persistent coding issues, and general guidance during the whole process. We would like to extend a large thanks to Staffan Qvarnström and Thomas Östberg for guidance with respect to electric and mechanic problems. Furthermore, we would like to thank Jannis Angelis for inspiring the analysis about the emergence of this technology in real life. Finally we would like to thank all the students in the mechatronics lab as well as down in the workshops for all the help with things large and small.
## Contents

### List of Figures

### List of Tables

### Introduction
- Background ........................................... 2
- Purpose .............................................. 2
- Method .............................................. 2
- Scope ............................................... 3

### Theory
- Ackermann Steering .................................. 4
- Docking .............................................. 5
  - Dynamics of articulated vehicles ............... 5
  - Human parking behaviour ......................... 5
- Ultrasonic Distance measurement ................. 6
- PID Controller ...................................... 7

### Demonstrator
- Electronics .......................................... 8
- Sensors ............................................... 9
- Arduino .............................................. 9
- Adafruit motor shield ............................... 10
- Truck ................................................. 10
- Steering ............................................ 10
- Sensors ............................................. 11
- Drivetrain .......................................... 11
- Stepper Motor ...................................... 11
- Rear axle .......................................... 12
- Cogwheels and Timing Belt ......................... 12
- Trailer ............................................. 12
- Front ................................................ 12
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Ackermann Steering [4]</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Kinematic model made by Amro Elhassan [3]</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>Fully constructed demonstrator</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>Demonstrators drivetrain</td>
<td>11</td>
</tr>
<tr>
<td>5</td>
<td>Cogwheel that is mounted on rear axle as well as motor axle. Picture rendered with Keyshot 6 [15]</td>
<td>13</td>
</tr>
<tr>
<td>7</td>
<td>Test area and vehicle path. Made with Microsoft PowerPoint</td>
<td>17</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th></th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Dimensions of the truck.</td>
<td>9</td>
</tr>
<tr>
<td>2</td>
<td>Arduino Uno specifications.</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>Adafruit motor shield v2.3 specifications.</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>Experiment results from the ten tests conducted.</td>
<td>18</td>
</tr>
</tbody>
</table>
List of Abbreviations

*IDE* Integrated Development Environment

*PID* Proportional, Integral, Derivative

*PVC* Poly Vinyl Chloride

*PWM* Pulse Width Modulation

*RC* Remotely Controlled
Introduction

Background

Reversing a vehicle is considered by many human drivers to be the hardest part of driving, especially with a trailer attached, due to the somewhat counter-intuitive movements required to position the trailer correctly. Professional truck drivers are of course better at handling their vehicle even though it is not an easy task. When docking with loading bays the vehicle often has to be reversed which makes this a regular occurrence. An automated system should in theory be able to perform these actions better than a human driver could.

Several companies are working today with the total or partial automation of trucks. Tesla is constructing their own truck [2] and Volvo says that ”We believe that automation will redefine the commercial transport solutions that most of us rely on every day.” [20].

Purpose

The purpose of this report is to answer the following question:

*How can a parking algorithm be implemented to park an articulated vehicle into a trailer loading dock?*

To answer this question an autonomous demonstrator vehicle will be constructed. The tests will be made with the primary focus on if the vehicle is able to park safely between two trailers. The tests will use the PID-algorithm and a pre defined path to park itself. A docking is considered successful when the trailer is within the designated docking spot.

Method

A model with the scale of 1:20 compared to a regular Scania truck [14] is constructed to resemble a truck with four wheels, Ackermann steering and rear wheel drive. The vehicle will utilize an Arduino UNO microcontroller in order to control the motors that handles the movement and steering as well as receiving signals and processing inputs from the robot’s surroundings. Four ultrasonic sensors are used to measure the distance to objects in proximity to the robot. One sensor will be placed on
the left side of the vehicle to be able to measure the size of the parking spot. The trailer will have one sensor at the rear and one on each side close to the middle of the trailer length. The performance of the vehicle will be simulated with a small scale test that is constructed to resemble a truck parking in a loading dock. A test run will begin with the autonomous vehicle travelling along a line of parked trailers until it finds a suitable parking spot which it will park itself into. The test run will be deemed successful when the vehicle managed to align itself between two trailers without colliding into any of the trailers.

Scope

The robot is assumed to be driven along a straight line of vehicles parked perpendicularly to the robots path on a flat surface. The path is modelled after a normal parking path. The only obstacles facing the vehicle during testing will be the parked trucks represented by cardboard boxes. To develop the project further a path planner could be included to allow the vehicle to traverse more complex environments. The programming will not allow for manual control after the parking program has been started. The vehicle will not be able to try again if a failed parking occurs without manual resetting. A failed docking means that the vehicle got too close to the obstacles, crashed into them, or that it did not enter the docking spot or did not properly align itself inside the docking spot.
Theory

Ackermann Steering

When performing parking lot maneuvers at low speed the tires of a vehicle do not develop lateral forces. Instead the tires roll with no slip angle while turning as seen in figure 1. In the figure the principles of Ackermann Steering is shown.

![Ackermann Steering Diagram](image)

Figure 1: Ackermann Steering [4].

Ackermann steering and the geometry behind it is a geometric arrangement where the linkages in the steering of vehicles are designed to avoid the need for tires to slip sideways when following paths around curves. The slip is caused by the wheels on the inside and outside having to trace out circles of different radii.

The Ackermann steering principle is implemented to compensate for this by having all wheels and their axles arranged as radii of circles with a common centre point. When the rear wheels are fixed, the center point moves on a line extending from the rear axle. The steering arms are angled inwards in order to steer to a greater angle than the outside wheel. This angling allows the Ackermann angle to
be determined. In order to do this symmetry lines have to be drawn through the steering arms to intersect with the rear axle’s centre line. The vehicle’s curve radius can then be measured as the distance between the intersection of the front wheel axes symmetry lines and the centre of rear axle.

Docking

This vehicle will be able to identify and locate a suitable parking spot to the left while driving in a straight line. A suitable docking spot is one measured by the vehicle. The chosen size was double the width of the demonstrator and was decided with regards to the demonstrators pre-defined docking path. The goal is to achieve a safe and reliable park.

Dynamics of articulated vehicles

Articulated vehicles have many benefits compared to non articulated vehicles, but are much harder to drive in reverse. The main difference in reversing an articulated vehicle compared to parking a normal vehicle is that the hitch angle, defined as the angle between the trailer and vehicle, is not directly steerable. Furthermore it is also possible to get into a state called ”Jackknifing” where it is impossible to recover and not crash the truck into itself without driving forward. On the constructed model this should be 90° as the limitation will be the kinematic constraint, as the trailer does not hit the truck before that.

A kinematic model of the dynamics is described by Amro Elhassan [3] with the following equations

\[
\begin{align*}
\dot{x}_C &= v \cos(\theta_1 - \theta_2) \cos(\theta_2) \\
\dot{y}_C &= \cos(\theta_1 - \theta_2) \sin(\theta_2) \\
\dot{\theta}_1 &= \frac{v \tan(\phi)}{L_1} \\
\dot{\theta}_2 &= \frac{v \sin(\theta_1 - \theta_2)}{L_2}
\end{align*}
\]

(1)

This leads to the hitch angle being defined by the steering angle as equation 2

\[
\delta = \dot{\theta}_1 - \dot{\theta}_2 = \frac{v \tan(\phi)}{L_1} - \frac{v \sin(\theta_1 - \theta_2)}{L_2}
\]

(2)

Figure 2 shows how all the variables are defined in the model.

Human parking behaviour

In the scope of this report a path planner is not included. Therefore something else has to plan the path of the vehicle. The path taken will be inspired by how a professional driver today parks its truck in the situation that is modelled which
is docking between two parked trailers. The truck is driven perpendicular to the parked trailers until the whole trailer has passed the open spot. Then the driver turns the steering wheel fully and starts to reverse until the hitch angle is almost critical at around 80°. The driver first quickly, then more gradually starts reducing the steering angle until the hitch angle is zero and the trailer is in the middle of the already parked trailers. The final step in the demonstrator is to regulate the hitch angle to stay as close to zero as possible while completing the reversing motion until the trailer is fully backed in. [8]

**Ultrasonic Distance measurement**

Ultrasonic Distance sensors sends out a high-frequency pulse of ultrasonic sound waves and times how long it takes for the echo of the sound to reflect back. The ultrasonic sound waves are above the human audible frequency range. The time difference $t$ between sending and receiving the sound pulse and the speed of sound which is approximately 343 m/s at room temperature 25°Celsius and close to sea level and allows the distance $s$ to the object to be calculated with the equation (4).

$$s = \frac{vt}{2}$$  \hspace{1cm} (3)

The equation is divided by 2 because the sound wave has to travel to the object and back. When a demonstrator is constructed the conditions above are assumed and considered satisfactory.
PID Controller

Control systems are widely used in a lot of different systems, some examples are balancing robots, cruise control in cars and autopilot in airplanes just to name a few.

Proportional-integral-derivative controller, (PID) controller is a control loop feedback mechanism which calculates a continuous error value $e(t)$ between a current measured value and a specified desired value, in order to then calculate the required change $u(t)$ to the control signal.

The PID controller equation is presented in equation (4) where the behaviour of the system is dependent on three parameters: proportional gain $K_p$, integral gain $K_I$ and derivative gain $K_D$. [5]

$$u(t) = K_p e(t) + K_I \int_0^t e(\tau) d\tau + K_D \frac{d}{dt} e(t)$$  (4)

The system characteristics impacted by the choices of parameters are overshoot, static error, rise time and settling time.

Overshoot is when the signal exceeds its target value and is often caused by over-regulating the system.

Static error (steady state error) is a constant error term which remains over time, the error is proportional to process gain and inversely proportional to proportional gain. Steady state error is usually compensated for by implementing an integral term.

Settling time is the time from the instantaneous step response to the time at which the output has entered and remained within a specified margin of error from the desired value.

When regulating angles in control systems, too long of a rise time and too much overshoot might lead to undesired behaviours such as collisions during parking. This may occur even though the system initially works and behaves in a desirable way.
Demonstrator

An overview picture of the completed demonstrator can be seen in figure 3.

The demonstrator was constructed to have the outer measurements at a 1/20 scale compared to a Scania P360 truck [14], with proportional axle distance and wheel size. However the internal measurements and the weight were not made proportional to the Scania P360 truck. The measurements in table 1 were used when constructing the truck. The trailer had equal width to the truck and with a length of 46 cm.

Figure 3: Fully constructed demonstrator
Electronics

Sensors

Ultrasonic sensor

The testbed uses 4 HC-SR04 Ultra sonic-sensors. HC-SR04 was chosen due to its small size and easy implementation. How the sensor works is described in chapter.

Rotary Angle Sensor

An off-the-shelf potentiometer was modified with a steelrod, so that it could work as a tow ball for the trailer. The sensor produces analog output between 0 and Vcc (5V DC) on the A3 connector with an angular range up to 300 degrees and a linear change in value. Some manual adjustments were made in the code in order to make sure that a certain value corresponded to the trailer being aligned.

Arduino

The Arduino Uno is an open-source microcontroller which in this project is used to read input signals and turn them into output signals. Instructions for how the Arduino handles input and output is programmed using Arduino IDE software. The Arduino handles input from the sensors and sends the output to the Adafruit motor shield v2.3 in order for it to control the motors. The Arduino Uno was chosen for this project because of its relatively small size, while still being able to handle all signals as well as take in and process the sensor data within a reasonable timeframe.

Table 2 includes the most necessary specifications for the Arduino Uno.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Operating Voltage</td>
<td>5 V</td>
</tr>
<tr>
<td>Input Voltage</td>
<td>7 V - 12 V</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16 Mhz</td>
</tr>
</tbody>
</table>

Table 2: Arduino Uno specifications.
**Adafruit motor shield**

In order to control all of the motors used in the project an Adafruit motor shield v2.3 was implemented. The board connects to the Arduino Uno with regular pin headers and has a dedicated PWM driver chip onboard. The chip allows the user to handle all of the connected motors with different speed controls as well as allows the required motor voltage of 6 V and 1 A to be supplied to the motor without damaging the Arduino. Furthermore it has an Arduino software library for controlling the motors which simplifies software development.

The stepper motor is connected to a motor port on the Adafruit, and is controlled with Adafruit’s Stepper library.

Table 3 includes the most necessary specifications for the Adafruit motor shield v2.3.

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power Supply Voltage</td>
<td>15 V</td>
</tr>
<tr>
<td>Output current</td>
<td>1.2 A (ave) / 3.2 A (peak)</td>
</tr>
<tr>
<td>Number of Digital I/O Pins</td>
<td>14</td>
</tr>
<tr>
<td>Number of Analog Input Pins</td>
<td>6</td>
</tr>
<tr>
<td>Number of PWN Digital I/O Pins</td>
<td>2</td>
</tr>
</tbody>
</table>

Table 3: Adafruit motor shield v2.3 specifications.

**Truck**

**Steering**

The Ackermann steering was modeled based on the design from a bachelor thesis, Following Car,[18] but was remodeled to comply to the Ackermann principles and with the outer dimensional constraints of the demonstrator. The reasoning behind choosing Ackermann steering was partly based on the efficiency of said method as well as the geometrical benefits.

The Ackermann steering pars were made with Solid Edge ST9: Siemens PLM Software [16] and 3D printed using Ultimaker 2 and Ultimaker 3 [19]. The steering pars are held together with standard nuts and bolts.

A servo motor, Hitec Hs-300 RCD Apollo 5 was used to control the steering, as it can guarantee that the required steering angle is met by the demonstrator, which is very important in a non holomorphic system when reversing the vehicle, where failure to meet the demanded steering angle could quickly result in a non recoverable state.
**Sensors**

**Ultrasonic sensor**

One HC-SR04 Ultra sonic-sensor was mounted on the middle left side of the truck in order to measure and identify a suitable parking spot.

**Rotary Angle Sensor**

The Rotary Angle Sensor was used to calculate the hitch angle between the trailer and the truck. It was mounted centrally above the rear axle of the truck, although on real trucks it is mounted in front of the rear axle. This was done because it simplifies the transfer functions greatly [3] without impacting the dynamics much in reality.

**Drivetrain**

A picture of the demonstrators drivetrain can be seen in Figure 4.

![Figure 4: Demonstrators drivetrain](image)

**Stepper Motor**

The chosen stepper motor, a Vexta Stepping Motor Model PK243M-01A 2-Phase with 0.9 °phase, was used for the drivetrain even though DC motors are usually used in RC cars. The decision to use a stepper motor was made with the dynamics of parking in mind, as these are different from when racing a RC car. The high level of control needed in parking, constrained by the demonstrators small size, made a stepper motor the perfect match as there was no need for an external encoder.
The stepper motor is easily commanded to drive a certain number of steps, which translates to a set number of centimeters driven, simplifying path design.

Motormounts were modeled in Solid Edge ST9: Siemens PLM Software [16] and 3D printed using Ultimaker 2 and Ultimaker 3 [19] in order to hold the motor in place under the truck.

Rear axle

The rear axle is a 5 mm thick steel axis which is held in place by two mounts. The mounts were made with Solid Edge ST9: Siemens PLM Software [16] and 3D printed using Ultimaker 2 and Ultimaker 3 [19]. The mounts were screwed onto the plate using nuts and bolts. The wheels on each side are held in place by a screw in order to lock them in the axial direction.

Cogwheels and Timing Belt

Two cogwheels with a timing belt between them are used to transfer torque from the motor to the rear axle. The cogwheels were fitted tightly onto the axles.

In figure 5 the cogwheel mounted on the rear axle is shown. A similar cogwheel is mounted on the motor axle but with a different hole diameter. The cogwheel on the rear axle is locked in place axially with a screw and the cogwheel on the motor is pressed onto the Stepper motor axle. The use of a timing belt ensured that the desired amount of steps translated exactly to the driving axle.

The timing belt transfers torque from the Stepper motor to the rear axle. RS Pro Timing Belt 85 Tooth was chosen for this purpose. Mainly due to the small size of the belt as it suits the demonstrator well. The timing belt is 172.2 mm long and 6 mm wide.

Trailer

Front

The front was laser-cut from a PVC-board, in the shape of a T, to allow realistic hitch angle of 90° while not taking space from the trucks electrical components. The upper part of the tow ball mount was designed in Solid Edge ST9: Siemens PLM Software [16] and 3D printed using Ultimaker 2 and Ultimaker 3 [19], which was mounted centrally on the laser cut parts. On the wide part there are two holes for fastening the extenders. The front was attached to the rear with two extenders. The pair of extenders were laser-cut from PVC with two holes at each end for fastening.

Rear

The rear was also laser-cut from acrylic plastic. It has three Ultrasonic sensors SR-04 mounted on top of it, one in the middle, facing rear, and two symmetrically placed facing right and left. These are connected to a small breadboard, which in
Figure 5: Cogwheel that is mounted on rear axle as well as motor axle. Picture rendered with Keyshot 6 [15]

turn connects with a longer cable to the truck. At the front there are holes for fastening the extenders. The two sensors facing to the sides were not implemented in the final version of the code.

**Electronics**

The Adafruit Motor Shield is powered by four AA batteries supplying 1.5 V each. The Arduino Uno and all the other components connected to it is driven by a battery pack through USB.

**Software**

The truck moves in a straight line along objects and measures the distance to its left side. If the distance is longer than a certain value for a set distance driven, a parking spot is assumed to have been found. The truck now enters its pre-defined parking path. When the path is completed the trailer is sufficiently straight for the PID calculations to begin. The PID’s task is to fully align the trailer with the truck.
by minimizing the hitch angle while reversing. While the truck is reversing the ultrasonic sensor at the back of the trailer is continuously measuring the distance left to the back of the parking spot. If that distance is less then a certain value the main loop is exited and the truck stops.

A complete flow chart of the algorithm is described in Figure 6 below. The Arduino Code can be found in Appendix.

Figure 6: Flow chart of the algorithm. Made with draw.io [10]
Emergence of automating docking in logistics centers

This work have focused on the technical solutions enabling an articulated vehicle to autonomously reverse into a loading dock. The demonstrator and the scope of this work would not in itself allow for completely automated loading areas, but could be a central part of a larger system, handling traffic and docking of trailers at a logistics centre.

This section will attempt to broadly describe the conditions and actors present in the sector today, and their impact on the emergence of this technology in the field.

Possible benefits of implementing automated docking

Traditional loading of a trailer takes up to 30 minutes but can take as low as 2 minutes with automated loading [9]. The large difference in loading time creates two different cases for automated trailer management. With 30 minutes loading time, truck drivers could arrive with one trailer and leave instantly with another, however this procedure would probably not be efficient with the short loading time.

In Sweden the average hourly salary for a truck driver is 140 kr [11]. Saving 30 minutes at each end would therefore only yield a saving of 140 kr per transported trailer. This could probably be realized easier with an automated loading system than with a complex system of changing trailers only viable at very large reloading centers.[9]

However at very large loading areas the time to find the right dock, get checked in etc, adds up, and the benefit of automating grows. Amazon has been investing heavily in making their supply chain more efficient and automated. In the fall of 2017 Amazon launched an app called “Relay” with the special goal of optimizing the check in and docking process[17]. Automating the whole docking process would further Amazon’s stated goals, and could be possible given Amazon’s opportunity to own or control all of the parts in the logistics chain.

This suggests that the benefits of implementing this technology would mainly be visible for very large actors.
Who would make the investment?

The logistics/transportation market is divided into three major players.[1]

- The customers, who want goods moved at the lowest price with the highest convenience, for example Zalando, often owns warehouses and sometimes logistics centers.

- The logistics firms, who plan routes and optimize paths for specific parcels, owns the trailers and the planning algorithms, and faces both the end customers and the freight forwarders.

- The freight forwarders contacted by the logistics firms to drive a certain route, often owns the trucks.

This dynamic creates a difficulty in finding an actor willing to invest in further developing this technology. The technological developments in autonomous trucking have mostly been focused on automating highway driving, as it is the easiest to automate, while also taking up a lot of driving time [13].

The logistics companies do not own the trucks, where most of the tech would have to be placed, and do not control the loading areas, which would have to be optimized to utilize the technology. They would therefore be secondary actors in the development of self parking trucks. Developing this technology would therefore need an actor with a more vertically integrated supply-chain (as noted in the section about economical impacts).

Conclusions

The fragmented market structure and limited profits for smaller actors implementing automated docking suggests that this type of automation will be implemented further into the future, as a part of a more generally automated logistics chain, rather than as a small automated part in a generally human dependant system.

Suggested by the technical conclusions of this thesis using automated docking systems, where a central system reliant on local positioning systems controls the planning of trucks, and control their adherence to their paths would be a better system than with the internal sensors and systems used for automated driving on outside roads. This could create a "handover", where the systems used on the closed area for docking hands over control to the trucks internal system, when the truck and trailer leaves the gates.

The structure of the market, and the moves made by Amazon, together with the fact that only already autonomous trucks can be remotely controlled safely suggests that the primary driver of innovation and the path of emergence will be chiefly influenced by Amazon or another player of similar characteristics.
Experiments

Test Area

The test area was designed to resemble a large loading area with multiple docks in a row. In some of the docks trucks were already parked in order for the ultrasonic sensors to find an empty spot. A large loading area would in reality be very safe to automate, since speeds are low, and the area can be closed off to the public. The other trucks and trailers are represented by cardboard boxes.

Figure 7 shows a simple sketch of the test area as well as a part of the designated path.

![Figure 7: Test area and vehicle path. Made with Microsoft PowerPoint](image)

Experiment description

The starting position of the demonstrator was set at a distance shorter than 40 cm from trucks represented as cardboard boxes, with a steering angle of 0°. The demonstrator drives in a straight line until the distance becomes longer than 40 cm and if that distance continually is longer than 40 cm while the demonstrator drives 18 cm forward, it has identified a suitable parking spot. The demonstrator enters the parking mode function and parks itself, the last step is aligning the trailer with the truck using a PID regulator.
For a successful experiment the demonstrator has to find a parking spot and enter the parking spot without colliding with anything.

The trailer’s angle from the parking spot will be measured after a successful parking in order to determine how straight the trailer aligned itself.

**Parking results**

The demonstrator was able to identify a suitable parking spot in all of the ten tests performed. The difference of the demonstrators angle from the center line was the only calculation taken into consideration due to the fact that the actual position inside the parking spot was solely dependent on the first distance driven in the pre defined path.

When the demonstrator completed it’s parking two measurements were measured in order to determine how close to $0^\circ$ hitch angle the demonstrator was aligned. The first measurement was the distance from the parking spot’s right side to the center of the truck’s rear axle. Secondly, the distance from the parking spot’s right side to the center of the trailer was measured. The distances were calculated into the angle difference from the center line of the parking spot.

Table 4 presents the angle difference from the center line of the parking spot from the ten tests conducted. The complete data from the experiments can be found in Appendix.

<table>
<thead>
<tr>
<th>Test number</th>
<th>Calculated angle</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5°</td>
</tr>
<tr>
<td>2</td>
<td>7.3°</td>
</tr>
<tr>
<td>3</td>
<td>12.4°</td>
</tr>
<tr>
<td>4</td>
<td>14.5°</td>
</tr>
<tr>
<td>5</td>
<td>11.0°</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>1.5°</td>
</tr>
<tr>
<td>8</td>
<td>5.1°</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>18.7°</td>
</tr>
</tbody>
</table>

Table 4: Experiment results from the ten tests conducted.
Discussion and conclusions

Performance of drivetrain

The stepper motor and timing belt did in a majority of test cases deliver the correct driving distance with good accuracy. However, the open control combined with the fact that the stepper motor was working near maximum capacity lead to some times with more challenging conditions (i.e. dirt on the path, or large hitch angles) non completed steps. This resulted in the vehicle driving too short of a distance, leading to completely failed parking attempts.

Problems with using a pre defined path

The main problem of using a pre defined path is that the system becomes very sensitive to disturbances. Compared to earlier works [7] concerning parking paths with non articulated vehicles, the disturbances do not amplify as much as in the situation with an articulated vehicle. Furthermore if a small deviation in the hitch angle occurs in the beginning, it quickly amplifies if not corrected and the only possible solution for correcting it is to drive forward.

To implement a controller for following a path, whether pre-set for a situation or with a full scale path planner sensors outside the vehicle would be needed to position the vehicle in an xy-plane. In addition the relationship between the trailer and truck i.e the hitch angle is important to take into consideration. In the scope only internal sensors were used which proved to be an important limitation on the end result.

Implementing a path planner for automating vehicles requires "automated driving under controlled environments" [6]. Automation and path planning for uncontrolled environments require different kinds of path planning. [12]

Performance of steering and PID

For the main part of the parking path the full possible steering angle in either direction is used and the constructed steering worked very well at achieving full angles quickly and correctly. However, in the final parking when the PID-algorithm was used to try and align the trailer with the truck, problems arose. The construction
was not sufficient at achieving exact turning angles inside the physical limits, this
lead to problems during the tuning of the PID. The PID could not correct small
errors before they grew large, leading to oscillating system that kept turning the
trailer in the direction it was in when the PID first was engaged. This behaviour
leads to the trailer being angled in the same direction in every successful parking
test. Further tuning of the PID could have improved the behaviour, but with the
limitations in the steering the end result would not improve notably.

Conclusion

The demonstrator achieved a successful park 8 out of 10 times, which is reasonably
good performance. However the high accuracy needed for docking with a loading
dock, was only achieved once in the tests. Earlier work, including real world cars
have shown the possibility of parking a simple vehicle with a predefined path using
only on-board sensors. Noting the sources of error, and limitations in the demon-
strator, it does not seem advisable to design a completely autonomous parking
algorithm for an articulated vehicle.
Bibliography


Arduino Code

Arduino Uno Code
// Lucas Molin and Oscar Andersson
// Arduino code for bachelor project in Mechatronics 2018, AutoTruck

#include <Wire.h>
#include <Servo.h>
#include <Adafruit_MotorShield.h>
#include <PID_v1.h>

#define ROTARY_ANGLE_SENSOR A3  // Sensors mounted on the left side of the trailer
#define ADC_REF 5  // Reference voltage of ADC is 5v. If the Vcc switch on the Arduino board switches to 3V3, the ADC_REF should be 3.3
#define GROVE_VCC 5  // VCC of the grove interface is normally 5v
#define FULL_ANGLE 300

int tv[2] = {4, 5};  // Sensors mounted on the left side of the trailer
int th[2] = {2, 3};  // Sensors mounted on the right side of the trailer
int tm[2] = {8, 11};  // Sensors mounted in the middle of the trailer
int bv[2] = {6, 7};  // Sensors mounted on the left side of the truck
int inPin = 12;  // Switch read

double langd = 29;  // 29 // 39 // 49

String str;

// PID
double Setpoint;  // Will be the desired value
double Input;  // Hitch angle
double Output;  // Steering angle

// PID parameters
double Kp= 3, Ki=2, Kd=5;

Adafruit_MotorShield AFMS = Adafruit_MotorShield();  // Create the motor shield object with the default I2C address

Adafruit_StepperMotor *myStepper = AFMS.getStepper(400, 2);  // Connect a stepper motor with 400 steps per revolution (0.9 degree) to motor port #2 (M3 and M4)

Servo servol;  // Declare servo

PID myPID(&Input, &Output, &Setpoint, Kp, Ki, Kd, DIRECT);  // Declaring a PID object

void setup() {
  pinMode(LED_BUILTIN, OUTPUT);  // Set pin #13 to Output
  Serial.begin(9600);  // Set up Serial library at 9600 bps

  pinMode(ROTARY_ANGLE_SENSOR, INPUT);  // Set analog pin #3 to Input
  AFMS.begin();  // Create with the default frequency 1.6KHz
  AFMS.begin(1000);  // OR with a different frequency, say 1KHz initialize the variables we're linked to
  Input = analogRead(ROTARY_ANGLE_SENSOR);

  int distance(int trigPin, int echoPin);  // Define distance
  servol.attach(9);  // Attach a servo to pin #9
  servol.write(40);  // Set the servo steering angle to 40
  pinMode(inPin, INPUT);  // Set servo pin to input

  // PID continuing
  Setpoint = 156;

  // Turn the PID on
  myPID.SetMode(AUTOMATIC);  // Set the pin to automatic
```c
myPID.SetTunings(Kp, Ki, Kd); // Declare the PID parameters
myPID.SetOutputLimits(0, 80); // Set output limits for the PID to send to servo

// Setup the stepper
myStepper->setSpeed(120); // Speed of 120 rpm
}

// Function to read and return distance from Ultrasonic sensors
int distance(int lista[2]){
    int total;
    int trigPin = lista[0];
    int echoPin = lista[1];
    pinMode(trigPin, OUTPUT); // Set every trig pin to output
    pinMode(echoPin, INPUT); // Set every echopin to input
    long duration, distance;
    digitalWrite(trigPin, LOW); // Added this line
    delayMicroseconds(2); // Added this line
    digitalWrite(trigPin, HIGH); // Writes a high signal to trig pin
    digitalWrite(trigPin, LOW); // Writes a low signal to trig pin
    duration = pulseIn(echoPin, HIGH); // Reads the time for the measurement
    distance = (duration/2) / 29.1;
    return distance;
}

// Drive function, takes distance the truck should be driven as input
// Sends output to stepper
void drive(float cm) {
    float steps = (cm*200)/(3.5*3.141592);
    if (steps>0)
    {
        myStepper->step(steps, FORWARD, DOUBLE); // Declare how many steps the stepper takes forward
    }
    if (steps<0)
    {
        float rsteps = -1*steps;
        myStepper->step(rsteps, BACKWARD, DOUBLE); // Declare how man steps the stepper takes backwards
    }
}

// Steering function for servomotor
void steer(int angle){
    int servo_angle = angle+40;
    servo1.write(servo_angle); // Writes an angle to sevro motor
    int wait = 15*angle;
    int wa = abs(wait);
    delay(wa);
}

// Calculating the hitch angle
float hitch(){
    int sensor_value = analogRead(ROTARY_ANGLE_SENSOR);
    float voltage = (float)sensor_value*ADC_REF/1023;
    float input_angle = (voltage*FULL_ANGLE)/GROVE_VCC;
```

```
float hitch_angle = input_angle - 150;
return hitch_angle;
}

void pid(){
  Input = map(analogRead(3), 0, 1024, 300, 0); // Rotary angle sensor mounted on pin 3, maps analog value to rotary angle
  // PID calculation
  myPID.Compute(); // Computes the values for PID
  servo1.write(Output); // Write the output as calculated by the PID function
  // Send data by serial for plotting
}

// If a parking spot is identified this function is called upon
// Parking path
void parking_mode(){
digitalWrite(LED_BUILTIN, HIGH); // Writes high signal to LED
delay(1000);
steer(0); // Steers 0 degrees
  // Distances driven calculated as a factor of the length of the trailer
drive(langd*2.256410256); // Drives a distance depending on length
steer(-50); // Steers -50 degrees
drive(langd*-0.948717949); // Drives a distance depending on length
steer(50); // Steers 50 degrees
drive(langd*-1.025641026); // Drives a distance depending on length
steer(50); // Steers 50 degrees
drive(langd*-0.769230769); // Drives a distance depending on length
  // Measuring distance from the back of trailer to end of parking spot
int kvar = 100;
while (kvar > 5 or kvar == 0){
drive(1); // Continous drive forwards with 1 cm
  d = distance(bv); // Calls upon distance function for sensors mounted on the truck
  Serial.println(kvar);
drive(-1); // Continous drive backwards with 1 cm
pid(); // Calls upon PID function
}
digitalWrite(LED_BUILTIN, LOW); // Writes low signal to LED
exit(0); // Stops the truck from reversing by exiting main loop
}

// State in which the truck searches for a large enough parking spot
bool canpark(){
  for(int i; i <18;i++){
    drive(1); // Continous drive forwards with 1 cm
    int d = distance(bv); // Calls upon distance function for sensors mounted on the middle of the trailer
    if (d <40){
      digitalWrite(LED_BUILTIN, HIGH); // Writes high signal to LED
      return false;
    }
  }
digitalWrite(LED_BUILTIN, LOW); // Writes low signal to LED
return true;
}

void loop(){
drive(1);  // Continous drive forwards with 1 cm
if (canpark()){
parking_mode(); // Calls upon parking mode function
}

// The following code allows for the truck to be steered and driven manually by entering serial inputs
if(Serial.available() > 0) {
    str = Serial.readStringUntil(' ');
    x = Serial.parseInt();
    if (str == "d"){
        drive(x);
        Serial.println(str);
        Serial.println(x);
    }
    if(str == "s") {
        steer(x);
        Serial.println(str);
        Serial.println(x);
    }
    if(str == "p") {
        for (int i ;i < x; i++){
            pid();
            drive(-1);
            Serial.println(str);
            Serial.println(x);
        }
    }
    if(str == "c") {
        int bv = distance(bv);
        Serial.println(str);
        Serial.println(bv);
    }
}
## Raw Data from Experiments

<table>
<thead>
<tr>
<th>Test number</th>
<th>Calculated angle</th>
<th>Identified spot</th>
<th>Distance, back [cm]</th>
<th>Distance, Front [cm]</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9.5°</td>
<td>Yes</td>
<td>16</td>
<td>9.5</td>
</tr>
<tr>
<td>2</td>
<td>7.3°</td>
<td>Yes</td>
<td>15.5</td>
<td>10.5</td>
</tr>
<tr>
<td>3</td>
<td>12.4°</td>
<td>Yes</td>
<td>17</td>
<td>8.5</td>
</tr>
<tr>
<td>4</td>
<td>14.5°</td>
<td>Yes</td>
<td>19.5</td>
<td>9.5</td>
</tr>
<tr>
<td>5</td>
<td>11.0°</td>
<td>Yes</td>
<td>15</td>
<td>7.5</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>7</td>
<td>1.5°</td>
<td>Yes</td>
<td>9</td>
<td>8</td>
</tr>
<tr>
<td>8</td>
<td>5.1°</td>
<td>Yes</td>
<td>10</td>
<td>6.5</td>
</tr>
<tr>
<td>9</td>
<td>-</td>
<td>Yes</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>10</td>
<td>18.7°</td>
<td>Yes</td>
<td>26</td>
<td>13</td>
</tr>
</tbody>
</table>