Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control
Institut für Dynamische Systeme und Regelungstechnik

Hans Muster

FUCK DSPACE

How a company stole our money

Bachelor's Thesis

Institute for Dynamic Systems and Control
Swiss Federal Institute of Technology (ETH) Zurich

Supervision

First Supervisor
Prof. Dr. Second Supervisor

December 2016

IDSC-XX-YY-ZZ

Abstract

Abstract goes here

Keywords: First keyword, Second keyword.

Contents

Chapter 1

Introduction

The purpose of this work was to design a drive-by-wire system for future use in the field of autonomous self driving cars. The idea of the overall project is to have a fleet of autonomous vehicles, mainly to test fleet management algorithms. This go-kart prototype was made as a proof of concept and for early testing of the software and hardware.

Our objective therefore was plan and realise a solution for every critical component, i.e. throttle, brake and steering. In order to test A.I. and the fleet management algorithms, a stable and reliable low level platform needs to be in place. Requirements for the indiviual components in the drive-by-wire system were i.e. precision, speed, reliable communication and flexibility. Planing included steps such as finding the parts, how to build it into the kart, communicating with the part and supplying the right power. Designing a drive-by-wire system poses many challenges, which can be approach in many different ways. Most drive-by-wire vehicles are being conceptualised with safety as their highest priority. This is because those vehicles transport passengers and their well-being is of utmost importance. That is why these system contain many fallback levels, which are rather difficult to realise. Our kart does not contain such fallback levels, simply because it is self-driving and the drive-by-wire system will not be active when a human is at the wheel.

Chapter 2

Overview Hardware

2.1 Components

In the following section, we will shortly describe the relevant parts already built into the kart, as well as all other components we installed. We will also explain their importance and why we decided for these compontents.

2.1.1 Built into Go-Kart

The go-kart used for this project was a SinusiON, an electric kart manufactured by Rimo Germany. An electric kart offers an easier implementation of a throttle-by-wire system over more common petrol-fueled kart, as the basis for such a system is already in place. It's weight prior to modifying it was roughly 170 kg, making it much heavier than a petrol-fueled kart. The dimensions (l/w/h) are 2020mm/1390mm/600mm.

ACD 4805 Motor controller

Each of the electric motors is controlled by an ACD 4805 motor controller, mounted on top of the motor. The ACD communicates via CANOpen and allows for easy modification of parameters and performance curves. The motor controller was likely the most important in-built part, as the whole throttle-by-wire system was based on it.

Brake

Rimo offers a dual-circuit hydraulic disc brake system. A simple lever arm connects the brake cylinder to the brake pedal. This configuration requires a precise actuator, as the way difference between a mild and hard brake was minimal.

Steering

The go-kart's steering mechanism was realized with a bell-crank linkage. This setup resulted in a non-linear steering behaviour, which needed to be accounted for when configuring the power steering. The steering shaft was short and it's surroundings offered limited space, therefore the required steering servo needed to be small as well.

Battery

The main battery consists of 16 x 3.2 V LiFeMnPO4 cells and offers 40 Ah of battery charge. The nominal on-board voltage is 48 V. Because of the capacity of the battery, it makes a separate power supply for most of the additional electric actuators redundant. Only the power steering required 12 V and 90 A peak current. Because most converters do not support such a high peak current and the idea of capacitor seemed impractical, a small 12 V battery with a capacity of 3 Ah had to be installed in the front of the car.

Motor

The kart features two "PMS 100 R" 2.8 kW double-sided synchronous motors, each controlled by one of the two ACD 4805 motor controllers. The motors also offered regenerative braking, offering longer driving time and assisting in braking the car.

2.1.2 LinMot

In order to actuate the brakes a linear motor was used. The characteristics of being fast and precised make the motor suitable for various brake maneuvers. The motor is an electromagnetic drive in tube-form. The linear motion simulating a mechanical brake can be generated electrical without any form of mechanical interconnection between motor and motion. The linear motor is composed by two parts: the stator and the runner. The runner consists of a series of neodym-magnets, placed in steel tube. In the stator, the winding as well as the bearing for the runner, position detection and supervision of the motor fit in. The internal position sensors can dynamically transmit a position signal. Therefore position can be controlled real time. This guarantees a high level of safety and flexibility. In order to communicate with the motor, an LinMot motor-drive is employed. Target position, maximum velocity and acceleration can steadily be adjusted.

2.1.3 Power steering

Thyssenkrupp Presta offered a compact and powerful solution. The 3 Nm of rated torque was more than sufficient for steering the kart. The unit communicates via CANOpen, with TKP offering a Simulink Blockset for easy implementation.

2.1.4 DC-DC converter

In order to provide 24 VDC for powering the Microautobox and the linear motor drive, an SD-50C-24 DC-DC converter was used.

2.1.5 Microautobox (MABX)

The power steering manufacturer implemented a blackbox model of their unit in Matlab's Simulink. Their model was programmed to only run on dSpace's Microautobox. The Microautobox is a real-time system, used for performing fast function prototyping.

If needed, the MABX can easily be connected to a more powerful computer via the in-built ethernet connector. The MABX communicates with the go-kart's components via CAN. In it's basic configuration, the system does not support CANOpen, therefore an additional Simulink blockset had to be bought. The scarce time during the project justified the purchase of MABX, otherwise a much cheaper option could have been acquired. A less expensive option obviously calls for more

programming and offers less plug-and-play, which surely would have slowed down our progress significantly.

2.1.6 cases/cables/adapters

A wide variety of cables, adapters were used in order to ensure seamless integration of the components into our system. Most of the cables we soldered ourselves were used for power supply. For the CAN communication we used a preexisting solution, where no soldering was needed. Previously, a RJ-45 cable connected the linear motor drive to the can network. A break out adapter was bought, in order to gain easy access to the CAN high and CAN low connectors.

A metal sheet case was bought and mounted in the back of the kart, where the DC-DC converter, the MABX and the Linmot motor drive were safely stored.

Chapter 3

Overview Software

For this project, a variety of software was being used, namely dSpace Control Desk, LinMot Talk and Matlab Simulink. Without going into too much detail, we will elaborate on their use in this project and how they worked together.

Matlab is a numerical computing environment, with Simulink being a block diagram environment for multi domain simulation and model-based design.

dSpace Controldesk is a experiment software, used to develop and test operating ECUs. If offers data capture across different platforms and access to common Bussystems, including CAN and CANOpen.

LinMot Talk is LinMot's own drive-configuration software, allowing for easy access to the drive and controlling of the linear motor.

When the system is online, the Microautobox will receive commands from a higher layer, such as a simple rc controller or an A.I. autonomously controlling the vehicle. For this, a model of the communication interface needed to be set up in Simulink.

dSpace provided us with numerous Simulink blocksets, containing blocks of all necessary components, such as ADC, DAC and CAN. With these, a model can easily be formed and compiled. The compiler creates a C/C++ code, which then will be flashed onto the Microautobox. Controldesk can then be used to gain information about the state of the system's components and change values in real-time.

LinMot Talk was mainly used to find out the most efficient way to communicate with the linear motor and become familiar with it's characteristic. The software includes a configurable oscilloscope, where the variables and parameters of interest can be plotted. This allowed us to determine the optimal performance of the motor in the given circumstances. The software is also needed to flash the configurable firmware onto the linear motor's drive. This needed to be done once, as all of the parameters and commands can be changed afterwards via CANOpen.

To debug our programmed software and monitor the can bus, we used a PCAN USB adpater, together with the PCAN View software to send and receive messages and PCAN Stats to manage the CAN network and adjust the baud rate.

Considering Matlab Simulink required the most work, we will now elaborate on our process of creating our Simulink model.

Chapter 4

Implementation

4.1 Steering

As already mentioned in chapter one, we used a compact power steering unit, with a rated torque of around 3 Nm. In order to install the unit, the kart's steering column was cut up, as was the power steering's drive shaft. The column and to the drive shaft were then joined together, by using press-fit joints.

4.2 Braking

Braking presented itself to be the most difficult aspect in the drive-by-wire system. The kart offered limited space, therefore powerful actuator was needed. Early testing showed that for a full brake roughly 250-350 N of force was required. However, this testing was done with the kart moving, therefore we could not fully rely on these results.

Early on we set a few requirements for our brake actuator, namely speed, force and precision. After some research we decided to look further into a high end linear motor by LinMot and neglect the idea of a rotary motor. Our main reasons being that a rotary motor generally was significantly slower and usually required external sensors to be precise enough. With our battery providing 48 VDC, LinMot's range of motor narrowed down quite a bit. The PS01-48x240F-C was our first idea, as it provided up to 572 N of maximum force. We quickly realized that our limited space would not allow for such a big motor. After a personal consultation with one of LinMot's employees, we decided for a PS01-37x120F-HP-C with a 300 mm runner. Because of the smaller size, we now had various options of where to put the linear motor. Again, we strived for an easy installation. Therefore we determined the best location to be on top of the brake cylinder, just in between the two pedals. Pushing the pedal, resulted in a small vertical downward motion of the point where the motor's runner was attached to the lever. As the linear motor should not be under radial load, a design was realized with bearing on the front end, which allowed the motor to tilt slightly. This helped reduce the radial load, as the downward motion could be compensated with tilting the motor at an angle.

4.3 Throttle

To control the throttle, we had to access the ACD 4805 motor controller of each electric drive. After a simple start up sequence, we were able to control the velocity of each wheel via CANOpen.

4.4 Testing

To find the optimal parameters of the linear motor, we used a simple spring setup and the LinMot Talk software. With the software, we were able to adjust any motor parameters in real-time, have it perform different moves at various speeds and accelerations. On the spring gauge, which was clamped in between the linear motor and the beam, we could read off the force and compare it to the calculated force in LinMot Talk. We found out that the maximum stroke was mostly limited by the maximum velocity. The maximum required stroke of around 55 mm can be reached with a maximum velocity of around 1.5 m/s and maximum acceleration. A maximum velocity higher than that, leads to the motor shutting down, which is something we certainly wanted to avoid.

Another test was performed to check the brake's force and speed. Using simple buttons to trigger a move by the linear motor on the LinMot drive, the driver was able to brake the kart at his command. Starting of with a rather short stroke, resulted in a minimal braking force. It however proofed our concept, and showed that the linear motor can be used while on the kart. We were there independent of external power supplies as well as our laptops. As soon as we increased the stroke and therefore the brake force, one of our fuses blew. We hadn't realized, that the fuse was rated for up to 10 A. The linear motor's peak current however was around 25 A. This is why the fuse blew, and we replaced it with a appropriate fuse. Now that we were able to increase the stroke to maximum length without risking short circuit, the brake force was enough to stop the wheels at almost full speed. This showed, that our brake design fulfilled our requirements of force and speed. The precision of the linear motor was not yet determined fully, which would follow in subsequent tests.

Chapter 5

Communication

To handle the communication between all devices, a fast and easy to implement data exchange method was required. RIMO, as well as the other manufacturers of our components made use of the standardized industrial application CANOpen. Because CANOpen is based on CAN, we will first describe the CAN bus protocol.

5.1 CAN Basics

CAN stands for Control Area Network and consists of two main layers, namely the physical layer and the data link layer. CAN was developed in 1986 and is used for data exchange between different stations in a network, based on serial communication. Messages are received and transmitted via broadcasting, making every message available for all of the connected stations.

The rise in electronification in many parts of the industry called for simpler and more efficient means of communication. Extensive wiring still resulted in rather limited data exchange. A way out of this was presented by serial bit data exchange and connecting up all electronic control units to a single bus. Depending on the bus length, CAN offers up to 1 Mbit/s of data rate, while remaining robust and reliable, even in a noisy environment.

The physical layer consists of a twisted pair of wires, which can be shielded if required. The value of a bit was determined by the difference in voltage of the two wires. If the voltages are the same, the bit is recessive. If the difference is higher than 0.9 V, the bit is dominant. As both wires are affected by the same electromagnetic disturbances, their difference in voltage will not vary. The data link is therefore immune to electromagnetic disturbances. By twisting the wires, the magnetic field generated will also be reduced significantly.

To reduce reflections in the data cables with rates higher than 125kbit/s, it is recommended to terminate the ends of the communication lines with termination resistors with 120 Ohm.

Each CAN message contains the following structure.

Especially the role of the identifier bit is important, as it handles and represents the message priority.

5.2 CANOpen

CANOpen is a higher-layer protocol based on the aforementioned CAN. In the following, we will provide information on the project-relevant aspects of CANOpen. All real-time data is exchanged via process data objects (PDO). The adress of each individual PDO can be found in a standardized table by CAN in Automation (CiA). PDO's contain data from a single or different objects from the object dictionary. This maps each bit of the data section of a PDO to a certain object. To change PDO mapping or customise parameters in the object dictionary, service data objects (SDO) are used. These however do not transmit and receive real-time data, but are only used for service purposes. They usually contain the index, subindex and the new value of the respective object.

Each CAN device has a personal node, ranging from 1 to 127. This ensures that every message reaches it's corresponding device.

The standard table of CAN-Id's can be found below.

There are various transmission types avaible in CANOpen for PDOs. Some are manufacturer specific, therefore only the types relevant for our project will shortly be described.

Transmission type 0 (acyclic synchronous)

Transmission type 1-240 (cyclic synchronous)

This transmission type relies on sync messages. After every n sync messages, the PDO transmits it's data. So for example, if the transmission type is 6, the PDO will send after every 6th sync message.

Transmission type 254/255 (asynchronous)

These transmission types are event-triggered, where the event is manufacturer specific for 254 and for 255 defined in the CANOpen device profile.

COB	**Function Code**	**Resulting CAN-ID**
NMT	0000_b	$0(000_h)$
CODE	0001_b	$128(080_h)$
TIME	0010_b	$256(100_h)$
EMCY	0001_b	$129(081_h)$– 255 ($0FF_h$)
PDO1(tx)	0011_b	$385(181_h)$ – 511 ($1FF_h$)
PDO1(rx)	0100_b	$513(201_h)$ – 639 ($27F_h$)
PDO2(tx)	0101_b	$641(281_h)$ – 767 ($2FF_h$)
PDO2(rx)	0110_b	$769(301_h)$ – 895 ($37F_h$)
PDO3(tx)	0111_b	$897(381_h)$ – 1023 ($3FF_h$)
PDO3(rx)	1000_b	$1025(401_h)$ – 1151 ($47F_h$)
PDO4(tx)	1001_b	$1153(481_h)$ – 1279 ($4FF_h$)
PDO4(rx)	1010_b	$1281(501_h)$ – 1407 ($57F_h$)
SDO(tx)	1011_b	$1409(581_h)$ – 1535 ($5FF_h$)
SDO(rx)	1100_b	$1537(601_h)$ – 1663 ($67F_h$)
NMT error control	1110_b	$1793(701_h)$ – 1919 ($77F_h$)

Table 1, COBs and there corresponding CAN-IDs

5.2.1 Implementation

ACD

In order to communicate with the ACD, the CAN Node-Id of each ACD had to be determined. One of Rimo's pdf files, namely "Setting up ACD Controller & Connection Diagram", serves exactly this purpose. To find the respective node-id's, we had to check if either pin 12 (DI5) or pin 20 (DI6) was connected to any other pins on the ACD 4805 K1 connector. In our case, pin 12 of the right ACD was connected to pin 1, making it node 6. Pin 12 of the left ACD was not connected and therefore making it node 5. Afterwards our findings were confirmed when we connected the go-kart to our CAN bus. With the help of a CAN-USB adapter, we were able to receive and send messages, and after a while controlling the kart. Because the go-kart does not communicate via CAN by default but only for service and remote control purposes, we had to put all nodes into operational mode by sending a NMT start messages to all nodes, namely node 5 and 6. The ID of a NMT message is 0x000, to start the nodes the instruction code 0x01 had to be used. To reach all nodes simultaneously, the node adress needed to be 0x0.

Issues with CANOpen and dSpace.

When we tried to control the ACD via CANOpen with the Microautobox, the communication presented itself to be an issue. The wheels were not turning smooth, but rather interrupted from time to time. It was clear that something had to be disrupting the communication. With the help of a CAN-USB adapter, we were able to monitor the CAN bus. After checking all physical layers, we came to the conclusion that it must be a software problem. Our first approach was to change synchronisation times. This however did not have any effect, neither did changing the baud rate or adjusting the step size for Matlab's solver.

We established connection between MABX and a receiver, in our case a laptop connected to the MABX CAN via CAN USB.

Two's complement is convenient way to store integers, such that adding and subtracting with negative numbers becomes very easy. This was used on the ACD 4805, where certain values, such as the rotational speed of the wheels, can be negative. The basic principles of two's complement are the following.

- Zero is represented by all 0's. e.g. 0 0 0 0 = 0

- The maximum positive integer is $2^{numberofbits-1}-1$. So for 4 bits, the biggest integer is therefore 0 1 1 1 = 7, and not 1 1 1 1 = 15 as in the standard notation.

- if the integer is negative, 1's and 0's switch roles, starting from all one's, e.g. 1 1 1 1 = -1. This increases the range for negative numbers by one.

So for 4 bits it looks as follows.

0 0 0 0 = 0	0 1 0 0 = 4	1 1 1 1 = -1	1 0 1 1 = -5
0 0 0 1 = 1	0 1 0 1 = 5	1 1 1 0 = -2	1 0 1 0 = -6
0 0 1 0 = 2	0 1 1 0 = 6	1 1 0 1 = -3	1 0 0 1 = -7
0 0 1 1 = 3	0 1 1 1 = 7	1 1 0 0 = -4	1 0 0 0 = -8

So an easy way to find the negative integer of a positive integer is to convert the desired decimal number to binary, inverting all 0's and 1's and then adding 1. A more hands on approach is to again convert to binary, starting from the right to find the first 1 and inverting all bits to the left of it.

So in order to have a rotational speed of -500 revolutions per minute, the following steps have to be taken for a signed 16 bit integer.

500 = 0 0 0 0 0 0 0 1 1 1 1 1 0 1 0 0

Now invert all bits.

1 1 1 1 1 1 1 0 0 0 0 0 1 0 1 1

And add 1

1 1 1 1 1 1 1 0 0 0 0 0 1 1 0 0 = -500

The second method would result in the following steps.

Starting from the right, find first 1.

500 = 0 0 0 0 0 0 0 1 1 1 1 1 0 **1** 0 0

Invert all consecutive bits to the left of it.

1 1 1 1 1 1 1 0 0 0 0 0 1 **1** 0 0 = -500

If a signed integer is postive or negative is easy to spot, as it's most significant bit determines if the number is negative or positive. 1 = negative, 0 = positive.

LinMot

important settings

For some reason, TPDO3 and TPDO4 would not transmit their data when a sync message was sent. This was tried to solve by setting the intern event timer to around 10 ms and changing the transmission type to 254. However, after a while it would stop transmitting out of a unknown reason. As time was scarce, we simply remapped the pdos, such that the needed data would be transmitted via TPDO2.

Another issues presented itself while working on the brake. Previously we used a motion command called VAI go to position 16bit, which takes velocity, acceleration/deceleration and position as input and creates a curve for the linear motor, which results in the motion. However, a new command will only be executed, if the value of the motion command count has changed. In the easiest way bit 0 can be toggled. To avoid this, we tried a different setting, called PV Stream. This uses a constant stream of position and velocity inputs during a fixed streaming period, interpolates and executes the command. While this seemed very intriguing, it's implementation was not possible. For some reason an error arose, stating that our streaming was too slow. Even after checking the period time with PCAN and checking all setting, the issue could not be resolved. After that, we went back to the prior way of setting the position.

Chapter 6

Results and Discussion

Chapter 7

Conclusion

In conclusion, the go-kart fulfils it's requirements. With more time on our hands, we could have planned the project better. There were many unforeseen expenses, which mainly resulted from compatibility issues. Especially dSpace's CANOpen Master Solution led to a lot of issues. While it provided a somehow intuitive interface and helped in creating the needed blocks, it also lacked customisability. Many processes were somehow a black box model. This made debugging and understanding the errors very challenging or even impossible. The lack of documentation and support did not help either.

Chapter 8

Appendix

The following code is the definition of the bibliography entry of the document class IDSCreport [1].

```
@manual{IDSCreportClass,
author = {Andreas Ritter and Philipp Elbert and Christopher Onder},
title = {How to Use the {IDSCreport} {\LaTeX{}} Class},
language = {english},
howpublished = {Version 1.4.0},
organization = {Institute for Dynamic Systems and Control ({IDSC})},
address = {ETH Z\"{u}rich, Switzerland},
month = dec,
year = 2016
}
```

Bibliography

[1] A. Ritter, P. Elbert, and C. Onder, *How to Use the IDSCreport LaTeX Class*, Version 1.4.0, Institute for Dynamic Systems and Control (IDSC), ETH Zürich, Switzerland, Dec. 2016.

Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Institute for Dynamic Systems and Control

Prof. Dr. R. D'Andrea, Prof. Dr. E. Frazzoli, Prof. Dr. C. Onder, Prof. Dr. M. Zeilinger

Title of work:

FUCK DSPACE

How a company stole our money

Thesis type and date:

Bachelor's Thesis, December 2016

Supervision:

First Supervisor
Prof. Dr. Second Supervisor

Student:

Name: Hans Muster
E-mail: muster@student.ethz.ch
Legi-Nr.: ??-???-???
Semester: ?

Statement regarding plagiarism:

By signing this statement, I affirm that I have read and signed the Declaration of Originality, independently produced this paper, and adhered to the general practice of source citation in this subject-area.

Declaration of Originality:

`https://www.ethz.ch/content/dam/ethz/main/education/rechtliches-abschluesse/leistungskontrollen/declaration-originality.pdf`

Zurich, 23.6.2017: _____