KATHOLIEKE UNIVERSITEIT
LEUVEN

Faculteit
Ingenieurswetenschappen

Hardware Implementation and Control Strategy of a High-Dynamic CVT Transmission for an E-Bike

Jorrit Heidbuchel

Thesis submitted for the degree of
Master of science in
Mechanical Engineering

Supervisors:
Prof. dr. ir. W. Desmet
Dr. ir. B. Pluymers

Assessors:
M. Versteyhe
J. Croes

Mentors:
Ir. T. Keppens
Dr. ir. F. Cosco

Academic Year 2016 - 2017

Preface

I want to thank everybody who helped me with this thesis. Special thanks goes to Tomas Keppens for his great mentorship and inspiring enthousiasm. Also thanks to Francesco Cosco for the support and for always being available on short notice. Thank you to my promotors Wim Desmet and Bert Pluymers for aiding in acquiring the needed material and connections. I'm most grateful to my parents for supporting me in my education. My brother Rugen deserves a word of thanks for helping me out with some computer issues during the year. I would also like to thank the jury for reading this text.

Jorrit Heidbuchel

Table of Contents

Abstract

This text is about the hardware implementation and control strategy of a high-dynamic CVT transmission for an electric bicycle. Starting from a concept by ir. Tomas Keppens and from work done in the previous years, the goal of this year's thesis is to take the concept to a first operational hardware prototype. This task is divided into three subtasks. In a first step, the Simulink virtual testing environment and Kalman filter-based control of the bicycle CVT made in last year's thesis by Simon Deruyter are adapted to work in a hardware prototype. Deruyter's code was made for virtual testing on a desktop computer and was not designed for embedded operation on a microprocessor. The hardware configuration in the prototype is also different from the configuration Deruyter used in his work, so the bicycle model and Kalman filter need to be adapted to this new configuration. The control of the bicycle was only used last year for testing the Kalman filter performance in Simulink simulations and was not yet made for real-world operation. A new control that takes into account real-world, transient behaviour, system nonidealities and hardware limits is developed in this thesis. In the second step, the control program needs to run embedded on the prototype. For convenience and debugging capabilities, a three-processor setup is used with a Raspberry Pi minicomputer as a master controller, running the main control program containing the Kalman filter and the bicycle cadence and support level control algorithm. Two smaller MCU's programmed with Texas Instruments motor control software control the two electric motors in the bicycle, communicating with the master processor through SPI. The Raspberry Pi is programmed directly from Simulink using Simulink coder/ Embedded coder. Debugging the master processor is done via external simulation mode in Simulink. The third step is testing the prototype and tuning the control and the Kalman filter. Placing the bicycle on rollers, the Kalman filter covariances are tuned and the measurement covariances are determined. The behaviour of the cadence control can not be validated because the test setup on rollers is not representative for riding on the road. The behaviour of the Kalman filter is very satisfactory. In the last part of this text, the bicycle-cyclist system is looked at from a human-in-the-loop perspective. The problems of optimal cadence choice and intuitive control of the bicycle are discussed based on publications on optimal cadence for professional cyclists and on human-in-the-loop models for cars. A reliable human behavioral model and good estimation of not only the bicycle states but also of the riding mode and cyclist intention will be needed for optimal and intuitive control.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviatons

ADAS
Advanced Driver Assistance Systems 78
ADC
Analogue to Digital Converter 47
CCS
Code Composer Studio 46
CVT
Continuously Variable Transmission 13
DC
Direct Current 16
DVD
Cambridge university Driver-Vehicle Dynamics group 79

Symbols

F_g	Gravitational load [N]
m	Mass of the bicycle and cyclist [kg]
g	Gravitational constant
α	Road gradient [rad]
F_f	Friction force [N]
F_{fs}	Static friction force [N]
F_{fv}	Dynamic friction force [N]
c_r	Rolling friction coefficient
c_w	Viscous friction coefficient
F_{aero}	Aerodynamic drag [N]

c_x	Aerodynamic drag coefficient
A_{aero}	Frontal area of cyclist and bicycle [m^2]
ρ_{aero}	Air density [kg/m^3]
T_{load}	Load torque on bicycle [Nm]
r_w	Bicycle wheel radius [m]
ω_r	Rotational speed of ring gear [rad/s]
ω_s	Rotational speed of sun gear [rad/s]
ω_c	Rotational speed of planet carrier gear [rad/s]
ρ	Ratio of sun gear radius over ring gear radius in planetary gear set
r_r	Radius of ring gear [m]
r_s	Radius of sun gear [m]
r_c	Radius of planet carrier gear [m]
P_r	Power delivered by ring gear [W]
P_s	Power delivered by sun gear [W]
P_c	Power delivered by planet carrier gear [W]
T_r, T_{ring}	Torque on ring gear [Nm]
T_s	Torque on sun gear [Nm]
$T_c, T_{carrier}$	Torque on planet carrier gear [Nm]
θ_{sun}	Angle of the sun gear [rad]
$\theta_{carrier}$	Angle of the planet carrier gear [rad]
ω_{sun}	Angular velocity of sun gear [rad/s]
$\omega_{carrier}$	Angular velocity of planet carrier gear [rad/s]
$T_{cyclist,offset}$	Cyclist torque offset [Nm]
$T_{cyclist,max}$	Cyclist torque dubble amplitude [Nm]
ϕ	Cyclist torque phase offset [rad]
T_{MG1}	Torque delivered by MG1 [Nm]
T_{MG2}	Torque delivered by MG2 [Nm]
α_{eq}	Equivalent road gradient [rad]
$T_{cyclist}$	Cyclist total torque [Nm]
$\theta_{cyclist}$	Crankshaft angular position [rad]
$\boldsymbol{x}$	State vector
$\boldsymbol{u}$	Input vector
$\boldsymbol{y}$	Measurement vector
$\boldsymbol{w}$	Process noise vector
$\boldsymbol{v}$	Measurement noise vector
A, B, C, D	State-space matrices in continuous state-space model
θ_{MG1}	Angular position of MG1 [rad]
θ_{MG2}	Angular position of MG2 [rad]
α_{sun}	Angular acceleration of the sun gear [rad/s^2]
$\alpha_{carrier}$	Angular acceleration of the planet carrier gear [rad/s^2]
$T_{MG1,meas}$	Measured MG1 torque [Nm]
$T_{MG2,meas}$	Measured MG2 torque [Nm]
$K_{ring,crank}$	Gear ratio between ring gear and crankshaft
$K_{sun,MG1}$	Gear ratio between sun gear and MG1 shaft
F, G, C, D	State-space matrices in discrete state-space model
h	Timestep of discretisation [s]
n	Number of states in the Kalman filter
k	Index for timestep

I	Identity matrix
ω_{crank}	Angular velocity of the crankshaft [rpm] or [rad/s]
$\omega_{cyclist,opt}$	Optimal cadence (crankshaft speed) for the cyclist [rpm]
C_v	Velocity correction factor
C_T	Torque correction factor
v_{fix}	Velocity at which switching to CVT mode occurs [m/s]
v_{switch}	Velocity at which optimal cadence is maintained [m/s]
$T_{threshold}$	Torque threshold for non-zero torque [Nm]
$K_{crank,wheel}$	Gear ratio between crankshaft and rear wheel
$T_{s,control}$	Sample time of the control algorithm [s]
$T_{rear\ wheel}$	Torque exerted on the rear wheel by the driveline [Nm]
$T_{front\ wheel}$	Torque exerted on the rear wheel by MG2 [Nm]
f_s	Torque support factor
$a_{desired}$	Desired acceleration of the bicycle [m/s^2]
c_1, c_2	Constants in the linear decay acceleration model [m/s^2], [1/s]
$v_{desired}$	Desired end velocity after acceleration manoeuvre [m/s]
f_s^d	Differential torque support factor
$\boldsymbol{Q}$	Process noise covariance vector
$\boldsymbol{R}$	Measurement noise covariance vector
v_{min}	Minimum velocity to estimate road gradient [m/s]
T_{ampl}	Cyclist torque amplitude [Nm]
T_{offset}	Cyclist torque offset [Nm]

1 Introduction

1.1 Electric bicycle project

This master's thesis is one of several theses in the context of a project started by Ir. Tomas Keppens, concerning the development of a next-generation E-bike. The goal of the project is to make an electrically assisted bicycle that is intuitive and therefore safe to use, with a simple hardware configuration, making it both reliable and economical. These goals are set to meet future expectations of the growing market of electrical bicycles. Total E-bike sales are around 20% of total car sales in Europe. Worldwide, two E-bikes are sold for every five cars (Figure 1). The market has known an impressive growth over the last decade, E-bikes quickly gaining share in the total bicycle sales (Figure 2).

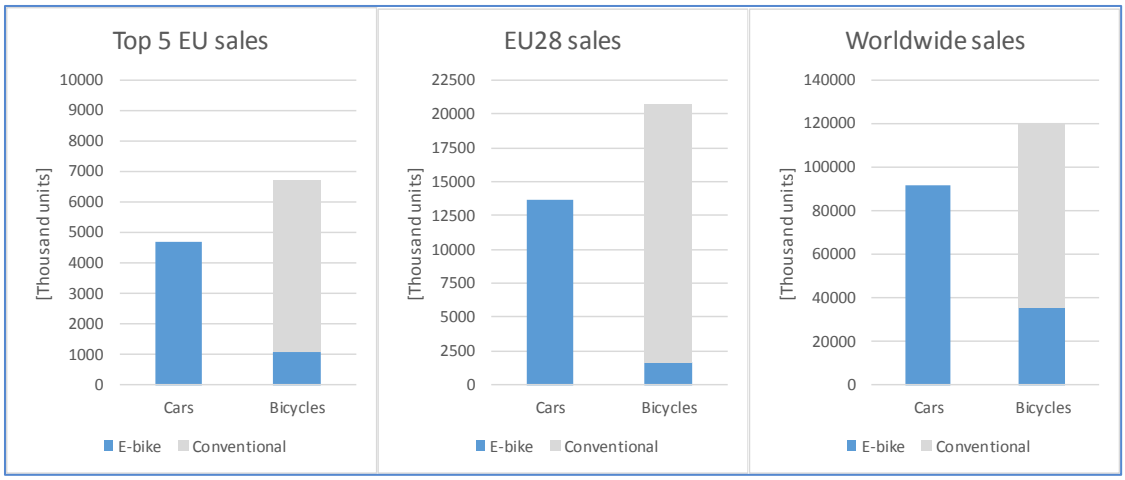

Figure 1: Comparison between bicycle sales and car sales worldwide and in the EU. Top 5 EU: Germany, Netherlands, Belgium, Austria, Denmark.

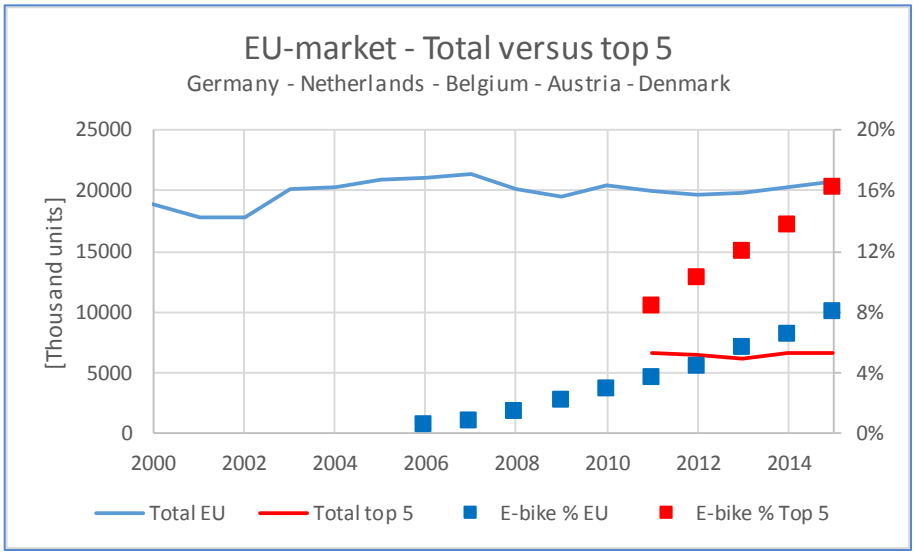

Figure 2: Bicycle sales and E-bike share 2000-2015.

The typical E-bike nowadays is characterised by both complex hardware and by a complex human-machine interface. Figure 3 shows a typical modern E-bike on the left of the figure. On the right is a close up of the Bosch drive system for electric bicycles. In this drive system, there are two one-way clutches, a torque sensor to measure the cyclist torque, and a lot of gears to make the coupling between the built-in electric motor and the crankshaft. The electric motor turns at speeds of about 3000 rpm, whereas the crankshaft typically rotates at 70 rpm. Apart from this drive unit, there are still manual derailer gears, which involve a freewheel in the rear hub, and also manual brakes. This makes the human-machine interface rather complex. For accelerating, the cyclist has to push on the pedals. This torque is then supported by the electric motor working in parallel to it. For decelerating on the other hand, you cannot use the pedals nor the electric motor in the drive unit. Deceleration is done by the hand brakes. Choosing the cadence at which the cyclist wants to cycle requires manually shifting gears, which is yet another control input mechanism. The derailer gear requires a freewheel. This means that the electric motor is not always coupled to the bicycle. The electric motor can therefore not be used for regenerative braking.

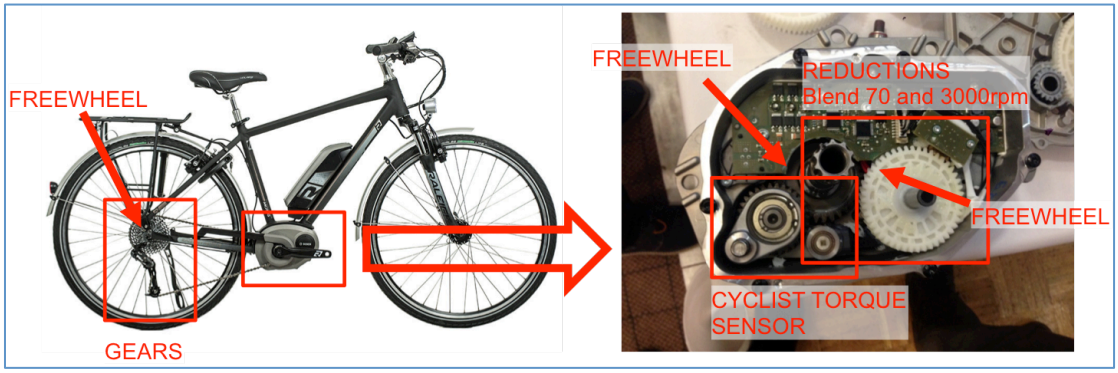

Figure 3: Left: conventional electric bicycle. Right: internals of the Bosch drive system.

The human-machine interface might be simplified by eleminating all but one input mechanisms. The obvious choice is to keep the crank as only input. To be able to do this, there must always be an input coming from the crank. Or rather: the bicycle should always be aware of what the cyclist is doing with the crankshaft. This means that the mechanical freewheel – the one-way clutch – needs to be eliminated. Only then can the bicycle always measure the torque the cyclist puts on the pedals. Without the freewheel, derailer gears cannot work. Besides, they make things more complex because the cyclist needs to shift gears manually. In order to get rid of these derailer gear and the one-way clutch, the bike uses a continuously variable transmission (CVT) instead that is also capable of an “infinite” gear ratio, i.e. the crank standing still with the wheels of the bike rotating (cfr. freewheel behaviour). The gear ratio between the crank and the driven wheel can now be adjusted automically in a continuous way. This control will have to be high-dynamic since the connection between the crank shaft and the bike can no longer be interrupted by a freewheel. The control will have to be able to create the freewheel behaviour and decouple the crank from the bike via the CVT system. The layout of the CVT system will be discussed later. It is an electrical system based on the Toyota Power Split Device (PSD). The electric motor serves as both a support motor and controls the gear ratio between crank

and bicycle wheel. Inherent to this system is that the support level (= amount of extra power delivered by the electric motor) provided by the motor is not indepent of the gear ratio control. By choosing the gear ratio, one also chooses the support level. To make the support level adjustable a second motor is added to the bicycle.
Very important in the control of the bicycle is, as mentioned before, reading the cyclist torque at all times. Last year's student Simon Deruyter worked on an estimator that can measure the cyclist input torque. The goal of this thesis will be to continue the work on this estimator and to implement it on a hardware prototype. With this prototype, tests can be performed to validate the estimator and the CVT control algorithm. A performant estimator and control algorithm are the first step towards controlling the bicycle using cyclist behavioral models. These models will have to link intention of the cyclist (accelerate, decelerate, emergency brake,…) to cyclist behaviour, i.e. in this case cyclist torque on the crank shaft. Using this cyclist behavioral model, the bicycle control can deduce the cyclist intention from the crank torque and act accordingly, choosing the correct gear ratio and support level.

1.2 Hardware configuration

As mentioned earlier, the electric bicycle this thesis is working on has no gears and no mechanical freewheel. It makes use of a CVT transmission instead. The CVT transmission is based on the Toyota Power Split Device (Figure 4). Toyota uses this transmission system in it's hybrid drive train, the Toyota Hybrid Synergy Drive (HSD) system. Essentially a planetary gear set, the PSD contains two electric motors, one coupled to the sun gear (MG1) and one coupled to the ring gear (MG2). The ring gear is also the output shaft, driving the wheels of the car. The car's internal combustion engine (ICE) is coupled to the planet carrier of the planetary gear set. MG1 is responsible for setting the gear ratio between the planet carrier and the ring wheel, making the PSD work as a CVT transmission. MG2 on the other hand can support or load the ring wheel. This makes it possible to have electrical support in fast accelerations, to drive purely electrical with the ICE turned off, or to use regenerative braking.
In the electric bicycle, the output shaft is not the ring gear but the planet carrier. The cyclist, which essentially replaces the ICE, is coupled to the ring gear instead.

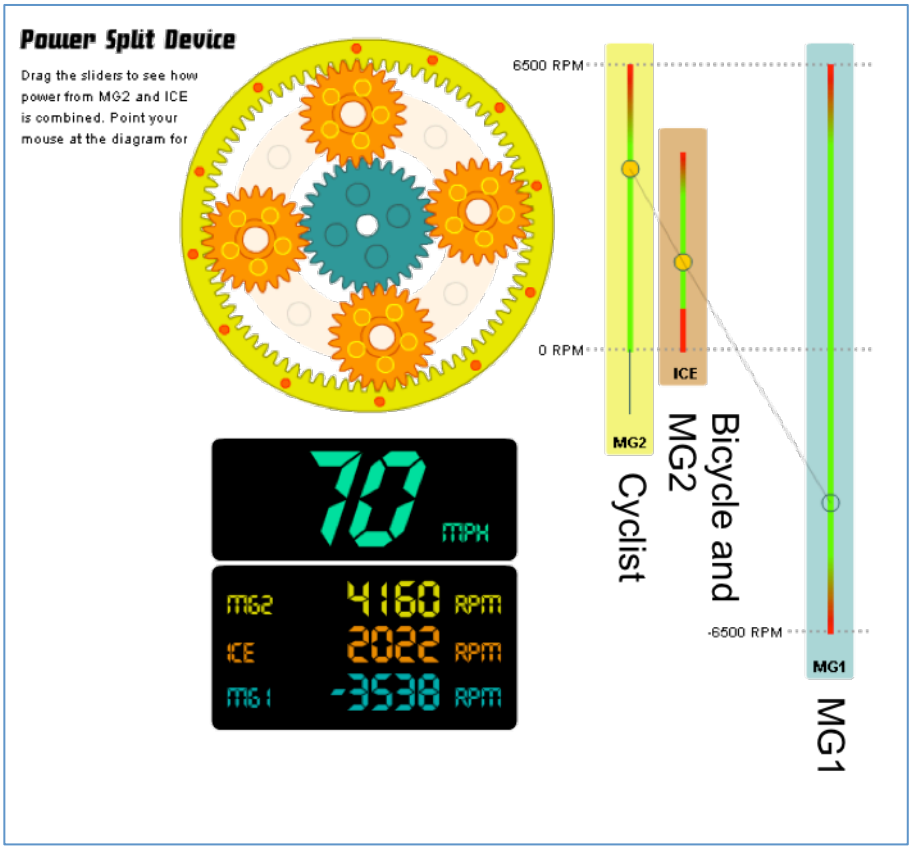

Figure 4: Power Split Device with nomogram. Under the sliders it is indicated what is coupled to the gear. (Source: eahart.com/prius/psd/)

In the Toyota system, when cruising, the power is delivered by the ICE and no net electric power is being consumed. The Prius for example is generally not a plug-in hybrid vehicle, meaning it should not consume energy from the battery in steady state working modes. The battery is only there for transients and to recuperate braking energy. To still have the hybrid behaviour however, MG2 still needs to be powered. MG1 is therefore used as a generator, not as a motor. This means that part of the energy from the ICE is flowing not through the mechanical system to the wheels, but via first converting it to electricity using MG1 and then feeding it to MG2. This serial path means there are serial losses in the system. An electric bicycle is a plug-in hybrid using two power sources: the human and the battery. The battery can continuously assist the cyclist and there is no need to have one MG working as a generator in nominal operation. To have both MG's working as motors, the layout of the system needs a small adaptation. Instead of coupling the wheels and MG2 to the ring gear and the human to the planet carrier, the human (crankshaft) is coupled to the ring gear and the wheels are coupled to the planet carrier. This way, the torque from MG1 and the cyclist work in the same direction and both can work as a power input. Only the wheels act as a load. Figure 5 shows a schematic summary of this.
Notice that, using the system as a CVT, by choosing the gear ratio between crank and wheels, also the support level is fixed. This is because the torque ratio between the gears is always the same. Changing the gear ratio will change the relative speed of the gears but not the relative torques, so the relative power will

change with the speeds, i.e. the gear ratio. MG2 is used as either a motor or a generator to make the support level adjustable.

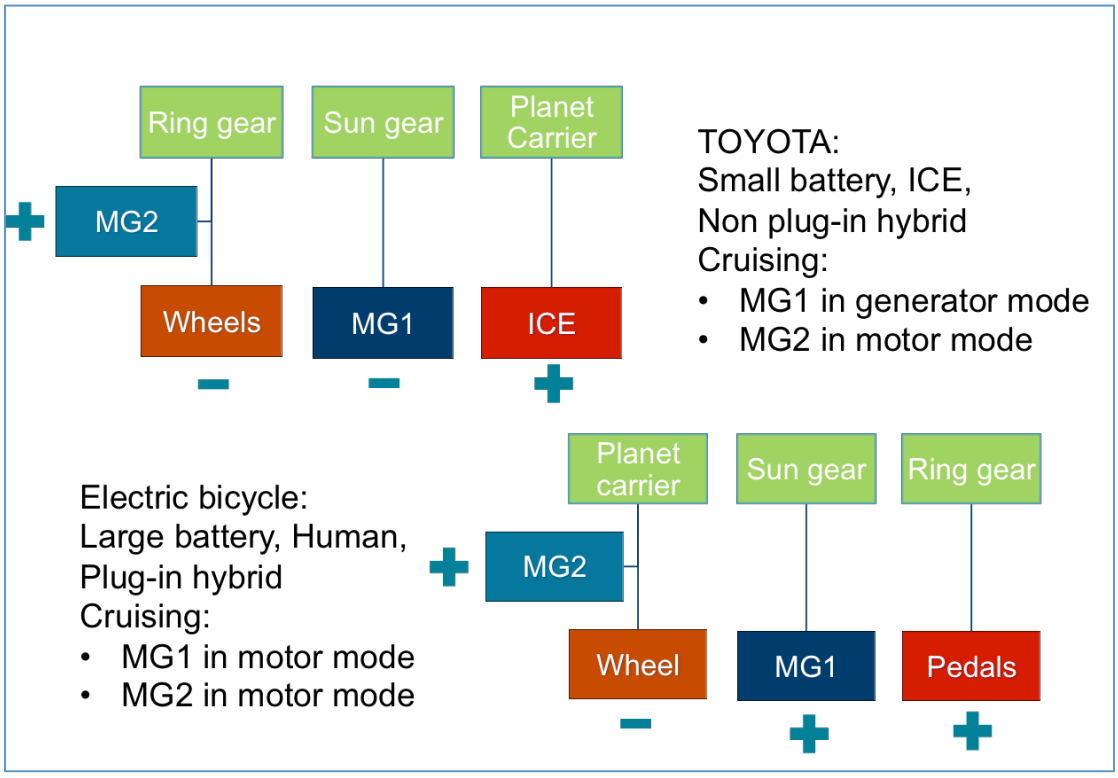

Figure 5: Toyota PSD vs configuration in an electric bicycle. The + and - signs indicate the subsystems working as either a motor (+) or a load (-). (Source: thesis Jonas Tant)

In the prototype made for this thesis by Tomas Keppens, this configuration has been realised as shown in Figure 6. The planetary gear set is integrated into the rear wheel hub. MG1 is connected to the sun gear via a belt drive that adds some reduction. The crankshaft is in the prototype connected to the ring wheel via a chain drive, but without a derailer. Later this chain drive may be replaced by a belt drive, since the chain adds play to the system that will make the estimator and control algorithm less accurate. The wheel is driven by the planet carrier. MG2 is integrated in the front wheel hub (not shown here). This was easier to make then to integrate it in the planetary gear set in the rear hub. Maybe the fact that the bicycle has now two wheel drive will prove to be an added benefit. For the first prototype tests, MG2 is not controlled and the system is used with only MG1.

The motors used in the prototype are 750W brushless DC motors. This gives the prototype a total installed electric power of 1500W, which is a lot for an electric bicycle. The choice was made for these large motors to make sure we would not run into issues concerning motor maximum torque or power. The prototype will be used to investigate the need for these motors. The production bicycle will most probably have two lower-powered motors rather than 750W.

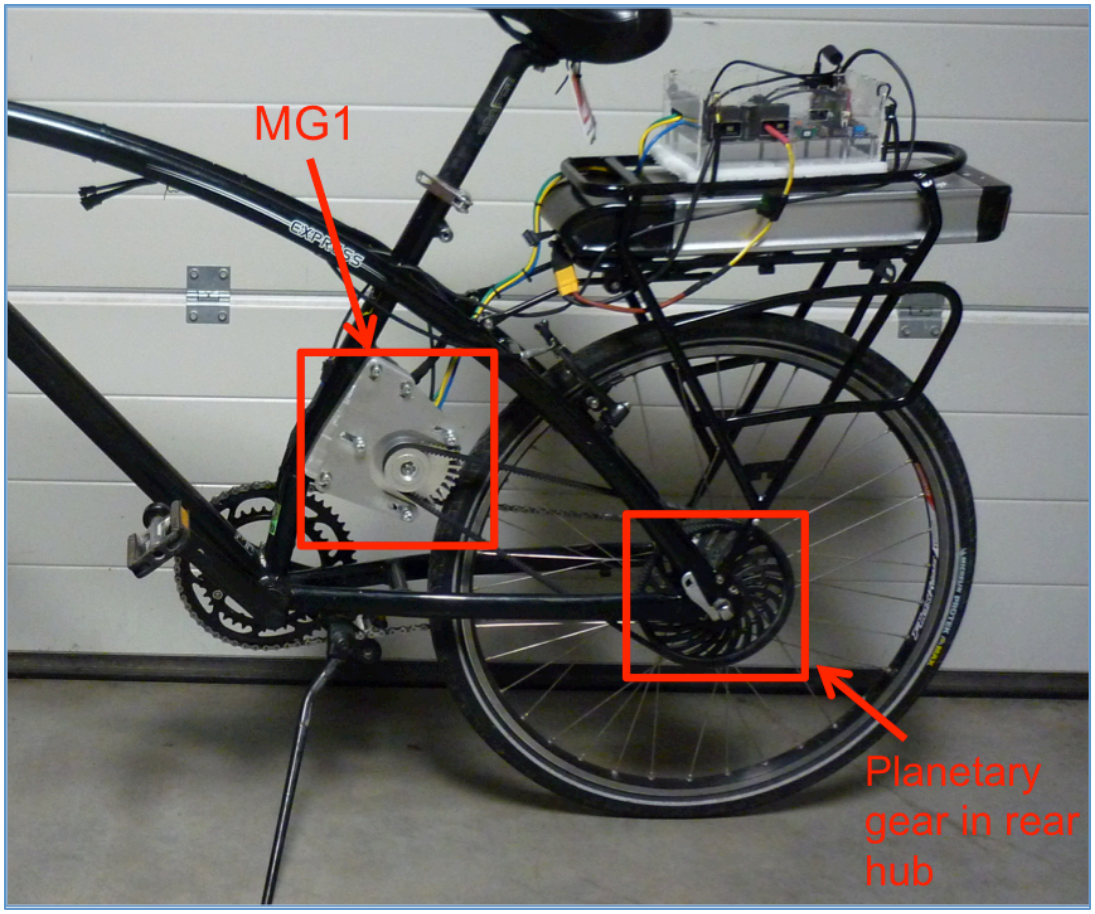

Figure 6: Hardware configuration of the CVT system in the prototype.

1.3 Goal of the thesis

As mentioned earlier, the main objective for this master's thesis is to build a working prototype that can later be used to perform tests on cyclist behaviour. For this, adaptations will have to be made to the work done in the previous years and the system will need to run embedded on a microcontroller. The second objective is to come up with (ideas on) a cyclist model that can be used to deduce cyclist intention from the torque the cyclist applies on the crank shaft. The control algorithm in the bike can then act accordingly and set the cadence or position of the crankshaft, set the gear ratio and adjust the support level.
Starting from last year's master's thesis by Simon Deruyter, the first objective is to adapt and run the estimator for the cyclist torque on an embedded processor. In order to have this estimator work on the prototype it should also run in real time. In last year's thesis, the estimator ran in a simulation only and far from real time. It will be a challenge to run the estimator faster on a smaller processor. Simon Deruyter also made a simple control algorithm for setting the cyclist cadence, gear ratio and support level. This control should also be refined for real world use and implemented on embedded hardware to run real time in order to be able to actually use the prototype for testing. Only when the prototype is fully up and running can the second objective start, namely experimenting on better control strategies that try to bring cyclist intention into account.

2 Previous work

During the past years, several theses have been done in the context of this electric bicycle project. In this chapter, I will give a concise overview of the work done by other students during previous years.

2.1 Modelling and control of first generation bicycle: Mattias Wilberts (2012-2013)

Mattias Wilberts developed a model-based control strategy for the first generation of the electric bicycle. The modelling of the planetary gear set and the load is relevant for this thesis.

2.1.1 Load and losses

The load on the bike consists of three components: a load due to gravity, a load due to rolling friction, and a load due to aerodynamic drag. The rolling friction consists of a viscous part, which is velocity-dependent, and a static part which is constant.

$$F_g = m \,.\, g \,.\, \sin\alpha$$

$$F_f = F_{fs} + F_{fv}$$

$$F_{fs} = m \,.\, g . c_r \,.\, \cos\alpha$$

$$F_{fv} = c_w \,.\, v \,.\, \cos\alpha$$

$$F_{aero} = 0.5 \,.\, c_x \,.\, A_{aero} \,.\, \rho_{aero} \,.\, v^2$$

The three components add up to a total load force that is quadratic in the velocity of the bike. This load force results in a torque on the wheel of the bike as follows:

$$T_{load} = r_w \,.\, \left(F_g + F_f + F_{aero}\right)$$

The total torque equation on the bike wheel is then:

$$\frac{d\omega}{dt} \,.\, (m \,.\, r_w^2 + I_w) = \; T_w - T_{load}$$

T_w is the torque on the wheel from the transmission.

2.1.2 Planetary gear set equations

The CVT transmission is essentially a planetary gear set. The gear set has 3 outgoing axes, ring gear, sun gear and planet carrier, and thus 6 variables: 3 torques and 3 speeds/positions. The kinematic equation for the gear set is:

$$\omega_r + \rho \,.\, \omega_s - (1 + \rho) \,.\, \omega_c = 0$$

The ratio between the radius of the ring over the radius of the sun is often used as a characteristic parameter for the planetary gear set,

$$\rho = \frac{r_s}{r_r}$$

together with the following relationship:

$$r_r = r_s + 2\,.\,r_c$$

There is also the torque equations, derived from the power balance.

$$P_s + P_r + P_c = 0$$

$$T_s.\,\omega_s + T_r.\,\omega_r + T_c.\,\omega_c = 0$$

$$T_r = \frac{T_s}{\rho}$$

$$T_c = -\left(1 + \frac{1}{\rho}\right).T_s = -(\rho + 1).T_r$$

Inertia of the gears is not taken into account in this equation. Between the 3 speeds there is one relationship, which means 2 speeds can be freely chosen and the third follows from these two. Between the 3 torques there are 2 relationships, so chosing one torque fixes all three torques. This is interesting because this means that the motor on the sun gear can derive from the torque it feels the torque of the two other shafts in the gear set. Since one of these shafts is connected to the crankshaft, MG1 can theoretically measure the crankshaft torque without the need for an extra – expensive – torque sensor.

2.1.3 Cyclist torque

Mattias Wilberts modelled the cyclist torque as a simple sine function with two variables: the cyclist mean torque and the crank angle[i]:

$$T_p = T_{p,mean}\,.\left(1 + 0.74\,.\sin\left(2\,.\,\theta_p - \frac{\pi}{2}\right)\right)$$

The 45° phase shift is there to make the top dead center of the pedals the reference position.

2.2 Mechanical layout of the transmission: Stijn Vanrysselberghe (2014-2015)

The mechanical layout of the system was updated by Stijn Vanrysselberghe as compared to the layout used by Mattias Wilberts. The prototype used in this thesis has yet another hardware layout. The most important update in the context of this year's thesis is that Stijn Vanrysselberghe added a second motor to the design, to be able to both adjust the gear ratio and the support level independently. In Vanrysselberghe's thesis, both motors were still combined in a single drive unit together with the planetary gear set. As explained in the introduction, the second motor is now in the front wheel and MG1 is connected to the planetary gear set via a belt drive. Vanrysselberghe's thesis explained the

Toyota hybrid synergy drive system, so I will explain this system also in this chapter.

2.2.1 Toyota Hybrid Synergy drive

The Toyota Hybrid Synergy Drive (HSD) is a hybrid system that is a combination of a purely serial and a purely parallel system. In a serial hybrid system, all the power from the ICE is converted to electricity before this electricity is used to drive the motors that propel the vehicle. This system has the advantage that the ICE can always work in it's optimal working point at every power demand. The battery in the vehicle serves as a buffer for transients and for recuperating braking energy. The main disadvantage of the serial system is that, even when the power output could be perfectly matched to the ICE via a mechanical transmission, the power from the ICE is still converted twice. This introduces so-called serial losses that can be significant and can make the overal efficiency much worse than if the ICE would have been coupled to the load mechanically.
The counterpart of the serial system is the parallel hybrid system. Here the electric motor and ICE work, as the name says, in parallel. They are coupled to the same axle and rotate at the same speed (sometimes with reductions in between) The torques delivered by the two motors add up to the total drive torque. Unlike the serial hybrid, the ICE is not fully free to operate at any working point, since the speed is fixed by the speed of the vehicle as in a traditional car. This means that the best a parallel system can do is to use the electric motor to provide the ICE with it's best possible torque output at the given speed. The electric motor can do this by either working as a generator, creating an extra load on the ICE, or by working as a motor, assisting the ICE. The advantage of the parallel system is that it can both use the electric motor for regenerative braking and transient behaviour, as well as acting as a purely mechanical driveline when the ICE is well-matched to the vehicle load. This eliminates serial losses when e.g. cruising on the highway.
In the Toyota HSD system, both the behaviour of a serial and a parallel hybrid can be achieved to a certain extent. The system contains two electric motors that can also act as generators. These two motors (MG1 and MG2) are coupled to the ICE via the Power Split Device (PSD) which is essentially a planetary gear set. MG2 is set in parallel to the outgoing shaft of the transmission that is connected to the wheels of the car. It is therefore very similar to a parallel hybrid configuration. MG1 is connected to the sun wheel of planetary set, where it controls the gear ratio between the planet carrier (ICE) and ring wheel. The working of the HSD system is easiest explained by looking at some different driving modes.

Acceleration (Figure 7**)**
When accelerating from standstill, initially only MG2 is used to propel the car. When the speed of the car is high enough and more power is required, the ICE is turned on. It is set to it's optimal working point for the power it needs to deliver by regulating the speed of MG1. The power from the ICE is added to the power from MG2. MG1 works as a generator at higher speeds, delivering some of the power needed for MG2 together with the battery. Notice that this is the serial path, so the system works as a serial hybrid also, but only for a fraction of the total power.

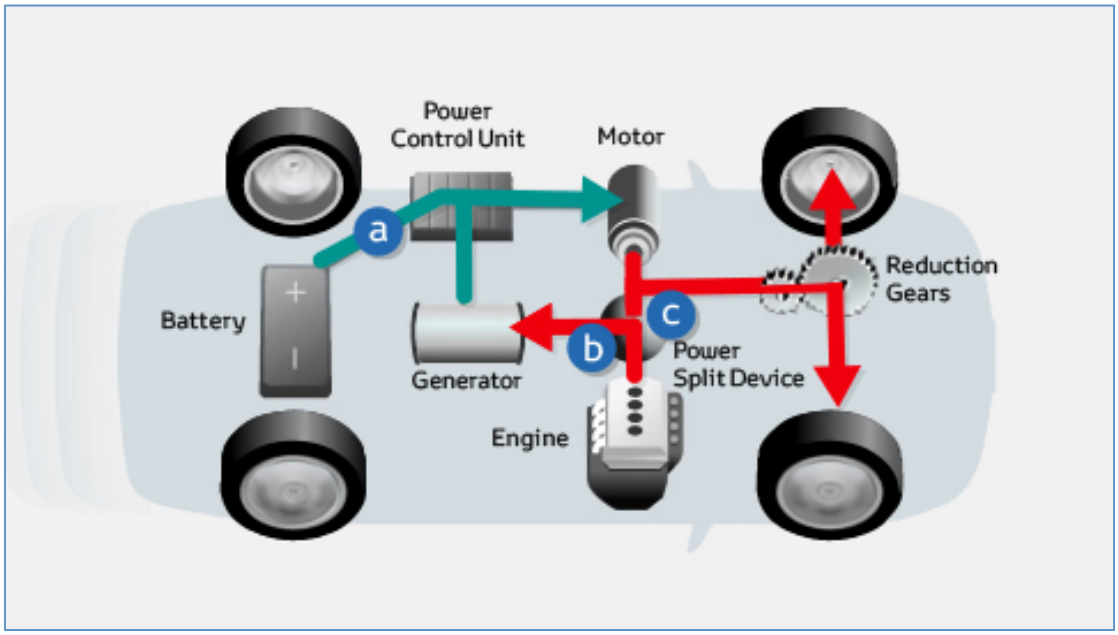

Figure 7: HSD system in acceleration mode.

Cruising (Figure 8**)**

When cruising at a constant speed and thus a constant load, when the battery does not need recharging, the ICE delivers all the power needed for maintaining the vehicle speed. The ICE is set to it's ideal working point for delivering this power by controlling it's speed using MG1. MG1 works as a generator, powering MG2 that works as a motor. Part of the power path is thus again purely mechanical, part is via converting the mechanical power to electricity and back to mechanical power. The system is dimensioned in such a manner that the serial power is minimised, since this introduces extra losses, while the ICE is operated at its most efficient rpm for each requested power output.

When charging of the battery is desired, MG1 is used as a generator, generating more power then MG2 needs. This extra power is used to charge the battery.

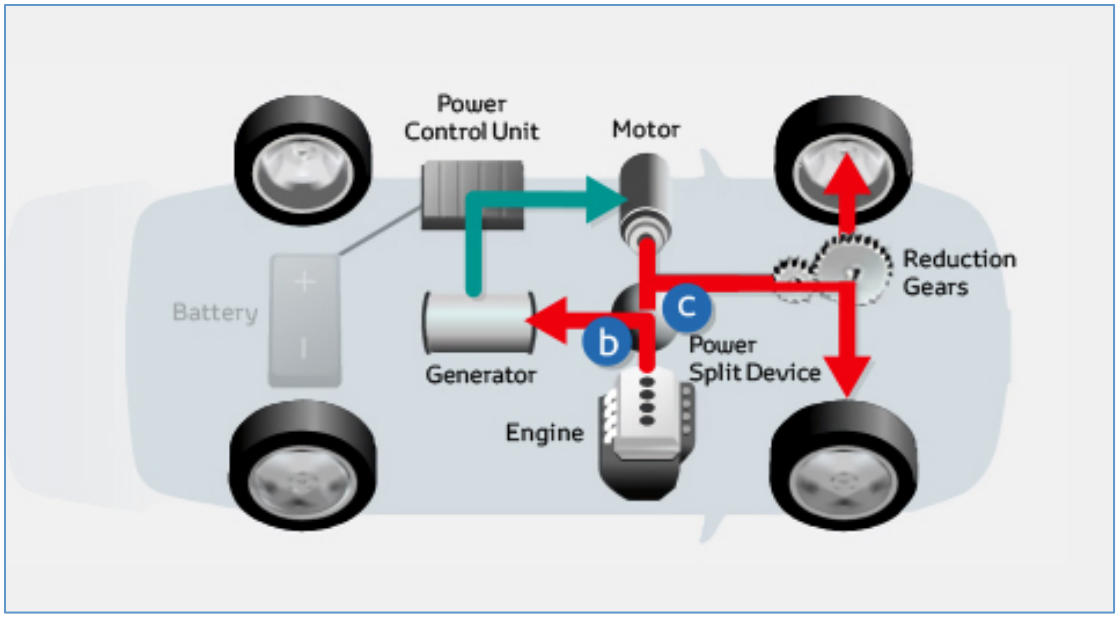

Figure 8: HSD system in cruising mode.

Deceleration (Figure 9**)**

When the battery is not fully charged, MG2 works as a generator recuperating braking energy and storing it in the battery. When the battery is fully charged, conventional engine braking is performed. This is the exact opposite of the acceleration procedure. MG2 now works as a generator, providing MG1 with the electricity it needs to control the gear ratio to set the ICE at it's desired working point.

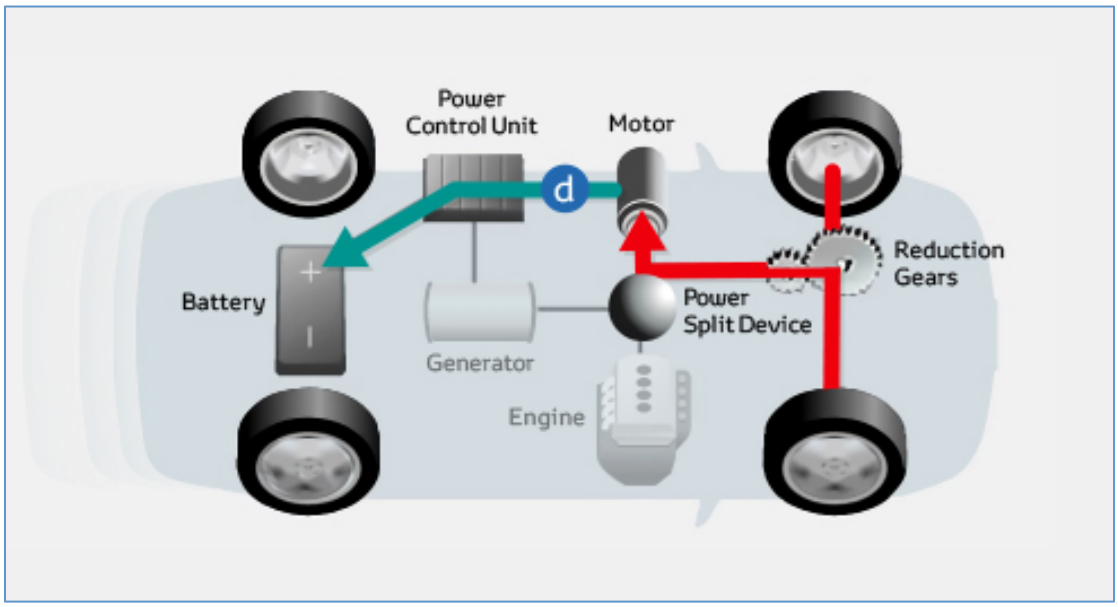

Figure 9: HSD system in deceleration mode.

2.3 Motor control options: Thomas Vleeschouwers (2014-2015)

Thomas Vleeschouwers investigated different control options for running the motors. The main focus was on comparing sensored motor control and

sensorless field-oriented control (FOC). Texas Instruments provides a very extensive set of motor control tools for sensorless motor control. Thomas Vleeschouwers compared these control tools from TI to a standard PI controller and to a sensored controller. For the TI motor control he used a TI F28069M microcontroller and a DRV 8301 motor driver board. Texas Instruments makes use of a so-called FAST (Flux, Angle, Speed, Torque) estimator to not have to use sensors. Except for very low-speed operation, the sensorless control performed as good as the sensored controller. The sensor can therefore be omitted to simplify the system, if the bicycle control can be made so that the motor control can stay away from the low speed operation. Also a conclusion from Vleeschouwers' thesis is that the TI motor control algorithm (SpinTAC) outperforms a standard PI controller. Based on these conclusions, this thesis will use TI motor control for running the two motors in the prototype. Also the TI hardware used by Thomas Vleeschouwers will be used in this thesis for the motor control. However, because of the required stability of MG1 in the bicycle around zero rpm, Hall sensors will be used in the prototype for position measurements of the motors.

2.4 Simulink model of cyclist and bicycle and Kalman state estimator: Simon Deruyter (2015-2016)

Simon Deruyter made a complete simulink model of the bicycle and of the cyclist input. This model can be used as a framework for performing simulations to test control strategies for the bicycle. Deruyter also made a Kalman state estimator to estimate the bicycle states: motor torques, speeds of the planetary gear set, cyclist torque and road gradient. This estimator was proven in Deruyter's thesis to make directly measuring the cyclist torque unnecessary. Based on this estimator, Deruyter also made a simple control algorithm to perform some simulations. Most importantly it was tested whether the cyclist torque estimation is fast enough for a freewheel-like behaviour to be simulated by the CVT transmission. Although the response was not as fast as in a normal bicycle, it is believed to be fast enough for comfortable operation. This will have to be tested on the hardware prototype.

2.4.1 Bicycle model

Using Simulink and Simscape, a complete model was made to simulate the behaviour of the electric bicycle. This model contains a model for the CVT transmission, made up of a planetary gear set and several reductions. Also the play of the chain drive and friction is included in the model. The driveline model is made as a sub-block with four mechanical connections (Simscape): MG1, MG2, Cyclist and Wheel. There is also a data output on the block. These mechanical connections are coupled to models of the respective subunits. MG1 and MG2 are modelled as brushless DC motors. MG1 is a speed controlled motor, MG2 is torque controlled. The control is via a simulink standard block (BLDC motor drive) and uses a PI controller for the motor control. The motor blocks get their target speed and torque as input, as well as two electrical voltage values, coming from the battery. They have a mechanical rotational port that is connected to the driveline. The cyclist is modelled based on research done by Simon Deruyter and on literature data. From literature, the measured torque shape of the cyclist is used as a normalised basis that is then scaled with a torque value, yielding the

output torque the cyclist exerts on the crankshaft. Deruyter also added a mass-spring-damper comfort model to take into account the reaction of the cyclist's legs on shocks coming from the driveline. In the cyclist model, there is also an emergency stop input. This input triggers a sudden stop in the cyclist's leg movement. The emergency stop was added to test the reaction of the bicycle to a sudden stop. As the last subunit, the wheel mechanical connection is coupled to a load model similar to the one used by Mattias Wilberts. The load on the bike is velocity and gradient dependent. The gradient is given as a parameter of the simulation. The velocity of the bike is translated to rotational speed of the wheel. The load equation is therefore written as a function of bike wheel rotational speed rather then bicycle speed.

2.4.2 Kalman state estimator

Simon Deruyter programmed a Kalman state estimator that is used to avoid measuring the cyclist torque directly via a crankshaft torque sensor. Theoretically this is not necessary as the planetary gear set defines that the torque of MG1 is always a measure of the torque put on the ring gear, i.e. the cyclist torque. The Kalman estimator is a combination of model-based feedforward estimation and sensor-based correction on the estimation.
The state vector for the bicycle is:

$$\boldsymbol{x}^{\boldsymbol{T}} = \left(\theta_{sun}\ \theta_{carrier}\ \omega_{sun}\ \omega_{carrier}\ T_{cyclist,offset}\ T_{cyclist,max}\ \ \phi\ \ T_{MG1}\ \ T_{MG2}\ \ \alpha_{eq}\right)$$

An important thing to note in this state vector is that the cyclist torque is not estimated as a single value. This did not yield satisfactory results because the torque changes dynamically with crank angle. Instead, a simple sine shape is presumed for the cyclist torque, as in the thesis by Mattias Wilberts, giving 'static' state parameters during cruising. The definition of the sine shape is somewhat peculiar. Figure 10 shows the meaning of the three cyclist torque states.

$$T_{cyclist} = T_{cyclist,max} \,.\, \left(0.5 \,.\, \sin\left(2 \,.\, \theta_{cyclist} + \phi\right) + 0.5\right) + abs\left(T_{cyclist,offset}\right).\, sgn\left(T_{cyclist,max}\right)$$

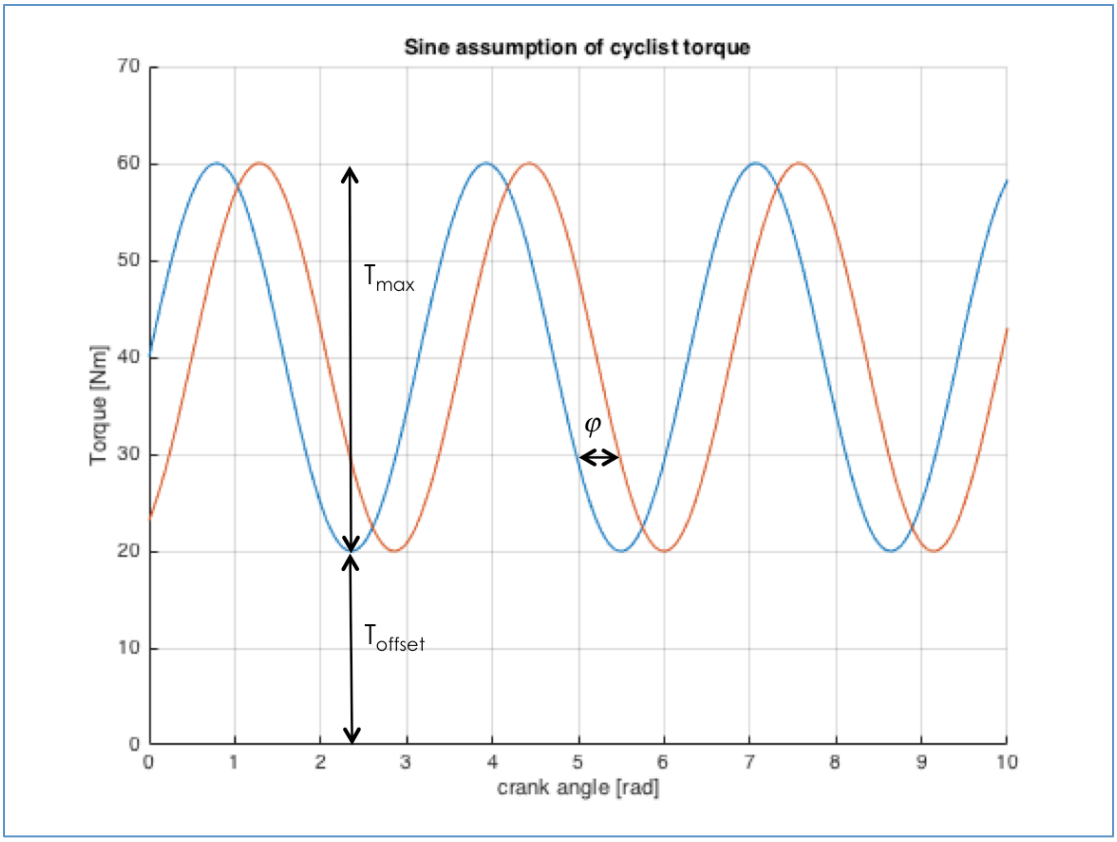

Figure 10: Meaning of variables in simple sine assumption for the cyclist torque.

The gradient state is added because the load torque on the bike is also depending on the road inclination. Not adding the gradient would make the bike model in the estimator inaccurate when riding uphill or downhill. It also accounts for losses not described in the model, in the form of an equivalent gradient. The state-space model is generally described as follows:

$$\boldsymbol{x}' = A.\boldsymbol{x} + B.\boldsymbol{u} + \boldsymbol{w}(t)$$
$$\boldsymbol{y} = C.\boldsymbol{x} + D.\boldsymbol{u} + \boldsymbol{v}(t)$$

x' is the time derivative of x. y is the output vector, which in the case of a Kalman estimator is the measurement vector. That is the predicted outcome of the measurements based on the state-space model.

$$\boldsymbol{y^T} = (\,\theta_{MG1}\;\omega_{MG1}\;T_{MG1}\;\theta_{MG2}\;\omega_{MG2}\;T_{MG2})$$

The measurements are the position, speed and torque of the two motors, coming from the motor's FAST estimators. Notice that the Kalman estimator makes it possible to "indirectly measure" states. The cyclist torque is not measured directly, eliminating the need for an expensive crank torque sensor.
In this particular case, there is no input vector:

$$\boldsymbol{u} = \boldsymbol{0}$$

w(t) and v(t) are time-dependent noise vectors, indicating that on top of the model prediction, the states and measurements can vary stochastically. The matrix A is the linearised form of the general state-space equation:

$$\frac{d\boldsymbol{x}}{dt} = F(\boldsymbol{x}, t) + \boldsymbol{w^T}$$

$$\boldsymbol{x'^T} = (\omega_{sun}\ \omega_{carrier}\ \alpha_{sun}(\boldsymbol{x})\ \ \alpha_{carrier}(\boldsymbol{x})\ \ 0\ 0\ 0\ 0\ 0\ 0\ ) + \boldsymbol{w^T}$$

The rotational accelerations α_{sun} and $\alpha_{carrier}$ are functions of the states. For the complete equations, see appendix. Since the cyclist torque states come into the equations in a sine function, these equations are non-linear. They can be linearised around a state, yielding a linear state-space model as in the general equations above. The matrices in this model need updating when the state changes, where in a linear model they are constant.
The Kalman estimator consists of a prediction step, based on the state-space model, and a correction step where the measurements come in. Simon Deruyter used a first order Taylor approximation to generate a linear state-space model at every timestep and then used the linear Kalman filter procedure.

$$\frac{d\boldsymbol{x}}{dt} \approx F(\boldsymbol{x}_i, t) + J(\boldsymbol{x}_i, t).(\boldsymbol{x} - \boldsymbol{x}_i)$$

F is the general non-linear state-space equation evaluated at the current state x_i. The matrix J is the jacobian evaluated at the state x_i. Writing this in the form of the linear state-space model gives:

$$A = J(\boldsymbol{x}_i, t)$$
$$B = F(\boldsymbol{x}_i, t) - J(\boldsymbol{x}_i, t)\,.\boldsymbol{x}_i$$

With an input vector:

$$\boldsymbol{u} = \mathbf{1}$$

The Kalman procedure is then the standard procedure for linear systems. It is not elaborated here. For this procedure, please refer to the course on Control theory by Goele Pipeleers and Jan Swevers[ii] or the wikipedia page on Kalman filters[iii].
Simon Deruyter showed in his virtual simulations that the torque estimation was accurate and fast enough to perform a simple control strategy for setting the cycling cadence and crank position and even to imitate freewheel behaviour with the CVT system.

2.4.3 Simple control of cycling cadence and crank position

In the final part of his thesis, Simon Deruyter programmed a simple control strategy based on four working modes: Forward, backward, standstill and a shock mode. In forward operation, the cycling cadence is linearly built up to the optimal cadence, which is given as an input to the control. At low speeds and low torques, the cadence is lower than this optimal cadence. When starting, the

cadence will start at zero to gradually increase up to a cadence of e.g. 70 rpm while the bike is speeding up. This is a very simple control, but is sufficient to perform simple simulations. In the backward mode, the pedals are put in a horizontal position and regenerative braking is applied proportional to the counter-torque the cyclist puts on the pedals. Standstill mode means the cyclist is not putting any torque on the pedals and the pedals are kept still. The shock mode was added to make a sudden stop of the pedals possible. In normal forward operation, the rate at which the crankshaft speed can change is limited. In shock mode, the control brings the pedals to a horizontal position as fast as possible, to simulate the behaviour of a mechanical freewheel where the cyclist can suddenly stop cranking and the pedals will come to an almost immediate standstill since there is only a very small inertia connected to them. The simple control strategy was made to perform some simulations, most importantly to check whether the cyclist torque estimation is fast enough to make freewheel behaviour simulation possible in the bike control. It was found that, while not as fast as with a mechanical freewheel (200ms) the control based on the cyclist torque estimation can bring the pedals to a complete standstill in 500ms. The prototype will be used to test whether this is fast enough to be comfortable.

3 Filter and control for use in the prototype

3.1 Adapting Kalman filter and bike model to prototype configuration

The prototype configuration is different from the hardware model Simon Deruyter used in his thesis. In the prototype, the planetary gear set is placed in the rear wheel hub. The planet carrier gear is mounted to the wheel. MG1, the speed controlled motor controlling the CVT, is coupled to the sun gear via a belt drive that adds a reduction ($K_{sun,MG1}$). MG1 is mounted on the bike frame, as in Figure 6. The crank shaft is coupled to the ring gear by means of a chain drive. This chain drive also adds an extra reduction between crank shaft and ring gear ($K_{ring,crank}$). MG2 is no longer coupled to the rear wheel as in last year's case. It is now mounted in the front wheel hub. This means that it is in a sense coupled to the wheel directly, since both wheels are of the same size. Dynamically this interpretation is valid for as long as the bicycle rides straight ahead and on an even surface. Measuring the wheel position on the other hand cannot be done reliably by means of MG2 as the coupling between front and rear wheel is "through the ground".

To be able to continue using the virtual testing framework made by Simon Deruyter, the hardware model of the bike is to be adapted to this new configuration. Last year's model only took into account the rear wheel of the bicycle, which poses a problem now that MG2 is in the front wheel. However, only operation when riding straight ahead was considered anyway, not taking into account any road roughness. With these assumptions, as mentioned, MG2 can be considered mounted to the rear wheel hub. This is the planet carrier shaft, the output shaft of the planetary gear set. Using this assumption, the model can be rather easily adapted to the new configuration by reconnecting all the ports of the simscape model as described. The chain drive is modelled as a reduction with some added freeplay.

Also the kalman filter equations need updating. MG2 is in the dynamic equations again considered to be placed at the rear wheel hub. The equations for the planetary gear set itself stay more or less the same. The added reductions from the belt and the chain drive are being added and the crank speed and ring speed are being decoupled (this was not the case in last year's configuration). The states of the system used in the filter do not change, since the entire system is still described by two kinematic parameters of the planetary gear set. Altough in theory this is correct, it might be more logical to take a different set of states for this hardware configuration. When talking about the measurement equations, this will be discussed further.
By moving MG2 to the front wheel, this motor is lost as a useful sensor to measure a position or speed of the planetary gear set. Only on a perfect road surface and when riding straight ahead, the position and speed measurement of MG2 reflects the position and speed of the planet carrier gear. The planetary gear set is fully determined in terms of kinematics and dynamics when one torque state is known and two positions. MG1 is still directly – via a belt drive – coupled to the sun gear. All the measurement from MG1 (position, speed and torque) are thus useful information. The MG1 torque measurement suffices to know all torques in the system. Apart from the MG1 position measurement, another position is necessary to fully determine the system kinematics. For this purpose an extra position sensor must be added on either the rear wheel (planet carrier shaft), the ring gear or the crank shaft. Both for practical reasons and because an absolute reference of crank position is important, the extra position sensor is placed at the crank shaft. Because the position of MG1 and of the crank position are the measured quantities, it might be more logical to use these variables as the states in the state vector, rather then the position of the sun and the planet carrier. For now, to avoid re-writing the entire filter, the state vector is left as it is. The measured variables are now simply a linear combination of the kinematic state variables, position of sun gear and planet carrier gear. Besides from being less intuitive, this should not influence the performance of the filter.
Because the prototype is tested on rollers first (Figure 11), two variants of both the Kalman estimator and the control algorithm have to be made. When on rollers, the front wheel of the bicycle is stationary and only the rear wheel is moving against a load provided by the rollers. Adding MG2 to the system states and control is not applicable in this case since the model presumes the front and rear wheel to have approximately the same speed at all times. Besides, MG2 should off course not be powered when the front wheel is not supposed to move. For this test case, a variant of the filter only taking into account MG1 is used. It is as if MG2 is not there and the front wheel is unpowered as in a normal bicycle. However, to not have to make an entirely new Kalman filter, the MG2 torque state is not removed from the state vector. The process noise covariance for this state is put to zero and this state is set to zero in each timestep. Zero MG2 torque is after all the equivalent of not having a motor in the front wheel. In the measurement equations, the MG2 measurements are taken out.

Figure 11: Prototype test setup with rear wheel on roller with eddy current brake.

The resulting state vector for the Kalman filter is the same as in last year's configuration:

$$\boldsymbol{x^T} = \left(\theta_{sun}\ \theta_{carrier}\ \omega_{sun}\ \omega_{carrier}\ T_{cyclist,offset}\ T_{cyclist,max}\ \phi\ \ T_{MG1}\ \ T_{MG2}\ \ \alpha_{eq}\right)$$

In the kalman filter used when MG2 is not operational, the process noise covariance of the T_{MG2} state is set to zero – this is to not loose optimality in the filter when setting the state to zero – and the value of T_{MG2} is set to zero in each timestep. For the complete kalman filter, with MG2 operational, the measurement vector is:

$$\boldsymbol{y^T} = \left(\ \theta_{cyclist}\ \theta_{MG1}\ \omega_{MG1}\ T_{MG1,meas}\ \ T_{MG2,meas}\right)$$

The measurement equations are:

$$\theta_{cyclist} = \left[(1+2\rho)\,.\,\theta_{carrier} - \rho\,.\,\theta_{sun}\right].K_{ring,crank}$$

$$\theta_{MG1} = \theta_{sun}\,.\,K_{sun,MG1}$$

$$\omega_{MG1} = \omega_{sun} \,.\, K_{sun,MG1}$$

$$T_{MG1,meas} = T_{MG1}$$

$$T_{MG2,meas} = T_{MG2}$$

When MG2 is not operational, the torque measurement of MG2 is left out of the measurement equations. It is of no harm in principal that the filter is left unadapted to the roller testing. When MG2 is not powered, it's torque measurement will always be zero and therefore also the state estimation will estimate the MG2 torque to be zero. However, it is of no use to estimate a state which you know should be zero at all times. This estimation can only add complexity and unaccuracy in such a case. Purely from a practical point of view it is necessary anyway to make a filter that does not require an input measurement of the MG2 torque, since in the first prototype MG2 will simply not be connected, meaning that it is not being powered and therefore also not returning any measurements of the torque.
Notice that, in contrast to Simon Deruyter's implementation, the position and speed measurements from MG2 are no longer taken into the filter. They are not useful anymore because MG2 is in the front wheel and the coupling "through the ground" of both wheels is not a reliable one. Besides, the biycle model does not take into account the front wheel and it's states, neither does it account for steering the bicycle. Making a turn will in the current model create an inconsistency between the position and speed measurements of MG2 and those of MG1 and the crank position sensor.
In Deruyter's implementation of the Kalman filter, there was a small mistake that has been corrected in the current implementation. No prediction step was performed on the covariance matrix in Deruyter's code, only on the states.
With both the simulation framework and the Kalman filter adapted, the work is validated by running the same simulation as Simon Deruyter did last year. All states are adequately estimated and the desired control action (including bringing the pedals to standstill quickly) is performed just as well. The results of this test are discussed in section 3.4.

3.2 Kalman filter integration schemes for real time operation

The kalman filter makes use of a discrete time state space model. The dynamics of the bike are however expressed in the form of a continuous time state space model. This continuous time model needs to be discretised. Because the model used is non-linear, it needs to be linearised every timestep. What this means is that the state-space representation of the model is also changing every timestep, being the result of the linearisation procedure. Where for a linear model the discretisation only needs to be done once, it needs to be done in every timestep for a non-linear model. Because this discretisation must be done in real time at a high rate (e.g. 1kHz), it is imporant that it takes a minimal amount of time.

3.2.1 Discretisation procedure

Starting from a continuous state-space representation of the system (linearisation around the current state):

$$\boldsymbol{x}' = A.\boldsymbol{x} + B.\boldsymbol{u}$$
$$\boldsymbol{y} = C.\boldsymbol{x} + D.\boldsymbol{u}$$

we want a discrete state space representation:

$$\boldsymbol{x}(k+1) = F.\boldsymbol{x}(k) + G.\boldsymbol{u}(k)$$
$$\boldsymbol{y}(k) = C.\boldsymbol{x}(k) + D.\boldsymbol{u}(k)$$

The measurement or output equation is already discrete, since the measurements are discrete. There is no input in the system. To discretise the continuous model, we need to integrate the continuous equation over a period of one timestep. This can be done with various integration schemes. Three of these schemes are worked out here and the performance of the filter is compared when using each of these schemes.

3.2.2 Zero-order hold method[iv]

Assuming zero-order hold on the input, the model can be discretised using the following procedure:

$$\begin{pmatrix} F & D \\ 0 & I \end{pmatrix} = e^{\begin{pmatrix} A & B \\ 0 & 0 \end{pmatrix}.h}$$

h is the timestep of the discretisation. N being the number of states, this procedure requires a matrix exponential of a matrix of size (n+1)x(n+1) and is the "exact" procedure for discretising the state space model. Proof of this is ommited here. Please refer to the reference in the section title for this.

3.2.3 Explicit (forward) Euler method[v]

In the explicit Euler method, the derivative at the current timestep is used to calculate the state at the next timestep. Integrating the state **x** using forward Euler integration gives:

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) + h\,.\boldsymbol{x}'(k)$$

Filling in the state-space expression for the derivative of the state vector gives:

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) + h\,.(A.\boldsymbol{x}(k) + B.\boldsymbol{u}(k))$$

Writing into the state-space form:

$$\boldsymbol{x}(k+1) = (I + h\,.A).\boldsymbol{x}(k) + h.B.\boldsymbol{u}(k)$$

and so:

$$F = (I + h\,.A)$$
$$G = h\,.B$$

And so the continuous state space representation has been discretised. The calculation of the discretised representation requires only a multiplication and a

summation of matrices of size nxn when using the explicit Euler method, which makes it a highly efficient method. Stability however is not guarenteed when using explicit Euler. The timestep should be small enough to ensure stability. When using the method in a kalman filter, this stability is less of an issue then when e.g. simulating a system. In the filter there are always the measurements that will keep the states from diverging when the integration is not stable. The explicit Euler method is of first order, meaning it's approximation error disappears in a linear way when the timestep h grows smaller.

3.2.4 Implicit (backward) Euler method[vi]

The implicit Euler method does not use the derivative at the current timestep but rather at the next timestep, that is still unknown. Integrating the state using implicit Euler integration:

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) + h\,.\,\boldsymbol{x}'(k+1)$$

Filling in the state-space expression:

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) + h\,.\,(A.\boldsymbol{x}(k+1) + B.\boldsymbol{u}(k+1))$$

And working out to state-space form:

$$\boldsymbol{x}(k+1) = (I - h\,.\,A)^{-1}.\boldsymbol{x}(k) + (I - h\,.\,A)^{-1}(h.B).\boldsymbol{u}(k+1)$$

So:

$$F = (I - h\,.\,A)^{-1}$$
$$G = (I - h\,.\,A)^{-1}(h.B)$$

This discretisation requires a matrix inversion of an nxn matrix, on top of a multiplication and summation. This will make the method more costly, but the implicit Euler method is always stable, no matter the size of the timestep h. It maps the continuous time stability region on a subset of the discrete time stability region. This method is also first order.

3.2.5 Tustin/trapezoidal method[vii]

Finally, the integration can also be done by using not the derivative at t or at (t+h) but rather the mean of these two. This is the Tustin method:

$$\boldsymbol{x}(k+1) = \boldsymbol{x}(k) + h\,.\frac{\boldsymbol{x}'(k+1) + \boldsymbol{x}'(k)}{2}$$

Working out in an analog fashion as the two previous methods yields:

$$F = \left(I - \frac{h\,.\,A}{2}\right)^{-1}\left(I + \frac{h\,.\,A}{2}\right)$$
$$G = \left(I - \frac{h\,.\,A}{2}\right)^{-1}\left(\frac{h\,.\,B}{2}\right)$$

This method requires also a matrix multiplication, on top of the inversion and summation steps of the implicit Euler method. All matrices are of size nxn. The Tustin method is, like the implicit Euler method, always stable and maps the continuous time stability region on the discrete time stability region. Stability in the continuous domain is thus preserved. This method is second order, meaning the approximation error disappears quadratically when the timestep grows smaller.

3.2.6 Considerations on integration scheme choice

The explicit Euler method is the least costly of the three methods in terms of computation, not requiring a matrix inverse. Because of the relatively small size of the matrices, not more can be said about computational cost of the methods just from the theory. Stability is not very important because of two reasons:

1) The estimator is used to estimate a high-dynamic input and a fast-changing state. This means that the sample time will have to be small for the proper working of the estimator, independent of the integration scheme used. The integration should be as fast as possible to obtain a high control bandwidth. This makes an explicit scheme a logical choice. Implicit schemes are more interesting when a large timestep is desired, as in false time stepping procedures.
2) The model of the bicycle is not accurate enough to predict the state of the bicycle with high accuracy. Measurements will be the major source of information. These measurements will always keep the states within bounds and will not allow them to diverge, even if the integration scheme in itself would be unstable.
3) Many of the estimated states are so-called augmented states. This means that they are rather inputs to the systems, that are being estimated through the use of a Kalman filter rather then measured directly. These inputs and thus also the states vary randomly. It is only the measurements that keeps these states within limits. The stability of the integration scheme is therefore not very important.

All three integration schemes are implemented in the Kalman filter and compared in terms of accuracy and speed (Figure 12, Figure 13, Figure 14). At 1kHz, all integration schemes give good results. The time needed to do the calculations of one timestep in the kalman filter is measured using the profiler report in simulink. Using exact ZOH integration, it takes approximately 10^{-4}s to run the filter on a 2,4 GHz laptop. With each of the approximation methods, it takes about $5 \cdot 10^{-5}$s to run the filter.

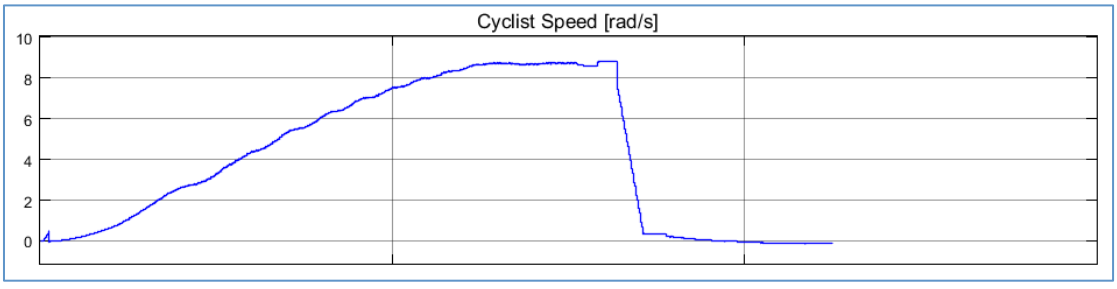

Figure 12: Explicit Euler integration scheme, 1 kHz.

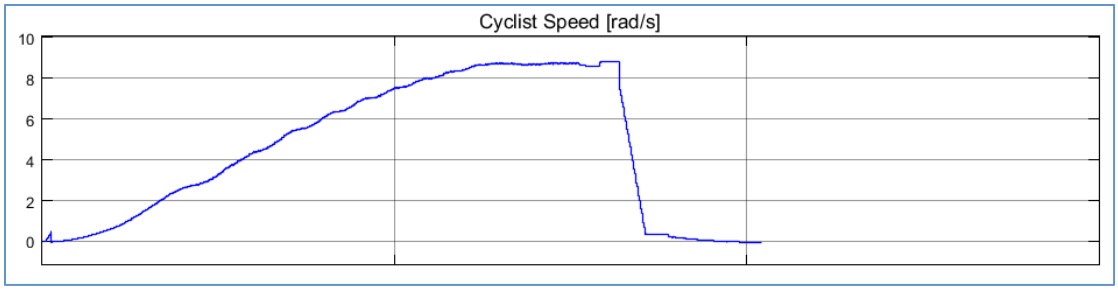

Figure 13: Implicit Euler integration scheme, 1kHz

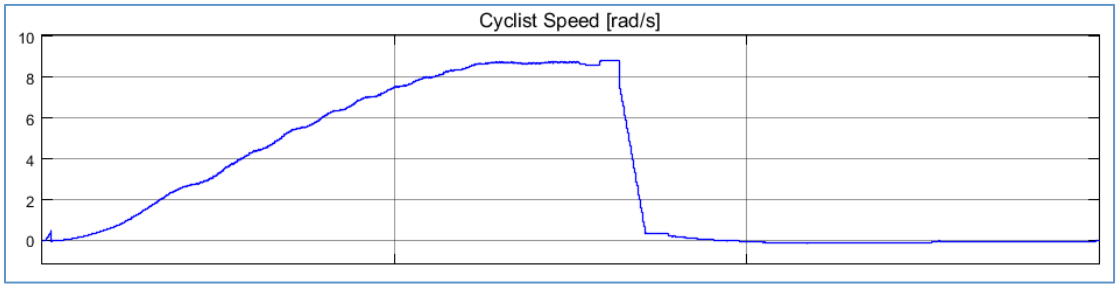

Figure 14: Tustin integration scheme, 1kHz.

3.3 Adapting the control algorithm

3.3.1 Desired bicycle behaviour

For the first prototype tests it is important to have a simple and straightforward control algorithm to make debugging easier. The control that Simon Deruyter came up with last year was based on working modes. The switching between modes was rather arbitrarily done. To make the control easier to debug and also to simplify, the mode-based control is removed and replaced by a rather simple look-up table approach. Before going into the actual control, it is best to consider the desired behaviour of the bike. Consider different scenarios:

1) **Standing still:** putting some torque on the pedals should not make them move, you want to be able to let your leg rest on your pedals. Below a certain torque treshold, you want your pedals to be kept still.
2) **Starting from standstill:** once the torque on the pedals is high enough to start moving, your pedals start to move as they would in a fixed gear bicycle. This fixed gear approach is kept to a certain speed until your optimal cadence is reached, from there, the cadence is not increased anymore with the bicycle speed.
3) **Stop pedalling at speed:** when at speed and you suddenly stop pedalling, you expect freewheel behaviour, namely your pedals stay still while your bicycle is still running at speed.
4) **Regenerative braking by pedalling backwards:** when at speed and putting a negative torque on the pedals (pedalling backwards) the pedals should be kept still so that a maximum braking torque can be applied on them. MG1 will then work as a generator.
5) **Taking motor limits into account:** MG1 has limits on it's torque and speed. When the bicycle speed is becoming very high, MG1 will have to run at high speed to maintain the optimal cadence. When the speed of MG1 reaches a maximum value, the cadence should go up to give the

cyclist feedback on this fact. Also, by going above the cyclists optimal cadence, the power input will be lowered because the cyclist's torque output falls. This will reduce the acceleration of the bicycle and will in this way protect the system from too high speeds without giving the cyclist an unnatural feeling. Mimicking freewheel behaviour can only be done up to a certain maximum bicycle speed. Slowing down the crankshaft requires MG1 to speed up for a given bicycle speed. The maximum bicycle speed is the intersection between the line from maximum MG1 speed to zero crank speed and the bicycle speed slider in the nomogram. The cyclist should be made aware when crossing this maximum bicycle speed, because freewheel behaviour will not be achievable above this speed. Also the maximum torque that MG1 can provide needs to be taken into account in the control. The torque on MG1 is always proportional to the torque put on the pedals, as is clear from the planetary gear equations. The control must prevent that the cyclist puts a torque on the pedals that is high enough to step through MG1. This might make the control unstable and unpredictable. To avoid this, the cadence should be speeded up when the cyclist torque goes above a certain treshold and the motor should be controlled to a working point where it can take up the torque. This treshold can be 90% of the maximum torque that MG1 can still deliver and it will be dependant on MG1 speed.

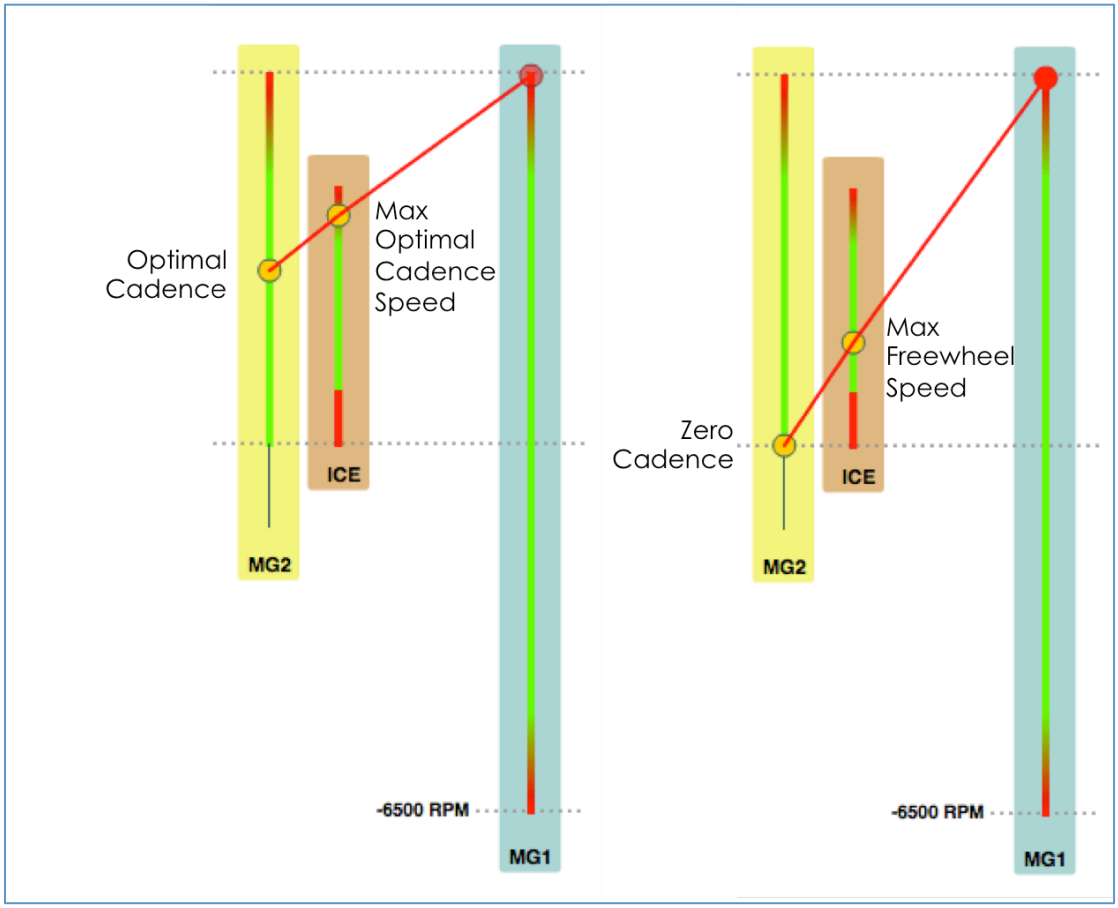

Figure 15: Principle of bicycle speed limits visualised in the PSD nomogram. (Note: in the bicycle: MG2 = crankshaft, ICE = rear wheel).

3.3.2 Choosing the desired cadence

The cadence control can be described using the following simple formula:

$$\omega_{crank} = \omega_{cyclist,opt} \,.\, C_v \,.\, C_T$$

C_v and C_T are correction factors determined by the bicycle speed and cyclist torque respectively:

$$C_v = f(v)$$
$$C_T = f(T), \qquad v > v_{fix}$$
$$C_T = 1, \qquad v \leq v_{fix}$$

T indicates the cyclist torque envelope. This is the DC torque that connects all the torque sine peaks (see section 2.4.2). The figures below show plots of the scaling factors. The velocity scaling factor is always taken into account. The torque scaling factor is not always taken into account and is simply set to 1 when the velocity of the bicycle is below v_{fix}, the velocity under which there is a fixed gear behaviour at all times. This means that when you move the bicycle back and forth when e.g. at a red light, the pedals will move with it. When at higher speeds, the idea is to have a fixed gear feeling until you reach the optimal cadence. This happens at a certain velocity v_{switch} which is a design parameter. Once this cadence is reached, it is kept constant when the bike is further accelerating. The torque scaling factor is there to allow for slower cranking when at high speeds. When cranking under a certain torque, the crank will move slower as it is clear in this case that not much power is needed from the cyclist. This is implemented now as a linear function, but research on the optimal cadence as a function of cyclist torque might lead to a different relation. As will be later discussed however, it is interesting already from quite low cyclist power to move to the optimal cadence. When the cyclist stops cranking completely, i.e. the cyclist torque falls to zero or even a negative value, the crank should immediately be stopped to mimic freewheel behaviour and to allow for a high negative torque to be put on the pedals in order to make regenerative braking possible.

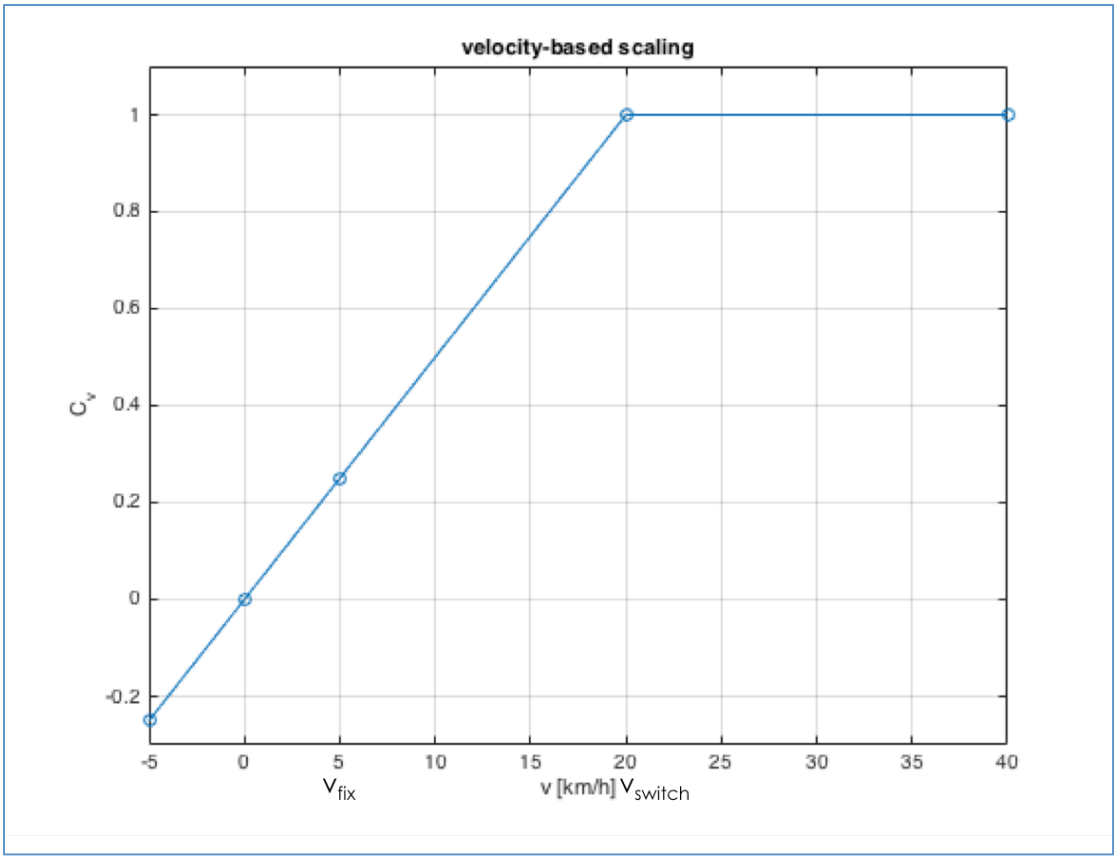

Figure 16: Velocity scaling factor

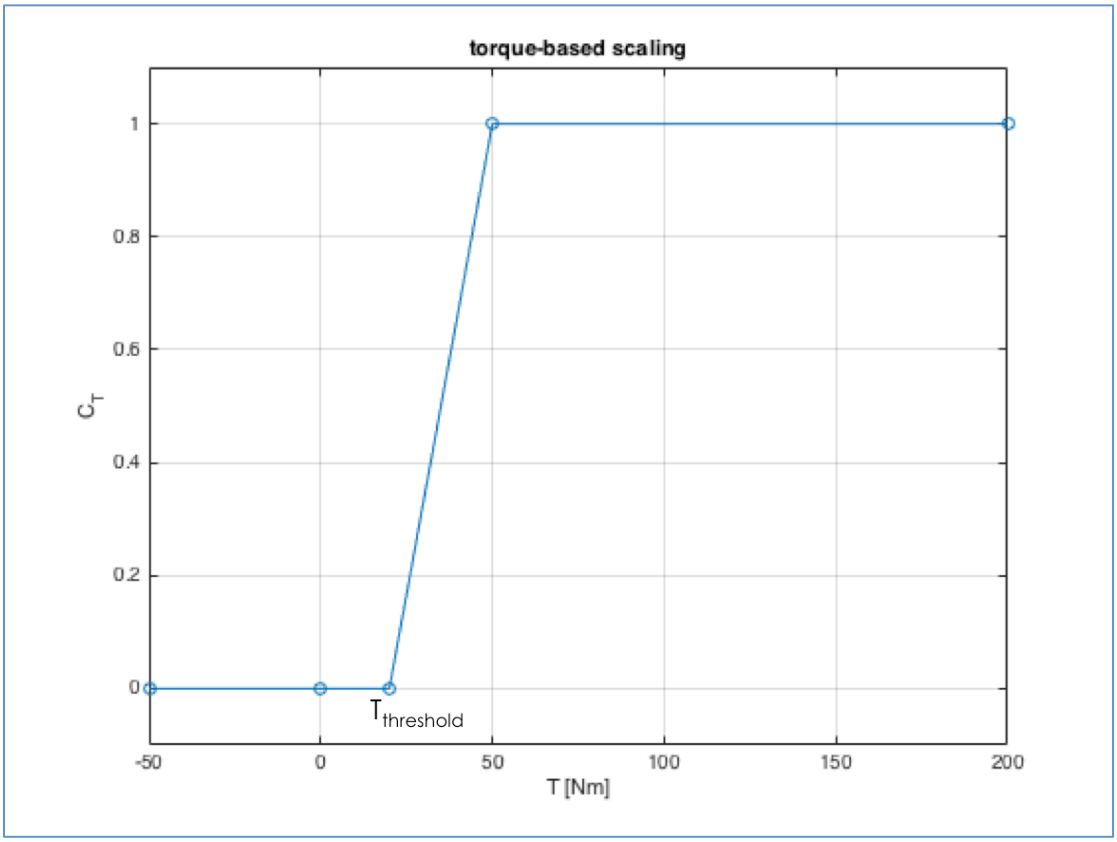

Figure 17: Cyclist torque scaling factor

3.3.3 Control for changing conditions/riding modes

The above discussion mentioned only steady state situations. Each situation has it's unique desired crank speed setpoint. Changing conditions or so-called riding mode changes evidently occur all the time. In fact it is this kind of dynamic behaviour in particular that will make the bicycle feel intuitive or not. Most of the dynamic behaviour is already taken up in the above mentioned control of the cadence. The lookup tables allow for a fluent evolution through the cadence setpoints without having discontinuities. By not using any modes, continuous operation is ensured as there are no shifts in the control behaviour. However, since the torque can vary very rapidly and this has an immediate effect on the crank setpoint as calculated by the control, measures have to be taken that do not allow the crank speed to vary to quickly. Therefore, when the desired crank speed has been calculated, it is rate-limited. This means that the rate of change of the crank speed control signal is limited by two extremes. The crank should never accelerate very quickly, this will have a negative effect in two ways. It will make it in some cases impossible to maintain a torque on the pedals as the pedals would seem to be "moving away" from the cyclist's legs. By limiting the rate at which the pedals can accelerate, this can never occur. The cyclist expects some inertia on the pedals. You could say that the rate limiter is some kind of artificial inertia built in to the system. A second reason is the changeover at the speed v_{fix}. When decelerating to a complete stop, the bicycle off course passes this velocity. When doing so, the control will set the cadence from zero to a positive value $C_v(v_{fix})$ in a very short time. To make this transition smooth, the rate at which it occurs is limited. Off course also for safety reasons it is important to limit the rate of change. When decelerating the crank, the rate of change can be a lot higher. To mimic freewheel behaviour, the crank should come to a complete standstill in a very short time (ideally 200ms).
The maximum and minimum rate can be chosen by trial and error off course and tuned in the prototype tests for best feeling, but it is already possible to determine good values based on the above-mentionen considerations. For the maximum rate of change, the comfortable time for the evolution from freewheel to fixed gear behaviour is the basis. Let's assume two seconds to be a comfortable duration for this evolution. Then:

$$\left(\frac{d\omega_{crank}}{dt}\right)_{max} = \frac{\omega_{opt} \,.\, C_v\left(v_{fix}\right)}{2s}$$

The minimum rate of change can be set based on the desired freewheel behaviour characteristics. In Simon Deruyters thesis, it was mentioned that in a normal freewheel bicycle it takes 200ms to go from the optimal cadence to a complete standstill. To allow for this rate of deceleration:

$$\left(\frac{d\omega_{crank}}{dt}\right)_{min} = \frac{\omega_{opt}}{200ms}$$

In order to have a stable standstill operation, it is of the highest importance that the motor stays at exactly 0rpm when the bicycle is not moving. In contrast to operational points where the bicycle and the crank are moving and some play on the movement of the crank can be tolerated, the crank should be kept perfectly

still when the bicycle is standing still. Even the slightest movement of the crank at this point will give the impression that the control is not working properly. As there is always some unaccuracy on the state estimates and the desired MG1 velocity is calculated from these states, this desired velocity will never be exactly zero. The result will be that the motor control will move the motor slightly back and forth in a random fashion, inducing some movement on the crank. This is unacceptable. To avoid this, a dead zone is introduced in the desired MG1 velocity. Only when the desired velocity is above 200 rpm will it be transmitted to the motor control. If it is lower than 200 rpm, the desired speed will be set to exactly zero. This 200 rpm limit might be too large or too small, depending on the unaccuracy of the state estimates. It should be experimentally tuned. The dead zone will result in a discrete "gear shift". The continuity of the CVT system is lost in this region. When maintaining the optimal cadence (e.g. 70 rpm) at low bicycle speeds, MG1 is rotating in the opposite direction (negative rpm). Speeding up, MG1 is also speeding up when the cadence is kept constant. When crossing zero MG1 speed, two discrete jumps will take place in the gear ratio: one from 200rpm negative to 0rpm and one from 0rpm to 200rpm positive. Defining the gear ratio as:

$$K_{crank,wheel} = \frac{\omega_{wheel}}{\omega_{crank}}$$

The gear ratio's at these three positions are, in the prototype configuration, calculated using the kinematic equations of the planetary gear set:

$$\omega_{crank} = \left[(1+\rho).\omega_{wheel} - \rho.\omega_{MG1}.K_{MG1,sun}\right].K_{ring,crank}$$

$$\omega_{wheel} = \frac{K_{ring,crank}.\omega_{crank} + \rho.\omega_{MG1}.K_{MG1,sun}}{1+\rho}$$

$$K_{crank,wheel} = \frac{K_{ring,crank}}{1+\rho} + \frac{\rho.K_{MG1,sun}}{1+\rho}\frac{\omega_{MG1}}{\omega_{crank}}$$

$$Prototype: \rho = 0{,}34\ ; K_{ring,crank} = \frac{21}{52}\ ; K_{MG1,sun} = 0{,}27$$

So that:

$$\omega_{MG1} = 200rpm:\ K_{crank,wheel} = 0{,}497$$
$$\omega_{MG1} = 0rpm:\ K_{crank,wheel} = 0{,}301$$
$$\omega_{MG1} = -200rpm:\ K_{crank,wheel} = 0{,}105$$

The relative change in gear ratio's is very large (300% even). A typical relative gear ratio change is 13%. The dead zone would lead to unacceptable gear ratio changes when used in continuous operation also. The obvious solution to this problem is to only implement the dead zone during standstill operation. To do this, the control will need to recognize the standstill situation (mode) and act accordingly. Such modes are discussed later.

Another option for keeping the crankshaft still is in switching to position control when standing still, or even in the entire speed range below v_{fix}. The position of the crank is measured with respect to a fixed reference so a simple PI controller could keep the crankshaft still.

3.3.4 Motor limits

MG1 is an electric motor and has limits regarding maximum speed and torque. The maximum speed of the motor can easily be taken into account by saturating the output target speed the control sends to the motor. This means that the output speed of the control can never be higher then e.g. 95% of the maximum motor speed. This will result in the cyclist having to crank faster then his optimal cadence in order to go faster then the bicycle speed corresponding to the optimal crank cadence and MG1's maximum speed. Because the cyclist will not be able to maintain his torque at these high speeds, it will automatically create a limit on the bicycle speed. This will feel very natural to the cyclist, just as if he were in the highest possible gear on a normal bicycle and still wanting to go faster.
Protecting MG1 against "step through" is more difficult. This means that the torque applied on the crankshaft should never be so high as to result in a torque on MG1 that it cannot deliver. The maximum torque MG1 can deliver is speed-dependant. In the ideal case, since MG1 is a DC motor, the torque declines linearly with the speed from it's maximum value at zero speed. The torque relation between the sun gear and the ring gear couples the torque on MG1 and the cyclist torque. Every timestep, it should be checked wether the cyclist torque is still in a safe region not to overload MG1. If this is not the case, i.e. the cyclist torque leads to a torque on MG1 that is greater then 95% of it's maximum torque, then MG1 should be controlled to a speed where it can handle this torque. This means slowing down MG1 to the point where the cyclist induced torque on MG1 is 90% of the maximum MG1 speed. This will result in an acceleration of the crank which in the same time will make it harder for the cyclist to continue exerting such a high torque. Figure 18 shows this principle.

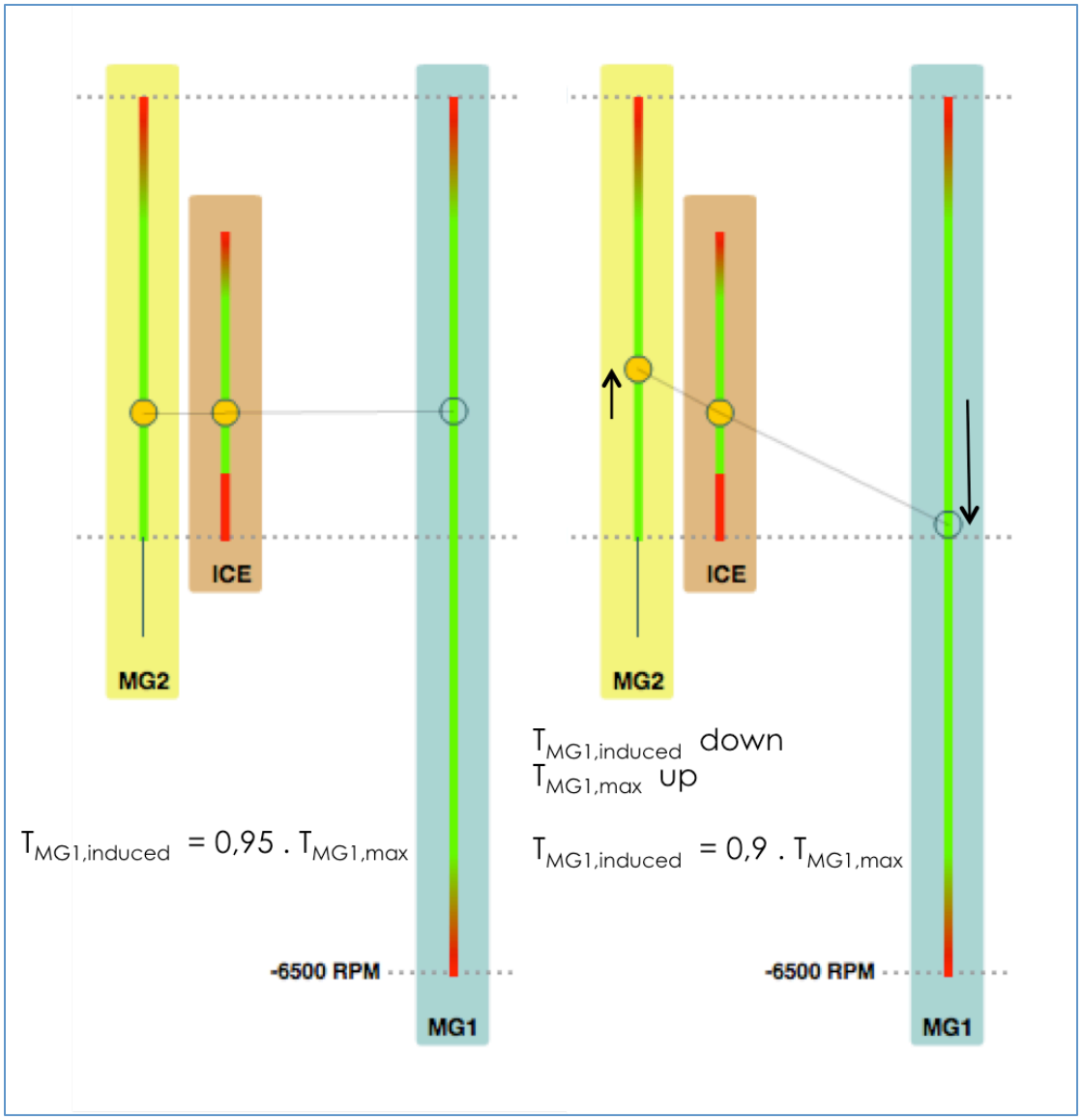

Figure 18: Controlling MG1 to protect it against step through (in the bicycle: ICE = rear wheel, MG2 = crankshaft).

When the cyclist induced torque on MG1 is higher then MG1's stall torque, then this control can no longer be applied and other measures have to be taken not to overload MG1.
One option is to accelerate the crank in order to prevent the cyclist to exert more torque on the pedals. This is done by simply decreasing the target speed in every timestep with a fixed rate of e.g. 4000 rpm per second. In the control this results in:

$$\Delta\omega_{MG1} = -4000rpm * T_{s,control} \left[\frac{rpm}{timestep}\right]$$

At a rate of 1kHz, this results in a decrease of 4rpm per timestep. Altough it might be an effective safety measure for MG1, this behaviour is not very safe and definitely not comfortable for the cyclist. When at standstill for example, exerting a large force on the pedals in order to accelerate quickly, this safety measure might cause a sudden loss of resistance on the crank, which may hurt the cyclist

as the pedals suddenly move much faster than he anticipated. The highest possible torque on the crank is expected to be at this standstill situation, where the cyclist can put his entire weight onto the pedals. Either the motor stall torque needs to be high enough to cope with this, or an additional measure should be taken to take up the high torque on the sun gear. A controllabe brake on the sun gear or the MG1 shaft would be a solution. When the torque on MG1 becomes too high for it to handle, the brake can be closed so that the cyclist can put maximum torque on the crank without overloading MG1. This will give the bike a good starting behaviour, because the high cyclist torque will allow for good acceleration at standstill. The fixed gear behaviour mentioned earlier can even be integrated in this system. When the sun gear is blocked by means of the brake, the gear ratio between the ring gear and the planet carrier, i.e. between the crankshaft and the wheel, is fixed. Because at low speeds, the required power is low, it is not a problem that MG1 is not delivering any power. Using the brake for the fixed gear behaviour will mean that the velocities are fixed by the hardware and can no longer be freely chosen. A controllable brake might be too expensive.
A purely mechanical one-way clutch can also be used to avoid motor step through. This clutch can be placed at either the motor itself (or the sun gear) or between the ring gear and the planet carrier.
When placed on the motor or the sun gear, the one way clutch will make it impossible for the motor to rotate in the opposite direction. The clutch will act as the boundary for the cyclist torque based MG1 control explained earlier. When the cyclist torque increases so that MG1 cannot deliver the counter torque at it's current speed, it will slow down until it is at a speed where it can deliver this required counter torque (Figure 18). When the motor reaches 0rpm in this way, it cannot deliver a torque higher than the stall torque, so the clutch will take up the torque at this point. One could say that this measure makes the stall torque "infinite" or at least very high. The problem with this solution is that when a high torque is put on the crank shaft at rather high bicycle speeds, this will lead to a very high crank speed as the motor will decrease it's speed possibly even to 0 rpm. It will be highly uncomfortable for the cyclist to have the crank at high speeds and therefore not being able to exert a high force on the pedals. The bicycle will feel like it is working against the cyclist.
The one-way clutch can also be placed between the ring gear and the planet carrier, so that the ring gear can never rotate faster than the planet carrier can. Every danger of stepping through the motor is hereby avoided because when the bicycle speed is positive, MG1 speed can never be negative. This is also clear from the planetary gear nomogram in Figure 18. The result of this clutch is also a fixed gear behaviour up to the point where the cyclist reaches the optimal cadence. From this point, MG1 can keep the cyclist on this cadence by speeding up, pulling op the bicycle speed. The fixed gear ratio between crank and wheel will be larger in this configuration then if the clutch is placed on the sun gear.
It is clear from the nomogram and the explanation above that adding a clutch in the system will limit the possibilities for controlling the system. MG1 can e.g. not be used in negative rpm's. This means that the optimal cadence can never be achieved at low bicycle speeds, e.g. when cycling uphill. The main disadvantage of adding a clutch however is that the system looses track of the cyclist torque. This torque is estimated via the measurement of MG1 torque. When a brake rather than MG1 is taking up the cyclist torque, there is no way to estimate this

torque anymore. Since the cyclist torque estimate is the most valuable information from the cyclist, using a brake or clutch should be avoided if possible.
The only option left if the MG1 stall torque is not sufficient for handling the maximum cyclist torque (full weight on the pedals) and without using brakes or one-way clutches is to control the bicycle in such a way that the cyclist is not likely to exert high forces on the pedals. This can be done by tuning MG2 to be very aggressive. The acceleration of the bicycle is the result of the torque on the rear wheel plus the torque on the front wheel. The torque on the rear wheel is completely controlled by the cyclist, MG1 can only provide countertorque but cannot add to the acceleration of the bicycle. MG2 delivers it's torque directly to the front wheel. This means that is can be used both as an aggressive motor and as an aggressive regenerative brake. Intuitively, when feeling a very strong effect from the force put on the pedals, the cyclist will be more cautious and not put his full weight on them. The assistance level or torque multiplication provided by MG2 can thus be used to encourage the cyclist to be lighter on the pedals, requiring less torque from MG1.
The resulting torque that the cyclist puts on the rear wheel can be easily calculated:

$$T_{ring} = T_{cyclist} \,.\, K_{ring,crank}$$

$$T_{carrier} = \left(1 - \frac{1}{\rho}\right) .\; T_{ring}$$

$$\rho = \frac{R_s}{R_r}$$

$$\Longrightarrow T_{rear\ wheel} = T_{carrier} = K_{ring,crank} \,.\left(1 - \frac{1}{\rho}\right) .\; T_{cyclist}$$

Define a torque support factor that is the ratio between the torque provided by MG2 and the torque provided by the cyclist:

$$T_{front\ wheel} = T_{MG2} = f_s \,.\, T_{rear\ wheel}$$

Assume a certain desired acceleration that the cyclist expects, so that:

$$m \,.\, a_{desired} = F = T_{total} \,.\, R_{wheel} = (1 + f_s).\, T_{rear\ wheel} \,.\, r_w$$

$$T_{cyclist} = \frac{m.\, a_{desired}}{(1 + f_s).\, r_w \,.\, K_{ring,crank} \,.\left(1 - \frac{1}{\rho}\right)}$$

or:

$$(1 + f_s(v)) = \frac{m.\, a_{desired}(v)}{T_{cyclist,max}(v) \,.\, r_w \,.\, K_{ring,crank} \,.\left(1 - \frac{1}{\rho}\right)}$$

The support factor can be obtained from the parameters of the bicycle, the maximum allowed torque on the crank because of MG1 limits and the desired acceleration. The maximum allowed cyclist torque depends on the speed of MG1 and this in turn depends on the bicycle speed, following the control of MG1 discussed earlier. Also the desired acceleration is taken to be velocity-dependent. According to a study done by M.C. Snare the Virginia Polytechnic Institute an State University in 2002[viii], the acceleration profile in passenger cars is shown to follow a linear decay model:

$$a_{desired} = c_1 - c_2 \,.\, v$$

With c_1 around $3 m/s^2$ and c_2 around $0.1\ s^{-1}$. These values seem to be independent of the driver and the type of car when test subjects were asked to perform regular driving.
In another study by D. Luo[ix], specifically for cyclists, the actual acceleration follows a rather parabolic line, peaking at about half the desired speed:

$$a_{desired} = c(v^2 - 2.v.v_{desired})$$

Maximum acceleration is around $0.7\ m/s^2$. The E-bike will probably be somewhere in the middle between both models because it is a hybrid between human power and electrical power. The hypothesis is that the acceleration behaviour will be the same as that of a normal cyclist if the total torque on the wheels is always proportional to the torque of the cyclist, i.e. the support factor is a constant. This is the case in most E-bikes nowadays. The linear decay model still holds (see Snare study) when the actual acceleration is lower than the maximum acceleration possible. In this case, the desired acceleration can be taken to be the actual acceleration, as there is no other factor determining the acceleration than the human intention itself. As a hypothesis one could state that the linear decay model is the best model for the desired acceleration and that, in case of limited power, this optimal desired acceleration cannot be met, resulting in the parabolic curve observed by Luo. For the specific bicycle, experiments will be needed to determine what the desired acceleration profile is. For now, it is presumed that the acceleration follows the linear decay model.
Combining the linear decay model and the torque-speed characteristic of MG1 in the formula above, the support factor can be calculated as a function of speed. MG2 is then controlled in the following way:

$$T_{MG2} = f_s(v)\ .\ K_{ring,crank} \,.\, \left(1 - \frac{1}{\rho}\right)\ .\ T_{cyclist}$$

To make the control more dynamic and more reponsive, the torque on MG2 can also be made not just proportional to the cyclist torque, but proportional to the rate of change of this torque. When a sudden acceleration or deceleration is required, MG2 can be more aggressive than in steady state situations. This is given by the differential support factor:

$$T_{MG2} = f_s(v).\ K_{ring,crank}\ .\left(1 - \frac{1}{\rho}\right).\ T_{cyclist} + f_s^d\ .\frac{dT_{cyclist}}{dt}$$

This differential torque support factor in particular will encourage the cyclist to be lighter on the pedals during transients. The bicycle will react more aggressively to a change in behaviour than to a steady state torque. In the prototype however, only the support factor is used until now.
Finally, MG2 can also be controlled simply proportional to the cyclist torque, except when this torque is too high for MG1 to handle. Suddenly accelerating the bicycle at this moment will discourage the cyclist to maintain this high torque. Prototype testing will determine what solutions are not only theoretically possible, but comfortable and intuitive to the cyclist.

3.3.5 Performance of the control

The adaptations to the control algorithm are tested using the exact same test situation that Simon Deruyter used in his thesis last year. The test starts from standstill, building up speed by applying a constant torque (i.e. ideal torque shape with constant parameters) on the pedals and then suddenly stopping (cyclist torque to zero). The desired behaviour in this case is that the cadence builds up steadily towards the optimal cadence and suddenly goes to zero when there is no torque applied on the pedals anymore. Figure 19 shows the cadence throughout the test. The time to standstill with this control algorithm is shorter than what Deruyter achieved. This is probably because there is no "shock mode" that first needs to be detected before appropriate action can be taken to stop the crank.

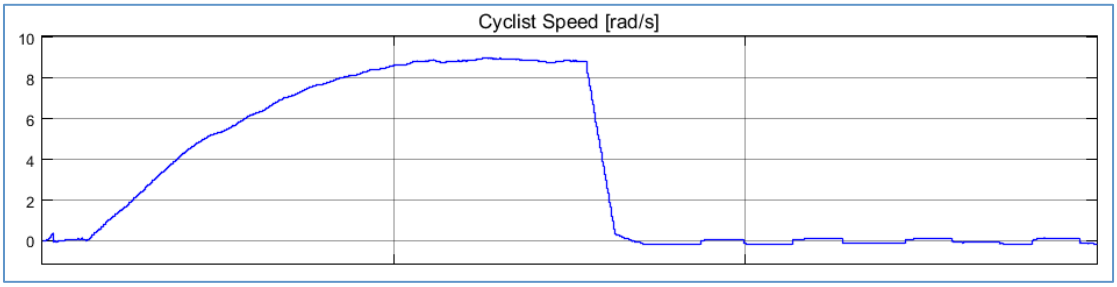

Figure 19: Cadence over a 15 second test with sudden stop in cyclist cranking.

4 Running Embedded

In this part, the embedded implementation of the state estimator and the control algorithm on the bicycle prototype will be discussed. The software for the prototype – and later for the actual bicycle – performs three major functions: state estimation, CVT control and motor control. Based on readings from the two electric motors and the crank sensor, the Kalman state estimator will estimate the state of the system and provide the CVT control with the information it needs to control the bike CVT. The output of the CVT control is a target speed for MG1 and a target torque for MG2. These targets are then fed to the motor control, continuously controlling the two motors as close as possible to their target speed and torque. These three functions should all be performed in real time of course, while the bike is running. Also, this functionality has to run on an embedded processor that can be built into the prototype.

4.1 Hardware

The hardware used for the prototype is a TI F28069M microcontroller with a 90MHz Piccolo family processor, plugged into a DRV 8301 motor driver board. All specifications of the microcontroller and driver board can be found on the TI website[x]. This MCU-driver board combination is used to drive a 750W brushless DC motor. Each motor – MG1 and MG2 – is controlled by it's own MCU-driver board configuration (Figure 20) in the prototype.

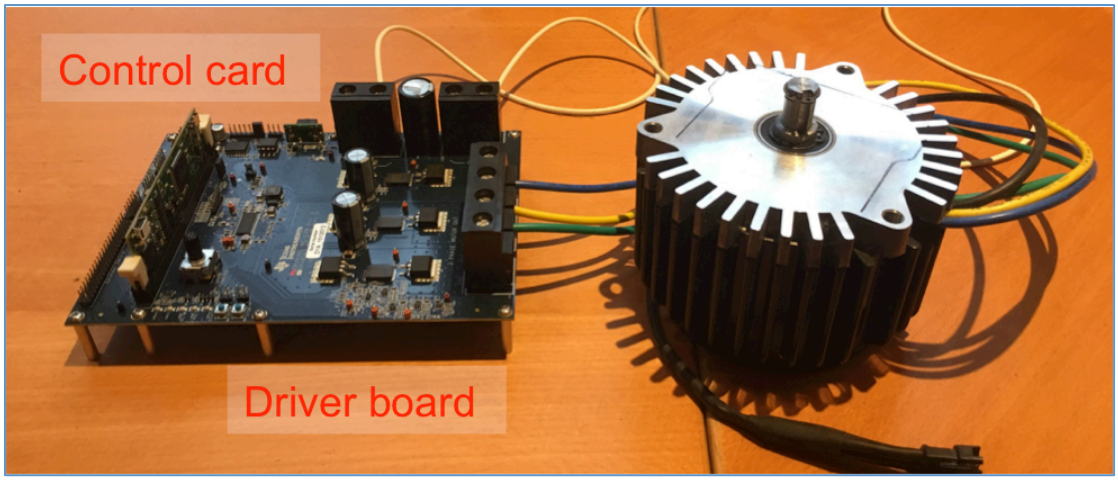

Figure 20: Hardware setup for running embedded

4.2 Software

4.2.1 Code Composer Studio (CCS) IDE

Code Composer Studio is TI's own integrated development environment (IDE) for programming and debugging microcontrollers. It contains compilers to generate optimised code for all TI processors from C-code programs written in the CCS IDE. The programs can be debugged in real time via a USB-JTAG connection that connects the microcontroller to the PC. The variables in the program can be monitored in real time in the watch window, variables can be plotted and changed while the program is running. There is also functionality for profiling the programs (timing).

4.2.2 Motorware

Motorware is a software package provided by TI containing a large collection of motor control solutions and example programs for motor control. It includes the TI instaSPIN FOC and instaSPIN MOTION programs. These are names for a collection of advanced motor control and identification algorithms. TI is very strong in sensorless control, using their FAST estimator (Flux, Angle, Speed, Torque) to replace a hall sensor or encoder. InstaSPIN MOTION includes the SpinTAC controller, that has been proven in Thomas Vleeschouwer's thesis to outperform the standard PI control. The example programs for motor control provided in Motorware will be used in this thesis as the starting point for the embedded program.

4.2.3 Embedded Coder[xi]

Typically, an embedded program is made and tested in a virtual environment, like Simulink, before it is deployed on embedded hardware for real-world

operation. This means that a program is, in most cases, written at least twice: once in the virtual environment (e.g. Simulink) and then again in a e.g. C-code in CCS for deployment to the hardware and debugging. For production purposes, this is probably the most efficient solution. But for prototyping and development, this approach takes a lot of time, since the iteration can not be done within a single software environment. Mathworks, the company behind Matlab and Simulink have developed a software package called Embedded Coder (EC) that has the ambition of completely integrating virtual testing and simulation and embedded operation. Embedded Coder provides specific tools for developing embedded programs. These tools are made for a stepwise approach for going from a completely virtual simulation on a PC, over simulation of microprocessors (emulation) in the simulation, via processor-in-the-loop simulation towards completely external simulation and debugging. In this external simulation, Simulink and Embedded Coder work like the watch window in a typical IDE like CCS, showing the variables of the program running on the microcontroller via the block diagram in Simulink. Combined with a support package for TI c2000 processors, Embedded Coder can also be used to generate optimised C-code for the specific microprocessor used and it can deploy the program directly to the hardware, without using CCS. The built-in functionality of the MCU is available through the use of Simulink library blocks that are part of the support package. This functionality is for example reading out ADC's, sending out analog signals, reading and setting GPIO (general purpose input-output) pins and communication protocols like SPI. TI motor control is, regrettably, not available in Embedded Coder. There are some basic functions that allow you to program a simple motor controller, but nothing close to the functionality available through Motorware.

4.3 Structure of the bicycle control program

As mentioned before, the control program needs to perform three main functions: state estimation, crank control and motor control. Not all of these tasks have the same priority. The motor control is what creates the actual interface with the cyclist. The cyclist feels the motors working. Also, the stiffness of the system as felt by the cyclist will depend on the bandwidth of the motor control. The motors also serve as sensors for the system, but since they work sensorless and thus rely on their FAST estimators, the motor control and motor state estimation (FAST) must always work perfectly. Once the motor control fails, the complete system will fail with it. The motor control must be guaranteed also for the cyclist's feeling. An interrupt of the motor control can result in a small shock or vibration felt by the cyclist. All this makes that motor control has the highest priority of the three tasks. The state estimation and crank control work together but also here, the control cannot work without the state estimation and so this estimation has a higher priority then the control. When the control runs a little slower, this will only make the bike reaction feel slow. When the estimation runs too slow, the bike will lose track of it's state and will become unstable.

These priorities have to be taken into account in the embedded program. On an embedded processor, there are three main levels of priority. The highest priority goes to basic functionality on the MCU, like running a timer, reading out ADC's and sending out interrupts. These functions are always running and when too

little calculation power is available, they will be the last to be stopped or delayed. Second on the priority ladder are the interrupt based functions. These functions are triggered by an interrupt signal and when the interrupt is given, the main program will be paused to run this interrupt based function. Once this function has been completed, the main program will resume. In the bicycle case, the motor control is interrupt based. The interrupts are sent out at a certain frequency (15kHz in current implementation), controlled by the timer functions that have, recall, the highest priority. This means that the motor control will always run at the specified frequency. At the bottom of the priority ladder is the main program. It runs in the background (therefore also called the background loop) and is paused every time a higher priority interrupt based function is triggered. This main program usually does not run at a specified rate, but rather as fast as possible. It is clear that the higher the rate of interrupts for e.g. motor control, the less time will remain for performing the main program. In the embedded program for the bike control, the state estimator and crank control algorithm are placed in the main program. This means that they will run as fast as possible, but at the lowest priority. Never will the motor control be slowed down by the estimator or the crank control. The priority ladder of the program is shown in Figure 21.

To summarize from highest priority to lowest: the program will read out ADC's from the sensors connected to the board as well as the motor drivers and run a timer. It will store both time and ADC data in data registers. Based on the timer, interrupts for various functions are being sent out at a specified rate. The motor control is such an interrupt based function, and it will run each time it receives an interrupt, pausing the main program. The motor control will thus run at the same rate the interrupts are given, provided that the function can be completed in the time between interrupts. The motor control will run the motors at the target speed and torque that is stored in a data register. Meanwhile it will itself store the data from the FAST estimator in a data register. The main function, that runs in the background, uses the ADC and motor data in the data registers to run the Kalman estimator. Based on the state estimation, the crank control is performed. This crank control updates the target values for the electric motor's speed and torque in the motor's data registers. The motors are thus at a very high rate controlling their speed and torque according to a certain target, but this target itself is updated at a lower rate.

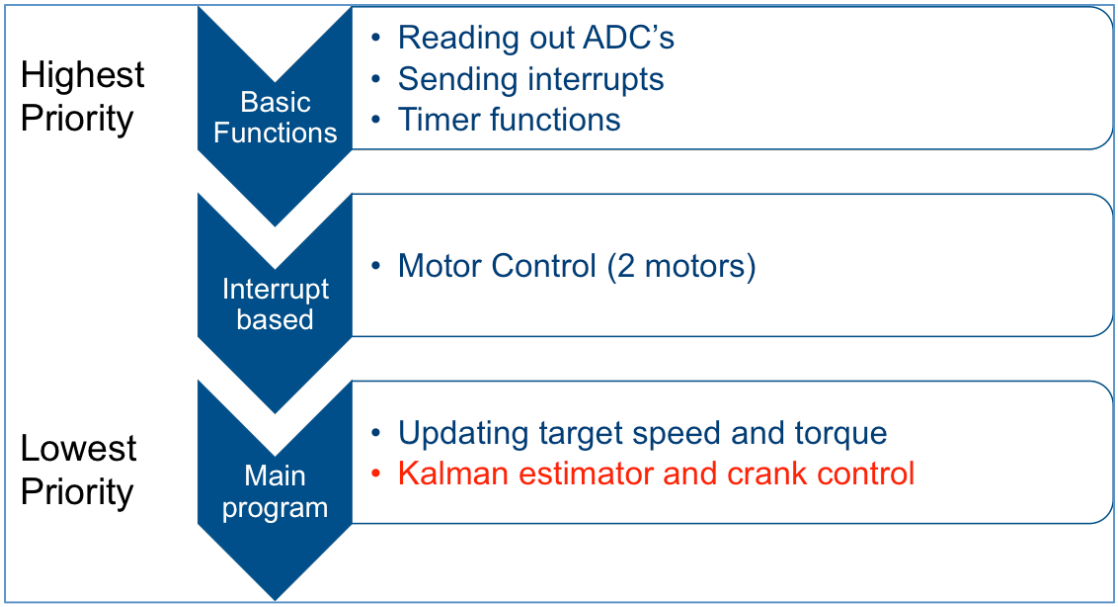

Figure 21: Priority ladder of the bicycle control program.

4.4 Running the program on a single processor

As a first milestone, the program as described above should run on a single microprocessor, using a laboratory hardware setup, and not necessarily in real time operation. This means: not necessarily already suited for the prototype. The MCU used is a TI F28069M. Later, as one processor proved not be enough for running the entire program at the desired rate (1kHz, see further), multiple MCU's will be used. This is discussed in later sections.

4.4.1 Write C-code directly in CCS

The obvious and most used way of making an embedded program is to directly program it in C-code in an IDE like CCS. The CCS functionality can then be used and loading the program on the MCU is straightforward. TI provides example programs with very sofisticated motor control that can be used as a starting point for making programs that require motor control. In figure 10 (priority ladder figuur), the blue text indicates the functionality that is already available in such an example program. The red text indicates what needs to be added. The Kalman estimator and crank control must be added in the main program. It is a major disadvantage of this approach that first testing the estimator and crank control algorithm in a virtual environment and then running it on the MCU requires first programming it in Simulink, and then a second time in CCS. Besides, knowledge of C-code is limited among mechanical engineers, let alone C-code that is optimised for MCU's. Although the direct C-coding approach might be interesting for production code, in the context of developing and testing it is not feasible.

4.4.2 Full use of Embedded Coder

Solving the above mentioned problem of having to do double work when it comes to developing an embedded program is exactly what Embedded Coder is designed for. Using Embedded Coder, the Kalman estimator and crank control programs tested in Simulink can, with minor adaptations, be compiled and loaded onto the MCU. There is no need of re-writing the program in C-code and all the work can be done in the Simulink environment, which is well known to

mechanical engineers in general. Support for the TI F28069M is good because of it being a widely used processor. Compiling the Simulink program for the microcontroller and deploying it to the board is no problem. As mentioned earlier however, TI motor control solutions are not available in EC. There are some simple examples for demonstration purposes and it is possible to program your own motor control in Simulink, but there are no ready-to-use programs available that have all the functionality of the TI motorware example programs. There are TI blocks for doing pulse width modulation (PWM) but the rest of the motor control needs to be programmed in Simulink manually. In Motorware, the TI SpinTAC controller can be used and the framework provides a lot of tuning and safety features that would all have to be implemented manually in the Simulink program. The example programs that are already available in EC are simply not good enough for doing the motor control in the bicycle. Since motor control is not the goal of this thesis and because programming motor control in Simulink as good as the TI control is a thesis on it's own, this option of using only Embedded Coder was abandoned.

4.4.3 Combination of Embedded Coder and CCS

In order to combine the use of the TI motor control example programs and the functionality of Embedded Coder and Simulink, Embedded Coder will be used not to program the MCU, but only to generate C-code for it. EC can generate C-code optimised for a specific MCU from a Simulink program. The Kalman estimator and crank control as tested in the Simulink simulation are being wrapped in subsystem blocks and isolated from the Simulink simulation in a different file. Embedded coder can now generate code for this file containing the estimator and crank control. In the settings of EC, the appearance of the generated C-code can be configured. It was configured so that the estimator and crank control were wrapped in functions. These functions use data from a data structure, also created by Embedded Coder, containing e.g. the current state of the system, to compute the target values for MG1 speed and MG2 torque. All the C-code generated by EC is stored in .c and .h files. Linking these files to the example program provided by TI, the functions for the Kalman estimator and crank control can be added to the program with minor adaptations. Some C-code needs to be written, but only minimal. This way, the estimator and crank control are implemented in the main loop of the example program as indicated in figure 6, without having to write the functions by hand. Updating the program is now faster of course then when the entire program needs to be hand written in C-code. It is however slower then when EC coder can be used directly for programming the MCU. This is because most adaptations in the Simulink blocks also require small adaptations in the C-program in CCS. Also the debugging needs to be done twice. An additional disadvantage is that the C-code generated by EC is not very readable. This makes it hard to optimally implement the EC C-code in the CCS program.

Using this approach, a working program is obtained. On the hardware setup consisting of two motors, one speed controlled and one torque controlled, the program can run both motors at the same time while updating the target speed and torque via the crank control algorithm. The estimator is of course not giving any reliable results since the hardware setup is not representative for the bicycle. Also an ADC readout representing the crank position sensor is

implemented in the program. Controlling the motors is fast and fluent, without any shocks or noise, as expected from the TI motor control. Also the control is easily tunable on-the-fly during a debugging session.
Important for using the program on the prototype is that it runs in real time. From the Simulink simulation of Simon Deruyter last year, 1kHz proved to be a good compromise between fast response of the state estimator and control and calculation power. A lower rate led to slower reaction of the Kalman filter to the sudden torque-drop on the crankshaft when the cyclist stops pedalling, making the controlled freewheel behaviour too slow. 1kHz is the goal for the embedded program also. Apart from this and probably more important however, the bandwidth of the control will be a major factor in the sensation of riding the bicycle. The higher the bandwidth, the stiffer the bicycle will feel. A low bandwidth will make for a "rubber band" feeling and a too high bandwidth will make the bicycle transmission uncomfortably stiff. The aim is to have a bandwidth in the control in the same range as the one of a chain drive in a conventional bicycle. Exact figures on this are difficult to find, but it will be lower than 1kHz probably. Altough the control in itself can have a somewhat lower bandwidth, the estimator needs a high frequency to pick up fast transients.
Adding a timer function to the main program (background loop), it is observed that this runs at a frequency of approximately 5Hz. This is off course way too slow. The updating of the target speed can be heard also at a frequency of 5Hz in the motor control. Because the Simulink program is not converted to a fixed point program and thus requires a lot of floating point calculations, it can be speeded up by using the MCU's floating point unit (fpu) for the calculations. Using the fpu enables speeds up to 50Hz, when only MG1 is controlled. This also is still much too slow. However these low frequencies of the estimator and crank control prove in a way the priority concept explained earlier. The motor control is completely unaffected by the rate of the estimator, it has no problem with maintaining the given target speed. The slow estimator rate can only be observed true the discrete steps in motor speed, every time this speed is updated by the crank control.
Table 1 shows a comparison of the three methods described above.

Table 1: Comparison of methods for programming 1 MCU with the bicycle control program.

	All C-code in CCS	Only Embedded Coder	EC C-code into TI motor control lab
Programming in C	Yes	No	Yes, but not a lot
TI motor control	Available	Not Available	Available
Use of EC tools	No	Yes	No
Fast update of algoritm	No	Yes	Yes, with small adaptations
Use of Matlab/ Simulink	No	Yes	Yes
Optimised Code	Yes	Yes	Not guarenteed
Practical problems	I'm no C-coder	Motor control in EC not good enough	Estimator runs too slow

4.5 Three-processor solution

Using only one processor, it is not possible to run real time at an acceptable rate with the state estimator and the control algorithm as it is. In order not to spend too much time on optimising the program to run on a single processor, three processor are therefore being used to run the program on the prototype. They are arranged in a configuration as in Figure 22.

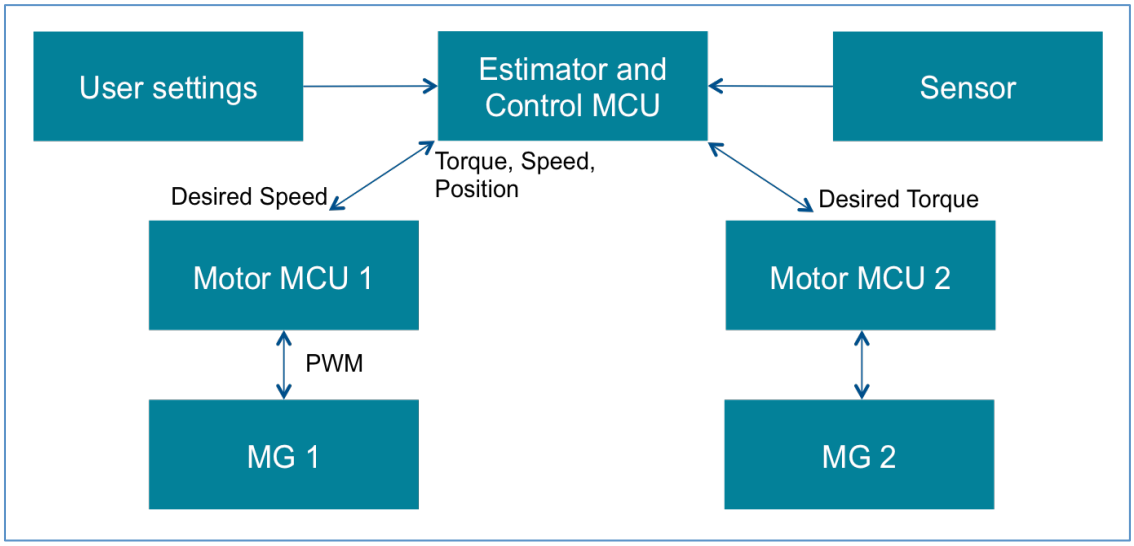

Figure 22: 3 processor configuration.

Each motor will use one processor for the motor control. These processors will receive their motor target speed and torque via SPI (Serial Peripheral Interface bus) communication and will communicate the motor speed, position and torque to the master MCU. This master MCU runs the Kalman estimator and CVT control. It receives data from both motors, as well as from sensors and maybe user settings, like a desired cadence. SPI is the communication protocol of choice

because it is a processor standard, it is fast and the bicycle system has a clear master-slave hierarchy with only two communication nodes. CAN, which is another much used standard, is more useful when communicating in a network with more than two nodes and without a clear hierarchy.
Using this configuration has several advantages in the context of developing the program. First of all, the master processor can be chosen based on the calculation power required for the estimator, creating some overhead. The motor control from TI, that only works on some TI MCU's, does not change with the master MCU choice. For the motor control processors, the CCS workflow can be followed as it is intended because no extra functionality needs to be added to the example motor control programs, except for SPI communication. For the master MCU on the other hand, Embedded Coder can be used to it's full functionality since no motor control needs to be done anymore on this MCU. Real-time debugging in Simulink is more convenient than in CCS because the maximum rate of the scopes is above 100Hz in Simulink compared to 2Hz in CCS. Towards production, this three processor solutions is probably not the most optimal, since it requires three processors instead of one. But it has clear advantages over the single processor approach when it comes to developing the prototype. Table 2 lists these advantages.

Table 2: Advantages of three processor solution for developing.

	EC for estimator, TI C-code for motor control
Programming in C	Yes, but not a lot
TI motor control	Available
Use of EC tools	Yes
Fast update of algoritm	Yes
Use of Matlab/Simulink	Yes
Optimised Code	Yes
Added benefit	Choice of processor for estimator independent of motor control processors

4.5.1 TI MCU as master processor

Simulink can be used to program TI MCU's via the Embedded Coder package. The TI master MCU is programmed in this way. The complete configuration as intended for the prototype is shown in Figure 23. On the first prototype, MG2 is not implemented yet because it is not useful in the tests on rollers.

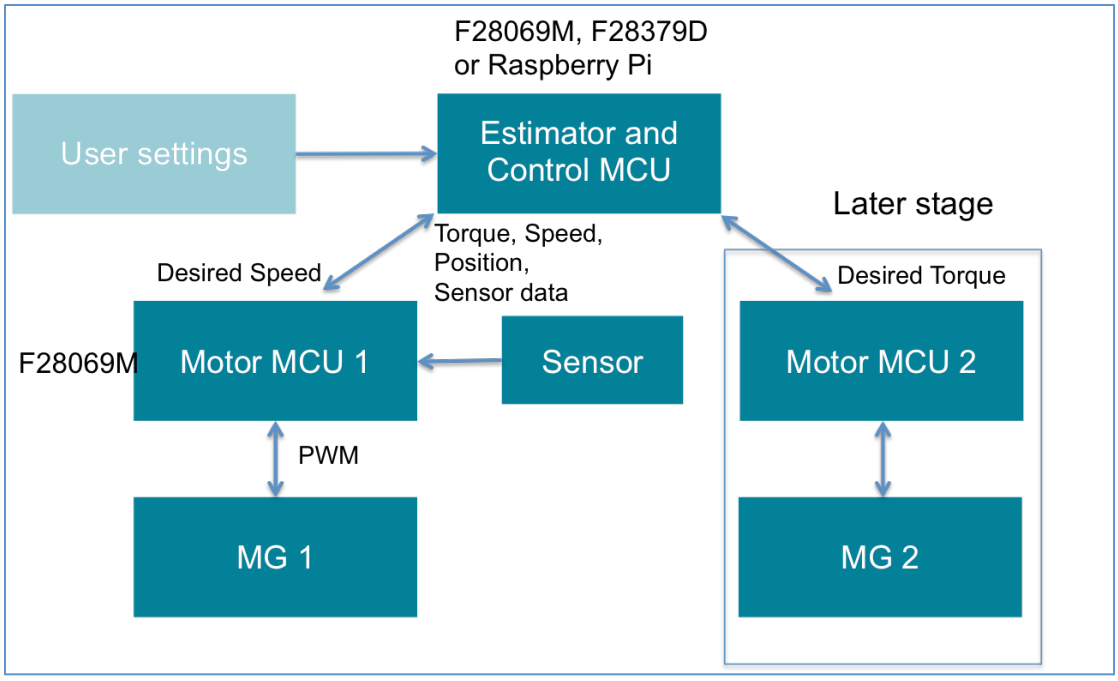

Figure 23: 3 MCU configuration for prototype.

The sensor readout is performed not by the master but by the slave MCU. Via SPI communication, 4 measurements are being sent from the slave to the master. On the master MCU, the state estimation for that timestep is performed and based on this state estimate, the target speed for MG1 is calculated by the control algorithm. This target speed is then sent back from the master to the slave. The flow of the program that runs on the prototype is shown in Figure 24. Notice that SPI communication only support uint16 values on the TI platform. The values have to be coded and decoded manually. In order to keep everything in sync, it is the SPI that triggers execution of the programs running on the MCU's.

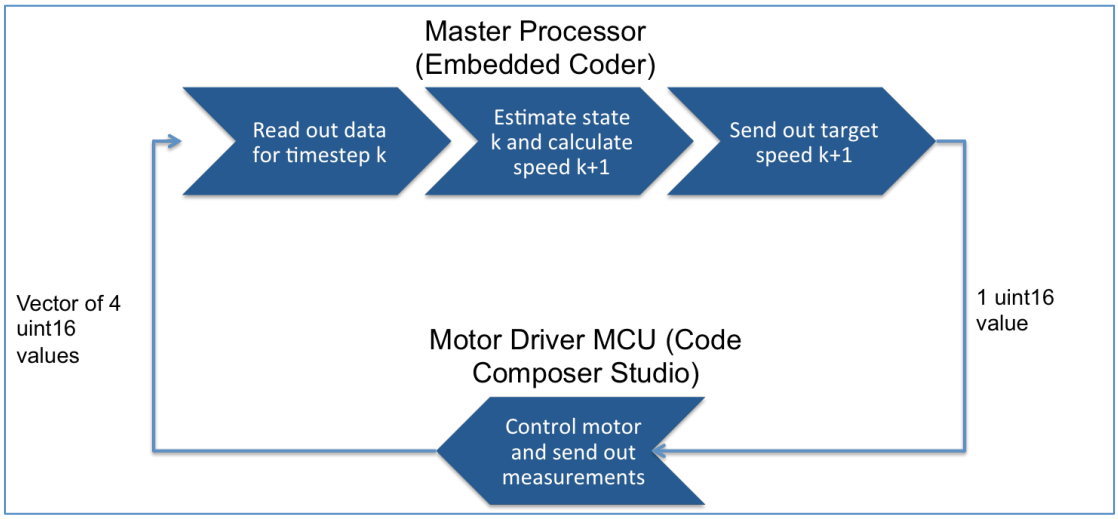

Figure 24: Flow of the prototype program.

In Figure 25, the program running on the master MCU is explained in further detail. The TI MCU's support 64 bit floating points but cannot calculate them directly in hardware. They are only 32 bit processors. Therefore all values in the

program have to be converted to 32 bit floating point numbers for the calculations.

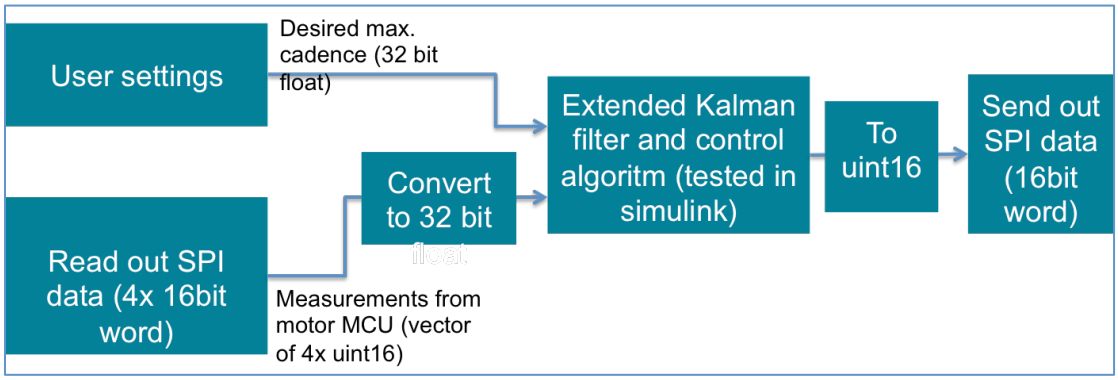

Figure 25: Workflow of the master MCU program.

Several problems occur in the implementation with three TI processors. The slave MCU's work perfectly as they are being programmed via CCS. The master MCU on the other hand is programmed via Embedded Coder. This gives rise to three major problems:

1) The SPI communication implementation for TI processors in Embedded Coder does not provide the functionality of sending or receiving words of multiple uint16 numbers, as is required to send the four measurements to the master MCU. Attempts to send the values one-by-one prove not to be robust enough to ensure a reliable data transfer.
2) Matlab and Simulink are 64 bit environments. The Kalman state estimator and control algorithm are written in Matlab code and contain a lot of parameters that are defined in this code. By default, these are all 64 bit floating point numbers. Embedded Coder does not provide a tool to automatically cast these numbers to 32 bit floating point numbers, and in the generated code they all appear as 64 bit floating point numbers for that reason. The TI processors support 64 bit numbers, so this does not call an error. However, since all calculations with these numbers need to be done in software instead of hardware, this slows down the program tremendously. Programming the master MCU with Embedded Coder, it is not possible to go beyond a rate of 20Hz. Theoretically though, simply estimating the rate based on relative clocking speed of the MCU compared to the PC, 120-130Hz should be possible with the F28069M. The only solution for this problem is to manually cast all parameters and values in the Matlab code to 32 bit floating point numbers. This would not only be a tedious job in itself, it would also make further development of the Matlab code more difficult.
3) Even the latest version of Embedded Coder does not support the specific F28069M MCU, which is slightly different from the normal F28069. Code generation is possible, but serial communication and debugging via the so-called external mode simulation is not. This makes debugging the master program very difficult, needing to manually send the data to be checked to an external scope to monitor it.

Because of these problems and also because of the rather limited potential in terms of calculating power of the TI MCU's, the TI processor are decided to be

unsuited for use as the master MCU in the prototype. Although Embedded Coder has some potential for programming the TI processors, the implementation of the Kalman filter and the control algorithm is too much for this platform.

4.5.2 Raspberry Pi[xii] as master processor

Mainly because of the unsatisfactory results obtained with the TI MCU solution explained in the previous section, Raspberry Pi model 3B (Figure 26) is used as the master processor. The workflow is entirely the same as in the implementation with the TI MCU as master. Raspberry Pi is no longer an MCU, but rather a small computer running it's own operating system. It has a 64 bit 1.2Ghz quad-core processor, eliminating the problem of converting all values to 32 bit floats. The Raspberry Pi is completely open source and is very well supported by Matlab and Simulink, even without the need for Embedded Coder. In contrast to TI, the SPI communication blocks provided for Raspberry Pi in Simulink are very intuitive to use and allow for sending multiple uint16 words. The external mode simulation for debugging works perfectly and all Simulink functionality like scoping, changing values on-the-fly and storing data in the workspace are supported in this external mode. Debugging is very convenient in this way.

Figure 26: Raspberry Pi Microcomputer.

Because it is running on an operating system, real time operation is not guarenteed on the Raspberry Pi platform. To ensure real time operation, a real time check is added to the program. When this check notices that real time has been lost, the motor is being shut down for safety. The real time check is performed on the slave MCU. The slave keeps track of the time between incoming SPI calls. Since the slave runs the motor control at a rate of 15kHz and the master runs at 1kHz, 15 motor control interrupts should be counted in between SPI calls. If it takes too long to receive the next call, then real time is lost on the master and the slave shuts down the motor. This is to ensure that the motor does not behave in an unpredictable way, possibly damaging the prototype setup or hurting the cyclist.

Using Raspberry Pi, a rate of 1kHz is easily achieved without any adaptations to the program in Simulink.

4.6 Data type considerations

Use of the proper data type is an important aspect of programming an embedded application. The data type has a large impact on the performance, accuracy and stability of the program. For microprocessor applications it used to be and in fact still is common practice to use fixed point data types because they can be used more efficiently in calculations. Modern MCU's have floating point units to handle also floating point numbers efficiently. Because the bicycle program is still in a development phase and therefore adjustments to the program are constantly being made, fixed point numbers are not very convenient to work with. The matlab and simulink standard number format is 64 bit floating point. To efficiently program, test and adjust the Kalman estimator and control algorithm in Matlab and Simulink, this standard format is used in all calculations. As mentioned before, the TI MCU's support 64 bit floating point numbers in software but they are only 32 bit processors. Using 64 bit floats on these MCU's results in a large overhead for doing the calculations in software. This is not the case for the 64 bit Raspberry Pi.
Some care should be taken when using floating point numbers. Unlike integers, that always have an accuracy of 1, the absolute accuracy of a floating point number varies with the absolute value of the number. This is because a floating point number is represented in the following way:

$$significand \,.\, 2^{exponent}$$

The significand is always a number between 0 and 2 and has a fixed number of bits for it's representation (52 bit in a 64 bit floating point). The exponent is an integer value. This means that large numbers (large exponents) have an absolute accuracy that is lower than small numbers (small exponents). It is important to keep this in mind when calculating absolute errors, like in the Kalman filter between the measurements and the predicted state-space outputs, because this error will be less accurate in absolute terms when the values that are compared are large. In most cases, 64 bit floating point numbers provide way more accuracy than is required and this does not pose problems. Only when a certain number treatens to become very large – as in the case of angle unwrapping or absolute time counting – it should be checked if the required accuracy is not lost at some point. In section 5.4, unwrapping angles is discussed.
Once the program is ready for production, it is advisable to reprogram it with optimised data storage and in fixed-point data types. 32 bit numbers are more than accurate enough for the program controlling the bicycle. At this point, the trade-off between efficiently updating the program and saving on computational power is probably more towards saving computational effort, since this will lower the cost of the microcontrollers.

4.7 Bicycle prototype configuration

The prototype uses three processors and two 750W brushless DC motors. Each of the motors has it's own DRV 8301 motor drive board with a TI F28069M MCU controlling the motor. The motor control MCU's run an example program

provided by TI in the Motorware package, including the spinTAC controller, that has been adapted to do SPI communication with the master and to read out an extra sensor. The MG1 controller reads out the crank position sensor, while the MG2 controller reads out a crank torque sensor that has been added to the prototype for validation of the cyclist torque estimation. In both controllers some safety features have also been added that stop the motors when the master looses real time or when the emergency stop button is activated. The TI MCU's are debugged and monitored via the CCS toolchain. In debug mode, the motors can be manually switched on and SPI communication can be checked while the program is running on the board.
The master controller is a Raspberry Pi Model 3B. It runs the Simulink model of the bicycle control program and is programmed directly from Simulink. The debugging is performed via the external mode simulation. The Raspberry Pi is connected to the debug computer via a wifi network set up by the Raspberry Pi itself. Via the scoping functionality in Simulink, all estimated states and incoming measurements can be very easily monitored and compared.
At first, the prototype is tested on rollers, powered not by the 36V battery but by a 24V power supply that has more safety systems to prevent overloading. Only the rear wheel is resting on a roller, the front wheel is sitting on the ground. Therefore in this setup, MG2 is not powered. Two rollers are being used: one with a simple fan load and one with an eddy current brake and a controllable load. The simple fan load is not dimensioned for the combined power of MG1 and the cyclist and proves too little load for testing the E-bike. The controllable load can deliver a maximum of 900W, which is sufficient to take up the combined power of the cyclist (100-150W) and MG1 (750W), although at low speeds the brake cannot deliver enough torque.
In later testing stages, first the power supply is replaced by the actual battery and later the bicycle is tested on road instead of on the roller stand. When testing on road, MG2 can be powered.

4.8 Conclusion

In order to combine the development, debugging and virtual testing capabilities of Simulink with the motor control solutions from TI Motorware, the prototype is equipped with three processors. A Raspberry Pi mini-computer serves as master, running the Kalman estimator and bicycle control at a rate of 1kHz. It also controls the SPI communication to both slave MCU's. These slave MCU's run a adapted Motorware motor control program, controlling MG1 and MG2 at a rate of 15kHz. The motor measurements are sent back to the master through SPI.

5 Tuning the Kalman filter

5.1 Determination of measurement noise covariance

Recall the bicycle state-space model:

$$\boldsymbol{x}' = A.\boldsymbol{x} + \boldsymbol{w}(t)$$
$$\boldsymbol{y} = C.\boldsymbol{x} + \boldsymbol{v}(t)$$

The vectors **w** and **v** indicate the noise on the state-space equations. **w** is the process noise, it accounts for deviations between the model and the actual system. The more reliable the model, the smaller the values that are required to be used in **w**. **w** can change over time, hence the general notation **w**(t). In the Kalman filter used in the prototype however, **w** is taken to be constant. **v** is the measurement noise vector. It accounts for the fact that the measurement values **y** are not exactly equal to the measured values C.**x**. Both **w** and **v** are assumed to have a normal distribution with zero mean. The variances of the distributions are given by the vectors **Q** and **R**:

$$\boldsymbol{w}(t) \sim \mathcal{N}(0, \boldsymbol{Q}) \, , \boldsymbol{v}(t) \sim \mathcal{N}(0, \boldsymbol{R})$$

R is the measurement covariance vector. The values in the vector are determined by the accuracy of the measurements. The measurements are assumed to have no bias (zero mean). The lower the values in **R**, the more accurate the measurements. It is very important to have good values for **R**, since this determines how much weight is given to the measurements in the Kalman filter.
There are four measurements in the system: MG1 position, speed and torque and crank position. For MG1 speed and torque, the variance of the measurements was determined in Simulink by means of the variance block on the measurement signal, while the motor speed and torque were being kept constant (zero speed and torque).
For the angle measurements, a more theoretical approach was followed. The position of MG1 is measured by means of a Hall sensor. This sensor works in discrete steps. One revolution is divided in 48 discrete steps. The resulting signal is digital and does hence not add any noise to the measurement. The approach in Simulink would result in a covariance of zero, since there is no noise on the signal itself. Because of the discrete steps, the measurement error is not actually normally distributed but uniformly, since there is equal probability of being anywhere within the range of one step. The variance of the measurement error can be calculated with the formula for variance of a uniform distribution:

$$\frac{1}{12}(b-a)^2$$

b and a being the boundaries of the interval. For the Hall sensor this gives:

$$b - a = \frac{2\pi \, rad}{48} = 0{,}139 \frac{rad}{step}$$

$$Var = \frac{1}{12}(0{,}139 \, rad)^2 = 1{,}4 \, . \, 10^{-3} \, rad^2$$

A similar reasoning can be followed for the crank position sensor. This is also a digital sensor. The maximum error this sensor can make is 1% of one revolution, occuring at the 360°-0° crossing. A 1% threshold is used here to avoid false values in between 0° and 360° when crossing zero. Only if the value falls in the direct proximity of 0° or 360° will it be used directly, otherwise it is set to either 0° or 360°, whichever is closer.

$$b - a = \frac{2\pi\ rad}{100} = 0{,}0628\frac{rad}{step}$$

$$Var = \frac{1}{12}(0{,}0628\ rad)^2 = 3{,}3\ .10^{-4}\ rad^2$$

The uniformly distributed noise on the position sensors is thus approximated by normally distributed noise with the same mean and variance. The complete measurement covariance vector is:

$$\boldsymbol{R} = (1{,}4\ .10^{-3}\ \ 0.1\ \ 0.001\ \ 3{,}3\ .10^{-4})$$

5.2 Tuning the process noise covariance

Whereas the measurement noise covariance is a property of the measurement system (i.e. the sensors) and as such may be determined empirically, the process noise covariance can be considered as a parameter of the Kalman filter that requires some level of tuning. The values of **w** indicate the "freedom" of the system to deviate from it's current state in the period of one timestep. For the actual system states, this means that the states can evolve in a way that is different from the one given by the equations:

$$\boldsymbol{x}' = A.\boldsymbol{x}$$

The model is never perfect and can only show a certain trend in the state change of the system. The exact states will always have to be determined based on the measurements. The higher the process noise covariance, the more the measurement will be used to steer independently the system states. If the process noise covariance is zero, this means that the model equation is exact and thus the states can only change according to this model equations.
For the augmented states – the estimated inputs and parameters of the system – there is no model-based evolution:

$$\boldsymbol{x}'_{augmented} = \boldsymbol{w}(t)$$

These states are assumed to change in a completely random way according to the random-walk model. The change of these states is normally distributed. A high covariance thus indicates that the corresponding state can change rapidly, whereas a low covariance means that the state only changes slowly. Tuning the covariances for the different augmented states should be done so that they reflect the expected dynamics of the states. The complete state vector of the system is:

$$\boldsymbol{x^T} = \left(\theta_{sun}\ \theta_{carrier}\ \omega_{sun}\ \omega_{carrier}\ T_{cyclist,offset}\ T_{cyclist,max}\ \ \phi\ \ T_{MG1}\ \ T_{MG2}\ \ \alpha_{eq}\right)$$

Both position states are determined very accurately via the position measurements on MG1 and the crankshaft. Therefore these states get a high covariance value in order to follow the measurements quickly. The rotational speed states change according to the model mostly, there are not enough

measurements to directly determine them. The covariance on these states should be an order lower than on the position states. This will make them more dependent on the rate of change given by the model. The model is off course not perfect, so the covariance cannot be zero. The cyclist torque states are augmented states, they are assumed to vary randomly. The covariance on these states should reflect their expected dynamics. In Deruyter's thesis, it was shown that the cyclist could bring the torque on the pedals to zero in around 200ms, coming from a nominal torque. This rate of change is an indicator for the order of magnitude of the covariance.

$$\left(\frac{dT_{cyclist}}{dt}\right)_{max} \cong \frac{T_{max}}{\Delta t} = \frac{100\ Nm}{200\ ms} = 500\frac{Nm}{s}$$

The covariance on the two cyclist torque states should be somewhere in this order of magnitude. The exact value of the covariance is always the result of experimenting and tuning. For the phase offset, a similar reasoning applies. Imagine the torque phase can change 180° within one revolution. When rotating at 120rpm (very high already), this would mean that this phase change happens in half a second:

$$\left(\frac{d\phi}{dt}\right)_{max} = \frac{\pi\ rad}{500\ ms} = 2\pi\frac{rad}{s}$$

This value should be used as a maximum since the phase is not expected to change this much. It does however give a good starting point for tuning the covariance in terms of order of magnitude.

Both motor torques are measured directly and should have a very high covariance. Choosing the covariance slightly lower will filter the measurements somewhat, topping of very big changes. In the application of the filter without MG2, the process noise covariance on MG2 should be set to 0 and the torque itself should be manually set to zero in every timestep. This way optimality of the entire filter is not lost. When MG2 is not being powered, both the value of the MG2 torque is known to be zero and this value is known not to change, so the covariance should indeed be zero also.

Finally, the gradient state is changing very slowly. This state includes both the actual road gradient and friction losses not included in the model. In both cases the expected dynamic is small compared to the rest of the states. Imagine riding up a ramp at an inclination of 30°. This would mean a change in gradient of approximately half a radian, in a time of about a second. Half a radian per second can be taken as an absolute maximum rate of change, occurring perhaps 0.1% of the time. Just to make an estimate of the variance:

$$3{,}29\sigma = 0{,}5\frac{rad}{s}$$

$$\sigma = 0{,}15\frac{rad}{s}$$

$$Var = \sigma^2 = 0{,}023\left(\frac{rad}{s}\right)^2$$

Using these values as a starting point and tuning the process noise covariance based on experiments to achieve the desired behaviour from the filter yields the following process noise covariance vector:

$$\boldsymbol{Q} = \left(1 \;\; 1 \;\; 0{,}1 \;\; 0{,}1 \;\; 100 \;\; 200 \;\; \frac{\pi}{2} \;\; 10000 \;\; 10000 \;\; 0{,}001\right)$$

5.3 State estimation based on working modes

Not all estimated augmented states are relevant at all times. Some augmented states are even not observable when the bicycle is in a certain state. The sine assumption for the cyclist torque only holds when the crank is rotating and the cyclist is in a more or less steady state regime. Once in this regime, all three augmented states adding to the actual torque on the crank shaft can be estimated. When the crankshaft is standing still however, the torque on it is constant and not a sine. The sine assumption is in this case not relevant and creates an extra degree of freedom in the system, because of the fact that the constant torque will be estimated as the combination of three augmented states:

$$T_{cyclist} = T_{cyclist,max} \,.\, \left(0.5 \,.\, \sin\left(2 \,.\, \theta_{cyclist} + \phi\right) + 0.5\right) + abs\left(T_{cyclist,offset}\right).\, sgn\left(T_{cyclist,max}\right)$$

In case of a constant cyclist torque, the three augmented states can be uniquely determined if the phase offset is fixed and the crank is rotating. This would make the maximum torque zero and the offset torque equal to the constant torque on the crankshaft. This situation will only occur when the cyclist torque is zero. When the crank is standing still however, there is no guarantee that the states can be uniquely determined when the phase offset is fixed (e.g. the maximum torque multiplied by zero would make this torque undetermined). In order to avoid this problem, the state estimation will only estimate a subset of states when the torque on the pedals is constant. 3 working modes are considered for this: a standstill mode, a constant torque mode and a sine mode. The standstill mode is used when the bicycle is not moving, with all shafts stationary. In this mode only the torque offset state is estimated, not the torque maximum and the phase offset. The phase offset is set to a default value of -45°. The maximum torque is manually set to zero. Because the values are manually set, also the process noise covariances are set to zero. When the bicycle is not moving, the gradient is also unobservable. This state is therefore also not estimated when in standstill mode. When the bicycle is moving but the torque is constant – i.e. when the torque is below the torque threshold or when it is negative – a “constant torque mode” is activated in which the gradient is estimated. When the bicycle is moving faster than a certain minimum speed this state is observable and relevant.

The mode switching works like a state machine. When in standstill mode and applying a force on the pedals, the crank will either start rotating in accordance with the bicycle speed (fixed gear behaviour) or stay still if the torque is not high enough to put the bicycle into motion. In either case, it is best to estimate the torque as if it were constant. When the bicycle is moving slowly and therefore the crank is rotating slowly, the torque can be estimated as a constant. Once the

bicycle speed goes above a minimum velocity where the gradient estimation becomes relevant, the estimator switches to the constant torque mode. The road gradient is now estimated. If the torque is above the threshold, which indicates the cyclist is cranking, the estimator switches to the sine assumption mode. The bicycle is now moving at speed and the cyclist is cranking, so the crankshaft is rotating. Notice that the intermediate constant torque state will be skipped often, as the bicycle will be accelerating most of the time because the cyclist is cranking. In the rare case where the bicycle would be accelerating just from e.g. rolling down a slope, the estimator would remain in the constant torque state. When the cyclist stops cranking with the bicycle moving at speed, the control will stop or slow down the crank and the estimator will switch to constant torque mode. Slowing down below the minimum speed for relevant gradient estimation will make the estimator switch back to standstill mode. No hysteresis is needed between standstill mode and constant torque mode because switching between these two modes creates no discontinuities. This state machine is shown in Figure 27. In the switching between constant torque mode and sine assumption mode, hysteresis is built in to prevent switching constantly between modes when the torque is close to the threshold. This would make the torque estimate useless as it would not have time to converge before switching to another mode again. The upper threshold value is 30Nm, the lower one is 20Nm. Putting these values further apart will make the mode switching more stable but will create a larger ambiguous region. The values should be as close together as possible without making the mode switching unstable.

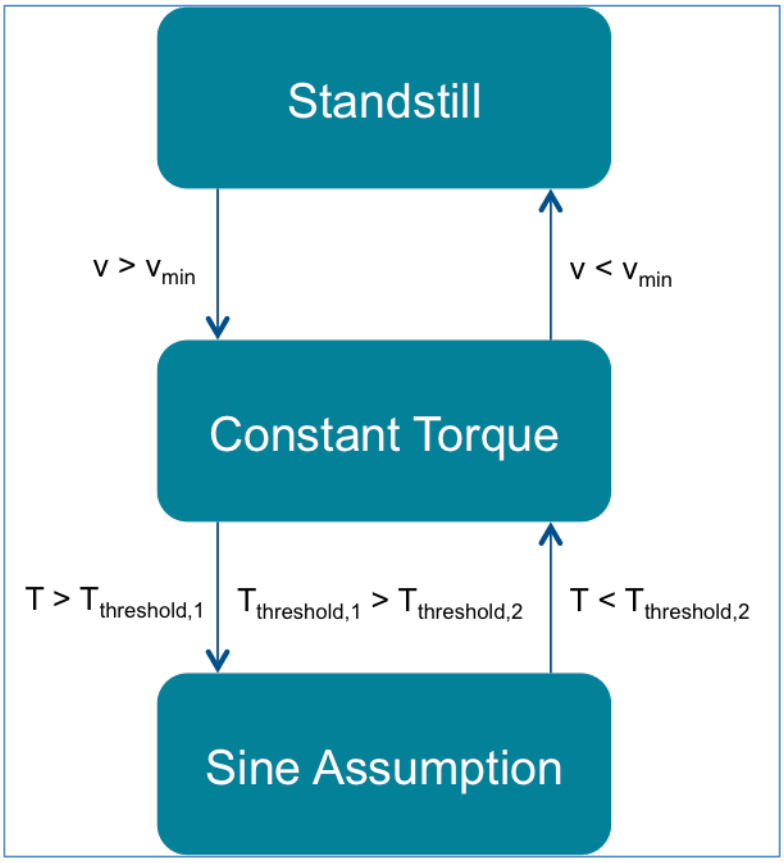

Figure 27: Estimator modes state machine.

5.4 Angle unwrap

The position states in the Kalman estimator are represented as unwrapped angles. This means that they can be any value, not necessarily in the interval [0,2π]. The reason for this is that the derivative (the rotational speed) can always be calculated as the finite difference between two consecutive position values. The alternative is to use wrapped angles, always in the interval [0,2π]. Calculating the derivative is then slightly more complicated as it is required to check whether the position has crossed zero. Using wrapped angles has the advantage of having no danger for overflow.

Two potential problems need to be considered when using unwrapped angles. The first problem is overflow. The angles are being represented by a 64 bit floating point number. Up to 2^{43} radians, all values are respresented up to an accuracy of 1mrad [xiii], which is the desired accuracy for the position measurements. Once higher then this maximum, accuracy is lost in the floating point respresentation and this can be named overflow. Assume the crankshaft to rotate at a constant speed of 120rpm or 2 rotations per second. This is 4π radians per second. It would take then:

$$\frac{2^{43}}{4\pi} = 7\,.\,10^{11}\ s = 1{,}94\,.\,10^{8}\ h = 22\,196\ years$$

To reach the maximum representable number. MG1 rotates faster than the crankshaft but it will still take literally ages before the angular position of the motor overflows. Moreover, MG1 rotates both forward and in reverse. Overflow is clearly not a problem.

The second problem is absolute accuracy. When floating point numbers become larger, the number of significant digits stays the same, since the fraction part of the floating point number remains the same size. This means that the absolute error between two floating point numbers becomes less and less accurate when the numbers become larger, whereas the relative error accuracy remains the same. In the Kalman filter, an absolute accuracy of 1mrad is desired as this is required for the resolution of the crank sensor (12 bit for 2π range = 0.0015rad resolution). As shown above, this accuracy is lost once the number becomes larger then 2^{43}. This will never occur in the lifetime of the bicycle. Besides, the values are being reset every time the bicycle is stopped and shut off. To conclude, using unwrapped angles should not be a problem in term of accuracy or overflow.

Incoming measurements are wrapped angles. In Simulink they are being unwrapped by means of an unwrap block. Such a block monitors the difference between the current value and the previous value: whenever this difference is larger than π, it means that the angle value has crossed the zero-2π boundary. In those cases, the sign of the difference can be used to decide the unwrapping direction.

6 Prototype test results

6.1 Tests on roller stand

The bicycle is first tested with the rear wheel on a roller stand (Figure 11). The rear wheel drives a roller attached to a load. The front wheel is resting on the ground. In the first tests, the bicycle is powered by a large 24V power source with overload protection and current regulation. Only when the tests prove that the system behaves as expected, the 36V battery is used. In this test setup, MG2 is not powered because the front wheel is not supposed to move. The adapted Kalman filter for MG1-only operation is used. Only one motor controller slave MCU is connected to the Raspberry Pi master in this first prototype.

Testing on the roller stand is useful to check the stability and performance of the Kalman filter, to tune it's parameters – the process noise covariances and the threshold values for switching of modes – and to determine experimentally the measurement noise covariance. Debugging the setup is easier on the roller stand because the bicycle is stationary. The control algorithm behaviour can only really be assessed during real road testing, but the basic behaviour and stability can be investigated already on the roller stand.

After tuning the process noise covariance of the filter and determining the measurement noise covariance values as explained in section 5.1, several tests are performed to investigate the performance of the state estimator and the control algorithm. The prototype control and state estimator are running at a rate of 1kHz in the experiments.

6.1.1 Constant MG1 speed estimator test

Before using the control algorithm to control the bicycle and setting the desired MG1 speed, MG1 is simply made to rotate at a constant speed to monitor the Kalman filter accuracy for different MG1 speeds (Figure 28, Figure 29, Figure 30). At 500rpm and 50rpm, measurements from MG1 are good. The estimator follows the measurements very well. This is also the case at 5rpm, but here the measurements from MG1 for speed and torque are clearly less accurate. This is because of the FAST estimator, that is only accurate above 10rpm.

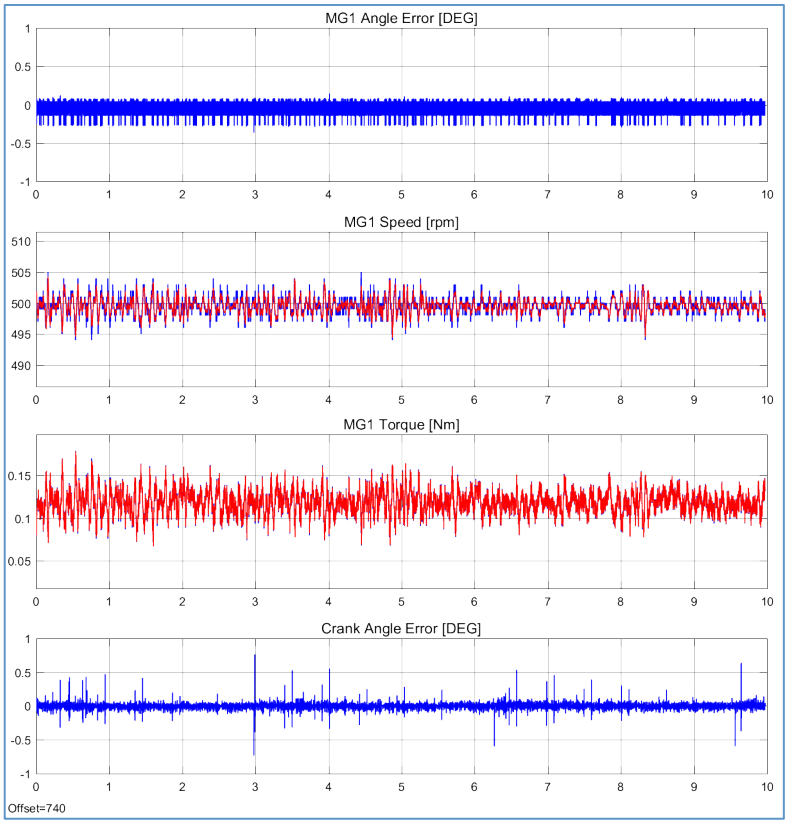

Figure 28: Kalman filter accuracy with MG1 rotating at 500rpm. (Subplots 1 and 4 show the absolute error between the measured angle and the estimated angle. Subplot 2 and 3 show the measurement in blue and the estimate in red)

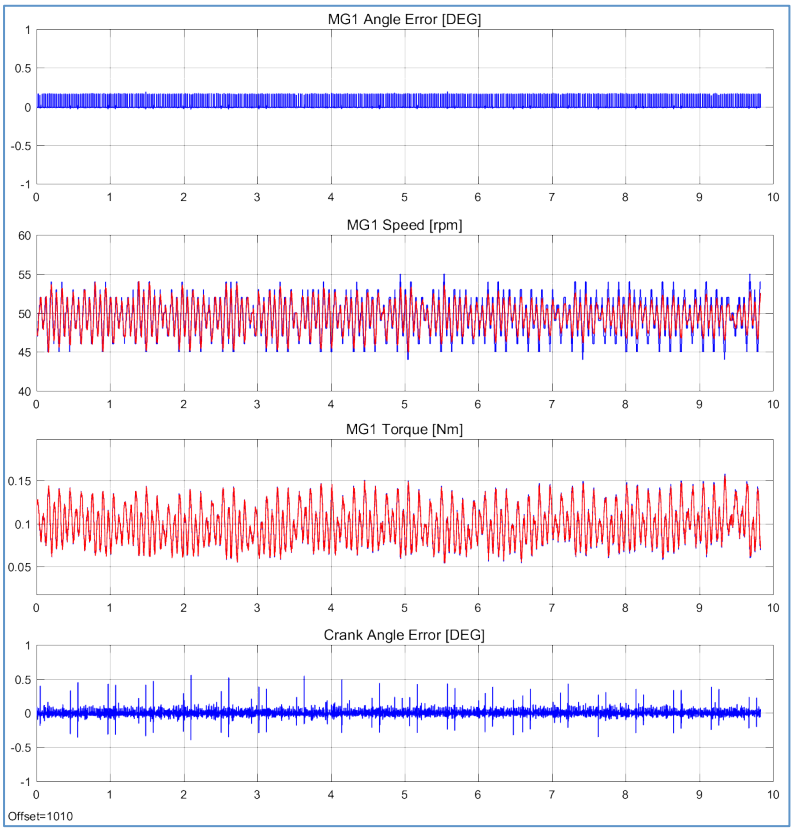

Figure 29: Kalman filter accuracy with MG1 rotating at 50rpm. (Subplots 1 and 4 show the absolute error between the measured angle and the estimated angle. Subplot 2 and 3 show the measurement in blue and the estimate in red)

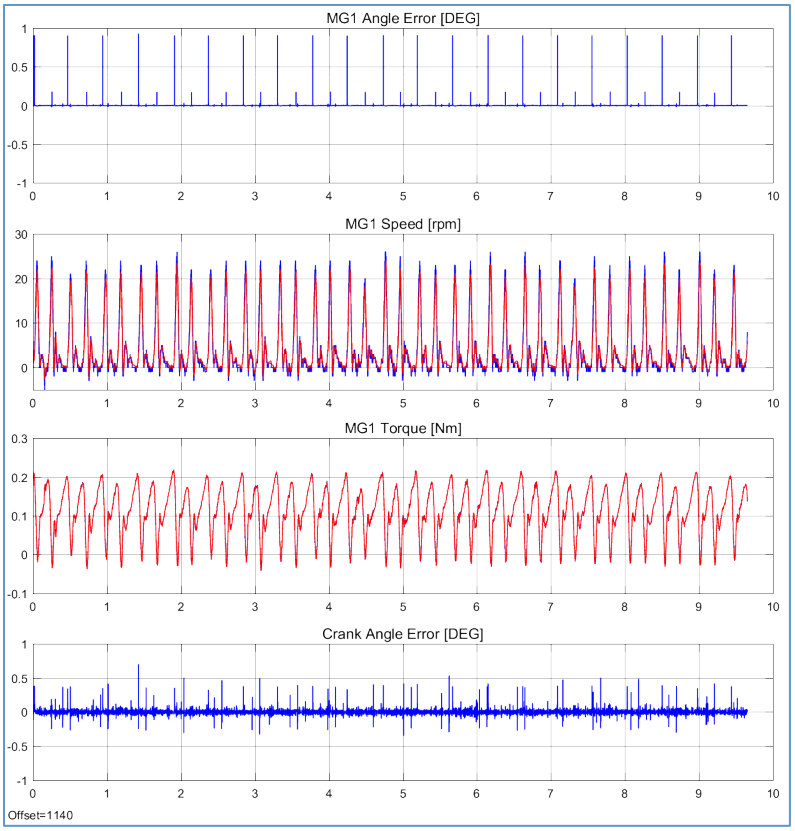

Figure 30: Kalman filter accuracy with MG1 rotating at 5rpm. (Subplots 1 and 4 show the absolute error between the measured angle and the estimated angle. Subplot 2 and 3 show the measurement in blue and the estimate in red)

6.1.2 Standstill operation

A fundamentally important though not very spectacular operating mode of the bicycle is simply standing still. When the bicycle is switched on, the pedals should not move and the crank should be kept perfectly still when the cyclist is not touching the bicycle or is not exerting any force on the pedals. As explained in section 3.3.3, when the desired MG1 velocity is calculated by the control algorithm to be below 200 rpm, it is set to zero. This is to avoid nervous behaviour of the crank when the bicycle is standing still.

6.1.2.1 Estimator performance

At the moment the system is switched on, the estimator converges within a second or so (Figure 31). The estimates of the measured states are very good in general. Since the bicycle is in standstill mode, the torque is estimated as a single value and the gradient is not estimated and presumed to be zero. Important to note is that the measurement of the MG1 torque is not very accurate at standstill. This is because the torque is not measured but estimated by the TI FAST estimator, that is only accurate when the motor is rotating. Although the bicycle state estimator perfectly estimates the MG1 torque in accordance with the measurements, it is not a good estimate of the actual torque on MG1. Also the estimated cyclist torque is therefore not accurate (Figure 32). In section 3.3 it was explained that the crank speed control at low speeds is performed based on the bicycle speed rather than on the cyclist torque. This proves to be a good approach. The bicycle speed is estimated with good accuracy also at standstill

(Figure 33). There is some freeplay in the planetary gear set. This is probably the cause of the fluctuations seen in Figure 33.
In terms of stability the estimator performs well. Even after large perturbations the filter converges quickly back to it's standstill state. No drift was observed in the state estimates even after long periods of time (1 hour and more).

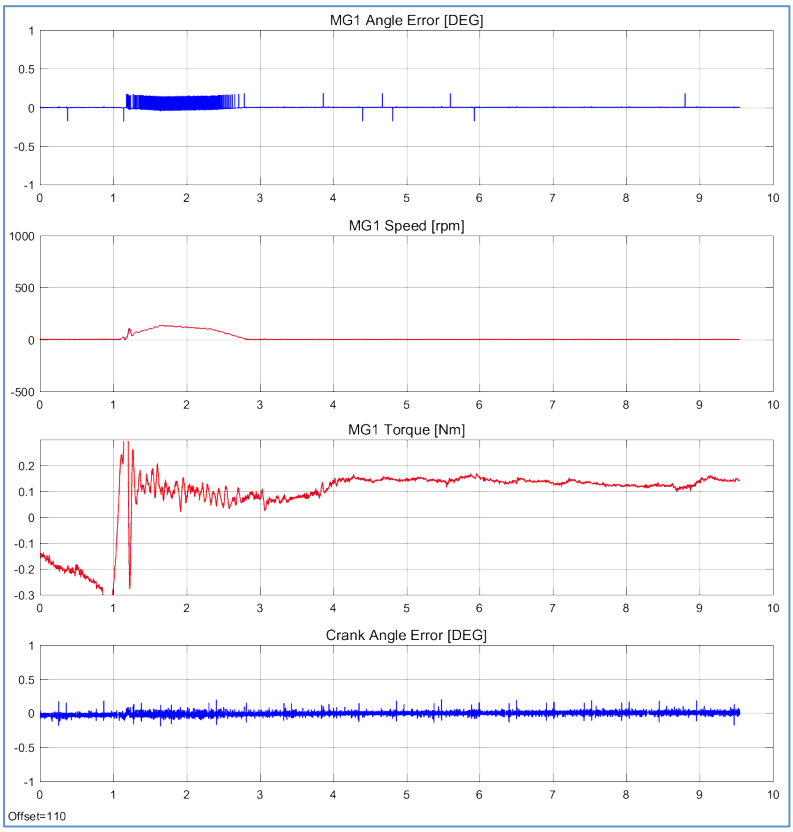

Figure 31: Filter accuracy when switching on the control at t=1s. (Subplots 1 and 4 show the absolute error between the measured angle and the estimated angle. Subplot 2 and 3 show the measurement in blue and the estimate in red)

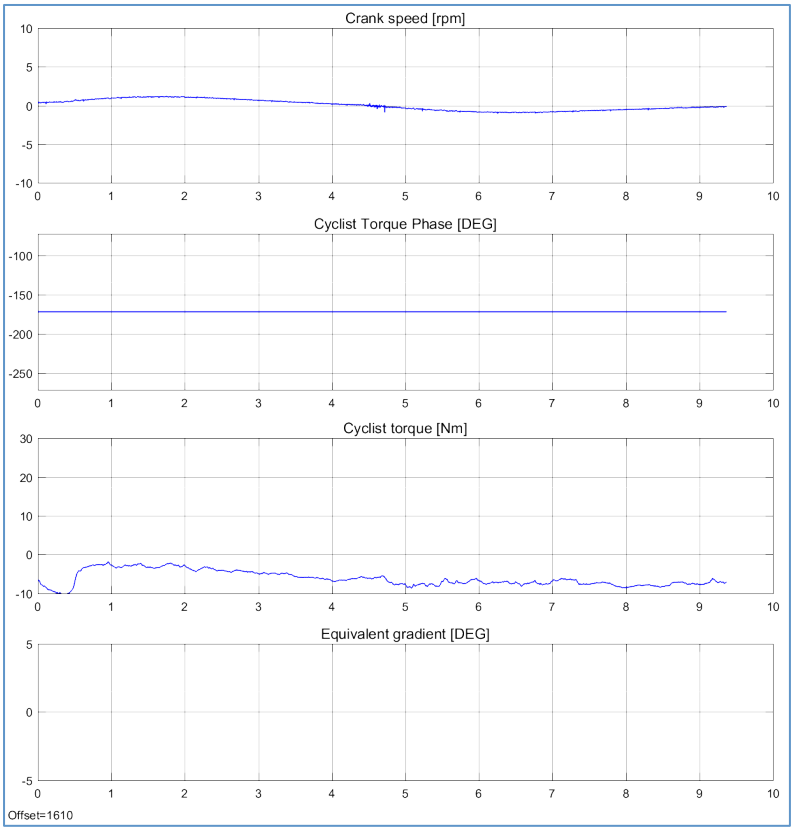

Figure 32: Estimated cyclist states at standstill.

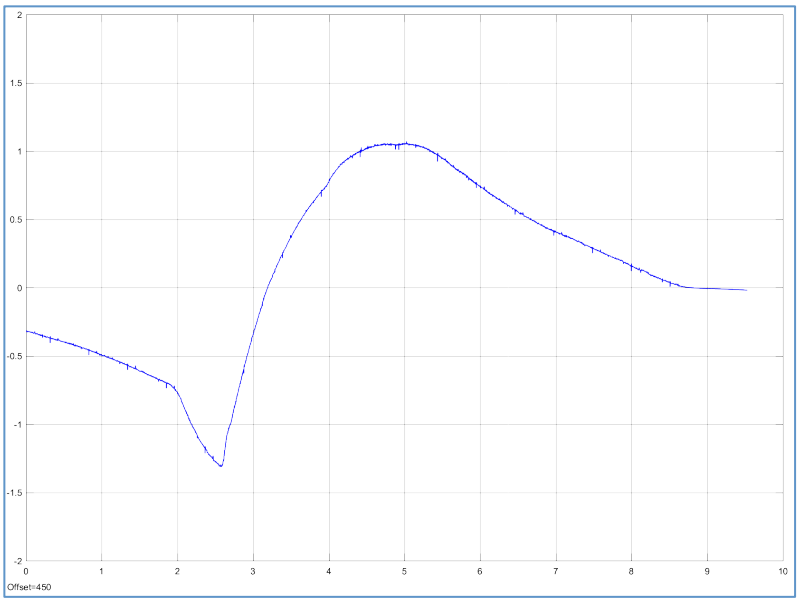

Figure 33: Bicycle speed estimate [km/h] at standstill.

6.1.2.2 *Control performance*

In the test, when switching the system on, the crankshaft moves slightly back and forth for about a second before the state estimator converges and the control keeps the pedals still. This is an issue that will have to be solved in the future as it is not acceptable in a production bicycle. As expected, the desired MG1 velocity

is perfectly zero because of the dead zone. The pedals stay perfectly still. This test is also performed without the dead zone implementation. Again as expected, the unaccuracy of the state estimator is translated by the control and the motor is making some random motion around it's standstill position. The crank is also moving slightly because of this.
To test the stability of the system at standstill, perturbations are applied to the pedals by kicking or jerking on them. Aside from the movement of the crank because of this, the control has no problem whatsoever to keep the crank still after the perturbation. No overshoot behaviour is observed.
Blocking the wheel of the bicycle by means of the rear wheel brake, the crank is directly coupled to the first motor. This way the ability of the motor to actively block the crankshaft can be tested, the motor stall torque in fact. The crankshaft is kept perfectly still up until the point where the cyclist steps through the motor. Using the SpinTAC controller, the system is very stiff at standstill. This is less so when making use of the simple PI controller also provided by TI. Stepping through the motor requires a large torque, but it is definitely possible at standstill, again showing the need for a good safety measure to avoid this.

6.1.3 Forward operation and cadence control

To test the control algorithm and the stability of the state estimator in varying working conditions, a forward operation test is performed on the roller stand. In this test, the cyclist simulates starting from standstill and building up to a comfortable speed. No very high torques are exerted on the crank shaft in this test, the goal is to gently go through the speed range of the bicycle.

6.1.3.1 Estimator performance

When speeding up, the estimator switches from standstill to sinusoidal mode (notice estimate of torque phase in Figure 35). This switching of modes works well and is not felt by the cyclist. The gradient is estimated and takes on expected values (Figure 35). The roller stand is not very representative for real road riding however so there is no way to say if the gradient is in accordance with the actual road gradient. The estimate of the gradient remains stable over the complete duration of the test. All other states are also estimated reliably. The estimate of the MG1 torque follows the measured value very well (Figure 34). This measurement of the MG1 torque is also reliable with the motor running at speed, so that the cyclist torque is also estimated correctly in most cases. When a very sudden increase of torque on the crankshaft resulted in motor stepthrough in early tests, the estimator did return clearly false values for the cyclist torque (4000Nm and higher). This problem was no longer observed when the motor was powered by the 36V battery and was probably also due to the low voltage from the power supply (24V). This lower voltage also reduces the maximum speed MG1 can achieve. The phase estimate of the cyclist torque is very stable and verifies the assumption that the cyclist torque is well described by a sinusoidal shape with near constant phase offset.

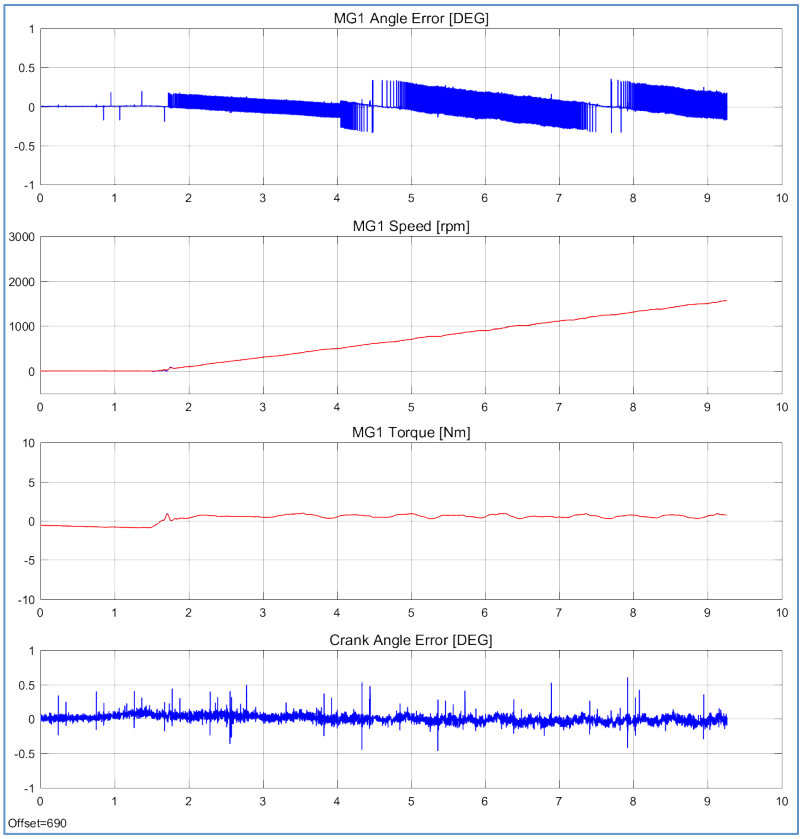

Figure 34: Estimator accuracy during acceleration test. (Subplots 1 and 4 show the absolute error between the measured angle and the estimated angle. Subplot 2 and 3 show the measurement in blue and the estimate in red)

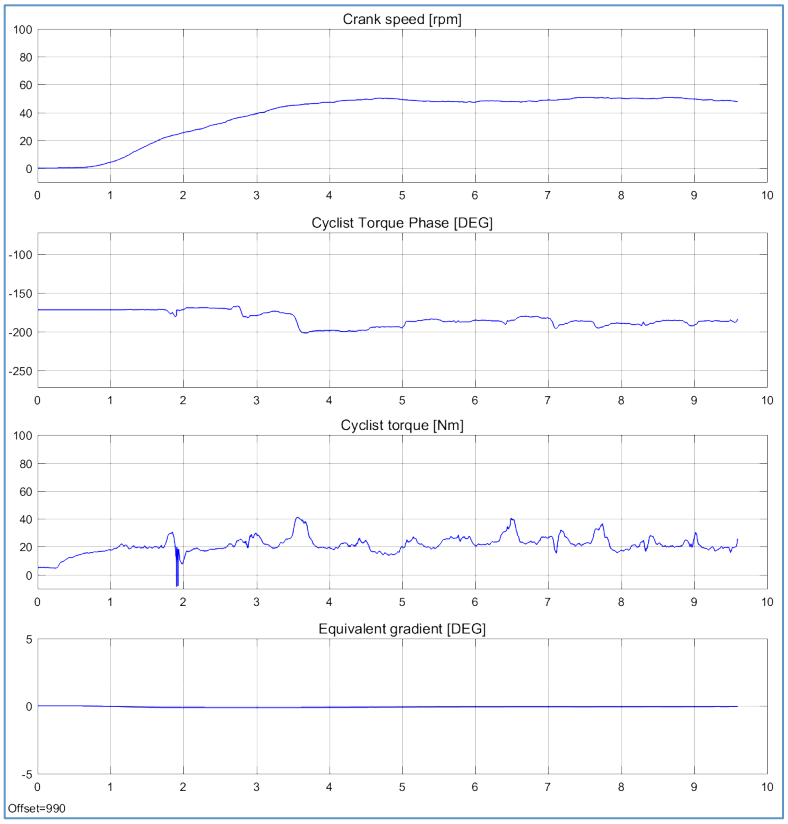

Figure 35: Cyclist state estimates during acceleration test.

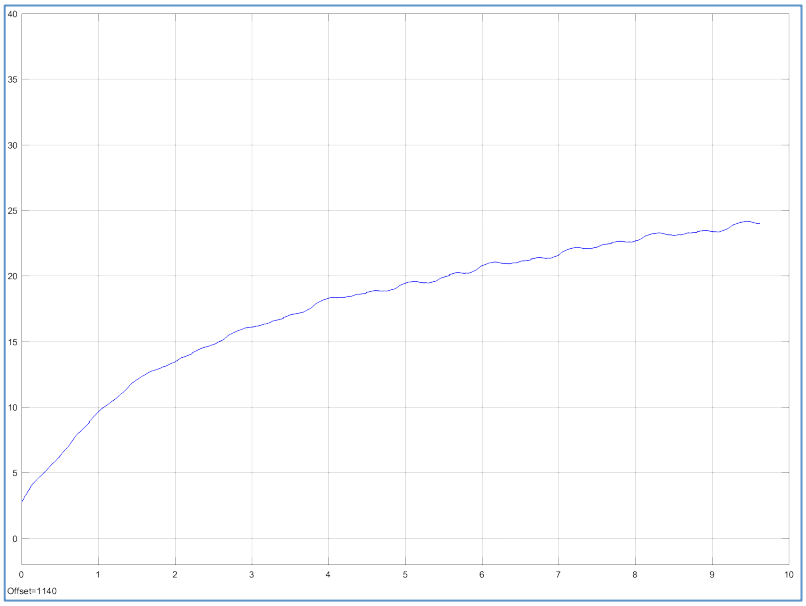

Figure 36: Rear wheel speed estimate [km/h] during acceleration.

6.1.3.2 Control performance

The control behaviour on rollers is not satisfactory. During one revolution of the crank, the resistance felt by the cyclist varies and the motor control is not able to keep the crank rotating at a constant speed within one revolution. Also at higher speeds, the optimal cadence is not maintained and when cranking hard, the cadence would go above the optimal cadence (set to 50, 60, 70, 80 rpm in the test). Even with the 36V battery this behaviour is observed.

Rather than a problem with the control itself, these observations are probably explained by the nature of the test setup. In normal operation on road, accelerating the bicycle requires to accelerate the entire inertia of the bicycle plus the cyclist. In the test setup on the other hand, there is hardly any inertia on the rear wheel since the roller has a very low inertia and it is not actively controlled to mimic this either. As a result, the rear wheel can be accelerated very quickly. The resistance and speed variations felt during one revolution of the crank originate probably in this lack of inertia, that makes that the wheel does not provide the reaction torque needed to control the crank by means of MG1. The unability of the control to maintain the optimal cadence is because the setup allows for very high bicycle speeds to be reached without much effort from the cyclist. At high speeds, the sensation of a high load should discourage the cyclist to crank faster than the optimal cadence. At these speeds where the load is becoming substantial (30km/h), MG1 is still able to maintain the optimal cadence. At very high speeds however, MG1 runs into the limits of it's speed range. The lack of speed sensation on the test stand also makes that the cyclist had no feedback regarding speed. In real road cycling, the cyclist would try to reach a comfortable speed and then opt to maintain it. On the rollers stand, there is no sensation of speed and so there is no reason to stop cranking very hard until the hardware limits of the system are reached.

6.1.4 Sudden stop and reverse operation

On the roller stand, a similar test is performed as was done in the virtual testing environment used by Simon Deruyter last year. When at speed, the cyclist suddenly stops cranking, applying a negative torque or no torque at all.

6.1.4.1 Estimator performance

The sudden change in torque is quickly detected by the estimator and the mode is immediately switched to constant torque estimation (notice estimate of torque phase in Figure 38). The phase offset is set to the default value. Although sometimes, despite the hysteresis, modes are switched from sine mode to constant torque mode and back multiple times, this does not influence the stability and accuracy of the estimator. The default phase offset is chosen close to the actual phase offset when cranking to make the discontinuity between the mode switches as small as possible (Figure 38).

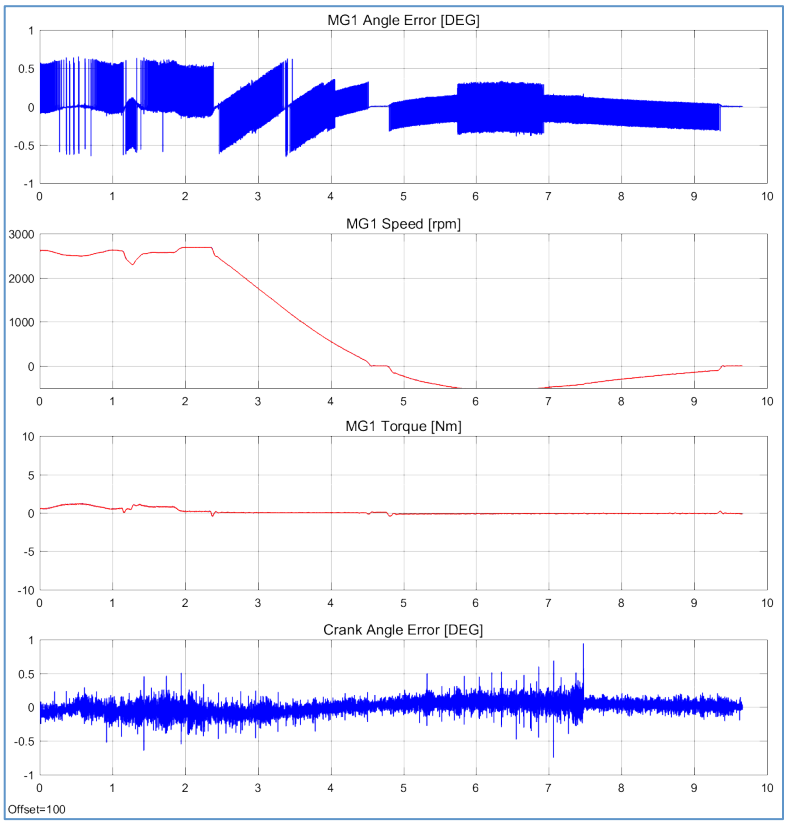

Figure 37: Estimator accuracy during sudden stop test. (Subplots 1 and 4 show the absolute error between the measured angle and the estimated angle. Subplot 2 and 3 show the measurement in blue and the estimate in red)

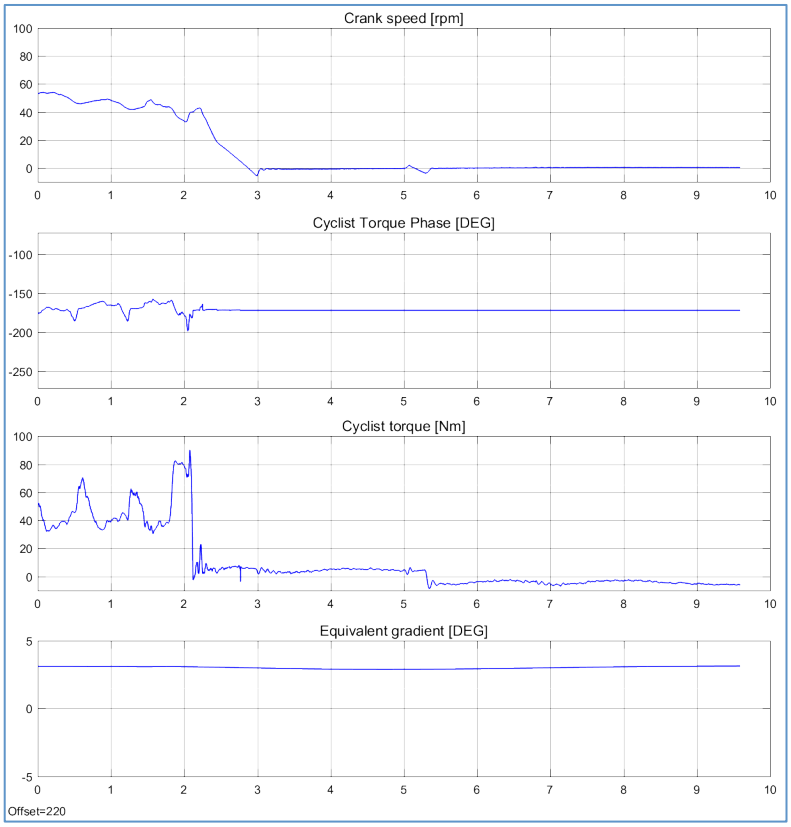

Figure 38: Cyclist state estimates during sudden stop test.

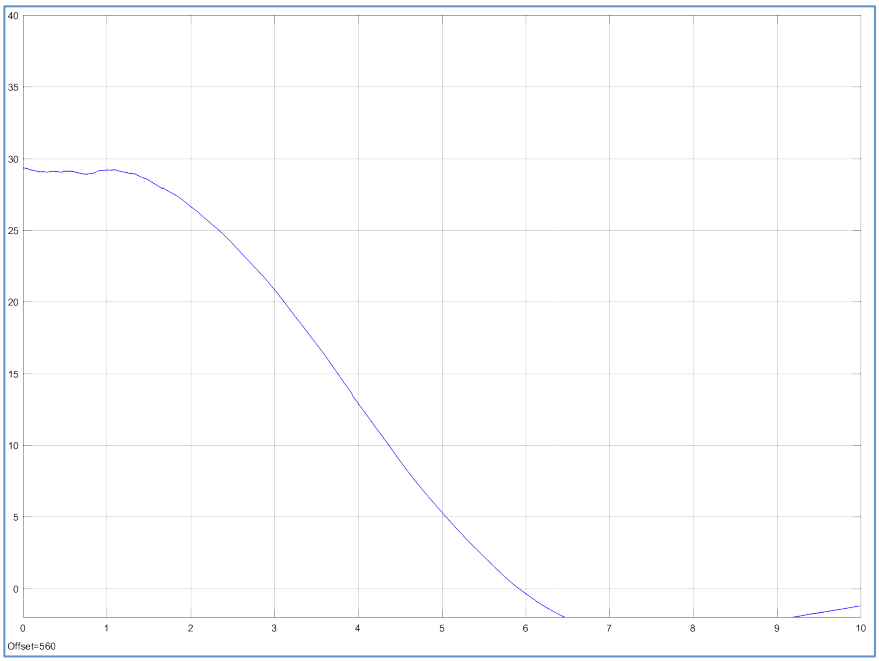

Figure 39: Rear wheel speed estimate [km/h] during sudden stop test.

6.1.4.2 Control performance

Because of the adequate estimator, the sudden torque change (at little past t=2s) is quickly detected and the control can act on it immediately, stopping the crank rotation (Figure 38). It takes almost a second to stop the crank, which is longer

than in the virtual simulation. Some oscillations are observed in the MG1 speed setpoint sent out by the control algorithm. With the deadzone on this speed setpoint set to 200rpm, these oscillations are eliminated in the eventual motor control and the crank is kept perfectly still when in standstill mode. The deadzone is noticeable in the crank speed when the motor needs to pass the zero speed point (Figure 38). The lack of inertia on the test stand is proving to be a problem here as well (notice rapid decrease in wheel speed, even going negative, in Figure 39), making it hard to judge whether it is the control stopping the crank or the cyclist himself, since the rotating rear wheel has so little inertia that it can easily be decelerated by the cyclist in a very short time. In real road riding, the bicycle speed should hardly change in the sudden stop test because of the bicycle inertia. The low inertia in the test setup also makes the standstill control of the crank hard when applying a reverse torque. This torque leads to a large deceleration of the rear wheel, making the time to go from 30 km/h to standstill only a few seconds. It is hard to judge the continuity of the control behaviour in this way. An additional problem is in the fact that there is no speed sensation on the test stand. Even though the crank behaviour in itself is good, there is no way to judge whether the bicycle feels intuitive when on the road (i.e. if the deceleration of the bicycle matches the expectations of the cyclist and if the crank behaves according to this expected deceleration).

6.1.5 Conclusions on the roller stand tests

For validating the Kalman filter performance and stability the roller stand tests are very useful. Since the bicycle can be coupled to the laptop during operation on the stand, all estimated states can be monitored in detail. The roller stand tests are less useful for testing the control algoritm. Although the main characteristics and behaviour can be tested, the test setup is not representative enough to actually validate the control behaviour. The lack of inertia and bicycle speed sensation in particular pose a problem. For the most part however, the control behaves as expected also on the roller test stand.

6.2 Tests on road

Due to technical problems with the SPI communication protocol required to control two slave boards, one for MG1 and one for MG2, road tests have not been performed thus far.

7 Cyclist intention estimation

7.1 The cyclist behaviour problem

An electric bicycle like the one in this thesis is a human-in-the-loop system. This means that the bicycle control does not only have to take into account the mechanical and electrical system, but also the human riding the bike. The system is schematically represented in Figure 40.

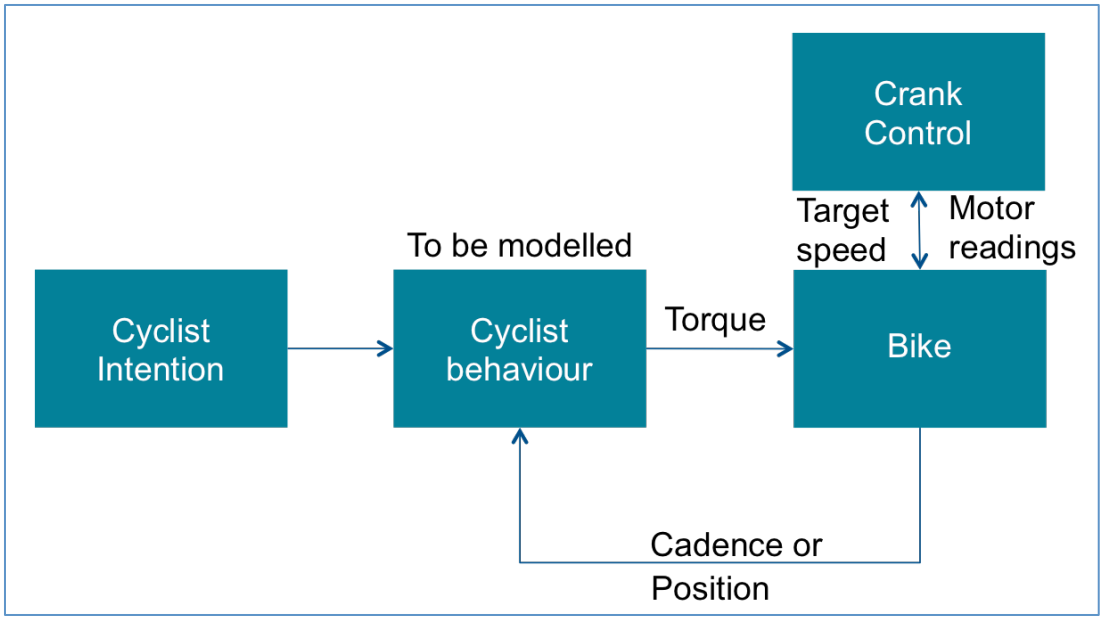

Figure 40: Block diagram of the human-bicycle system.

When riding the bike, the cyclist has a certain intention in mind, that will be manifested through his behaviour. This behaviour does not only depend on the cyclist's intention, but also on other factors. There are too many other factors that play a role to take into account, but one that is very relevant in the context of the electric bicycle is that of the pedal cadence or crank position. As a result of his intention, the cyclist will, depending on the cadence or crank position and it's rate of change, exert a certain torque on the crankshaft by pushing on the pedals. It is this torque that is being measured by the bicycle, directly by a sensor or indirectly like in our case with a Kalman state estimator. Based on the state of the bicycle and the knowledge of the cyclist torque, the crank control algorithm will send out a target speed for MG1, controlling the crank position or cadence. This cadence will be felt by the cyclist and he will act on that feeling through his behaviour, again exerting a torque on the crankshaft and closing the control loop. The goal of this control loop is to control the bike in such a way that it behaves according to the cyclist's intention, not necesseraly according to his behaviour. By doing what the cyclist intends, the bicycle can provide for a very intuitive cycling experience. The challenge is to know from the cyclist's behaviour (cyclist torque), what his intention is. This requires the cyclist behaviour to be modelled so that the input of the behaviour block (intention) can be derived from the output (torque) through knowledge of the effects of crank position and cadence (control action). For example: when the cyclist wants to accelerate (intention), we expect him to apply force on the pedals (behaviour), so that the torque will be higher. Depending on the position of the crank (control action), this torque will be very high (peek of the sine curve) or very low (dead center position), even though the intention hasn't changed. Also, when a change in torque accelerates the pedals very quickly, the torque exerted on the crankshaft by the cyclist will drop suddenly, even though the cyclist still wants to accelerate.
Although this appears to be a rather simple concept, it is being complicated by a lack of determinism. What this means is that there is no clear relation between cause and effect in the cyclist behaviour. A cyclist behavioral model will always be a statistical average of human behaviours. In most cases where human

behavioral models are being used, this is not very problematic because they are used for safety features. Electronic stability program (ESP) in cars is a simple example. Without any knowledge on how humans drive a car, ESP could just as well have been tuned to systematically oversteer a car (steering more than the driver asks for). A driver would react on this with a countersteering reflex that will create an unstable situation of steering the car from left to right in an attempt from the driver to stabilize it. Therefore the ESP system will deliberately understeer the car slightly. The driver's reflex will be to steer a little more, but not in a sudden way since he or she is already steering in that direction. This combined system will stabilize the car much faster then if the ESP would oversteer the car. Because ESP is a safety feature, it is not a problem that the car understeers somewhat, as long as the behaviour of the car is safe. If ESP would work with the driver in every turn, then it should have much better knowledge of the drivers steering behaviour in order not to create the sensation that the car responds inadequately to the driver's steering inputs. This is exactly the difficulty in the bicycle project. The control is working all the time, not just in an emergency situation. Therefore it cannot feel like it is taking over the bicycle or changing it's behaviour. It should feel very intuitive to the cyclist.
An interesting question on this intuitiveness issue is whether it is the bicycle that should be adapted to the cyclist, or whether the control should be made so that the cyclist can quickly adapt to the bicycle. Both from a pragmatical point of view and as a hypothesis on human behaviour, the second statement is taken as the approach in this project. Human behaviour is very complex and the causality between human intention and human behaviour is probably not reliable enough to base the control of the bicycle on. Knowledge on human behaviour should rather be used as a guideline for the basic control principles. Humans adapt very quickly if the system they are working with is straight-forward and easy to understand. From this hypothesis, it is of the utmost importance that the control is, above all, predictable, no matter how simple or complex. This way, cyclists will immediately feel how the bicycle reacts to certain conditions and inputs and adapt to the bicycle.

7.2 Literature on cyclist behaviour

Literature on human-in-the-loop control of bicycles is practically non-existant, probably because the current state-of-art of E-bikes does not require much specific knowledge in this field. Current E-bikes still work with all the usual bicycle controls – brakes, derailer gears and freewheel – and therefore do not require knowledge of the cyclist intention or behaviour, since there is hardly any direct interaction between the cyclist and the bicycle.
As mentioned in the previous section, there are two important aspects in the bicycle control. The first is the factor of cadence. To effectively control the CVT transmission, knowledge about cadence in steady state cycling is important. More specifically: what cadence is optimal in terms of comfort and/or performance and what does this optimal cadence depend on? There is extensive literature on this topic in the context of professional competitive cycling. The useful conclusions from this research will be discussed in the next section.
The second factor is the dynamical interaction between the cyclist and the pedals. The project aims for the pedals to be the only or at least by far the most important interface between the cyclist and the bicycle. Therefore it is important

to know how the cyclist reacts to changes in torque and speed of the crankshaft. Call this crankshaft feedback or pedal feedback. Ideally, the control should not only have a model of the bicycle, but also of the cyclist, so that it can predict the cyclist's reaction to a certain control action. There is no literature on this specific problem. However, as touched upon with the ESP example earlier, very similar problems occur in the automotive domain. Advanced driver assistance systems (ADAS) encounter exactly the same kind of problems we are faced with in the bicycle crank control. In section 7.2.2, the Cambridge driver steering behaviour model is discussed.

7.2.1 Freely chosen cadence and optimal cadence

In the context of professional cyclists, the optimal cadence for high performance is a topic of interest. Two terms pop up frequently: Freely Chosen Cadence (FCC) and Energy Optimal Cadence (EOC). The FCC is the cadence a cyclist adopts when he can choose for himself. The EOC is the cadence at which the oxygen uptake is optimal and therefore the cadence at which the energetic efficiency is highest. It was observed by Vercruyssen et al[xiv] that the EOC is generally much lower than the FCC. This falsified the theory that the FCC is aimed at maximizing efficiency. More surprising from Vercruyssen's research is that the EOC does not give the best performance. Better performance is generally achieved at the FCC then at the EOC, even though the energetic efficiency is significantly lower.

Hanssen et al[xv] make a very convincing case on the theory that, like walking, cycling cadence has a fundamental neurological basis. Just like walking cadence, the central nerve system controls the cycling cadence. Other then walking however, where the freely chosen cadence is the energy optimal cadence, this is not the case for cycling. A possible explanation for this is that energy optimal cycling hasn't got the evolutionary advantage that energy optimal walking has, and therefore humans have evolved towards energy optimal walking cadences, but not energy optimal cycling cadences. The work of Hanssen contains some interesting conclusions for the bicycle control. First of all, the freely chosen cadence should be the target cadence for steady-state cycling, since it gives the best performance in submaximal conditions. Second of all, a cadence that is lower than the FCC is not too problematic, performance will drop slightly but the subjective feeling of comfort will not drop too much. A cadence that is set too high on the other hand is very problematic. Performance will drop significantly and the comfort is considered to be very low. In controlling the bicycle the cadence will thus have to be close to the FCC, but better too low than too high. Since the FCC has a neurological basis, it changes very slowly for a single person. It falls back about 3rpm per decade[xvi]. Between people it can vary significantly. Practically, it might be worthwhile to measure the FCC when selling a bicycle to someone. This will then make the control optimally adapted to the cyclist and since the FCC changes very slowly, it will not need updating probably during the bicycle's lifetime.

All the previous is about steady-state cycling cadence. Little can be found in literature about transient behaviour. For example, how does a cyclist behave in a turn, when accelerating, when making an emergency stop? Good control of the bike in transient situations is what will most probably make the bike intuitive to use and fun to drive. Using the prototype, it is this transient behaviour that will have to be investigated in order to make an adequate control algorithm. As

mentioned in the previous section, it will be above all important that the control behaves predictably. The cyclist will then quickly adapt his behaviour to the bicycle.

7.2.2 The Cambridge model for driver steering behaviour

The Driver-Vehicle Dynamics (DVD)[xvii] group at Cambridge university led by David Cole, proposes an integral approach for controlling and designing vehicles. Rather than looking only at the vehicle itself and manually tuning it's parameters (shock absorber damping e.g.) in a later stage based on experiments, Cole et al. propose to model the entire driver-vehicle combination in a single model. Figure 41 shows a diagram of the complete driver-vehicle system. The purple coloured block is the actual vehicle, with it's nonlinear dynamics. The green blocks are part of the driver. In the DVD approach, the driver control behaviour is postulated to be made up out of two main parts. In the brain, the driver runs a model predictive controller that makes use of an internal model. This internal model has been formed by the driver through learning. The model predictive controller in the brain generates neural signals that travel to the neuromuscular system. This system generates a torque on the steering wheel and turns it, while at the same time sending out feedback signals that are being picked up by the brain. The result is a closed-loop system in which the brain controls through the neuromuscular system (NMS) the vehicle, taking into account the vehicle motion and the feedback signals from the NMS. The input of the system is the desired path to travel, more in general: the intention of the driver. The output is the vehicle motion. Notice the analogy with the diagram in section 7.1. The behaviour block in section 7.1 is modelled here as three separate blocks.

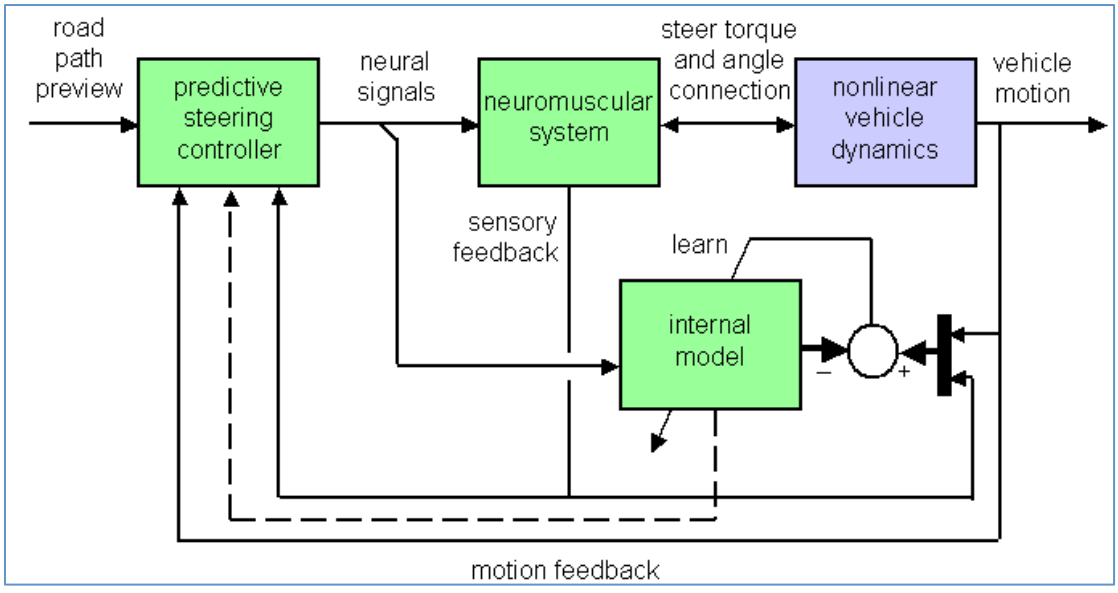

Figure 41: Diagram of the Cambridge approach to driver steering behaviour[xviii].

More in detail, the driver-vehicle model looks like the diagram in Figure 42[xix]. The path following control is the model predictive controller in the brain. There is some delay in the decision and transmission of the desired control action. The desired control action is sent directly to the muscles in the arms, that in turn rotate the steering wheel. This is feedforward control. At the same time, the

expected handwheel angle – calculated by a forward model – is compared to the actual handwheel angle as observed by the driver's senses. Reflexes will correct any discordance between these two. This is feedback control.

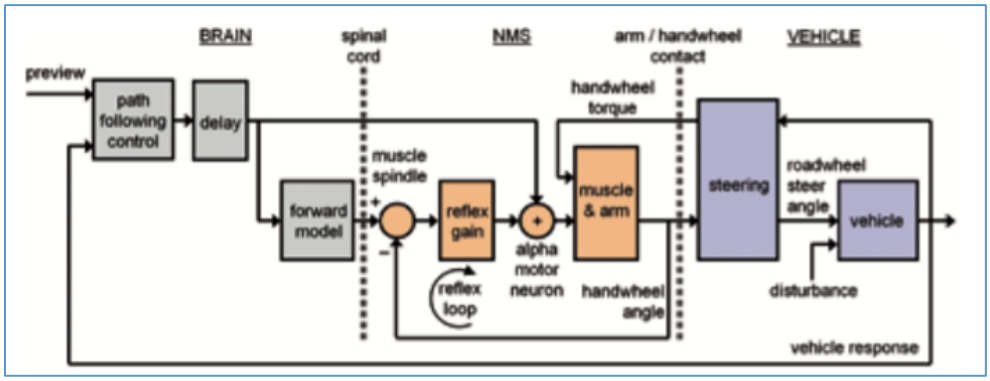

Figure 42: Diagram of the driver-vehicle model as proposed by David Cole[xx]

Determining the parameters of the vehicle model and the steering model is relatively straight-forward. On the other hand, the parameters of the NMS and brain model are more complicated to determine. Cole et al. developed a set of tests for this[xxi]. A first test is used to determine the dynamical properties of the driver's hands and arms. The test subject holds a steering wheel that is being moved in a random way by a servo motor, applying a torque on the wheel. The angular movement of the wheel is measured. The resulting frequency response function can be used to determine stiffness, damping, inertia and reflex gain. Path following control characteristics cannot be determined in this way. Driving simulator tests are used for this.
Although not directly applicable to the bicycle control, the approach by Cambridge's DVD group is a good guideline for future research on cyclist behaviour modeling. There is a clear analogy between both problems.

8 Further work

The obvious next step in the process is to test the bicycle on the road, using either a simple control for MG2 as explained in the 3rd chapter, or using only MG1. This is simply debugging the current implementation of the bicycle.
Some minor adaptations to the Kalman filter are useful to make it more intuitive. The sine shape definition of the cyclist torque is not very logical and can better be re-written as follows:

$$T_{cyclist} = T_{ampl} \cdot sin\left(2.\left(\theta_{crank} + \phi\right)\right) + T_{offset}$$

Also the state vector should be re-written so that the states reflect the measured values. Instead of the sun and the planet carrier position and speed, the crank position and speed and MG1 position and speed should be used.
More interestingly, a lot more potential is in the further modeling of the bicycle and the cyclist. The current implementation uses a high fidelity model for the driveline, but is has no knowledge about the complete bicycle, let alone the cyclist. Modeling the complete bicycle will open up a completely new set of

possibilities when it comes to controlling the E-bike. A logical first step would be to model not only the driveline dynamics but also the movement of the bicycle and the rotation of the front wheel, with the steering angle taken into account. The coupling between MG1 and MG2 can than be more reliably described. Also, monitoring and controlling the stability of the bicycle becomes possible through the use of MG2, like some sort of ESP for bicycles. Not only the bicycle itself but also the cyclist plays a major role in the handling and stability of the bicycle. Being able to estimate the cyclist's weight and position (center of gravity) will not only give extra information on the cyclist's intention, but will also make stabilizing and steering the bicycle more adequate. The cyclist is after all more than two thirds of the total mass of the combination.
As mentioned in chapter 7, knowledge about the cyclist behaviour in the form of a reliable model makes it possible to optimise the bicycle control for the combined cyclist-bicycle system. An approach like the one proposed by Cole et al. is a good starting point for building such a cyclist model.
Also in the interpretation of the cyclist torque there is still work to be done. Rather than looking only at the torque envelope, the complete "torque signature" of the cyclist should be used. The position of the peak in the sine shape, the sharpness of the curve, the relative importance of the offset torque compared to the torque amplitude,… These characteristics contain potential information about the cyclist's intention and profile. Research on the relation between torque signature and cadence will be very useful to estimate the optimal cadence of a cyclist in real time rather than having to ask it as an explicit input from the cyclist. The torque signature at different cadences might say something about the optimality of that cadence.

9 Conclusion

This thesis consists of three main parts. In the first part, using a virtual testing environment in Simulink, last year's work done by Simon Deruyter is resumed. The bicycle model is adapted to the most recent concept, that is to be used in the first hardware prototype. Also the Kalman filter is updated and refined to handle varying conditions and more transient behaviour, always yielding a reliable estimate of the bicycle driveline states and the input torque from the cyclist. Modes are introduced to adapt the estimator to different working system states. Calculation effort needed for the filter is reduced through the use of approximative integration schemes. A control algorithm that is both simple and performant in transient cases is developed to control the prototype. It takes into account the reliability of the Kalman filter in different system states and avoids ambiguous behaviour in order to not only efficiently debug the prototype, but also make it easy for the cyclist to understand what is happening. Care is also taken to make the algorithm as efficient as possible to execute, requiring less calculation power from the processor running the algorithm.
In a second stage, the bicycle control program needs to run embedded on the hardware prototype in real time. Three processors are used for this. One Raspberry Pi mini computer works as a master, running the main control program including the Kalman filter and the control algorithm. This master processor communicates the desired MG1 speed and MG2 torque over SPI to the two Texas Instruments slave processors. These slave processors in turn run a

motor control program, speed-controlling MG1 and torque-controlling MG2. At the same time, measurements from MG1 and MG2 are collected by the slave processors and sent back to the master, where these measurements are inputs to the Kalman filter.
Once the prototype is operational, tests are performed on a roller stand to validate the performance and stability of the Kalman filter and the control algorithm. Kalman filter state estimates are reliable in most working conditions and convergence is fast. Only at standstill, when MG1 is not rotating, the TI FAST estimator is not very accurate and therefore also the estimate of the cyclist torque is not very accurate. A 10Nm offset is not exceptional. The control takes this inaccuracy at standstill into account. Little can be said about the performance of the control. The test setup with the bicycle on rollers is not representative enough to give relevant results. A lack of bicycle inertia makes the rear wheel change speed too rapidly. Lacking any form of speed feedback, it is impossible to say whether the control is intuitive. Real road testing will be much more useful in this regard.

10 References

[i] Wilberts, M. (2013); *Ontwikkeling van een optimale controlestrategie voor de CVT van een elektrische fiets;* KULeuven.
[ii] Pipeleers G. (2015); *Control Theory, part C, Slides*; VTK. See C12, p.34 for Kalman estimator procedure.
[iii] https://en.wikipedia.org/wiki/Kalman_filter
[iv] https://en.wikipedia.org/wiki/Discretization
[v] https://en.wikipedia.org/wiki/Euler_method
[vi] https://en.wikipedia.org/wiki/Backward_Euler_method
[vii] Haugen F. (2005); *Discrete-time signals and systems*; Techteach. See section 8.3.
[viii] Snare, M.C. (2002); *Dynamics model for predicting maximum and typical acceleration rates of passenger vehicles;* Virginia Polytechnic Institute and State University.
[ix] Luo, D. (2014); *Modelling of cyclists acceleration behavior using naturalistic data;* KTH Royal Institute of Technology.
[x] http://www.ti.com/tool/drv8301-69m-kit
[xi] https://nl.mathworks.com/products/embedded-coder.html
[xii] https://www.raspberrypi.org/magpi/raspberry-pi-3-specs-benchmarks/
[xiii] https://en.wikipedia.org/wiki/Double-precision_floating-point_format
[xiv] Vercruyssen F. et al (2008); *Which factor determine the freely chosen cadence during sub-maximal cycling?*; Journal of Science and Medicine in Sport; Elsevier.
[xv] Hansen E. et al (2009); *Factors affecting cadence choice during submaximal cycling and cadence influence on performance*; International Journal of Sports Physiology and Performance; Human Kinetics.
[xvi] Hansen E. et al (2009); *Factors affecting cadence choice during submaximal cycling and cadence influence on performance*; International Journal of Sports Physiology and Performance; Human Kinetics.
[xvii] http://www2.eng.cam.ac.uk/~djc13/vehicledynamics/intro.html
[xviii] http://www2.eng.cam.ac.uk/~djc13/vehicledynamics/research.html, David Cole.
[xix] Cole, D. (2008); *Steering Feedback: Mathematical simulation of effects on driver and vehicle;* Autotechnology.
[xx] Cole, D. (2008); *Steering Feedback: Mathematical simulation of effects on driver and vehicle;* Autotechnology.
[xxi] Cole, D. (2008); *Steering Feedback: Mathematical simulation of effects on driver and vehicle;* Autotechnology.