Cruise Control Solar Car

Ingibjörg Þorsteinsdottir (s203862), Kristoffer Erbo Kjær (s203829),
Rasmus Kurdahl-Martinsen (s194337), Xandra Huryn (s203864)
Department of Electrical and Photonics Engineering
Technical University of Denmark
Kongens Lyngby, Denmark
31015 - Introductory project - Group 22

***Abstract*—This paper presents the cruise control project for the DTU Roadrunners Solar Team (ROAST). The purpose of the cruise control is to conserve energy by maintaining a constant speed and taking some of the operating workload of the driver.**

The speed controller which is the core of the cruise control, ended up being a PI-controller with anti-windup. It is implemented on a STM32 microcontroller, which results digital implementation of the controller. The controller receives a RPM which it will try to match by controlling the motor via a control voltage. The cruise control module receives feedback through the car's CAN bus from the motor, or in the case of this project, a simulation.

A 5 km/h step results in an overshoot of 0.58%, a rise time of 2.65s, and a settling time of 4.40s.

The resulting speed controller ended up operating to a satisfactory standard. However the control parameters will have to be redone when it gets implemented into the car, as many values are subject to change.

***Index Terms*—Cruise control, CAN, PI-controller, digital control implementation**

I. Introduction

The objective of this project is to create a cruise control function for use by the DTU Roadrunners Solar Team(ROAST) [1]. ROAST works towards creating an electrical car with solar panels to participate in the cruiser class of the Bridgestone World Solar Challenge [2] located in Australia. It is a race spanning over 3000 km across the continent. The solar car has limited solar panels and recharging opportunities, which emphasizes the need to create an energy-efficient solar car to win the race.

The most efficient use of energy is when the car drives with a constant speed. This is difficult to achieve by the driver due to the vast distances which will be covered. In an attempt to address this issue, and increase crew comfort, a cruise control function will be implemented.

The cruise control function will regulate the speed of the car without constant user input (i.e pressing down the pedal). This will be done by designing a linear speed controller, which will act on data from the motor driver through the car's on-board CAN (Controller Area Network) bus. The energy efficiency is an overall theme for the entire solar car project. This will also be a focus for the operation of the cruise control function. However the main focus will be on developing an operational cruise control function which interacts with the CAN.

A. Problem definition

The purpose of the project is to implement a cruise controller function for the solar car so that it can maintain a constant speed without correcting commands from the driver. The implementation should include a simple user interface connected to a microcontroller which can communicate with the motor driver through the already existing CAN bus. Some form of feedback should be included ensuring the desired speed is achieved. In every part of the system there will be a focus on how to improve safety.

- Which microcontroller(s) should be used in the project?
- How can a DAC be implemented to make the signal from the microcontroller compatible with the motor driver?
- How can an intuitive and easy-to-use user interface be designed and implemented?
- How can a stable feedback loop be designed and implemented to regulate speed?
- How can the safety of the drive-by-wire system be improved?

II. Overview

The developed cruise control system consists of two STM32 microcontrollers [3] as well as a CAN-driver for each. The inputs to the system consists of a brake signal, buttons for turning the cruise control on and off, and buttons for adjusting the speed with $\pm$5km/h. The first STM32(UI) handles these inputs. The second STM32(control) handles the speed regulation as well as the output to the motor driver through a DAC [4], as the motor driver requires an analogue signal.

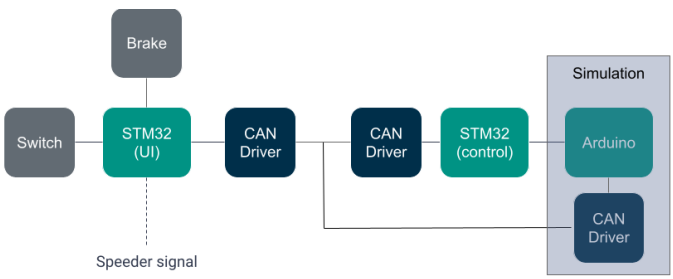

Fig. 1. Block diagram of system

A. Simulation vs. physical test

Since the solar car has not been assembled, it is not possible to test the cruise control function physically. It would not have been useful to test the motor alone, as there would be no external resistance emulating air resistance, hills etc. Thus we have chosen to use a simulation implemented on an Arduino for tuning and testing.

B. Choice of MCU (Microcontroller Unit)

The STM32 was chosen to match what the other solar car projects use, which will make it easier to integrate the projects. The STM is configured on a NUCLEO board [5] [6], for easier development and integration with the arduino framework. The STM32 do also have on-board debugging available. Lastly it has an integrated CAN controller, which can be implemented in an other project to avoid using the SPI (Serial Peripheral Interface) connection between the MCU and CAN controller.

The code is written in C++ with the Arduino library and uploaded to the microcontrollers with the help of PlatformIO. This setup is very useful because the code can be uploaded to both the STM32s and the Arduino. This means that all our CAN related code do not have to be rewritten when switching microcontrollers.

III. Simulation

We opted for modelling a simulation of the solar car based on the available specs for the car's engine and gearbox. The simulation was implemented on an Arduino Uno 3 [7] which will have elevation angle and the motor control voltage as inputs and sends the motor RPM via CAN. The motor driver would also send the mechanical RPM via the CAN bus. As the solar car is not fully designed, we have used data for a generic Toyota Prius, as we asses this to be in the ballpark of the size and aerodynamic shape of the future solar car. The simulation is based on the following force equation

$$F_{\mathrm{i}} = F_{\mathrm{s}} + F_{\mathrm{r}} + F_{\mathrm{a}} - F_{\mathrm{t}} \tag{1}$$

from [8]. F_t is the traction force between the road and the wheels.

$$F_{\mathrm{t}} = \frac{T_{\mathrm{m}}(V_{control})\, i_{\mathrm{x}}\, i_0\, \eta_{\mathrm{d}}}{r_{\mathrm{wd}}} \tag{2}$$

All the constant values except the torque can be seen in table I. The torque $T_m(V_{control}) = T_{\mathrm{m}}\, \frac{V_{control}}{5\mathrm{V}}$ is produced by the motor. It is scaled by the control voltage $V_{control}$, which ranges from $0\mathrm{V} - 5\mathrm{V}$ [9]. We assume that this scaling is linear. However the motor cannot deliver maximum torque at higher RPM where it will exceed the rated power, $14.5\mathrm{kW}$ [9], which is defined as $P_{\mathrm{max}} = T_{\mathrm{m}}(5\mathrm{V})\, RPM\, \frac{60\mathrm{s}}{2\pi}$. Thus the torque will be expressed by two different equations. The first case, $T_{\mathrm{m}}(V_{control}) = \frac{V_{control}}{5\mathrm{V}}\, 130\mathrm{Nm}$, is when the maximum torque results in a power lower than P_{max} due to low RPM. The other case, $T_{\mathrm{m}}(V_{control}) = \frac{P_{\mathrm{max}}}{\mathrm{RPM}}\, \frac{60\mathrm{s}}{2\pi}\, \frac{V_{control}}{5\mathrm{V}}$, is when the torque is restricted by the rated power.

F_s is the road slope force which accounts for the elevation of the road. The angle α of the road is measured in radians.

$$F_s = m\, g\, \sin\alpha \tag{3}$$

F_r is the road friction force.

$$F_r = (m\, g\, \cos\alpha)\, c_{\mathrm{r}} \tag{4}$$

F_a is the aerodynamic drag force. Note that this force is influenced by the speed of the car.

$$F_a = \frac{\rho\, c_{\mathrm{d}}\, A\, v^2}{2} \tag{5}$$

F_i is the inertial force of the car, expressed as $F_i = m\, a$. The acceleration in eq. (1) is isolated.

$$a = -\frac{F_s + F_a + F_r - F_t}{m} \tag{6}$$

Now we have an acceleration which is dependent on the the speed squared. We can by using $v = v_0 + a\, t$, where t is a very small interval, calculate the current speed. Using a step-wise calculation with the speed of the last step, we can calculate the current drag force. The drag force would continuously change as the speed continuously changes, however we are assuming that the change between each step is so small that it does not have a practical impact, as the time interval T is 5ms.

$$v = v_{-1} + a(v_{-1})\, T \tag{7}$$

The motor driver would return RPM. Thus the simulation calculates the car speed to RPM and returns it with CAN.

$$\mathrm{RPM} = \frac{v}{2\pi\, r_{\mathrm{wd}}}\, 60\mathrm{s}\, i_0\, i_{\mathrm{x}} \tag{8}$$

TABLE I
Sim-model values

Constants	Symbol	Units	Values
Transmission gear ratio	i_{x}	-	1
Final drive ratio	i_0	-	1.67
Drive line efficiency (guesstimated)	η_{d}	-	0.98
Dynamic wheel radius [8]	r_{wd}	m	0.343
Car mass	m	kg	800
Gravitational acceleration	g	$\frac{\mathrm{m}}{\mathrm{s}^2}$	9.816
Air density at sea level and 288.5K	ρ	$\frac{\mathrm{kg}}{\mathrm{m}^3}$	1.3
Air drag coefficient (Prius) [11]	c_d	-	0.25
Vehicle frontal area (Prius) [12]	A	m^2	2.4
Road friction coefficient (Prius) [10]	c_{r}	-	0.015
Maximum motor torque [9]	T_{m}	Nm	130

IV. PI controller

In order to design a controller, a transfer function for the system is needed. This was obtained by logging a step for the motor from $1.5\mathrm{V}$ to $2.5\mathrm{V}$ in the control voltage. The data was processed in MATLAB to get the following transfer function.

$$Gs = \frac{8.683}{s + 0.02067} \tag{9}$$

The system should be rather slow and stable, as the cruise control has to be to be energy efficient, safe, and comfortable for the driver and passenger(s). Thus no to little overshoot is prioritized over a fast settling time. With this in mind a PI-controller is a good choice. The P-controller is good for speed convergence. The integrator can remove stationary error,

because it increases the loop gain at lower frequencies, which is desirable. To design our PI-controller we set the following tuning parameters:

$$\gamma_m = 85^\circ \qquad Ni = 9.2 \tag{10}$$

With MATLAB we calculated the constants in our controller using the method from [13] to be:

$$Kp = 0.1125 \qquad \tau_i = 9.362 \tag{11}$$

The small Kp value means the system is not very aggressive.

A. Bodeplots

A bodeplot of the system, the open loop transfer function, and closed loop transfer function with the controller can be seen in figure 2.

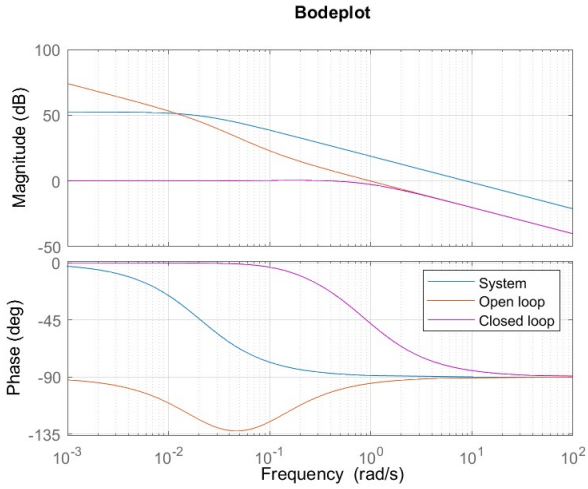

Fig. 2. Bodeplot of the system, open and closed loop

B. Simulink model

The system is modelled in Simulink in order to simulate a step response, which can be compared to the data from the STM32. The saturation block with a feedback loop is included to prevent wind-up. This anti-windup structure uses back calculation, where the gain in the feedback loop is $\frac{1}{Tt}$, where $Tt = \tau_i$, which is a typically used value [14]. One of the advantages of this anti-windup structure is that when the control output is within the voltage limit, it works as a normal PI-controller. To have a smooth step a pre-filter has been implemented. The last gain factor "K-" is simply a conversion from RPM to km/h.

C. Digital implementation

The controller is made and tuned in the Laplace domain, which means it is made with an assumption of continuous time. But to implement the controller on the microcontroller in C++, a z-transformation is necessary. Therefore the s is replaced with the Tustin approximation [13]

$$s \approx \frac{2(z-1)}{T(z+1)} \tag{12}$$

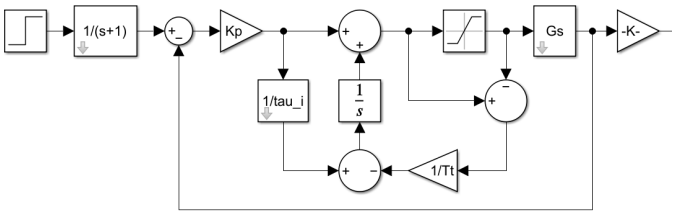

Fig. 3. Simulink model

to make the transfer function of the controller discrete. The sampling time T is set to be 10 times shorter than the settling time [13]. This means $T = 0.44s$.

After the controller is digitally implemented as a C++ function it is used on the controller STM. This STM's basic code structure can be found in the flowchart on figure 4.

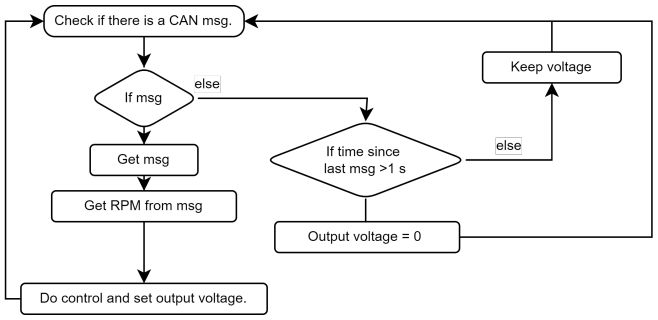

Fig. 4. Overview of the controller STM code.

D. Stability test

All poles and zeroes of the continuous time transfer function with the implemented controller lies in the RHP, which means the system is stable. However this does not guarantee stability in discrete time. To check the stability of the system in discrete time the Jury Stability test [15] is carried out. Using MATLAB the zeroes and poles of the discrete time transfer function is plotted 5. The poles are placed within the unit circle, which means the system is stable.

E. Step response

When the PI-controller is implemented a 5 km/h step response is made. Below the step response in Simulink and the data from the STM32 is shown.

The responses look very similar. The real cruise control acts as expected from the simulation. The response has an overshoot of 0.58%, a slow rise-time, $t_r = 2.65$s, and a slow settling time, $t_s = 4.40$s, which is desirable in this case.

V. CAN

The solar car uses CAN as its primary communication. The network communicates through two wires which will either be equal or unequal from each other, which will give 1 or 0. Due to both using Arduinos and STMs, we have used MCP 2515 [16] CAN controllers and MCP 2551 [17] CAN transceivers which sends, receives and processes the data. The CAN controllers are

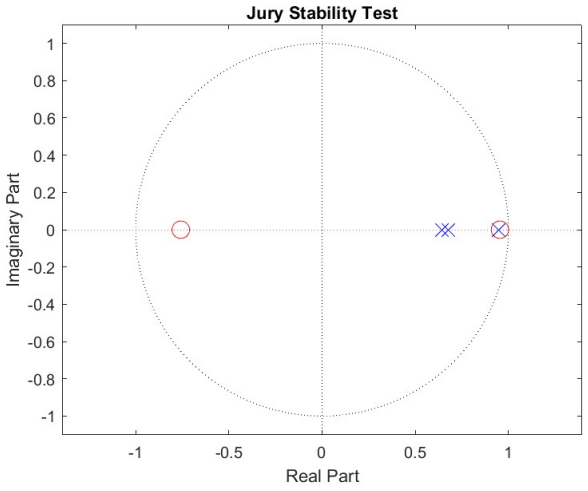

Fig. 5. Zero-pole plot

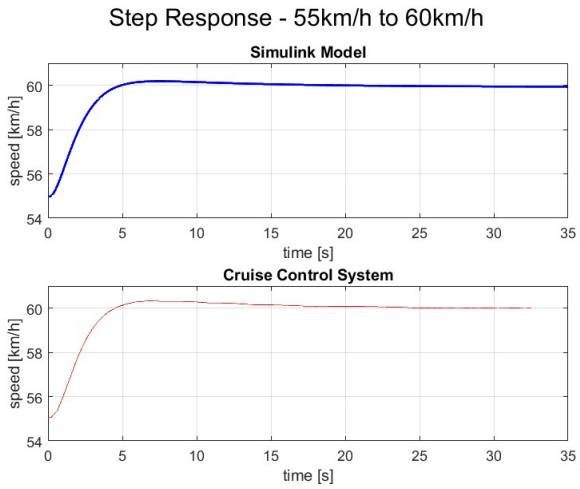

Fig. 6. Step response of simulation and data

connected to the microcontrollers with SPI (Serial Peripheral Interface). The STMs have an integrated CAN controller, but in order to save time and have a standardized method in the project, we used the MCP 2515 for all microcontrollers.

The "CAN_BUS_Shield" version 1.20 library published by Seeed Studio is used to interact with the CAN network. However some changes were made in the library in order for it to work with the STMs.

The CAN can transmit eight bytes, however we are only using three bytes. Of these three bytes, the most significant byte is used to communicate if the brake is initiated and if the cruise control is enabled. The two other bytes are used to transmit the RPM, whether it be the reference RPM from the UI STM to the control STM, or be it the motor RPM sent from the Arduino.

VI. User interface

The user interface of the cruise controller should be intuitive and easy-to-use. This is achieved with a push-button for enabling and disabling the cruise control along with a LED indicating whether it is turned on or off. Furthermore, there are two buttons placed horizontally, with the button on top being +5 km/h and the button below being -5 km/h. For this prototype, a brake button has been implemented with a red LED to indicate brake on. The buttons are connected to an inverting Schmitt trigger(MC14584B) [18] and an inverter(HEF4069UBP) [19]. The Schmitt trigger is to eliminate noise from the buttons, as they are somewhat sensitive. This is important in order to improve safety so that the cruise control does not turn on/off unexpectedly. The circuit diagram for the user interface can be seen in fig. 7.

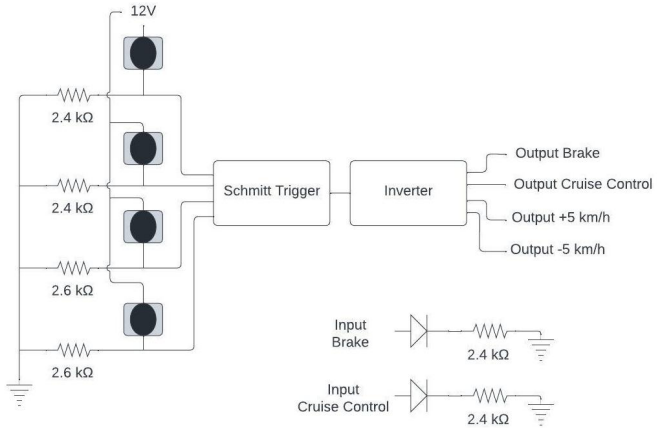

Fig. 7. Circuit diagram of the user interface

As the push-buttons used in our prototype will not be used when the cruise control system is integrated in the solar car, we have chosen some simple components which were available to us.

These buttons are used as input to the UI STM. The code implemented to control the user inputs and send the desired speed to the controller STM is described in the simple flowchart on figure 8.

VII. Digital to analog conversion

The motor driver takes in an analogue input varying between 0 and 5V, which the analogue output from the STM32 is unable to deliver as it is 0-3.4V. Therefore a DAC0808 [4], which is a 8-bit digital to analogue converter, is used to convert digital output from the STM32. The DAC outputs an inverted signal, accordingly -5V to 0V. The Arduino that handles the simulation of the car and in this case acts as the motor driver, can not read negative voltage values, which the motor driver can not either. To invert the output from the DAC we use an opamp (LM301AN) [20] [21] and a low pass filter. It is set up as shown in the circuit diagram 9.

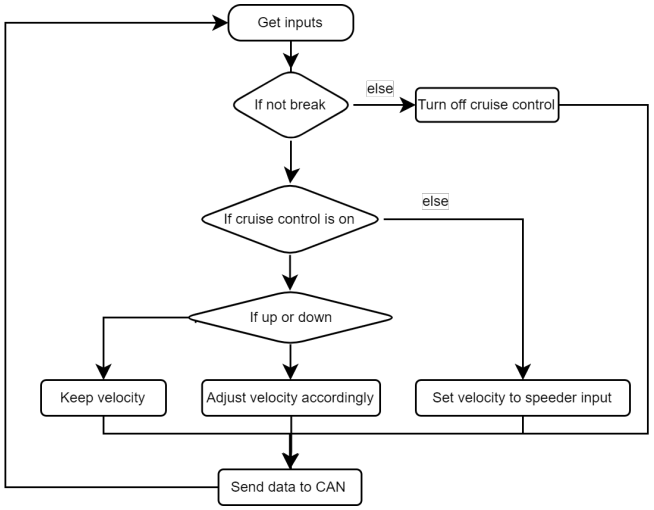

Fig. 8. Overview of the user-interface STM code.

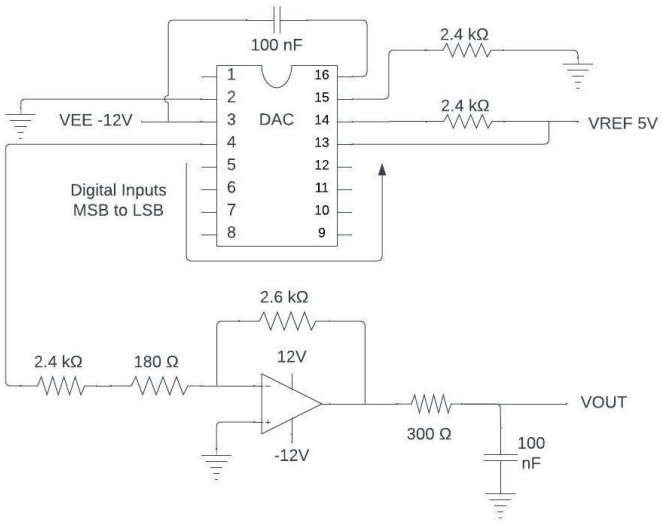

Fig. 9. Circuit diagram of the digital to analogue conversion

A. Inverter

The gain of the inverter should ideally be -1. The exact relationship between the resistors is $\frac{R_f}{R_{in}} = 1.0078$. The opamp had to be balanced, as the inverter had an offset of about 900mV with a input of 0V. A circuit diagram of how this is done is shown in figure 10. This balanced circuit is from the datasheet fig. 32 [20]. The potentiometer makes it possible to adjust the offset which means it is also possible to eliminate a potential offset from the measuring device.

VIII. Voltage regulator

A voltage regulator (LM7805) [22] is needed to convert from 12V to 5V. The circuit diagram can be seen in figure 11.

IX. Testing and validation

All the hardware was tested thoroughly on the breadboard before soldering the circuits to a prototype board. The testing

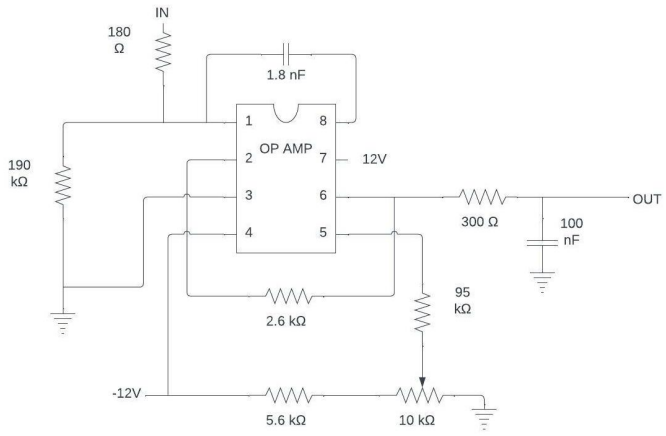

Fig. 10. Circuit diagram of the inverter circuit

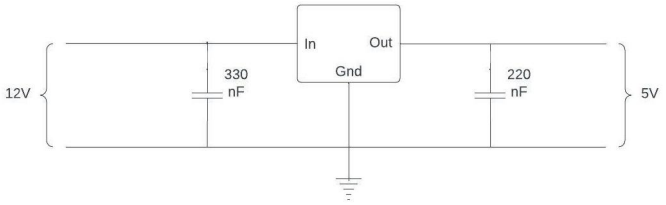

Fig. 11. Circuit diagram of the 12V to 5V voltage regulator circuit

was both done with the oscilloscope and with the microcontrollers. Each module was tested separately before testing everything together.

The software has also been tested thoroughly, both the standalone functions and when connecting the whole system together. It has been tested with edge-cases and random cases to make sure that the system handles every situation as expected.

The PI-controller has been simulated and tuned in MATLAB, where the step responses for different steps were evaluated and the stability of the system assured. Then this PI-controller has been implemented on the microcontroller. The system was then tested to ensure that the MCU's and hardware had the expected behaviour.

X. Discussion

This project is the first iteration of a cruise control function for the solar car. Therefore there are features that have not yet been implemented. We have therefore throughout our project attempted to make it possible to add elements to our design. For example we chose to use CAN transceivers for both STM32, as it makes it possible to implement a Ping-module, which will improve safety of the car. However, implementation of this is outside of the scope of this project.

The speeder-signal was missing during the project which is why the system, as it is now, does not take a speeder-input. When there is a speeder-signal it is possible to integrate it with the cruise control.

At this point, the system gets 12V and -12V from the voltage supply. Later on the 12V is supposed to come from the battery. Then it could be a good idea to use a DC-DC inverter [23]

circuit to invert the 12V to -12V, in order to only have one voltage supply to the prototype board and then later to a PCB.

In extension to the features and integration problems mentioned above we have discovered a major safety issue. As of now if the controller STM looses power the pins communicating with the DAC are not pulled low. This results in an output that the inverter interprets in a way so it's output is just under 5V. This means that the car would try to accelerate uncontrolled if the STM lost power. We propose that this is solved with a hardware implemented logic circuit that pulls the output low in the case that the controller STM does not output clear signals (ie. when the MCU have lost power).

When the safety and integration issues have been solved, improvement of the driving experience may be achieved by implementing different controllers for different driving cases. An example of this could be that the controller for maintaining a constant speed might not be optimal for accelerating. The implementation of this will however be more relevant when the development of the solar car is further along, since further knowledge of the most optimal acceleration and speed is needed to select the most relevant controller parameters.

XI. Conclusion

The purpose of this project was to make a cruise control system for the DTU Solar Car. Overall the project has successfully fulfilled the problem definition.

The chosen microcontroller is a STM32, which was an agreement with the other Solar Car groups. To make the STM32 compatible with the motor driver a DAC [4] with an inverter circuit has been implemented successfully. As it is a prototype the user-interface consists of only of two LED's and four buttons, which is simple and intuitive, but will have to be further developed when it is implemented in the car. A stable and smooth PI-controller has been successfully designed and tuned. Some of the safety features included is that the brake overrules cruise control, and with the CAN communication it is possible for further development to implement a ping module to improve the safety. The PI-controller is currently tuned to the somewhat simple simulation of the future solar car, and is going to need re-tuning when it is implemented in the actual car.

The scope of the project was that the cruise control system should be able to maintain a constant speed, but the project has exceeded this and implemented the ability to change the set speed by ± 5km/h.

This first iteration of a cruise controller for the solar car has successfully been designed and implemented, fulfilling the initial requirements and primed for further development.

XII. Acknowledgment

We would like to thank our mentors Christian Kruuse Kamp, Louise Julie Sternholdt-Sørensen and Sebastian Zeest Rydahl for all the guidance, help and meetings. We would also like to thank the Solar Car team, and especially Claus Suldrup Nielsen, for help and financial support. Lastly we would like to thank Hans Henrik Niemann for facilitating the course and for giving guidance on digital control.

References

[1] DTU Roadrunners. *DTU Roadrunners Solar Car*. [Online]. Available: https://dtucar.com/wiki/index.php?title=Main_Page#Solar_Car

[2] Bridgestone World Solar Challenge. *Bridgestone world solar challenge*. [Online]. Available: https://www.worldsolarchallenge.org/

[3] STMicroelectronics, "STM32F103x8 STM32F103xB, Medium-density performance line Arm®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces", STM32F103x8 datasheet, Aug. 2013, [Revised Mar. 2022]

[4] Texas instruments, "DAC0808 8-Bit D/A Converter", DAC0808 datasheet, May 1999.

[5] STMicroelectronics, "UM1724 User manual", STM32 Nucleo-64 board datasheet, Feb. 2014, [Revised Aug. 2020]

[6] STMicroelectronics. *NUCLEO-F103RB*. [Online]. Available: https://os.mbed.com/platforms/ST-Nucleo-F103RB/

[7] Arduino, "Arduino Uno 3", Product Reference Manual, May 2021, [Revised May 2022]

[8] A. Stark. (2022). *Vehicle acceleration and maximum speed modeling and simulation* [Online]. available: https://x-engineer.org/vehicle-acceleration-maximum-speed-modeling-simulation/

[9] C.K. Kruuse. (2021, May)."Implementation and Test Platform Development for PMSM, Driver and Controller for Electric Vehicle".[Online]. Available: https://dtucar.com/wiki/index.php?title=Motor

[10] Chegg. *Calculate The Road Load Power For The Case Toyota Prius* [Online]. Available: https://www.chegg.com/homework-help/questions-and-answers/calculate-road-load-power-case-toyota-prius-fronatal-area-08m2-sliding-street-coefficient--q79069651

[11] A. Biddle. (2015). *Efficiency by design: the aerodynamics of Toyota Prius* [Online]. Available: https://mag.toyota.co.uk/efficiency-design-aerodynamics-toyota-prius/

[12] Car and Driver (2014). *Drag Queens: Aerodynamics Compared* [Online]. Available: https://www.caranddriver.com/features/a15108689/drag-queens-aerodynamics-compared-comparison-test/

[13] C. Andersen, I. Santos, H.H. Niemann and O. Jannerup, *Feedback control techniques for practical PID design F2022*. Lyngby: Polyteknisk Boghandel & Forlag, 2022.

[14] C. Bohn and D.P. Atherton, *An Analysis Package Comparing PID Anti-Windup Strategies* [Online]. Available: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=375281

[15] K.M. Moudgalya, "Jury's Stability rule" in *Digital Control*, Chichester: John Wiley & Sons Ltd, 2007, pp.87-89.

[16] Microchip. "Stand-Alone CAN Controller with SPI Interface," MCP2515 datasheet.

[17] Microchip. "High-Speed CAN Transciever," MCP2551 datasheet.

[18] ON semiconductors, "MC14584B Hex Schmitt Trigger", MC14584B datasheet, May 2014

[19] Texas instruments, "HEF4069UB Hex inverter", HEF4069UB datasheet, Jan. 1995, [Revised Nov. 2011]

[20] Texas instruments, "LM101A/LM201A/LM301A Operational Amplifiers", LM301AN datasheet, Sep. 1999, [Revised March 2013]

[21] STMicroelectronics, "LM101A - LM201A LM301A SINGLE OPERATIONAL AMPLIFIER", LM301AN pinout datasheet, Oct. 1995

[22] Texas instruments, "LM340, LM340A and LM78xx Wide VIN 1.5-A Fixed Voltage Regulators", LM7805 datasheet, Feb. 2000, [Revised July 2016]

[23] Maxim, "-5V/-12V/-15V or Adjustable, High-Efficiency, Low IQ DC-DC Inverters", DCDC inverter datasheet, 1994.