



Designing a VVT Control System for a Single Cylinder Research Engine

Development and Implementation of a Turnkey Variable Valve Timing Solution for a Research Spark-ignited Engine

Master's thesis in Product Development

GABRIEL BERG & AXEL FELDT

DEPARTMENT OF MECHANICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026
www.chalmers.se

MASTER'S THESIS 2026

Implementing Variable Valve Timing on a Research Engine

Development and Implementation of a Turnkey Variable Valve
Timing Solution for a Research Spark-ignited Engine

GABRIEL BERG
AXEL FELDT



CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Mechanical Engineering
Division of Energy Conversion and Propulsion Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2026

Implementing Variable Valve Timing on a Research Engine
Development and Implementation of a Turnkey Variable Valve Timing Solution for
a Research Spark-ignited Engine
GABRIEL BERG & AXEL FELDT

© GABRIEL BERG & AXEL FELDT, 2026.

Supervisor: Petter Dahlander, Department of Mechanical Engineering
Examiner: Petter Dahlander, Department of Mechanical Engineering

Master's Thesis 2026
Department of Mechanical Engineering
Division of Energy Conversion and Propulsion Systems
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: 3D-render of CAD model with AI generated display contents.

Typeset in L^AT_EX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2026

Implementing Variable Valve Timing on a Research Engine
Development and Implementation of a Turnkey Variable Valve Timing Solution for
a Research Spark-ignited Engine
GABRIEL BERG & AXEL FELDT
Department of Mechanical Engineering
Chalmers University of Technology

Abstract

Four-stroke combustion engines operate at a variety of speeds, typically between 1000 rpm and 7000 rpm, affecting the flow of gases in and out of the combustion chamber. Flow of gases is controlled by opening and closing of poppet valves. This opening and closing is typically done by a camshaft driven by and running synchronously to the crankshaft. Variable valve timing (VVT) allows for a dynamic (while operating the engine) phase between camshaft and crankshaft, allowing dynamic changes in gas exchange and engine characteristics as different valve timing is preferred at different engine speeds and loads.

This thesis covers the development of a VVT control system for an AVL one-cylinder four-stroke engine located in test cell C at The Division of Energy Conversion and Propulsion Systems (ECaPS) at The Department of Mechanical Engineering (ME) at Chalmers University of Technology. The development is based on the engine's existing hardware, such as crankshaft and camshaft sensors, hydraulic channels and VVT solenoids controlling oil flow; therefore, this project has covered the development, manufacturing and installation of a control system and complete electrical installation.

The control system was developed iteratively with five rounds of prototyping and testing, aiming to produce a working product fulfilling user requirements of being stand-alone, simple, robust, and based on standard components.

The final product is the control system, based on a Teensy 4.1 microcontroller, which allows the user to perform independent on-the-fly cam phasing. Although it was not possible to procure the necessary cam gears to implement the hardware side of the solution, the system has proven itself during real operating conditions, connected to the engine test rig.

Keywords: control system, product development, variable valve timing, VVT

Acknowledgements

We would like to express our sincere gratitude to the people who have supported us throughout the work on this thesis.

First and foremost, we wish to thank our supervisor and examiner, Professor Petter Dahlander, for his guidance, expertise, and valuable feedback in the field of internal combustion engines.

We would also like to extend our appreciation to Robert Buadu, senior research engineer, for his assistance and support throughout the project. His practical knowledge and experience, especially in the electronics field, have been a great resource to us.

We would also like to thank Vincent Larsson, doctoral student, for his help with running the lab engine during the experimental work.

Gabriel Berg & Axel Feldt, Gothenburg, May 2026

List of Acronyms

Below is the list of acronyms that have been used throughout this thesis listed in alphabetical order:

BDC	Bottom Dead Center
CAD	Crank Angle Degree
CDM	Crank Degree Mark
DOHC	Dual Overhead Camshaft
Dual VVT	Variable Intake and Exhaust Valve Timing
ECaPS	(Division of) Energy Conversion and Propulsion Systems
ECU	Engine Control Unit
EGR	Exhaust Gas Recirculation
EMA	Exponential Moving Average
EVC	Exhaust Valve Closing
EVO	Exhaust Valve Opening
HC	Hydrocarbons
IVC	Intake Valve Closing
IVO	Intake Valve Opening
LCD	Liquid Crystal Display
ME	(Department of) Mechanical Engineering
MOSFET	Metal Oxide Field Effect Transistor
NA	Naturally Aspirated
OLED	Organic Light Emitting Diode
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
SI	Spark-ignited
SPI	Serial Peripheral Interface
TDC	Top Dead Center
UX	User Experience
VANOS	Variable Nockenwellensteuerung
VVT	Variable Valve Timing

Nomenclature

Below is the nomenclature of variables that have been used throughout this thesis.

Variables

g	Gap between plunger and end of coil
i	Current through coil
p_1	Oil pressure in advance chamber
p_2	Oil pressure in retard chamber

Contents

List of Acronyms	ix
Nomenclature	xi
List of Figures	xv
List of Tables	xvii
1 Introduction	1
1.1 Background	1
1.1.1 Four-stroke Spark-ignited Engine	1
1.1.2 Gas Exchange	4
1.1.3 Variable Valve Timing	5
1.2 Aim	8
1.3 Research Questions	8
1.4 Limitations	8
2 Theory	9
2.1 Current Engine Setup	9
2.1.1 Camshaft Sensors	10
2.1.2 VVT Solenoids	10
2.1.3 AVL Angle Encoder	11
2.1.4 Oil Channels	13
2.2 Signal Theory	13
2.3 Controlling Solenoids	16
3 Identified Usage Requirements	19
3.1 User Questions	19
3.2 User Needs	19
4 Prototyping and Testing	21
4.1 Hardware Platform Pre-study	21
4.2 Phase 1: Initial Software and Arduino Testing	22
4.3 Phase 2: Teensy Microcontroller Testing	23
4.4 Phase 3: Initial User Interface and Peripherals Testing	25
4.5 Phase 4: Updated User Interface and Hardware	26
4.6 Phase 5: Development of Electronics for Standalone Use	27

5	Result	29
5.1	Final Product	29
5.2	Components	31
5.2.1	MOSFET	31
5.2.2	Gate Driver	31
5.2.3	DC-DC Converter	32
5.2.4	Diode	32
5.2.5	Logic Converter and Voltage Divider	32
5.2.6	Rotary Encoder	33
5.2.7	Liquid Crystal Display (LCD)	34
5.2.8	Capacitor	34
5.2.9	Fuses, Cable Dimensioning and Routing	34
5.2.10	Noise, Shielding, Signal Treatment, Interference	35
5.3	Installation	35
5.3.1	Connectors	36
5.3.2	Junction Box	37
5.4	Test Run and Signal Capture	38
5.5	Software	38
5.5.1	Initialization	40
5.5.2	Functions	43
5.5.3	Loops	48
6	Conclusion	57
	Bibliography	61
A	Schematic of Final Product	I
B	Schematic of Junction Box	III

List of Figures

1.1	Labeled diagram of a four-stroke DOHC engine. Note. From Wapcaplet (2005), Wikimedia Commons, licensed under CC BY-SA 3.0.	1
1.2	Sketch showing cam lobe with base radius r_{base} and max radius r_{max}	2
1.3	The four strokes of the four-stroke cycle.	2
1.4	Valve opening relative to piston position.	3
1.5	AI-generated sketch of camshaft phaser internals. Generated by Google Gemini, 2026.	6
1.6	AI-generated diagram of VVT solenoid oil flow path. Generated by Google Gemini, 2026.	6
1.7	Missing link for VVT.	8
2.1	Front view of engine setup.	9
2.2	Cam sensor and camshaft trigger wheel system.	10
2.3	VVT solenoid mounted on top of valve cover.	11
2.4	AVL rotary encoder.	12
2.5	Components with oil channels.	13
2.6	Close up of measured CDM signal.	14
2.7	Close up of measured trigg signal.	14
2.8	Camshaft sensor signal.	15
2.9	Geometry of BMW N20 emulator trigger wheels.	16
2.10	“Geometry” of research engine emulator trigger wheels with first camshaft.	17
2.11	“Geometry” of research engine emulator trigger wheels with second camshaft.	18
4.1	Diagram of the testing setup.	24
4.2	Early prototype.	25
4.3	Second prototype.	26
4.4	Third prototype.	27
5.1	Inside of box.	30
5.2	3D-render of CAD model with AI generated display contents.	30
5.3	Crankshaft signal using logic converter.	33
5.4	Rotary encoder internals.	34
5.5	Rear of box with connectors.	36
5.6	Camshaft sensor contact.	37

5.7	Inside of junction box.	38
5.8	Software flowchart.	39

List of Tables

1.1	Typical intake and exhaust valve opening and closing times.	3
5.1	Cable list	35

1

Introduction

This report covers the development of a variable valve timing (VVT) control system for a 0.5 L 1-cylinder lab engine. In this chapter, background, aim, research questions and limitations for the project will be described.

1.1 Background

In order to understand VVT, the fundamentals of the four-stroke engine are first covered, later narrowing down to focus on valvetrain and VVT technology.

1.1.1 Four-stroke Spark-ignited Engine

Figure 1.1 illustrates a four-stroke spark-ignited (SI) engine. A SI-engine is a machine that converts chemical energy to rotational energy by combusting fuel and air, producing torque on the crankshaft (C). Air and fuel is combusted in a chamber, which creates pressure. This pressure results in a force on the piston (P) that pushes it down. The linear translation of the piston is converted to torque by the connecting rod (R) and crankshaft (C). A four stroke engine gets its name from the fact that a full engine cycle (two revolutions) consists of four piston strokes.

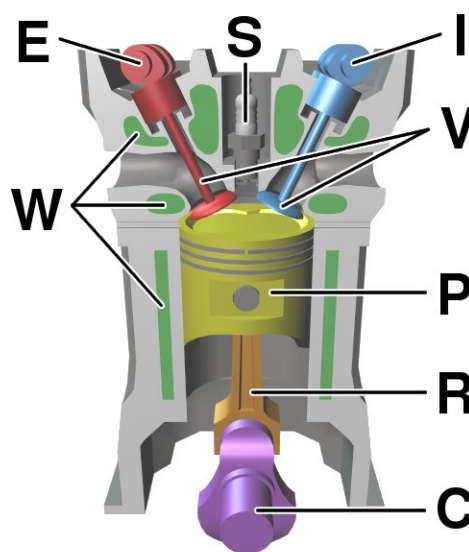


Figure 1.1: Labeled diagram of a four-stroke DOHC engine.

Note. From Wapcaplet (2005), Wikimedia Commons, licensed under CC BY-SA 3.0.

The flow of gas in and out of the engine is controlled by the intake and exhaust valves (V), which in turn are driven by the intake (I) and exhaust (E) camshafts. As the camshafts rotate the varying radius of the cam lobe, shown in Figure 1.2, forces the valves to open. Closing of the valves is typically accomplished using springs, that push the valves back as the radius of the lobe decreases. The camshafts are typically driven by the crankshaft using either belts, chains or gears, and rotate synchronously with the crankshaft at half its angular speed. As the camshafts rotate, the lobes push on the valves. This opens the valves, allowing gas exchange to occur.

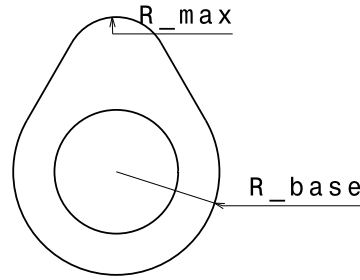
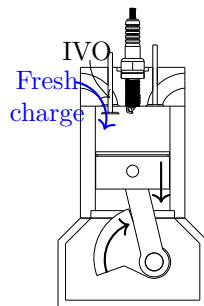
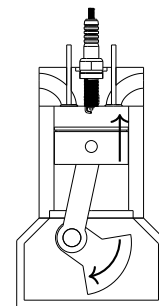


Figure 1.2: Sketch showing cam lobe with base radius r_{base} and max radius r_{max} .

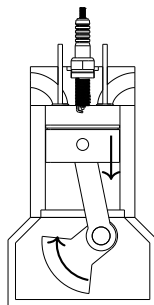
Figure 1.3 shows the four strokes of a four-stroke engine: intake (a), compression (b), power (c), and exhaust (d). The first stroke (a) of the four stroke cycle is the



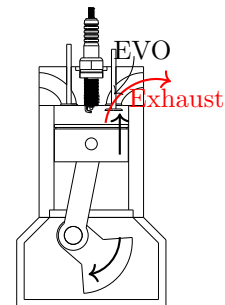
(a) Intake



(b) Compression



(c) Power



(d) Exhaust

Figure 1.3: The four strokes of the four-stroke cycle.

intake stroke. Here, the piston travels downwards from top dead center (TDC) as

the intake valve is opened and the exhaust valve is closed, allowing for intake of fresh air into the cylinder. The second stroke (b) is the compression stroke. As both intake and exhaust valves are closed, the piston travels up from bottom dead center (BDC), compressing the air in the case of direct-injected engines. Direct-injection is the process of injecting fuel directly into the combustion chamber. Other solutions include port-injection and carburetors, resulting in the intake and compression strokes working around an air-fuel mixture rather than just air. The third stroke (c) is the expansion or power stroke, where the rapid expansion of the combusting gases forces the piston downwards from TDC to BDC with both exhaust and intake valves closed. The fourth stroke (d) is the exhaust stroke where the piston travels upwards from BDC to TDC, forcing exhaust gases out through the opened exhaust valve. In practice, opening and closing of the intake and exhaust valves do not happen at exactly TDC and BDC. The reasoning behind this is both that valves do not open instantaneously, as one realizes studying the profile of a cam lobe, as well as for reasons related to air flow and gas exchange, which are discussed in Section 1.1.2. Table 1.1 and Figure 1.4 show typical opening and closing times according to Heywood (2019).

Table 1.1: Typical intake and exhaust valve opening and closing times.

Intake valve opening	10°–15° before TDC
Intake valve closing	50°–60° after BDC
Exhaust valve opening	40°–60° before BDC
Exhaust valve closing	15° after TDC

Note. Data from Heywood (2019).

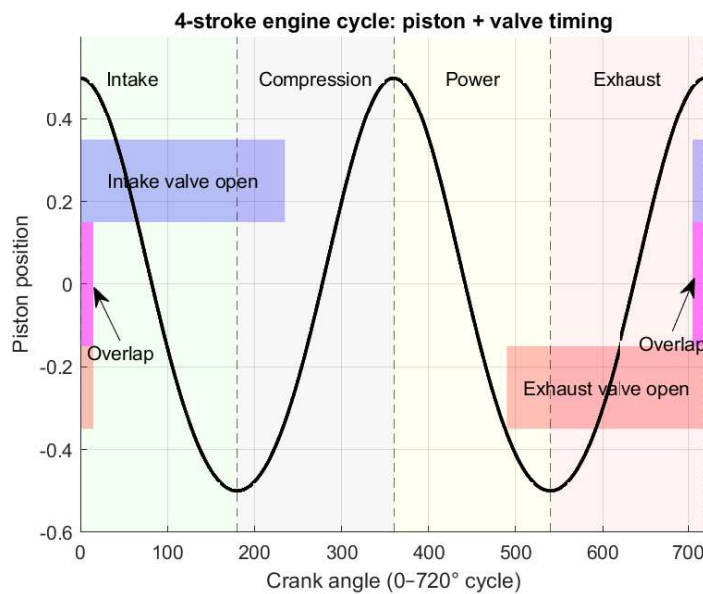


Figure 1.4: Valve opening relative to piston position.

1.1.2 Gas Exchange

Combustion engine characteristics are directly affected by gas exchange. The reason behind this is that the amount of fresh air in the cylinder affects how much fuel can be combusted which in turn affects maximum power output. There are several ways to improve gas exchange, including utilizing forced induction, decreasing flow restrictions, and tuning intake and exhaust ports and manifolds. Common for all of these is that they aim to improve scavenging and/or filling, meaning the extraction of exhaust gases from the cylinder and filling of fresh charge into the cylinder, respectively.

Forced induction — In a naturally aspirated (NA) engine, the filling of the cylinder is only the result of the volume increase. In an engine with forced induction, the intake pressure is increased to higher than atmospheric by for example a turbocharger (a compressor connected to a turbine driven by the exhaust gases) or a compressor (driven by the crankshaft), resulting in a higher mass of air filled into the cylinder and therefore the possibility to combust more fuel.

Decreasing flow restrictions — The geometry of valves, ports, and manifolds all exert a resistance against the flow of intake and exhaust gases. These restrictions can be decreased by for example increasing diameters, smoothing out bends or sharp edges.

Tuning intake and exhaust — Neither combustion nor gas exchange occurs continuously, but rather in pulses. Pressure pulses travel through both the intake and exhaust manifolds. Here, it is important to note that the intake and exhaust gases are the media, and that the gases themselves do not travel as waves. Rather, it is a pressure wave. Nonetheless, this can be used by tuning intake to manifold to a certain frequency in order to have the peak of the pulse hit the ports as the valves open, causing a higher port pressure, which in turn can increase cylinder filling. The opposite can be applied to exhaust, where exhaust scavenging can be increased by lowering exhaust port pressure as the exhaust valve opens.

As mentioned, all of these affect engine characteristics. For example, a production engine may have been designed with drivability and low end torque in mind whereas a race engine may disregard these drivability aspects and instead aim at increasing maximum power at high engine speeds. This can be accomplished by increasing lift and duration. More specifically, keeping the intake valve open longer makes for increased intake of fresh air at high speeds.

One could have overlap between exhaust valve closing and intake valve opening, meaning that the intake valve opens before TDC and that the exhaust closes after TDC. For high engine speeds, this may result in better scavenging/expulsion of exhaust gases and improved filling. On the other hand, overlap at low engine speeds may lead to worse scavenging.

1.1.3 Variable Valve Timing

A limitation of earlier described solutions is that typically, they are optimized for a certain range of engine speeds. Camshafts, for example, may have valve opening and closing that increase performance at high engine speeds, but decreases performance at lower engine speeds. Since these camshafts are physically synchronized with the crankshaft using a belt, chain, or gears, the degree at which the valves open and close cannot be altered during engine operation.

If one wishes to achieve the characteristics of both the production and race engine at the same time, in order to increase torque over a range of engine speeds, as well as improve drivability, a variable valve timing system (VVT) is one solution. VVT allows for dynamic (during engine operation) change of the phase between the crankshaft and camshaft(s). This enables, for example, delayed intake valve closing (IVC).

VVT is common in cars and has been used for a certain time, being mentioned as early as 1974 (Schiele et al., 1974). It typically operates using the hydraulic pressure of the engine oil to retard or advance valve timing.

One implementation is BMW's Variable Nockenwellensteuerung (VANOS). It works by connecting the cam gear to the camshaft via a pair of coaxial helical gears (one internal, one external). As the ECU adjusts a hydraulic valve, a hydraulic piston translates the gears relative to each other, causing rotation (phasing) between the cam gear and camshaft (National Research Council, 2015).

Hattori et al. (2008) describes the development of Toyotas VVT-iE system. It is an alternative to hydraulically controlled VVT systems which instead uses a brushless electric motor paired with a cycloid reduction gear and a linkage. The objectives are, as with most VVT systems, to reduce fuel consumption and emissions while increasing performance. This is achieved by enabling internal EGR (by advancing intake valve opening (IVO)), reducing pumping losses and increasing expansion ratio (by retarding IVC). The main benefit over a traditional hydraulic VVT system is the independence of oil temperature (hydraulic VVT systems operate slowly with cold and viscous oil) and the ability to operate with the engine turned off. This in turn leads to reduced HC emissions at cold starts by enabling internal EGR to heat the intake ports and the possibility to decrease vibrations in start-stop situations by using the VVT system as a decompression system.

The most common VVT system is hydraulically controlled and is also used in modern Volvo Cars engines, to which the lab engine this project revolves around shares a close resemblance. The system utilizes oil pressure from the main oil pump to fill chambers in the camshaft phaser to rotate the cam wheels relative to the crankshaft, see Figure 1.5. In the figure a larger hole in the middle part is shown, which is for a locking pin. At startup there is very limited oil pressure and the locking pin keeps the cam phaser from rattling.

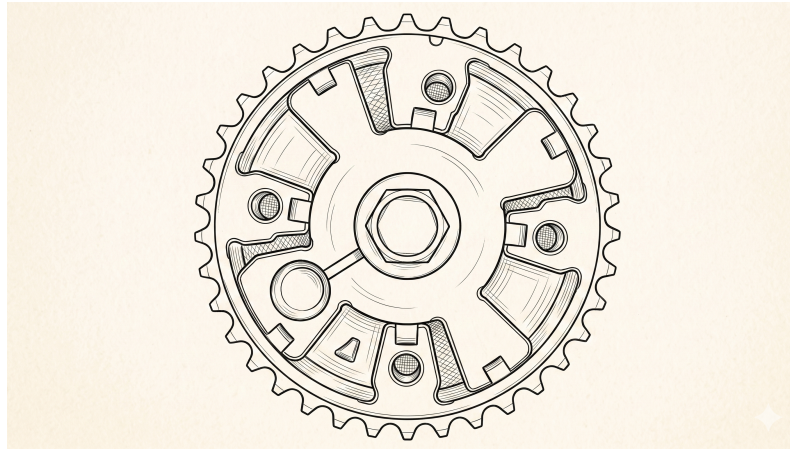


Figure 1.5: AI-generated sketch of camshaft phaser internals. Generated by Google Gemini, 2026.

Oil is channeled through the block and in channels to the head of the engine, which is further explained in Chapter 2.1. Oil flow to the camshaft gear is then regulated by a solenoid for each cam axle. The solenoids have five oil connections; one supply, two drain on opposite sides, and two ports going to the cam phaser, see Figure 1.6.

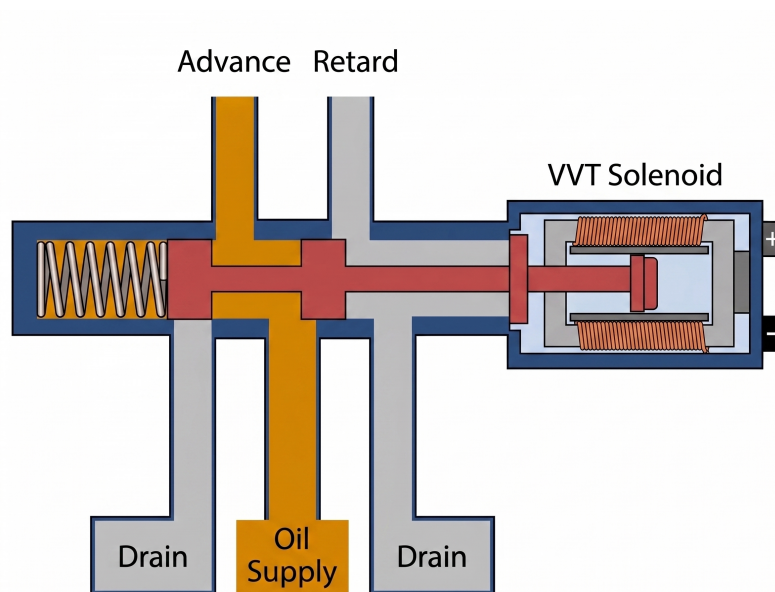


Figure 1.6: AI-generated diagram of VVT solenoid oil flow path. Generated by Google Gemini, 2026.

The solenoid has three main positions, even though the translation can be controlled continuously between them. The first position is the start position, when no power is sent to the solenoid. The spring pushes back and all oil flow is directed to the retard port. The second position is hold when both the red plungers block the flow of oil going to either the advance or retard port. While the engine is running

at steady state this is the most common position with small translation (opening the valves slightly) to adjust the cam phasers. The last position is advance which is shown in the figure. Oil is sent to the advance port and the retard port and drain are connected (National Research Council, 2015). The solenoid position is controlled using a PWM signal, typically between 0 V and 12 V, which because of the dampening of the oil allows for accurate positioning of the solenoid instead of the oscillation behavior induced by the system.

1.2 Aim

This project serves to develop, build, and install a VVT control system to be used in cell C at the Department of Mechanical Engineering at Chalmers University of Technology. The main goal of the project is to make it possible to move the camshaft gears relative to the crankshaft. Most of the hardware to allow for this is already in place but there is a missing part to make the system complete. To satisfy the main goal the missing link needs to be filled, see Figure 1.7.

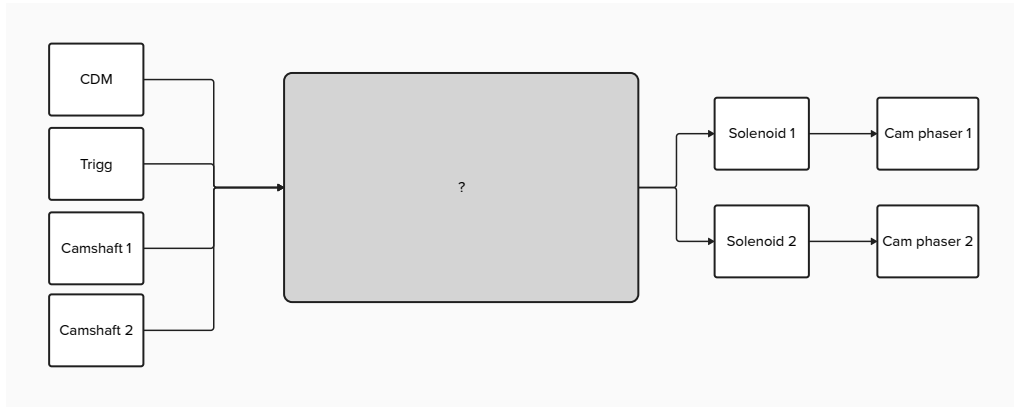


Figure 1.7: Missing link for VVT.

1.3 Research Questions

In conjunction with the physical development of the control system a set of research questions were formulated as a theoretical framework for the project.

- What hardware requirements exist when creating a control system for VVT?
- What software requirements are there to control a hydraulically based VVT hardware?
- What specific needs are there for a VVT system in a lab environment?

1.4 Limitations

There are several circumstances affecting this degree project which need to be taken into account. The system is built upon the existing engine test rig where some of the existing hardware supports VVT and will not be changed. This includes cam shafts and cylinder heads with oil channels for VVT, VVT solenoids mounted on the cylinder head, and camshaft sensors. The solution is to be adapted to the current setup and will be a pure VVT system. This means that only the phase/timing of the cam opening can be altered, not the duration or lift. The goal of the project is not to conduct any lab experiments but rather to develop the equipment used in experiments. The project has a time frame spanning from 19 January to early June and a limited budget.

2

Theory

The following chapter covers the theoretical background necessary to develop the VVT control system, first covering the current hardware of the engine lab, and secondly the signal theory involved.

2.1 Current Engine Setup

In order to develop the VVT control system adapted to the engine in lab cell C, the existing engine hardware was documented. The current lab setup is based around a 1-cylinder, 0.5 L AVL engine. The setup is modular, allowing for a range of different cylinder head configurations to be used. For example a head with a transparent see through port. The head for which this project is based on is custom made and utilizes two overlying camshafts, mounted in modular “units”. Figure 2.1 shows a front view of the engine, with labels showing camshaft sensors, VVT solenoids, cam gears, and crankshaft angle encoder.

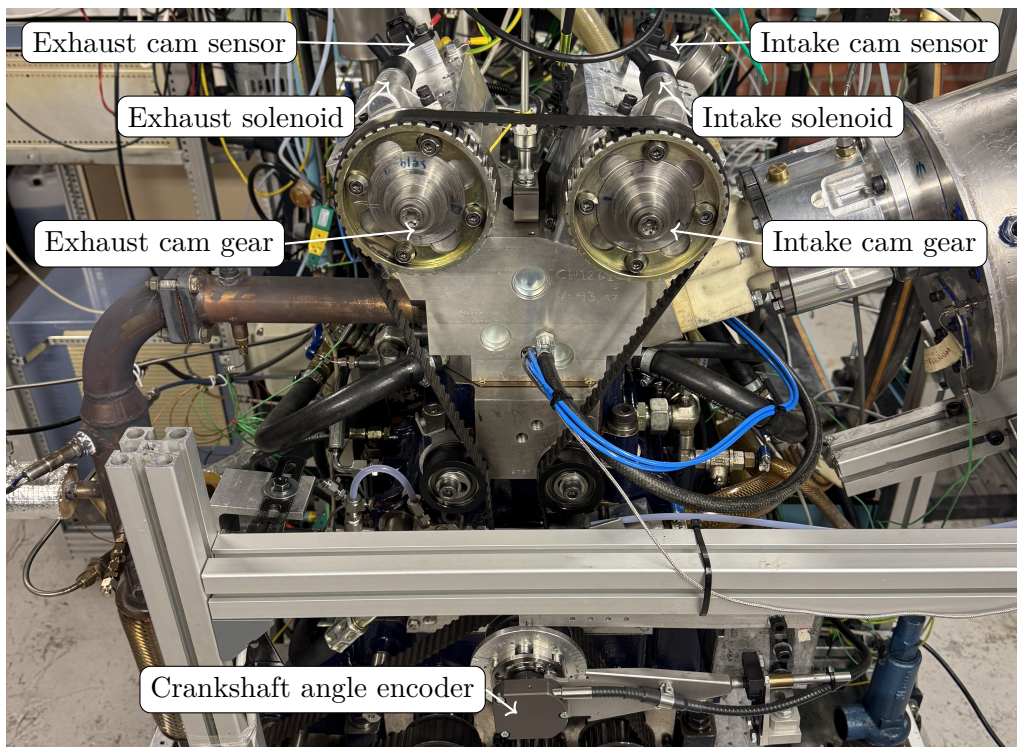


Figure 2.1: Front view of engine setup.

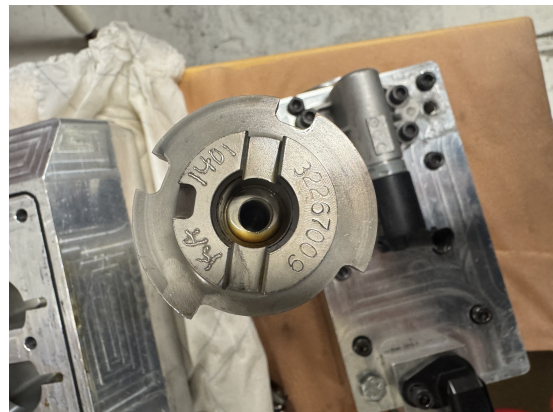
2.1.1 Camshaft Sensors

The camshaft sensors, see Figure 2.2a, are hall effect sensors made by Denso, with OEM part numbers Ford 6G9N-12K073-BA/Volvo 30713599. The sensors are of an open collector type with three pins: plus(1), ground(2) and signal(3). There was not much data on these types of sensors, therefore testing to get a correct signal was required. The sensor needed either 5 V or 12 V on pin 1. Pin 2 is sensor ground and pin 3 is the signal. There needs to be a pull-up resistor between signal and positive for it to output any signal. To simplify it there is a transistor between signal and ground. When metal is in front of the sensor the transistor is floating meaning that the signal has the same potential as the positive source and when no metal is in front of the sensor the transistor closes, pulling the signal to ground. This results in a digital high/low signal, see Figure 2.8.

Inside the sensor there is an element sensing if metal is nearby and gives out a signal. On the camshaft a trigger wheel is placed, see Figure 2.2. The trigger wheel design on this engine has three teeth resulting in three signals per full cycle (720°). The unsymmetrical pattern is designed to be able to give information about in which phase (combustion or exhaust) the full cycle is in. Earlier implementations of camshaft sensor only used one tooth on the trigger wheel which is enough to identify which phase the engine is in. The main reason for adding more teeth on the camshaft trigger wheel is to receive more frequent updates about the camshaft position which is essential for VVT control, as the PID controller can get information three times more frequently resulting in faster and more accurate response. For this application, which phase the engine is in is irrelevant as the only parameters the system needs are the camshaft positions relative to the crankshaft.



(a) Cam sensor



(b) Camshaft trigger wheel

Figure 2.2: Cam sensor and camshaft trigger wheel system.

2.1.2 VVT Solenoids

The solenoids used are Volvo solenoids with OEM number 31316664AD, see Figure 2.3. To aid in dimensioning of the control system, the impedance of the solenoids

were measured using a component tester and the resistance was tested with a multi-meter. Both intake and exhaust side solenoids have similar impedance and resistance and the small difference was deemed measurement tolerance.

Exhaust: $8.6\ \Omega$, 17.9 mH

Intake: $8.5\ \Omega$, 19.0 mH

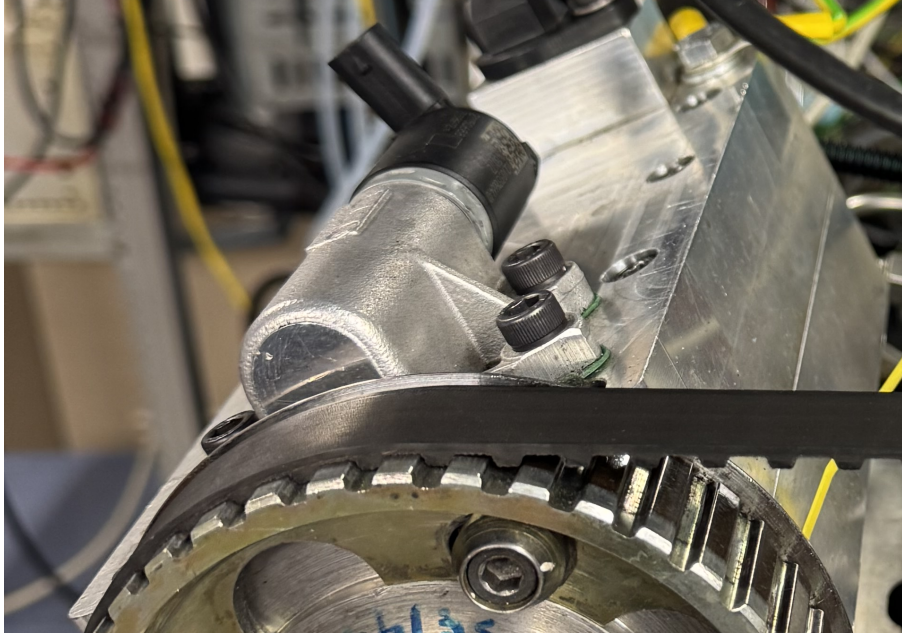


Figure 2.3: VVT solenoid mounted on top of valve cover.

2.1.3 AVL Angle Encoder

The crank angle sensor on the test rig is an AVL branded optical angle encoder, see Figure 2.4. The sensor is an optical incremental encoder that works similarly to a hall sensor but using lights instead. The sensor also differs by the way it sends the signal, instead of there being a larger gap for syncing the engine, the encoder sends two signals: crank degree mark (CDM) and trigger, where CDM is the rotation and trig is syncing the engine. A light is constantly shining and a disc with cutouts connected to the crankshaft lets light through periodically. The light that passes through the disc is detected by a light sensor. After the sensor there is a converter that converts the signal from the light sensor to a square wave signal. The AVL encoder used on the lab engine has 720 slots, which means that the crank position can be measured to a precision of 0.5° . The box that converts the optical signal to a usable square wave has the option to send out up to 3600 edges per second using interpolation (AVL List GmbH, 2010).

It should be mentioned that there also exists absolute optical encoders where several lights are used in combination with a disc with several rings with cleverly spaced cutouts. The different light patterns can then be interpreted as different positions, for example a sensor with four lights can have 16 different combinations. This effec-

tively means that one full rotation can be identified in 16 discrete slices $\frac{360^\circ}{16} = 22.5^\circ$. This resolution is relatively low, with the benefit being that the position is always known, from start and at standstill.

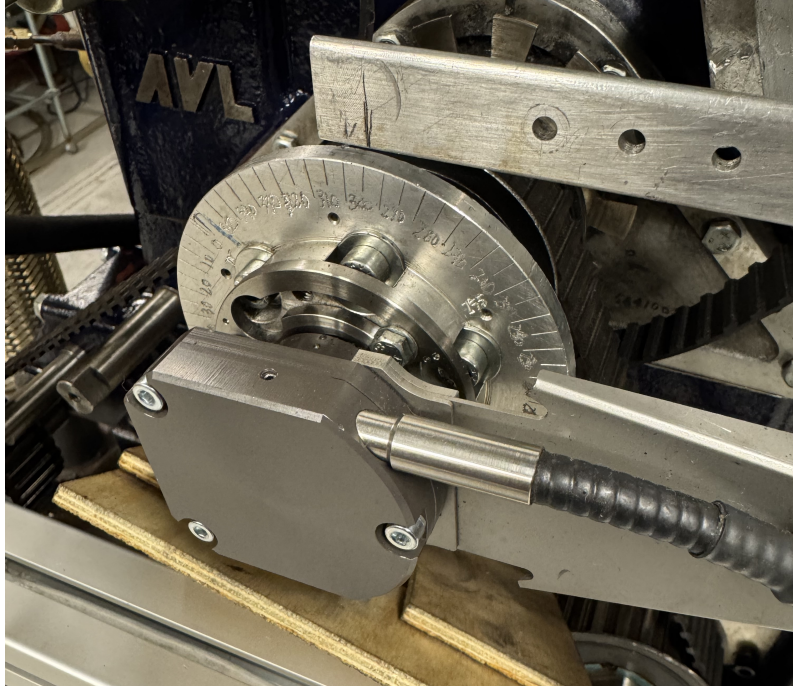
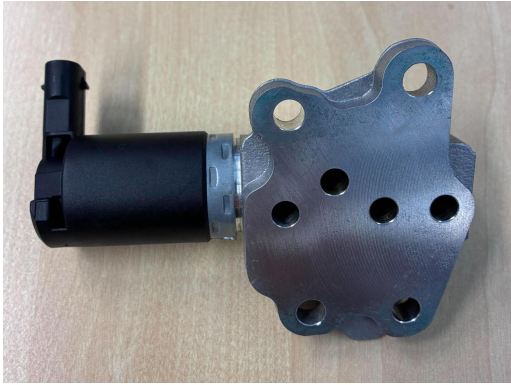


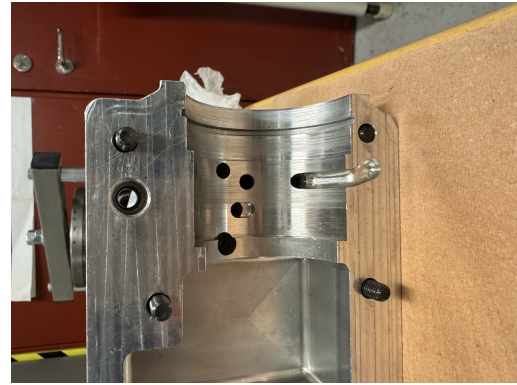
Figure 2.4: AVL rotary encoder.

2.1.4 Oil Channels

The camshafts and cam units are prepared for VVT as they have oil passages. Figure 2.5 shows a VVT solenoid, camshaft unit, and camshaft with oil channel holes. The camshaft units house the camshafts, with VVT solenoids and camshaft sensor mounted on top of the camshaft units as shown in Figure 2.1.



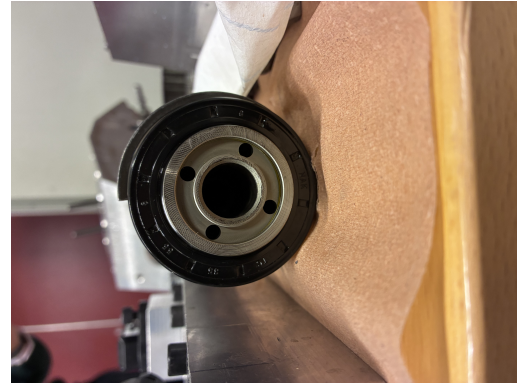
(a) VVT solenoid, bottom view



(b) Camshaft unit half



(c) Camshaft



(d) Camshaft

Figure 2.5: Components with oil channels.

The actuators and sensors used are the same ones used in Volvo Cars gasoline engines from circa 2002 to 2019. The camshafts used are single-cylinder versions of cams used in the previously mentioned Volvo engines. The effect of this is that parts are and will continue to be readily available.

2.2 Signal Theory

The VVT system in a mass-produced environment, at its core, needs to receive four signals and send out two signals. The first signal is the crankshaft position sensor, which also is a must for several different functions in an engine. On the crankshaft there is a toothed wheel with an irregularity, usually a missing or shorter tooth. A hall effect sensor is a magnetic sensor that detects when a magnetic material is close to it. The output of the signal is an on/off waveform. The irregularity on the

2. Theory

crankshaft trigger wheel is noticed by the hall effect sensor to synchronize the signal with where the actual position of the engine is. Depending on how many teeth there are on the wheel the accuracy of the signal can be altered, a common standard is to have 36 teeth on the wheel. With a cycle length of 720° this returns an accuracy of 20° . For the research engine the standard crankshaft sensor is replaced with an angle encoder as described above. The output of this is also split up in two channels: CDM and trigg, instead of using an irregularity for syncing. The measured output from these signals can be seen in Figures 2.6 and 2.7.

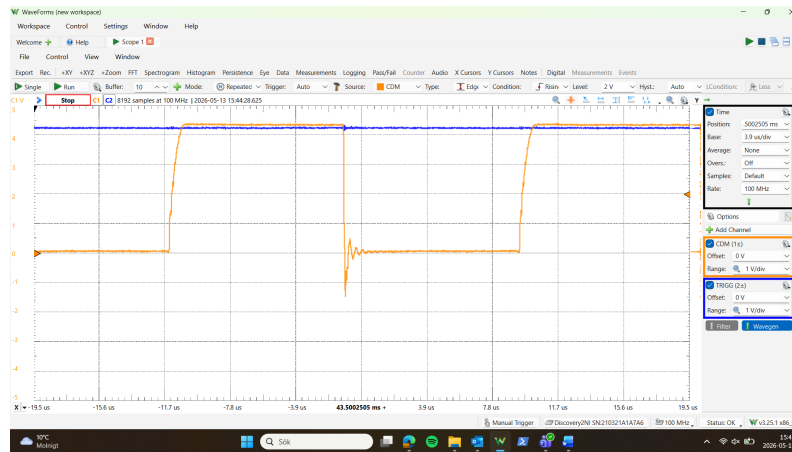


Figure 2.6: Close up of measured CDM signal.

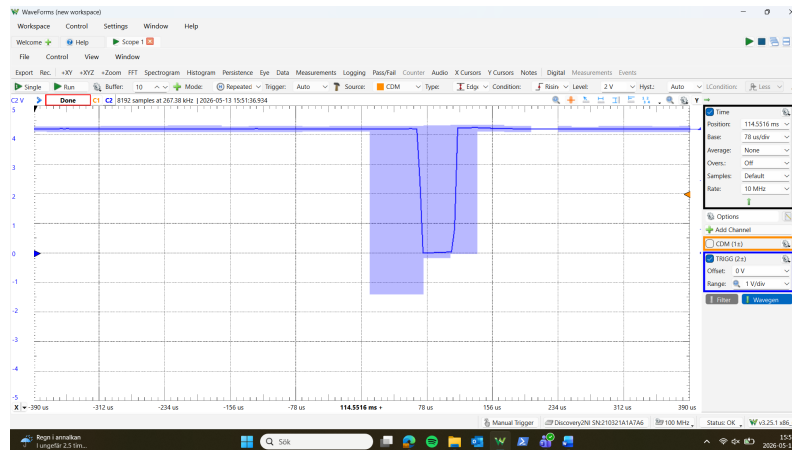


Figure 2.7: Close up of measured trigg signal.

The second and third input signals are the hall effect sensors for both the camshafts. The camshaft trigger wheel usually has fewer teeth than the crankshaft trigger wheel, leading to a lower resolution signal. The reason for this is that the combination of the position sensors create sufficient information. As the camshaft spins at half the speed of the crankshaft only the relative position to the crankshaft (if the engine is in its first phase 0° – 360° or second phase 360° – 720°) is needed, and in the case where

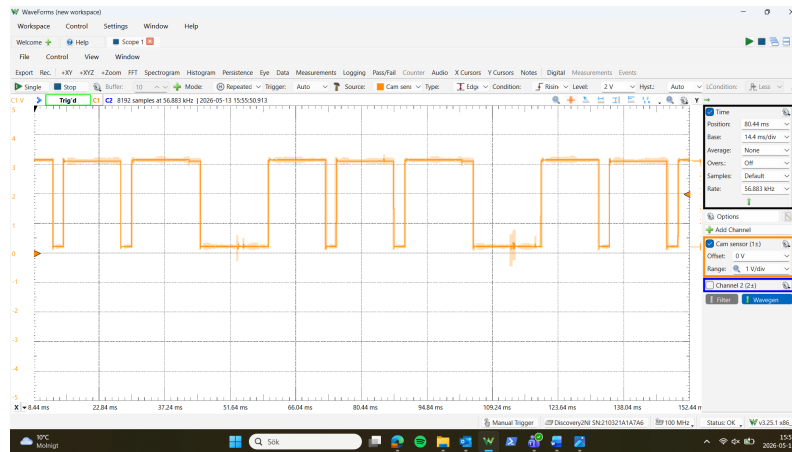


Figure 2.8: Camshaft sensor signal.

VVT is installed, to determine the relative shift from the crankshaft. For example if the camshaft trigger wheel has three teeth the relative angle can be checked and regulated three times per each full engine cycle $720^\circ/3 = 240^\circ$. Measured camshaft signals from the research engine can be seen in Figure 2.2a.

The last signal is the oil temperature sensor. At low oil temperatures, the oil flow is lower and the pressure higher as the fluid is more viscous, which affects the function of hydraulically controlled VVT systems. In a factory engine this sensor can be used to control if the VVT system should be switched on and adjust the PID regulator values to compensate. For this application this sensor is not necessary as a majority of testing is done with preheated fluids.

Initially, an emulator with the standard production engine signals were used for testing the system and software. These signals were created using the engine emulator Ardustim, running a BMW N20 engine (Stewart, 2024). This engine was chosen because it had a camshaft trigger wheel pattern similar to that of the lab engine. An illustration of the initial emulator trigger wheel geometry can be seen in Figure 2.9. After measuring the actual signals generated by the research engine a custom engine emulator could be created using the same Ardustim program with edited source code. This simulator recreated the AVL engine encoder by producing 360 identical teeth on the crankshaft with two inverted trigger pulses on one of the camshafts. It was not possible to produce the 3600 pulses actually sent out by the crankshaft angle encoder, so it was decided to reduce the pulses by a factor of 10. That meant that the regulator system math could be tested with a slightly lower resolution. The first camshaft that emulates the trigger pulses has two inverted teeth, 180° apart because the camshaft spins at half the speed, recreating the single gap that is seen per revolution on the engine. On the second camshaft an actual camshaft pattern could be produced to test the code on the regulator system. A figure of representative trigger wheel patterns for the research engine emulator can be seen in Figures 2.10 and 2.11.

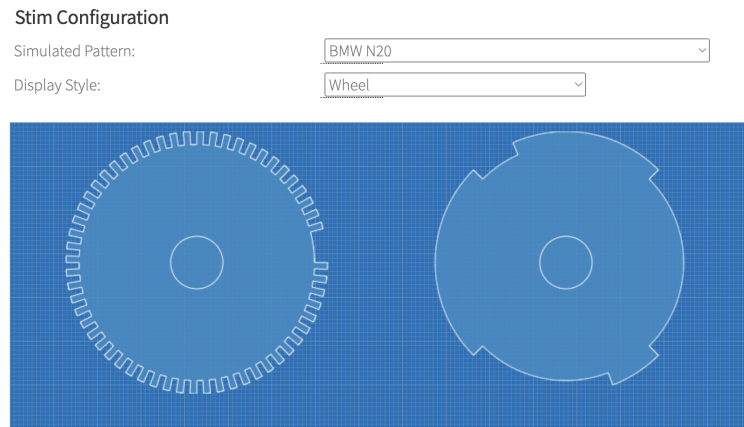


Figure 2.9: Geometry of BMW N20 emulator trigger wheels.

2.3 Controlling Solenoids

Automotive grade solenoids are linear solenoids. This means that when the solenoid is given a PWM signal linear translation can very precisely be controlled. It works by counteracting a spring. When no electricity is sent to the solenoid, the spring guarantees that the solenoid is in its base position. When electrical pulses are sent to the solenoid a magnetic field is built up that moves the shaft inside the solenoid. By balancing the time of each pulse a force balance between the magnetic field and the spring can be created. As the spring force is linear with translation, the absolute position can be controlled by adjusting the magnetic force. The magnetic force however is not linear:

$$F = \frac{(N \cdot i)^2 \cdot \mu_0 \cdot A}{2 \cdot g^2}$$

Apart from the constants there is i^2 and g^2 that affect the magnetic field. i is the current, meaning that changing the current affects the magnetic force quadratically. g is the gap, in this case between the metal core of the plunger and the end point of the solenoid. This creates an issue when trying to control the exact position of the solenoid as when the gap becomes small the magnetic force is proportional to $\frac{1}{g^2}$, meaning that when the gap is small the force approaches infinity, (this is a simplification of idealized magnetic circuit behavior). It creates a characteristic of the solenoid to jump to the end point when it gets close and stick there. In order to overcome the magnetic force the current almost has to go to 0 A before the spring then pushes the plunger all the way back. This is fine when you only want a solenoid that is on or off. Linear solenoids largely solve the issue by shaping the core like a cone that redirects some of the magnetic force from linear to radial force. The non-linearity of the magnetic force is one of the reasons a PID regulator is needed.

The PWM signal that needs to be generated has three main design variables: the frequency, voltage, and the duty cycle (the percentage of on-time). The voltage for which automotive solenoids are designed for is roughly 12 V to 14.4 V. The duty cycle is what is actively controlled to position the solenoid.

Stim Configuration

Simulated Pattern:

lab_motor_cell_c

Display Style:

Wheel

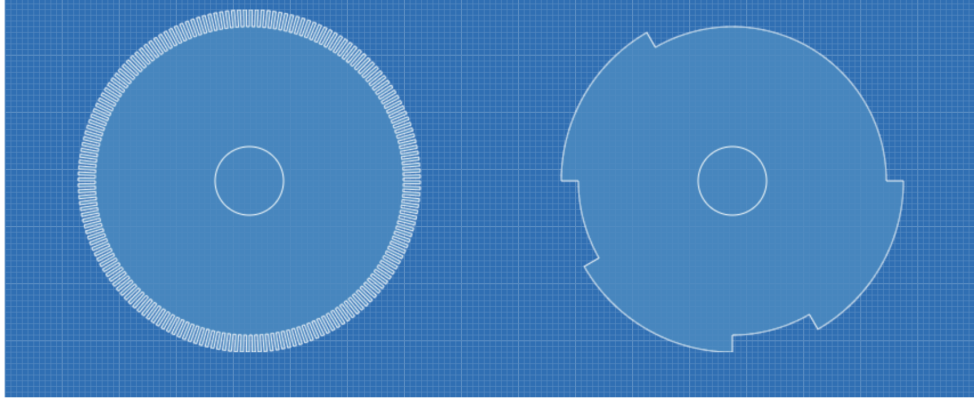


Figure 2.10: “Geometry” of research engine emulator trigger wheels with first camshaft.

The frequency depends on the solenoid and in what type of system it should be used in. Since a solenoid is an inductor, it takes some time to build up the magnetic field. The time constant τ , the time it takes to build up 63.2 % of the maximum magnetic field can be calculated using the coil’s resistance (R) and inductance (L):

$$\tau = \frac{L}{R}$$

Note that the 63.2 % threshold stems from the RL circuit step response equation $I(t) = \frac{V}{R} \left(1 - e^{-\frac{R}{L}t}\right)$. When $t = \tau$, the term $e^{-1} \approx 0.368$, leaving $1 - 0.368 = 0.632$.

$$\tau_{\text{vvt,solenoid}} = \frac{19 \cdot 10^{-3}}{8.5} = 2.23 \text{ ms}$$

Meaning that it takes 2.23 ms for the solenoid to build up 63.2 % of its magnetic field. Based on this, an appropriate frequency can be chosen $f = \frac{1}{s}$. For example if a relatively high frequency such as 1 kHz, the pulse would only be on for 0.5 ms at 50 % duty cycle and 1 ms at 100 % duty cycle before turning off again. That means that the magnetic field will never significantly be built up resulting in a solenoid that moves sluggishly or insufficiently. On the other hand, if a relatively low frequency such as 50 Hz was used for the PWM signal the pulse would be on for 20 ms meaning that a very strong magnetic field would be built up before turning off resulting in a solenoid movement that is oscillating. A frequency of 200 Hz, meaning that a full cycle takes 5 ms, was chosen as the first value to be tested with possible tuning when testing. At this frequency with 50% duty cycle, the pulse would be on for 2.5 ms, very close to the time constant. A lower frequency also has some benefits for this implementation. For this application, the solenoid is always filled with oil during operation, creating a damping effect, meaning that lower frequency can be

Stim Configuration

Simulated Pattern:

lab_motor_cell_c

Display Style:

Wheel

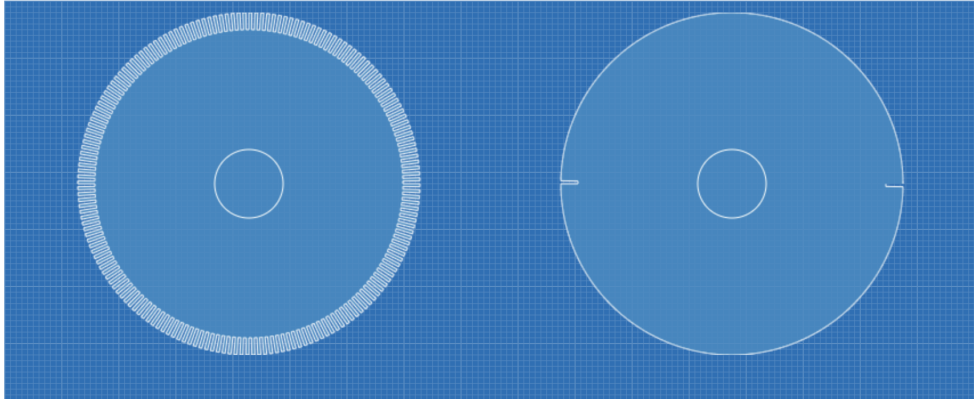


Figure 2.11: “Geometry” of research engine emulator trigger wheels with second camshaft.

used without noticeable oscillation. The low frequency also helps with sticking, especially at the end point. Another benefit is that the MOSFET spends less time between on and off, resulting in less heat generation.

3

Identified Usage Requirements

This chapter covers the engine lab setup which the VVT system is to be developed for. It includes a description of the engine as well as the monitoring, controls and measuring equipment. User needs have been extracted from meetings with the main user, where emphasis has been put on having a solution that is easy to use, understand, change, and repair. This aligns well with a stand-alone system made up of easily accessible standard components. It also calls for a clear description of all parts of the system and its development, mainly by this report and a manual.

3.1 User Questions

A set of user questions were formulated to extract user needs.

- Describe a typical experiment.
- What research is being conducted today?
- What research is being conducted tomorrow?
- Describe your dream lab setup?
- What is the best thing about the current setup?
- What is the worst thing about the current setup?

3.2 User Needs

Different needs can be extracted both from expressed needs as well as seeing the equipment in use:

- **Simplicity** — A solution which is simple is easy to use, alter, and repair. It is important that the user understands the solution and also that the solution is well documented. Lab equipment like this is often changed throughout the years and there is a significant value in being able to understand the solution when using it for the first time.
- **Fail-safe** — Because of the catastrophic failures that can occur in interference engines (meaning that the valves and piston can contact in case of for example cam belt failure) it is important that the solution in no way allows for interference between valve(s) and piston.
- **Stand-alone** — Related to simplicity. By having a separate measurement and control system for VVT instead of combining it into an engine control unit (ECU) we reduce unnecessary complexity. The user has also expressed the wish for separate physical control and monitoring of the VVT system, i.e.

3. Identified Usage Requirements

a separate box with a display and control “knobs” to be placed on the control desk.

- **Adaptability** — The camshaft and therefore the camshaft trigger pattern are subject to change as the lab engine is used for different tests. A need for either an automated sensing of the camshaft pattern or an easy way to edit the system was required.

4

Prototyping and Testing

This chapter describes the prototyping and testing performed from simple tests to the finished product.

4.1 Hardware Platform Pre-study

For the regulator to “keep up” with the motor some requirements were set up. Firstly the amount of edges that need to be read has to be identified. On the crankshaft there is an encoder which sends out 3600 pulses per revolution. On the camshafts there are trigger wheels with “teeth”, a hall effect sensor senses every time there is metal near it. With the current camshaft, three pulses per full cycle is generated for each camshaft. The max operating speed of the engine during testing is 4000 rpm. The regulator needs to be fast enough to read all of the edges and be deterministic to not miss any of the edges. The frequency of the crankshaft sensor pulses (f_{edge}) can be calculated using the engine speed (N) in RPM and the encoder resolution (Z) in pulses per revolution:

$$f_{\text{edge}} = \frac{N}{60} \cdot Z$$

$$f_{\text{edge}} = \frac{4000}{60} \cdot 3600 = 240\,000 \text{ Hz}$$

In addition to reading the crankshaft sensor, the regulator system also needs to be able to read both camshaft signals accurately. The camshaft spins at half the speed of the crankshaft and fewer teeth, in this case three teeth.

$$f_{\text{edge}} = \frac{4000}{2 \cdot 60} \cdot 3600 = 100 \text{ Hz}$$

It could be concluded that the system controlling the VVT, from a technical perspective, would need to be direct, high frequency and be able to produce PWM signals. A few systems initially were identified as possible solutions: Arduino, Raspberry Pi, Speeduino and aftermarket vehicle ECUs. Due to budget constraints, all possible solutions were to be examined theoretically before initiating physical testing. Ideally, physical testing would only be performed on a system that was deemed functional. However due to existing components possessed by the group, some initial testing could be performed on a few systems: Arduino R3/R4 and Raspberry Pi.

A few other solutions were ruled out instantly due to them not meeting requirements, for example cost or customization. A factory ECU would be a great way of controlling the VVT system as it was designed for it, however, a factory ECU does not allow for manual control of valve timing. The only possible way to do it is to replace the entire table of valve angles which depends on several factors such as engine speed and load. This would however require a flash of the ECU every time the valve angle was to be changed, taking several minutes. Using a factory ECU would also be expensive and provide unnecessary functionality that would not be used.

Using a Raspberry Pi was also ruled out because it is not deterministic as it uses an operating system instead of direct code.

Speeduino is an open-source project used for replicating an ECU with an Arduino Mega 2560. This is proven technology for the supported engines. Upon further investigation of Speeduino's VVT support, it was noticed that it only supports single tooth trigger wheel patterns on the camshaft. This is likely due to the slower speed of the Arduino which cannot identify and distinguish the different teeth at acceleration and deceleration of the motor. Another issue was that the Speeduino, similarly to a factory ECU, is a complete engine control system, meaning that cam phaser angle could not be chosen manually. At most some source code could be used as inspiration.

4.2 Phase 1: Initial Software and Arduino Testing

With all other solutions ruled out testing with already existing components initiated. This was done by running an engine emulator on an Arduino R3 which produced the square wave pattern produced by the crankshaft and both camshaft sensors. Regulating the system was done on the Arduino R4. With several iterations of code mixing AI assistance with source code from Speeduino a final evolution on this platform was achieved. As the engine emulator could not produce any phase shift of the camshafts the original position of the camshaft would have to be measured. Testing was performed both for sweeping speed between 1000 rpm and 6000 rpm and steady speed at 1000 rpm, while measuring the cam phase offset which in this case was supposed to be at 0° all the time.

The core logic behind the code was to:

- Read the crankshaft sensor to determine the rpm of the motor (to verify against the emulator) and finding top dead center of the motor using the two missing teeth on the trigger wheel. This is enough for controlling VVT while other functions in the engine require more information from the crankshaft sensor.
- Use the camshaft sensors for both knowing what cycle the engine is in and comparing the offset between the crankshaft and the camshaft.
- Integrate a PID-controller that would control the position of the solenoids depending on actual and desired phase shift.

Note that this code logic is the one used in the early stages of prototyping and has very little correlation with the final software. The first problem was that the camshaft teeth gaps would be difficult to notice as the “length” would be different at different speed and especially hard to predict during acceleration. Even after recording the exact pattern of the camshaft square wave pattern, the implementation remained unreliable. This was most likely due to slow processing which could not be solved without hardware changes. Then filtering of the camshaft signal was used to only use one of the camshaft teeth. That returned an actual usable result but the measured error was never at zero, ranging from a few hundredths of a degree to 0.3° . Additional methods like running the regulator in “sync mode” for a few rotations and fine tuning the latency between loops had no real performance benefit.

4.3 Phase 2: Teensy Microcontroller Testing

It was concluded in the first testing phase that an Arduino would not be a great fit for this project. This is due to several reasons. The first reason and the most fundamental to understand is the clock speed. Without going into too much detail the Arduino’s processor runs at 48 MHz, which was too slow for this application. The processor clock speed determines how many cycles of math can be completed in one second. When the Arduino senses a rising edge signal from the crankshaft it logs the time from when it last saw an edge, then the calculations for engine speed, offset and the PID regulator starts. If this calculation is not completed before the next tooth on the crankshaft arrives, it will continue doing the math inside the loop and note the next rising edge after it is done with the previous task. This causes the regulator to not know exactly where it is and was not acceptable for this project. This phenomenon is also caused by the way that an Arduino handles timers in software, the processor handles both timer interrupts and the calculations.

When the Arduino was ruled out, the search for a better suited microcontroller continued. Several platforms like STM32, ESP32 and Teensy were considered. ESP32, while running at fairly high clock speeds, was ruled out due to low determinism from the processor architecture and an operating system running in the background. These boards were more focused on internet connectivity and were not suitable for engine management.

The STM32 and Teensy platforms both have models with dedicated hardware for logging input captures. This means that even if the processor is busy, a separate system logs all timestamps of rising edges eliminating the drift seen in the Arduino. The Teensy platform stood out as its processor runs at 600 MHz, over 12 times faster than the Arduino while the most common boards from STM32 ran at similar speeds to the Arduino. There is a larger model of the STM32 platform that features the same Arm Cortex-M7 processor found in the Teensy 4.1 but no advantages were found with this system. The Teensy is smaller, even though space was not an issue, a smaller footprint was not deemed a negative. The main reason for choosing Teensy over the STM32 system, apart from size was the optimized libraries and similarities with Arduino programming which would speed up the prototyping phase.

The second test was conducted similarly to the first one with an Arduino R3 acting as an engine emulator. This test also incorporated some of the peripherals that would be needed for the final product like the two rotary encoders and display. This was done to start testing components to verify if they behaved as planned and that the controller could handle it both with enough pins in the board and from a performance view. One difference from the first test was that the Teensy runs natively on 3.3 V instead of the 5 V that was used in the previous Arduino R4. As the engine emulator outputs a 5 V signal three separate voltage dividers were used to bring down the signal voltage, according to:

$$V_{out} = V_{in} \cdot \frac{R_2}{R_1 + R_2}$$

$$3.33 \text{ V} = 5 \text{ V} \cdot \frac{R_2}{R_1 + R_2}$$

$$2R_1 \approx R_2.$$

2 k Ω and 1 k Ω resistors were used to achieve the correct voltage drop and to provide a good balance between minimizing current draw and maintaining signal integrity. Using a too low resistance would pull too much current from the sensor and using a too high resistance, the signal would start to be interfered with by the small capacitance in metal parts, softening the rising edges to undetectable smooth waves. A diagram of the parts and connections can be seen in Figure 4.1.

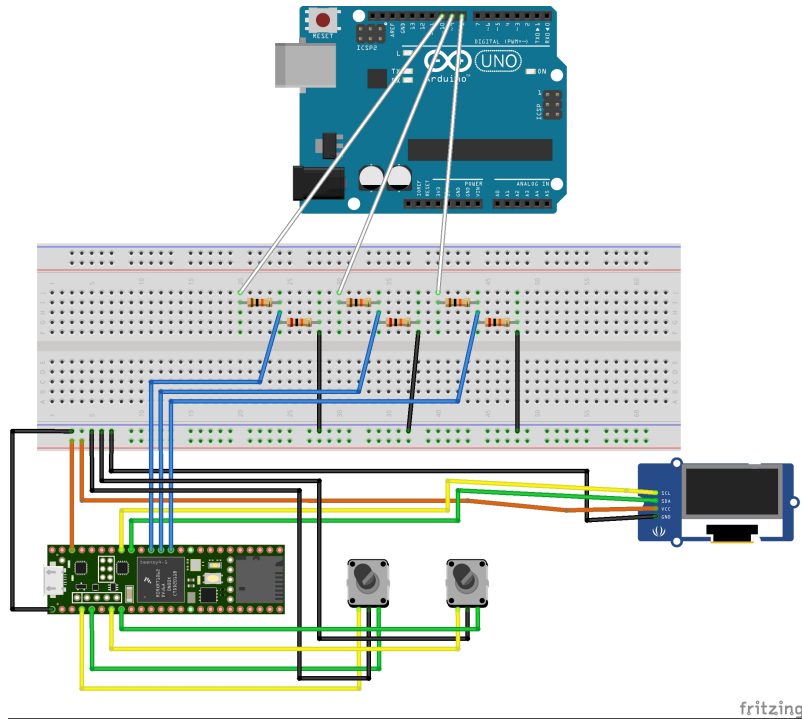


Figure 4.1: Diagram of the testing setup.

The result of this test was that the microcontroller behaved as needed and there was no lag or misalignment. The system saw minor fluctuations in the hundredth

of degrees, but it could not be concluded if this was due to the regulator or the emulator. Either way, the flickering was deemed within tolerance, especially with the dampening of the hydraulic system in a real engine.

The rotary encoders needed a filter to change the target offset reliably but other than that, all surrounding components also worked as intended. This meant that the next step was testing with a more realistic lab engine emulator, verifying that it would also work there and tuning the code for the different engine setup. A picture of the prototype can be seen in Figure 4.2.

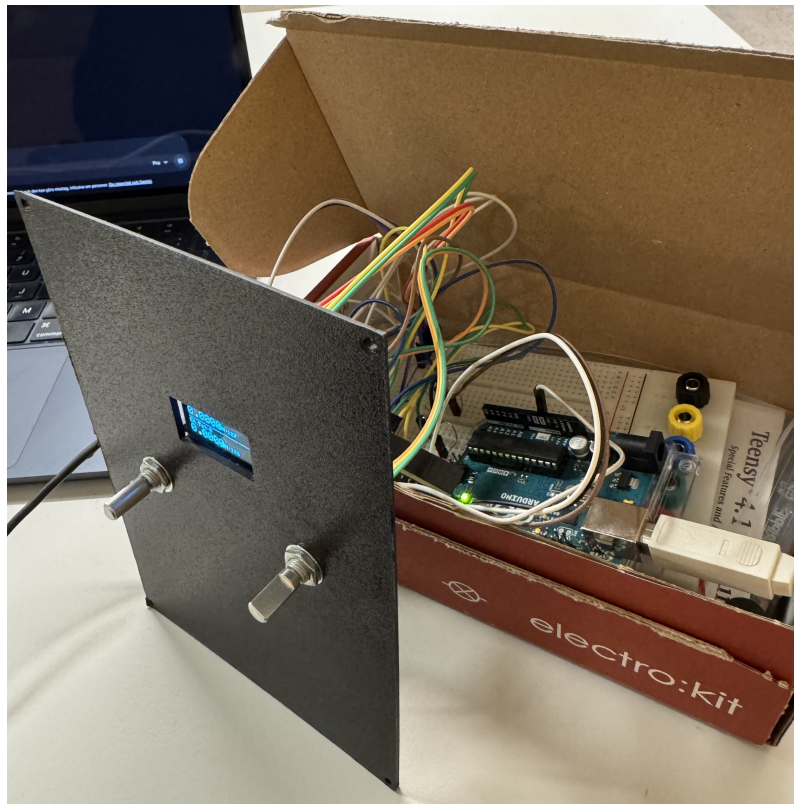


Figure 4.2: Early prototype.

4.4 Phase 3: Initial User Interface and Peripherals Testing

After a suitable microcontroller system was selected, other forms of testing was performed, relating more to the physical parts of the system. UX, form factor, connections and peripherals were tested.

An early model was designed to get feedback on and evaluate how the system could be designed, see Figure 4.3. From this prototype, the group in conjunction with the user concluded that the product needed to be bigger in several aspects, both physically to hold more components as well as the display and knobs. With this prototype the electronics were soldered instead of using a breadboard to get an early test of

how it would work and eliminate any bad connections or wires coming loose during testing. During this phase it was decided to split up the electronic boards to two sides, one housing the low voltage electronics and the controller and one side with 12V electronics and power supply for the other side. This meant that power could be drawn from the already existing power supplies and using a DC-DC converter to power the microcontroller.

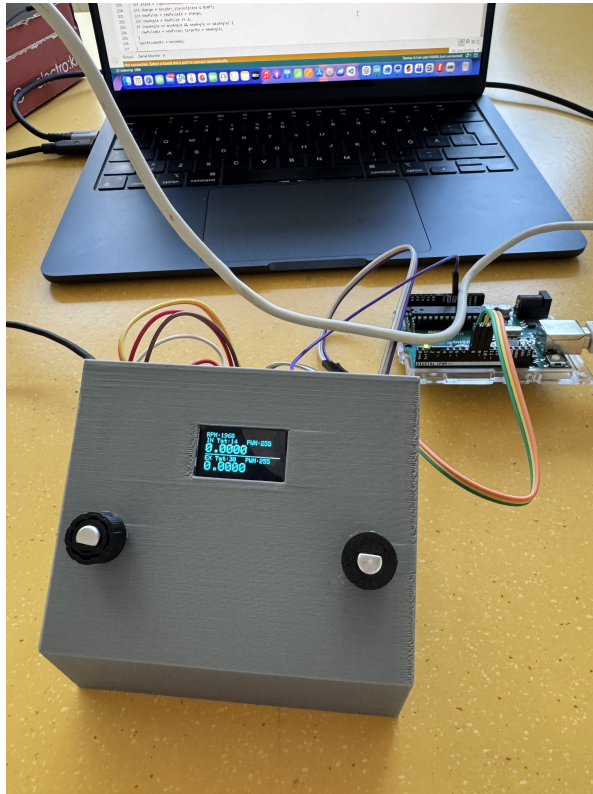


Figure 4.3: Second prototype.

4.5 Phase 4: Updated User Interface and Hardware

The second prototype was closer to a finished product. The casing had room for a larger screen, larger knobs, two separate matrix cards for different voltages and other peripherals like an on/off switch and a fan. With this prototype it was clear that cable routing internally required further testing and adding the inputs and outputs to the back of the box. See Figure 4.4.



Figure 4.4: Third prototype.

4.6 Phase 5: Development of Electronics for Standalone Use

It was decided to separate the electronics that operate at different voltages to two separate boards. One board would be powered with 5 V and the other with 12 V. The chosen microcontroller runs natively at 3.3 V but due to the switch to a larger display that runs on 5 V, it was decided for simplicity to only have two levels of voltage. This was made possible by the microcontroller having an onboard DC converter which outputs 3.3. The second board that runs the 12 V was “reinforced” with solid wires as the thin metal strips would not hold up to the higher current needed for the solenoids.

The solenoids are controlled using metal oxide field effect transistors (MOSFET). These components are included because the microcontroller cannot supply 12 V or handle the higher currents needed for the solenoids. The MOSFETs act like switches that can rapidly be switched on or off by a lower voltage, making it suitable for PWM regulation. The microcontroller will send a 3.3 V signal to the MOSFET which will then close the ground of the solenoid circuit. IRLZ44N was the chosen MOSFET as it requires low voltage to open and can handle up to 55 V @ 47 A, which gives headroom for controlling the solenoid. The MOSFET has three pins, one is for signal from the microcontroller, one for drain (the part that closes or completes the circuit when activated) and the third is for ground. When using MOSFETs that requires such low voltages to open a high resistance resistor is added between the signal pin and ground to make sure the switch does not close from any electrical noise. A solenoid uses coils to move a pin. When this pin moves back and forth it creates a current which can damage the microcontroller. That is why a flyback diode was also added between the drain and positive rail of the board, which is the same as

adding it between the positive and negative lead of the solenoid. The diode makes sure that no current can be pushed out to the other components. A schematic of the electronics during this phase can be seen Appendix A.

Some potential improvement areas were discovered from the previous prototype:

- Investigate size for a balance between size and in housing all the components.
- Flip the screen to allow the box to be slight slimmer and have more easy access to pins.
- New rotary encoders due to durability issues with previous ones and alter the mounting.
- Figure out cable management throughout the box.
- Add interface on the back for on/off button, fuse, inputs and outputs.

5

Result

This chapter covers the result of this project, which includes a control box with electronics, electrical installation, and the control software developed.

5.1 Final Product

The “final product” is the final 3D-printed enclosure. Several modifications were done to the product during this phase after testing. It is referred to as the final product because all components had to be permanently assembled for final testing. For example connectors on the rear of the casing are not easily removable, see Figure 5.5. In this part of the development focus was put on wiring and finalizing the design. The 3D-printed box was modified with anchor points for cable ties. A fan enclosure was added to reduce the possibility for a wire going into the fan. This enclosure also acted like a slide in mount for the fan as the mounting holes for the fan were straight holes. Previously, 3D-printed pins were used as friction mounts and screws were used to secure the fan in place. This solution would require both washers as well as a larger head screw, taking up space in the box. Changing the way the fan was mounted also meant changing the rails for mounting the lid as it would otherwise create a gap at the top. In the previous phase the lid was mounted in between the sides of the box and with this change it was placed fully above the walls of the box. All DuPont cables have been replaced with proper contacts or permanent soldered joints.

Some modifications were done to the electronics. A 2-channel gate driver was added between the microcontroller PWM signal and the MOSFETs. The purpose of this is to send out a more powerful signal, closing the MOSFET switch faster. Another modification was done to the 12 V board, instead of reinforcing the busses with extra wire, all components were soldered directly to a wire, using the card more as an organizer instead of carrying any current. A capacitor was added between the main positive and negative leads coming into the box which would guarantee continuous power when the solenoids start pulsing. Lastly a hole was cut out of the existing prototype to place a panel mount USB-C port, for easy access to edit code without opening the box.

During testing some modifications needed to be made. A resistor (pull up) between positive lead and camshaft sensor signal needed to be added to be able to

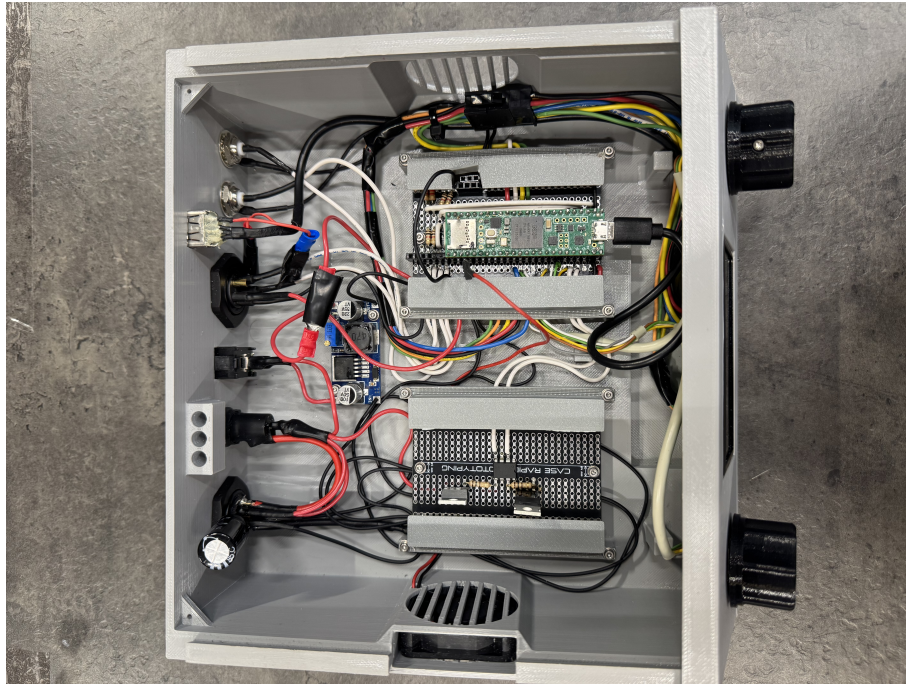


Figure 5.1: Inside of box.

read the signal. Using a pull up resistor between 3.3 V meant that the voltage divider for the camshaft could be removed and instead the left power rail on the matrix card was powered from the Teensy 3.3 V pin.



Figure 5.2: 3D-render of CAD model with AI generated display contents.

5.2 Components

In this chapter the various components used in the final product will be described and what purpose it serves. BOM:

- 1 Teensy 4.1 microcontroller
- 2 Logic level N-channel MOSFETS
- 1 2-Channel gate driver
- 1 LCD 128x64
- 2 Rotary encoders with detents and push button
- 1 LM2596 buck DC-DC converter
- 1 PC fan 5 V 45x45mm
- 2 Coaxial BNC panel mount female connectors
- 1 SY21 5-pin connector (male & female)
- 1 SY21 7-pin connector (male & female)
- 1 Switch on/off
- 1 Glass tube fuse holder
- 14 Machine screws M2 8mm
- 2 Machine screws M2 10mm
- 14 Washers 2.5 mm
- 4 $1k\Omega$ resistors
- 2 $2k\Omega$ resistors
- 2 1N4007 diode

5.2.1 MOSFET

A MOSFET (Metal Oxide Semiconductor Field Effect Transistor) is a very fast and efficient type of transistor. It could be described as a solid state relay capable of high-frequency switching. A transistor is used to control a high current signal with a smaller current signal. For example, the pins on the Teensy microcontroller can supply 3.3V with less than 100mA. The automotive solenoid used to control the camshaft phaser runs on 12V at several amps. So instead of running the solenoid directly of the Teensy, it sends a PWM signal to the MOSFET that then controls the solenoid. There are three pins on a MOSFET: gate, drain and source. For this product a N-channel (normally off) MOSFET is used with low side switch (controlling the ground). This means that the positive lead on the solenoid is always connected and the circuit is closed when power is sent to the MOSFET. The negative lead of the solenoid goes to the middle pin (drain). The left pin (gate) when looking at it from the side with text on is connected to the signal from the microcontroller. The right pin (source) is connected to common ground.

5.2.2 Gate Driver

A MOSFET also behaves like a small capacitor, meaning it requires loading up some charge to be fully closed. This means that when a MOSFET is driven by a

microcontroller with a small power output it remains partially closed for a short period. This is acceptable when using it for low-power applications or for slow switching, for example controlling a light with a push button. However, when using high current and fast switching like the 12 V high-frequency PWM signal for the solenoid, the MOSFET heats up very quickly and outputs a degraded signal. A gate driver solves this problem by amplifying the signal to almost instantly close the MOSFET.

5.2.3 DC-DC Converter

A DC to DC converter converts DC voltage to a higher or lower voltage. They are also known as a buck or boost depending on if they increase or lower the voltage. For this application 12 V is supplied in the engine room as it is needed for most automotive electrical systems. Several of the components used for this product run at lower voltages: the Teensy runs natively at 3.3 V while the display and fan require 5 V. Utilizing a DC-DC converter means that no extra power supply is required. It was decided to set the DC-DC converter to take the 13.8 V down to 5 V as the microcontroller can step down the voltage to 3.3 V internally and supply it. 3.3 V was only needed as for the pull up resistor for the hall sensor.

5.2.4 Diode

A solenoid is simply put a spool that creates a magnetic field that moves a shaft. The solenoid used for this application is spring loaded to keep it at its base position when it is switched off and allows the solenoid to be held at a steady position. The solenoid is driven using PWM meaning that the electricity is turned off for a fraction of a second. During this time the force from the spring pushes the shaft backwards and a high voltage current is generated by the spool and the polarity is switched. To counteract the current going back up the system and potentially damaging the rest of the electronics a flyback diode is used. This diode acts like a one way valve for electricity. The diode is placed between the positive and negative lead of the solenoid meaning that the current will be looped around in the solenoid producing a small amount of heat.

5.2.5 Logic Converter and Voltage Divider

The automotive sensors used for the crankshaft and camshaft require 5 V to operate and also output 5 V signals. The Teensy microcontroller used to regulate the system runs natively at 3.3 V meaning that the 5 V signal from the sensors would damage it. This is one of the trade offs that had to be made with choosing this microcontroller. An Arduino for example can handle the 5 V signal but was too slow and lacked dedicated hardware timers. To solve this initially a simple voltage divider using two resistors was used, basically grounding out part of the signal to reduce the voltage.

Another solution is using a logic converter. This circuit uses MOSFETs to raise or lower the signal voltage. This component has the advantage of completely separating the signals meaning that if a voltage spike occurred it could not damage

any of the other components. The logic converter was tested, however looking at the output signal it was noticed that it created shark fin shaped curves that would heavily reduce the reliability of the system it was decided to go back to using voltage dividers, see Figure 5.3. It was also possible that the logic converter got damaged during testing and time restraints also had an impact on the choice to go back to voltage dividers.

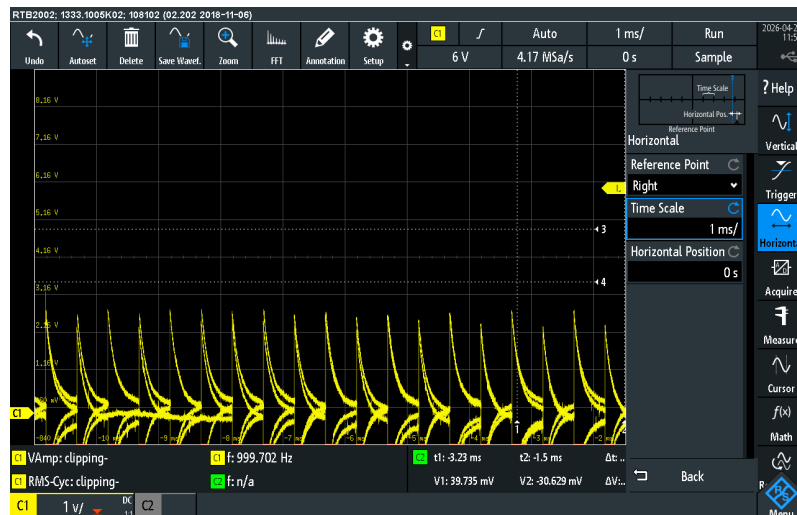


Figure 5.3: Crankshaft signal using logic converter.

5.2.6 Rotary Encoder

The rotary encoder used in the project is a five pin encoder which adds a button feature over a three pin encoder. The left and right pin on the side with three pins is connected to digital pins on the microcontroller with ground on the middle pin. This side controls the rotation output. On the other side with two pins, one is connected to a digital pin and one to ground. These pins are interchangeable on the encoder side. In Figure 5.4, the internals of a rotary encoder can be seen. Starting from the left, there is the metal casing, then the physical knob that is turned. The copper colored bracket is the detents spring which generates the tactile clicks when the knob rotated. Component 4 and 5 from the left is the signal part of the rotation, a small metal arm brushes over the pads on component 5 creating a quadrature signal. This signal can then be interpreted with an encoder library in software to identify one detent left or right. The right most green part is the push button circuit, which is simply a metal dome that connects a circuit when pressed.

The reason for using a rotary encoder for this projects is to input the target cam phaser angle. Compared to other rotary knobs like a potentiometer, there are version of encoders that don't have any physical end points, clear indents for each step and it makes it possible to set the value to 0° via software which is a must during startup.



Figure 5.4: Rotary encoder internals.

5.2.7 Liquid Crystal Display (LCD)

An LCD was added to display the relevant information such as target and actual value of the camshaft offset as well as engine speed and PWM signal duty cycle for verification. A smaller organic light emitting diode (OLED) display was initially tested but it was too small to easily read the numbers and there was also a risk of burn in when the system is turned on for long periods. The display uses two sets (positive and negative) of leads for power, one for the backlight and one for the LCD panel. On the back of the display there is a potentiometer where contrast can be set, this is just required to do once. There is an extra ground that just shorts a circuit to run in serial peripheral interface (SPI) mode. The microcontroller sends out 8-bit packages of information to be displayed on the screen.

5.2.8 Capacitor

A capacitor was added between the main incoming positive and negative leads. This was added to provide a steady level of current when the solenoids, which take fairly high current, are switched on. In the worst case the voltage would drop so low when the solenoids are turned on that the supply voltage going to the microcontroller would be so low that it would shut off.

5.2.9 Fuses, Cable Dimensioning and Routing

A main fuse was placed just after the main positive lead coming into the control box, for safety. With the calculated current of approximately 1.5 A per solenoid, a 4 A slow fuse was used.

All DuPont cables were removed in the box and replaced with more permanent soldered wires. Some DuPont headers were left for running the simulator, but this

would not impact the performance when running the system in the lab. Most of the wires used inside the control box were aluminum 20 AWG cable, roughly 0.5 mm^2 . For the 3.3 V and 5 V cables it was beneficial as the wire had a very similar diameter to the matrix card holes. For the solenoids it was decided to use the same 0.5 mm^2 sized cable as cable supply was limited. 0.5 mm^2 cable is generally rated for 5 A in continuous use, however our cable was made out of aluminum which has a higher resistance meaning that the current would be fairly close to the rated current. It would be preferable to use slightly thicker wires but it could not be too thick as that would interfere with the tight spacing of the MOSFET pins. As the distance was so short within the box and provided active cooling it was deemed safe to use 0.5 mm^2 cable even for the solenoids. The main positive and negative lead inside the box was a 1.5 mm^2 copper wire that were then split up the thinner cable going to various places in the box. The cables going out from the box were substantially thicker to minimize voltage drop, which was required as they span over 10 m.

5.2.10 Noise, Shielding, Signal Treatment, Interference

The two cables going from the box into the engine room were shielded, with one end of each shield connected to engine ground, both to protect the signal going to the box but also to protect the extensive amount of other cables that ran nearby.

5.3 Installation

Practical installation of the system entailed some challenges and design inquiries as cables needed to pass through the wall between the test cell and the control room. To minimize the space taken up on the existing cable ladder and holes, the amount of cables were kept to a minimum, only running one signal cable and one power cable between the test cell and control room. Table 5.1 shows the cables used in the installation. The setup includes a control box containing the Teensy microcon-

Table 5.1: Cable list

A	Cable	B
Control box	4x0.34	Junction box
Control box	7G1	Junction box
13.8 V supply	2x1.5	Junction box
Exhaust cam sensor	3x0.7	Junction box
Intake cam sensor	3x0.7	Junction box
Exhaust VVT solenoid	2x1.5	Junction box
Intake VVT solenoid	2x1.5	Junction box
Engine ground	1x1.5 (Y/G)	Junction box

troller, a LCD for output, and two rotary encoders for user input is placed on the desk of the control room.

12 V power was available inside the lab cell room and was used to power the control

system as well as the VVT solenoids. It was decided that the MOSEFETs should be placed in the control box, as even though the square wave form of the PWM signals are concerning regarding noise, shielded cables and limited currents should mitigate this. One alternative would have been to place the MOSFETs in the control room as to not run power cables alongside signal cables. Another alternative could be to have the microcontroller placed in the lab cell, instead routing cables for the display and encoders between the cell and control room. This is a trade-off between running power (relatively low at 10 A), next to signals (relatively unsensitive as they are digital), running a few 5 V signals for a longer distance (<10 m), and running more 5 V signals for a longer distance (<10 m), with a goal of tolerable loss and noise, and routing minimal cable area through the wall.

CDM and trigg signals from the crank shaft rotary encoder were taken from a control cabinet in the control room.

5.3.1 Connectors

Connectors were used on the back of the box to make it removable and for easier installation, see Figure 5.5. With the fairly high current draw from the solenoids, the connectors were dimensioned to not be the bottleneck. 7-pin and 5-pin 21 mm industrial contacts were used for the 6-lead power cable and the 4-lead camshaft signal cable. The 7-pin connector which had a lower amperage rating than the 5-pin connector, was rated for 15 A. Smaller connectors were on the boundary of the system amperage output. A smaller connector could have been used for the camshaft signal cable but the larger connector offered no drawbacks either with price or functionality. The different pin count of the connectors also made it impossible to plug in the wrong connector.



Figure 5.5: Rear of box with connectors.

For the crankshaft signals (CDM and trigg) standard BNC coaxial connectors were used. It was important here to use connectors with matching impedance at 50 Ω . Most coaxial panel mount connectors use the casing of the connector for ground and are mounted to a metal surface where only one wire needs to go from the chassis to the sensor ground. In this application where the system casing is made out of plastic an extra ground ring connected to the connector casing, needs to be installed. This ring can then be connected to common ground.

On the engine side automotive contacts were used to connect to the engine sensors and solenoids. A kit with pins and contacts to attach to a cable were ordered and installed, see Figure 5.6



Figure 5.6: Camshaft sensor contact.

5.3.2 Junction Box

To lessen the amount of cables going through the wall, a junction box was be used. It was mounted on a cable ladder above the engine. In the junction box, power is split up, and cables from the individual components connected to a multi-conductor cable. See Figure 5.7. Screw down mini-DIN terminals were used inside a electrical box. Two M16 cable glands on the top of the box and four on the bottom were drilled out and mounted. This would provide strain relief on the cables. One larger M20 cable gland had to be used on the top as the power cable was too large to fit in the smaller model. It was also decided to place the diodes that would protect the MOSFETs inside the junction box to keep the reverse pulses from the solenoids as close to the source as possible. A schematic of the junction box connectors can be seen in Appendix B.

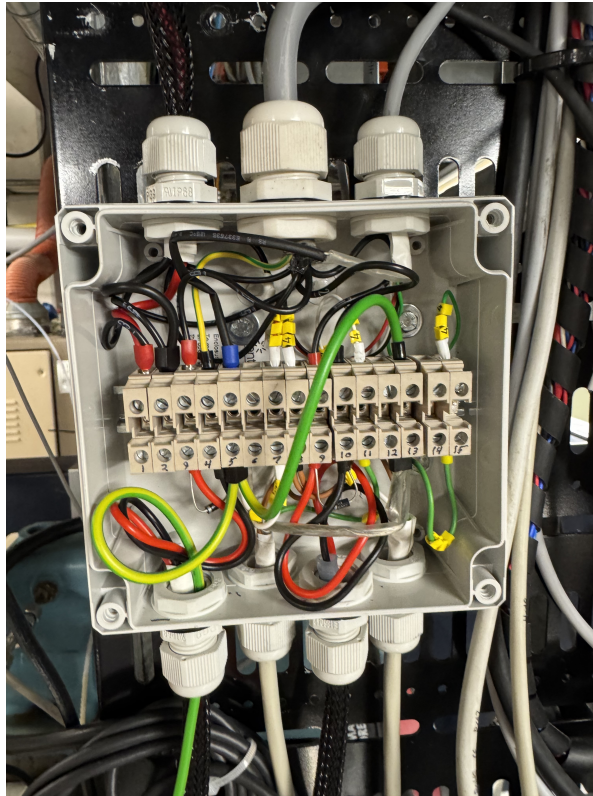


Figure 5.7: Inside of junction box.

5.4 Test Run and Signal Capture

In order to verify function of the system in its working environment, input signals were recorded after wiring was completed. This meant that any noise or other issues that would be present during operation of the final product could be identified. Exhaust and intake camshaft sensors were connected to the control box, first through two separate 3x0.7 cables, via the junction box and finally through a single 4x0.34 cable. From the control box, these were then connected to the analog channels of a Analog Discovery 2 card. CDM and trigg signals were connected from the cabinet in the control room to the digital channels of the same Analog Discovery 2 card. During testing, it was noted that the camshaft sensors were not giving of any signals. It was concluded that the camshaft sensors were of open-collector type as connecting one 1.8 k Ω pull up resistor in parallel between each sensors 5 V and signal connection resolved this issue. These four signals were studied using WaveForms 3 (Digilent Inc., 2025).

5.5 Software

In this section the code for the final product will be displayed and explained. The code is subject to minor changes during testing and final tuning. A simplified flow chart of the code logic can be seen in Figure 5.8. The most important part about the

code is identifying where the crank and the relative positions of the camshafts are. As stated in previous chapter the crankshaft encoder sends out 3600 CDM pulses per revolution and one trigg pulse. The trigg pulse is used for syncing (setting the counter to zero). This is actually done every other trigg signal as the trigg pulse is sent out on every rotation not full cycle. For this system we only need to know the relative position of the camshafts compared to the crankshaft, so in which phase the engine is in does not matter. After two trigg signals, the code counts each CDM pulse which is 0.1° rotation of the engine. The system also has a timer between CDM pulses. When a camshaft pulse is sent identified, the exact crankshaft position is “saved” for example 1128 CDM pulses + interpolating with the time since last CDM pulse. The engine speed is calculated between all CDM pulses which means that the interpolation is very accurate, very little acceleration can occur during 0.1° crank revolution.

When the system is first switched on the PWM signal duty cycle is 0%, meaning that the camshafts are in a fixed, end position. Then the camshaft teeth positions are recorded in a matrix with the same logic as above.

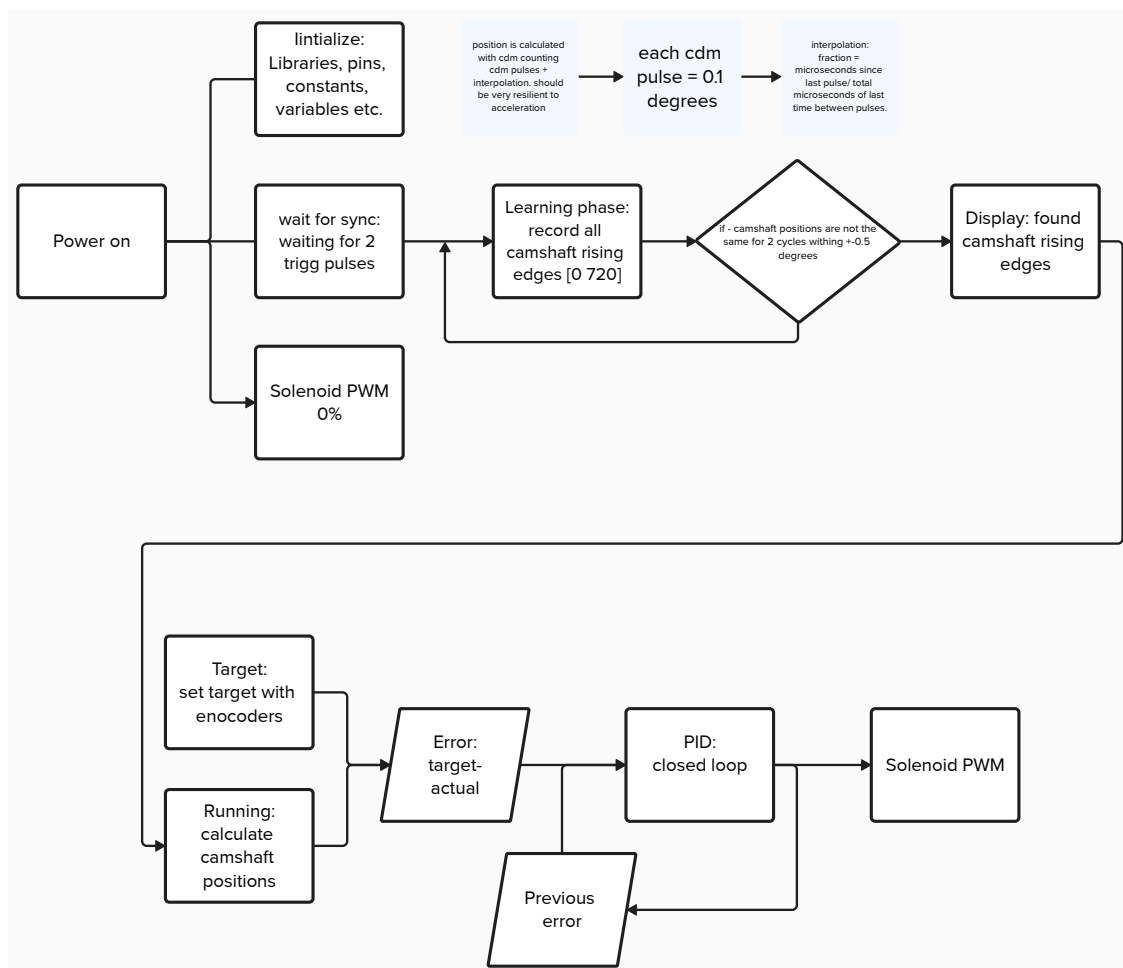


Figure 5.8: Software flowchart.

5.5.1 Initialization

In the first part of the code, the used libraries are included. After that a section with easy to edit variables are placed to fine tune the system or to make necessary changes when changing physical components. Most of these are self explanatory. Cam debounce is a filter meant to disregard pulses that are induced by wires being close to each other. Phase filter weight is how much of new data used for the regulator. In a dynamic system, belt tension and valve springs can create high-frequency jitter which will cause the solenoid to chase small variations that are actually resonance. It was found that more noise was seen on the exhaust side, likely due to the belt tensioner being on that side. the downside of amplifying this filter (lower value) is that the system will be slower to react. Max phase window is set as the camshaft phasers can not physically move more than 24° , ignoring any pulses that are not actually possible. Then the pins for inputs and outputs on the microcontroller are defined, if a cable is physically moved on the matrix card these need to be edited. In engine definitions, constants of the engine sensors are defined. If the amount of CDM pulses sent from the AVL encoder are ever edited, this needs to change. After that, until interrupt service routines, variables used in the code are defined.

```
1 #include <Arduino.h>
2 #include <SPI.h>
3 #include <U8g2lib.h>
4 #include <Encoder.h>
5
6 // =====
7 // VVT & PID CONFIGURATION (EASY EDIT BLOCK)
8 // =====
9 // 1. PID Tuning Gains
10 float Kp = 4.0; //Proportional
11 float Ki = 1.5; //Integral
12 float Kd = 1.0; //Derivating, to not overshoot target (sensitive to
    signal noise)
13
14 // 2. PID Output Limits
15 const float PID_BASE_PWM = 470.0; // initial PWM value at start,
    PID regulator will find exact baseline for steady state
16 const float PID_INTEGRAL_LIMIT = 250.0; //previous error limit
17 const int PWM_MIN_LIMIT = 0;
18 const int PWM_MAX_LIMIT = 1000;
19
20 // 3. Encoder Target Phase Limits, Set within physical cam phaser
    limits in crank angle degrees.
21 const float TARGET_IN_PHASE_MIN = -1.0;
22 const float TARGET_IN_PHASE_MAX = 24.0;
23 const float TARGET_EX_PHASE_MIN = -24.0;
24 const float TARGET_EX_PHASE_MAX = 1.0;
25
26 // 4. Sensor Filtering & Safeguards
27 const uint32_t CAM_DEBOUNCE_US = 50; // Ignore induced
    crosstalk within 50 microseconds, 0 = Off
28 const float PHASE_FILTER_WEIGHT_IN = 0.05; // 5% smoothing for
```



```

    Intake (Stable)
29 const float PHASE_FILTER_WEIGHT_EX = 0.01; // 1% aggressive
    smoothing for Exhaust (Jittery)
30 const float VALIDATION_TOLERANCE = 1.0; // Second cycle
    validation tolerance in crank angle degrees
31
32 // Dynamic Validation Window: Automatically scales to your highest
    target + 2 degrees safety buffer.
33 const float MAX_PHASE_WINDOW = max(TARGET_IN_PHASE_MAX,
    TARGET_EX_PHASE_MAX) + 2.0;
34
35 // 5. Dither Settings
36 const bool ENABLE_DITHER = true; // true = ON, false = OFF
37 const int DITHER_AMPLITUDE = 20; // +/- PWM wobble amount (
    on the 0-1000 scale)
38 const uint32_t DITHER_INTERVAL_MS = 10; // Milliseconds between
    wobbles (10ms = 50Hz dither)
39
40 // 6. Hardware Reversal & Pin Swapping
41 // If your intake and exhaust sensor plugs are physically swapped,
    swap these pins:
42 const int CAM_IN_PIN = 4; // Default Intake Cam (Rising)
43 const int CAM_EX_PIN = 7; // Default Exhaust Cam (Rising)
44
45 // If a camshaft gear is flipped (working in reverse), change the
    direction multiplier from 1.0 to -1.0
46 // 1.0 = Normal (Positive error increases PWM), -1.0 = Reversed (
    Positive error decreases PWM)
47 const float PID_DIR_IN = 1.0;
48 const float PID_DIR_EX = -1.0;
49
50 // =====
51 // PIN DEFINITIONS (Teensy 4.1)
52 // =====
53 const int CRANK_PIN = 0; // CDM stream (Rising)
54 const int TRIGG_PIN = 2; // AVL 1 pulse/rev (Inverted/Falling)
55
56 const int SOLENOID_IN_PIN = 23; //PWM output to gate-driver channel
    1
57 const int SOLENOID_EX_PIN = 22; //PWM output to gate-driver channel
    2
58
59 // --- FULL BUFFER SOFTWARE SPI FIX ---
60 U8G2_ST7920_128X64_F_SW_SPI u8g2(U8G2_R0, 13, 11, 10, 12);
61
62 // --- ENCODER PINS ---
63 const int ENC_EX_A = 14;
64 const int ENC_EX_B = 15;
65 const int ENC_EX_BTN = 18;
66
67 const int ENC_IN_A = 20;
68 const int ENC_IN_B = 19;
69 const int ENC_IN_BTN = 21;
70
71 Encoder encIn(ENC_IN_A, ENC_IN_B);
72 Encoder encEx(ENC_EX_A, ENC_EX_B);

```

```
73
74 // =====
75 // ENGINE DEFINITIONS
76 // =====
77 #define CRANK_TEETH 3600
78 #define DEGREES_PER_TOOTH (360.0 / CRANK_TEETH)
79 #define TOTAL_CYCLE_TEETH (CRANK_TEETH * 2)
80
81 // =====
82 // TIMING & STATE VARIABLES
83 // =====
84 volatile uint32_t lastCrankCycles = 0;
85 volatile uint32_t toothIntervalCycles = 0;
86 volatile int crankToothCount = 0;
87
88 volatile uint32_t lastCrankTime = 0;
89 volatile uint32_t lastTriggTime = 0;
90 volatile uint32_t lastCamInTime = 0;
91 volatile uint32_t lastCamExTime = 0;
92
93 volatile int triggPulseCount = 0;
94
95 enum EngineState { // 3 regulator phases, affects void loop
96     execution
97     WAITING_FOR_SYNC,
98     LEARNING_PHASE,
99     RUNNING
100 };
101 volatile EngineState currentState = WAITING_FOR_SYNC;
102
103 bool showLearnedDisplay = false;
104
105 // --- Dither Variables ---
106 uint32_t lastDitherTime = 0;
107 int ditherOffset = 0;
108 bool ditherState = false;
109
110 // =====
111 // DYNAMIC VVT PHASE VARIABLES (DUAL CAM)
112 // =====
113 const int MAX_CAM_TEETH = 10;
114
115 // Intake
116 volatile float camInMatrix[MAX_CAM_TEETH];
117 volatile int camInToothCount = 0;
118 volatile int activeCamInTeethCount = 0;
119 volatile int tempCamInCount = 0;
120 volatile float actualInPhase = 0.0;
121 volatile bool inLocked = false;
122
123 // Exhaust
124 volatile float camExMatrix[MAX_CAM_TEETH];
125 volatile int camExToothCount = 0;
126 volatile int activeCamExTeethCount = 0;
127 volatile int tempCamExCount = 0;
128 volatile float actualExPhase = 0.0;
```

```

128 volatile bool exLocked = false;
129
130 // Validation & Targets
131 volatile bool isVerifying = false;
132 float targetInPhase = 0.0;
133 float targetExPhase = 0.0;
134 long lastEncInPos = 0;
135 long lastEncExPos = 0;
136
137 bool inFineMode = false;
138 bool exFineMode = false;
139 int lastInBtnState = HIGH;
140 int lastExBtnState = HIGH;
141
142 // =====
143 // PID LOOP VARIABLES
144 // =====
145 uint32_t lastPidTime = 0;
146 const uint32_t PID_INTERVAL_MS = 20;
147
148 float integralIn = 0.0, prevErrorIn = 0.0;
149 float integralEx = 0.0, prevErrorEx = 0.0;
150 int pwmIn = 0;
151 int pwmEx = 0;

```

5.5.2 Functions

The interrupt service routine functions are ran when a pulse is identified. The first function is crankISR. This part keeps track of the CDM pulses, counting them and checking the speed. The speed is calculated by measuring the time between CDM pulses. When switching from the simulator that used 360 pulses to the real engine with 3600 pulses per revolution the timer was switched from using microseconds to using the clock speed of the processor instead. When running the engine at 2000 rpm there are roughly 8 μ s between each pulse, so there could be instances where rounding errors could affect the interpolation. This is a bit overkill as the mechanical noise is of much greater magnitude than the microsecond rounding errors.

The second function triggISR is for the trigg pulses. Firstly it applies the filters. If a trigg pulse arrives within the set cam debounce time it is ignored. And if the motor has stalled it does not run the rest of the code in the function. At the top and bottom of the function there are resets for the trigg pulse counter. If the two trigger pulses have been identified it resets it back to zero to start a new engine cycle. At the bottom there is a line that sets the CDM pulse count to 3600 at halfway of a cycle as a safety feature in case a CDM pulse is missed. What this function block does depends on what state the regulator is in (waiting for sync, learning or running). For example if the current state is waiting for sync, this function being run means that it has found sync and should go to the next phase which is the learning phase and resets variables such as intake camshaft tooth count.

The camISR function are two completely identical but separate functions to handle

sensor data from both intake and exhaust camshafts. It uses the same filters as the trigg function. After that it calculates the exact angle relative to the crankshaft it identified the camshaft rising edge. It does that by counting the amount of CDM pulses and interpolation. Then similarly to the trigg function, what this function does depends on the state of the regulator. If the regulator is in its learning phase the camshaft position are stored in a matrix. After the learning phase a validation cycle is performed where if camshaft positions are found outside the allowed tolerance, the state is set to waiting for sync again. When the camshaft trigger wheel teeth positions are learned and verified it uses the camshaft position to calculate an offset between the learned position and the actual position. When a pulse triggers the function and the exact angle is calculated the system has no way of knowing which teeth that it just recognized. Using nearest match was deemed the most robust way of identifying teeth. If the camshaft has rotated 12° camshaft degrees relative to the crankshaft it is impossible to mistake that pulse to another camshaft teeth. It is important to remember that this way of identifying pulses will not work with trigger wheels with very narrow and close spaced teeth. It is possible to use a counter in these edge cases but that solution would be very sensitive to signal noise.

```
1 // =====
2 // INTERRUPT SERVICE ROUTINES (ISRs)
3 // =====
4
5 void crankISR() {
6     uint32_t currentCycles = ARM_DWT_CYCCNT; //Using processor clock
7     speed for meassuring time 600MHz
8     uint32_t currentTime = micros(); // Timer used for stall
9     detection
10    uint32_t cycleDiff = currentCycles - lastCrankCycles; //time
11    between CDM pulses
12
13    crankToothCount++; // Counter for CDM pulses [0 7200]
14    if (crankToothCount >= TOTAL_CYCLE_TEETH) { // Reset counter at
15        7200
16        crankToothCount = 0;
17    }
18
19    if (lastCrankCycles != 0) {
20        toothIntervalCycles = cycleDiff; //Ignores very first CDM pulse
21    }
22
23    lastCrankCycles = currentCycles;
24    lastCrankTime = currentTime;
25 }
26
27 void triggISR() {
28     uint32_t currentTime = micros();
29
30     if (currentTime - lastTriggTime < CAM_DEBOUNCE_US) return; //
31     Remove induced crosstalk pulses
32     if (lastCrankTime == 0 || (currentTime - lastCrankTime > 200000))
33         return; // Stall detection
34
35     triggPulseCount++; // Reset every 2 trigg pulses (full cycle)
```

```

30  if (triggPulseCount >= 2) {
31      triggPulseCount = 0;
32      crankToothCount = 0;
33
34      if (currentState == WAITING_FOR_SYNC) {
35          currentState = LEARNING_PHASE;
36          camInToothCount = 0;
37          camExToothCount = 0;
38          activeCamInTeethCount = 0;
39          activeCamExTeethCount = 0;
40          isVerifying = false;
41          inLocked = false;
42          exLocked = false;
43          Serial.println("--- 720    SYNC ESTABLISHED! MAPPING CYCLE 1
          ---");
44      }
45      else if (currentState == LEARNING_PHASE) {
46          if (!isVerifying) {
47              if (camInToothCount > 0 || camExToothCount > 0) {
48                  activeCamInTeethCount = camInToothCount;
49                  activeCamExTeethCount = camExToothCount;
50                  isVerifying = true;
51                  tempCamInCount = 0;
52                  tempCamExCount = 0;
53                  Serial.println("--- CYCLE 1 MAPPED. VERIFYING IN CYCLE
          2... ---");
54              } else {
55                  Serial.println("--- CYCLE 1 TIMEOUT: No teeth detected.
          Retrying... ---");
56                  camInToothCount = 0;
57                  camExToothCount = 0;
58              }
59          }
60          else {
61              if (tempCamInCount == activeCamInTeethCount &&
          tempCamExCount == activeCamExTeethCount) {
62                  currentState = RUNNING;
63                  showLearnedDisplay = true; // Halts output and waits for
          user confirmation
64                  Serial.println("--- SAFEGUARD PASSED! TARGETS VERIFIED &
          LOCKED ---");
65              } else {
66                  currentState = WAITING_FOR_SYNC;
67                  triggPulseCount = 0;
68                  Serial.println("--- SAFEGUARD FAIL: Tooth count mismatch!
          Restarting sync ---");
69              }
70          }
71      }
72  } else {
73      crankToothCount = CRANK_TEETH; // Remove possible cumulative
          error, Set crank tooth counter to 3600 at "second trigg
          pulse"
74  }
75
76  lastTriggTime = currentTime;

```

```

77 }
78
79 void camInISR() {
80     uint32_t currentCycles = ARM_DWT_CYCCNT;
81     uint32_t currentTime = micros();
82     if (lastCrankCycles == 0 || (currentTime - lastCrankTime >
83         200000)) { lastCamInTime = currentTime; return; }
84     if (currentTime - lastCamInTime < CAM_DEBOUNCE_US) return;
85
86     uint32_t cyclesSinceLastCrank = currentCycles - lastCrankCycles;
87     //interpolate fraction between CDM pulses when a camshaft
88     //rising edge is identified and calculate exact relative
89     //position
90     float fraction = (toothIntervalCycles > 0) ? ((float)
91         cyclesSinceLastCrank / (float)toothIntervalCycles) : 0.0;
92     if (fraction > 1.0) fraction = 1.0;
93     float exactAngle = (crankToothCount * DEGREES_PER_TOOTH) + (
94         fraction * DEGREES_PER_TOOTH);
95
96     if (currentState == LEARNING_PHASE) {
97         if (!isVerifying) {
98             bool isDuplicate = false;
99             for (int i = 0; i < camInToothCount; i++) {
100                 float diff = abs(exactAngle - camInMatrix[i]);
101                 if (diff > 360.0) diff = abs(diff - 720.0);
102                 if (diff < 45.0) isDuplicate = true;
103             }
104             if (!isDuplicate && camInToothCount < MAX_CAM_TEETH) {
105                 camInMatrix[camInToothCount++] = exactAngle;
106             }
107         } else {
108             if (tempCamInCount < activeCamInTeethCount) {
109                 float expected = camInMatrix[tempCamInCount];
110                 float diff = abs(exactAngle - expected);
111                 if (diff > 360.0) diff = abs(diff - 720.0);
112                 if (diff <= VALIDATION_TOLERANCE) tempCamInCount++;
113                 else { currentState = WAITING_FOR_SYNC; triggPulseCount =
114                     0; }
115             }
116         }
117     }
118     else if (currentState == RUNNING) {
119         float minDiff = 720.0;
120         for (int i = 0; i < activeCamInTeethCount; i++) {
121             float diff = exactAngle - camInMatrix[i];
122             if (diff > 360.0) diff -= 720.0;
123             if (diff < -360.0) diff += 720.0;
124             if (abs(diff) < abs(minDiff)) minDiff = diff;
125         }
126         if (abs(minDiff) <= MAX_PHASE_WINDOW) {
127             if (!inLocked) { actualInPhase = minDiff; inLocked = true; }
128             else actualInInPhase = (actualInPhase * (1.0 -
129                 PHASE_FILTER_WEIGHT_IN)) + (minDiff *
130                 PHASE_FILTER_WEIGHT_IN);
131         }
132     }
133 }

```

```

124     lastCamInTime = currentTime;
125 }
126
127 void camExISR() {
128     uint32_t currentCycles = ARM_DWT_CYCCNT;
129     uint32_t currentTime = micros();
130     if (lastCrankCycles == 0 || (currentTime - lastCrankTime >
131         200000)) { lastCamExTime = currentTime; return; }
132     if (currentTime - lastCamExTime < CAM_DEBOUNCE_US) return;
133
134     uint32_t cyclesSinceLastCrank = currentCycles - lastCrankCycles;
135     float fraction = (toothIntervalCycles > 0) ? ((float)
136         cyclesSinceLastCrank / (float)toothIntervalCycles) : 0.0;
137     if (fraction > 1.0) fraction = 1.0;
138     float exactAngle = (crankToothCount * DEGREES_PER_TOOTH) + (
139         fraction * DEGREES_PER_TOOTH);
140
141     if (currentState == LEARNING_PHASE) {
142         if (!isVerifying) {
143             bool isDuplicate = false;
144             for (int i = 0; i < camExToothCount; i++) {
145                 float diff = abs(exactAngle - camExMatrix[i]);
146                 if (diff > 360.0) diff = abs(diff - 720.0);
147                 if (diff < 45.0) isDuplicate = true;
148             }
149             if (!isDuplicate && camExToothCount < MAX_CAM_TEETH) {
150                 camExMatrix[camExToothCount++] = exactAngle;
151             }
152         } else {
153             if (tempCamExCount < activeCamExTeethCount) {
154                 float expected = camExMatrix[tempCamExCount];
155                 float diff = abs(exactAngle - expected);
156                 if (diff > 360.0) diff = abs(diff - 720.0);
157                 if (diff <= VALIDATION_TOLERANCE) tempCamExCount++;
158                 else { currentState = WAITING_FOR_SYNC; triggPulseCount =
159                     0; }
160             }
161         }
162     }
163     else if (currentState == RUNNING) {
164         float minDiff = 720.0;
165         for (int i = 0; i < activeCamExTeethCount; i++) {
166             float diff = exactAngle - camExMatrix[i];
167             if (diff > 360.0) diff -= 720.0;
168             if (diff < -360.0) diff += 720.0;
169             if (abs(diff) < abs(minDiff)) minDiff = diff;
170         }
171         if (abs(minDiff) <= MAX_PHASE_WINDOW) {
172             if (!exLocked) { actualExPhase = minDiff; exLocked = true; }
173             else actualExPhase = (actualExPhase * (1.0 -
174                 PHASE_FILTER_WEIGHT_EX)) + (minDiff *
175                 PHASE_FILTER_WEIGHT_EX);
176         }
177     }
178     lastCamExTime = currentTime;
179 }

```

5.5.3 Loops

The code in the loop function runs as fast as the processor allows. However, the PID loop is reduced to running at 50 Hz to allow for the physical system to keep up. If the PID regulator ran as fast as the microcontroller would allow, the previous error would accumulate fast and the system would overshoot. The display is updated at 20 Hz as this was deemed fast enough. Updating the display even faster would for example cause the engine speed value, which changes during the rotation, flicker even more. The PID loop calculates the outgoing PWM signal for the gate driver and MOSFETs. It also turns the signal off when stall is detected or before the system is turned on.

Handle inputs is the code for the rotary encoders. Depending in which state the system is in they do different things. After the learning and validation is done the encoders are used to either start the system or relearn the camshaft teeth positions, by pressing the encoder button. When the system has started turning the encoder dials changes the target values for the intake and exhaust offset. Pressing the encoder button puts the encoders in “fine tuning mode” where target can be set with 0.1° accuracy.

The last loop function handles the display. Similarly it changes depending on regulator state. Waiting for sync is displayed when the engine is turned off. When trigger pulse is found the learning and validation phase is done in a fraction of a second and then the learned edges are displayed. If more then 3 teeth are found the display scrolls through the values. The user then has a choice to relearn the values or start the system. When the system starts the state and engine speed is shown at the top of the display. Below that the display is split by a vertical line for exhaust (left) and intake (right). In each column the target offset value, actual offset value and PWM in percentage is shown. The display likely uses a controller for a 256x32 display. Due to a defect in the display the three bottom rows relative to the 256x32 controller could not be used as it would create a vertical row of darker pixels. This affected placement on the screen slightly but is hardly noticeable.

```
1 void checkCrankWatchdog() {
2     noInterrupts();
3     uint32_t lastCrank = lastCrankTime;
4     interrupts();
5
6     uint32_t nowUs = micros();
7     if (lastCrank != 0 && (nowUs - lastCrank > 200000)) {
8         noInterrupts();
9         toothIntervalCycles = 0;
10        interrupts();
11
12        if (currentState == RUNNING || currentState == LEARNING_PHASE)
13            {
                currentState = WAITING_FOR_SYNC;
            }
```



```

14     triggPulseCount = 0;
15     Serial.println("--- CRANK SIGNAL LOST! DROPPING TO
        WAITING_FOR_SYNC ---");
16 }
17 }
18 }
19
20 // =====
21 // VVT PID CONTROLLER
22 // =====
23 void calculatePID() {
24     uint32_t now = millis();
25     if (now - lastPidTime < PID_INTERVAL_MS) return;
26     lastPidTime = now;
27
28     noInterrupts();
29     uint32_t safeIntervalCycles = toothIntervalCycles;
30     float currentIn = actualInPhase;
31     float currentEx = actualExPhase;
32     interrupts();
33
34     float currentRpm = 0.0;
35     if (safeIntervalCycles > 0) {
36         currentRpm = ((float)F_CPU / ((float)safeIntervalCycles *
            CRANK_TEETH)) * 60.0;
37     }
38
39     // Safety Cut: Disable solenoids if engine isn't fully running,
        RPM is too low, or user is still reading the Mapping screen.
40     if (currentState != RUNNING || currentRpm < 400.0 ||
        showLearnedDisplay) {
41         pwmIn = 0; pwmEx = 0;
42         analogWrite(SOLENOID_IN_PIN, 0);
43         analogWrite(SOLENOID_EX_PIN, 0);
44         integralIn = 0.0; prevErrorIn = 0.0;
45         integralEx = 0.0; prevErrorEx = 0.0;
46         return;
47     }
48
49     // --- 1. Intake PID Calculation ---
50     // Applying PID_DIR_IN to flip the response direction if needed
51     float errorIn = (targetInPhase - currentIn) * PID_DIR_IN;
52     integralIn += errorIn;
53     integralIn = constrain(integralIn, -PID_INTEGRAL_LIMIT,
        PID_INTEGRAL_LIMIT);
54     float derivativeIn = errorIn - prevErrorIn;
55     float outputIn = PID_BASE_PWM + (Kp * errorIn) + (Ki * integralIn
        ) + (Kd * derivativeIn);
56     pwmIn = constrain((int)outputIn, PWM_MIN_LIMIT, PWM_MAX_LIMIT);
57     prevErrorIn = errorIn;
58
59     // --- 2. Exhaust PID Calculation ---
60     // Applying PID_DIR_EX to flip the response direction if needed
61     float errorEx = (targetExPhase - currentEx) * PID_DIR_EX;
62     integralEx += errorEx;
63     integralEx = constrain(integralEx, -PID_INTEGRAL_LIMIT,

```

```

        PID_INTEGRAL_LIMIT);
64 float derivativeEx = errorEx - prevErrorEx;
65 float outputEx = PID_BASE_PWM + (Kp * errorEx) + (Ki * integralEx
    ) + (Kd * derivativeEx);
66 pwmEx = constrain((int)outputEx, PWM_MIN_LIMIT, PWM_MAX_LIMIT);
67 prevErrorEx = errorEx;
68
69 // --- 3. Dither Generation ---
70 int finalPwmIn = pwmIn;
71 int finalPwmEx = pwmEx;
72
73 if (ENABLE_DITHER) {
74     if (millis() - lastDitherTime >= DITHER_INTERVAL_MS) {
75         lastDitherTime = millis();
76         ditherState = !ditherState;
77         ditherOffset = ditherState ? DITHER_AMPLITUDE : -
            DITHER_AMPLITUDE;
78     }
79     // Only apply dither if the valve is active
80     if (pwmIn > 0 && pwmIn < 1000) {
81         finalPwmIn = constrain(pwmIn + ditherOffset, PWM_MIN_LIMIT,
            PWM_MAX_LIMIT);
82     }
83     if (pwmEx > 0 && pwmEx < 1000) {
84         finalPwmEx = constrain(pwmEx + ditherOffset, PWM_MIN_LIMIT,
            PWM_MAX_LIMIT);
85     }
86 }
87
88 // --- 4. Hardware Outputs (12-bit High Resolution Upgrade) ---
89 analogWrite(SOLENOID_IN_PIN, map(finalPwmIn, PWM_MIN_LIMIT,
    PWM_MAX_LIMIT, 0, 4095));
90 analogWrite(SOLENOID_EX_PIN, map(finalPwmEx, PWM_MIN_LIMIT,
    PWM_MAX_LIMIT, 0, 4095));
91 }
92
93 // =====
94 // INPUT HANDLING (Buttons & Encoders)
95 // =====
96
97 void handleInputs() {
98     int currentInBtn = digitalRead(ENC_IN_BTN);
99     int currentExBtn = digitalRead(ENC_EX_BTN);
100
101     // --- MAPPING SUCCESSFUL SCREEN OVERRIDE ---
102     if (showLearnedDisplay) {
103         // Check for Left Button (Relearn)
104         if (currentExBtn == LOW && lastExBtnState == HIGH) {
105             showLearnedDisplay = false;
106             currentState = WAITING_FOR_SYNC;
107             triggPulseCount = 0;
108             Serial.println("--- RELEARN REQUESTED BY USER ---");
109             delay(50); // Debounce
110         }
111         // Check for Right Button (Start)
112         else if (currentInBtn == LOW && lastInBtnState == HIGH) {

```

```

113     showLearnedDisplay = false;
114     Serial.println("--- STARTING PID LOOP ---");
115     delay(50); // Debounce
116 }
117
118     lastInBtnState = currentInBtn;
119     lastExBtnState = currentExBtn;
120
121     // Clear encoder buffers so turning the knobs during the freeze
122     // doesn't jump the target later
123     encIn.read();
124     encEx.read();
125     return; // Exit here so we don't change fine modes or target
126     // limits
127 }
128
129 // --- STANDARD RUNNING MODE ---
130
131 if (currentInBtn == LOW && lastInBtnState == HIGH) {
132     inFineMode = !inFineMode;
133     delay(50);
134 }
135 lastInBtnState = currentInBtn;
136
137 if (currentExBtn == LOW && lastExBtnState == HIGH) {
138     exFineMode = !exFineMode;
139     delay(50);
140 }
141 lastExBtnState = currentExBtn;
142
143 long newInPos = encIn.read();
144 long newExPos = encEx.read();
145
146 if (newInPos >= lastEncInPos + 4) {
147     targetInPhase += (inFineMode ? 0.1 : 1.0);
148     lastEncInPos = newInPos;
149 } else if (newInPos <= lastEncInPos - 4) {
150     targetInPhase -= (inFineMode ? 0.1 : 1.0);
151     lastEncInPos = newInPos;
152 }
153
154 if (newExPos >= lastEncExPos + 4) {
155     targetExPhase += (exFineMode ? 0.1 : 1.0);
156     lastEncExPos = newExPos;
157 } else if (newExPos <= lastEncExPos - 4) {
158     targetExPhase -= (exFineMode ? 0.1 : 1.0);
159     lastEncExPos = newExPos;
160 }
161
162 targetInPhase = constrain(targetInPhase, TARGET_IN_PHASE_MIN,
163     TARGET_IN_PHASE_MAX);
164 targetExPhase = constrain(targetExPhase, TARGET_EX_PHASE_MIN,
165     TARGET_EX_PHASE_MAX);
166 }
167
168 // =====

```

```

165 // DISPLAY HANDLING
166 // =====
167 void updateDisplay() {
168     u8g2.clearBuffer();
169     char buf[30];
170
171     // --- POST-LEARNING MAPPED RESULTS DISPLAY ---
172     if (showLearnedDisplay) {
173         noInterrupts();
174         int inCount = activeCamInTeethCount;
175         int exCount = activeCamExTeethCount;
176         interrupts();
177
178         // Auto-scroll index changes every 2 seconds
179         int scrollEx = (exCount > 3) ? (millis() / 2000) % exCount : 0;
180         int scrollIn = (inCount > 3) ? (millis() / 2000) % inCount : 0;
181
182         // -----
183         // TOP TITLE
184         // -----
185         u8g2.setFont(u8g2_font_6x12_tr);
186         u8g2.setCursor(0, 7);
187         u8g2.print("MAPPING SUCCESSFUL");
188         u8g2.drawLine(0, 9, 128, 9);
189
190         // -----
191         // HEADERS
192         // -----
193         u8g2.setFont(u8g2_font_ncenB08_tr);
194         u8g2.setCursor(0, 19);
195         u8g2.print("EXHAUST");
196         u8g2.setCursor(67, 19);
197         u8g2.print("INTAKE");
198
199         // Center divider
200         u8g2.drawLine(64, 10, 64, 51);
201
202         // -----
203         // DATA
204         // -----
205         const int rowStart = 31;
206         const int rowSpacing = 10;
207
208         // Exhaust
209         int exLimit = (exCount > 3) ? 3 : exCount;
210         for (int i = 0; i < exLimit; i++) {
211             int idx = (exCount > 3) ? (scrollEx + i) % exCount : i;
212             u8g2.setCursor(0, rowStart + (i * rowSpacing));
213
214             noInterrupts(); float exVal = camExMatrix[idx]; interrupts();
215             snprintf(buf, sizeof(buf), "%d: %5.1f", idx + 1, exVal);
216             u8g2.print(buf);
217             u8g2.print(char(176));
218         }
219
220         // Intake

```

```

221     int inLimit = (inCount > 3) ? 3 : inCount;
222     for (int i = 0; i < inLimit; i++) {
223         int idx = (inCount > 3) ? (scrollIn + i) % inCount : i;
224         u8g2.setCursor(67, rowStart + (i * rowSpacing));
225
226         noInterrupts(); float inVal = camInMatrix[idx]; interrupts();
227         snprintf(buf, sizeof(buf), "%d: %5.1f", idx + 1, inVal);
228         u8g2.print(buf);
229         u8g2.print(char(176));
230     }
231
232     // -----
233     // BOTTOM BUTTONS
234     // -----
235     u8g2.setFont(u8g2_font_ncenB08_tr);
236     u8g2.setCursor(0, 63);
237     u8g2.print("<-RELEARN");
238
239     int startStrWidth = u8g2.getStrWidth("START->");
240     u8g2.setCursor(128 - startStrWidth, 63);
241     u8g2.print("START->");
242
243     u8g2.sendBuffer();
244     return;
245 }
246
247 // --- STANDARD LIVE DASHBOARD (Dual Column, Flipped) ---
248 u8g2.setFont(u8g2_font_ncenB08_tr);
249
250 u8g2.setCursor(0, 9);
251 if (currentState == WAITING_FOR_SYNC) u8g2.print("WAITING trigg")
252     ;
253 else if (currentState == LEARNING_PHASE) {
254     if (!isVerifying) u8g2.print("MAPPING...");
255     else u8g2.print("VERIFYING...");
256 }
257 else u8g2.print("RUNNING");
258
259 noInterrupts();
260 uint32_t safeIntervalCycles = toothIntervalCycles;
261 interrupts();
262
263 int rpm = 0;
264 if (safeIntervalCycles > 0) {
265     rpm = ((float)F_CPU / ((float)safeIntervalCycles * CRANK_TEETH)
266         ) * 60.0;
267 }
268
269 u8g2.setFont(u8g2_font_6x12_tr);
270 snprintf(buf, sizeof(buf), "%4d RPM", rpm);
271 u8g2.setCursor(80, 9);
272 u8g2.print(buf);
273
274 u8g2.setFont(u8g2_font_ncenB08_tr);
275 u8g2.drawLine(0, 11, 128, 11);

```

```
275 u8g2.setCursor(0, 23);
276 u8g2.print("EXHAUST");
277 u8g2.setCursor(67, 23);
278 u8g2.print("INTAKE");
279
280 u8g2.drawLine(64, 13, 64, 63);
281
282 u8g2.setCursor(0, 41);
283 snprintf(buf, sizeof(buf), "TRGT:%4.1f", targetExPhase); u8g2.
    print(buf);
284 if (exFineMode) u8g2.print("*");
285
286 u8g2.setCursor(67, 41);
287 snprintf(buf, sizeof(buf), "TRGT:%4.1f", targetInPhase); u8g2.
    print(buf);
288 if (inFineMode) u8g2.print("*");
289
290 u8g2.setCursor(0, 51);
291 noInterrupts(); float printIn = actualInPhase; float printEx =
    actualExPhase; interrupts();
292 snprintf(buf, sizeof(buf), "ACTL:%5.2f", printEx); u8g2.print(buf
    );
293
294 u8g2.setCursor(67, 51);
295 snprintf(buf, sizeof(buf), "ACTL:%5.2f", printIn); u8g2.print(buf
    );
296
297 // --- PWM ROW: MIXED FONT SIZES ---
298 // Exhaust Side
299 u8g2.setCursor(0, 61);
300 u8g2.setFont(u8g2_font_5x8_tr);
301 u8g2.print("PWM: ");
302 u8g2.setFont(u8g2_font_ncenB08_tr);
303 if (rpm < 400) {
304     u8g2.print("CUT");
305 } else {
306     snprintf(buf, sizeof(buf), "%3d%%", pwmEx / 10);
307     u8g2.print(buf);
308 }
309
310 // Intake Side
311 u8g2.setCursor(67, 61);
312 u8g2.setFont(u8g2_font_5x8_tr);
313 u8g2.print("PWM: ");
314 u8g2.setFont(u8g2_font_ncenB08_tr);
315 if (rpm < 400) {
316     u8g2.print("CUT");
317 } else {
318     snprintf(buf, sizeof(buf), "%3d%%", pwmIn / 10);
319     u8g2.print(buf);
320 }
321
322 u8g2.sendBuffer();
323 }
```

This part is what is running the code. The void setup is ran once when the system

is started taking the definitions from the beginning of the code. For example in line `attachInterrupt(digitalPinToInterrupt(CRANK_PIN), crankISR, RISING);`, this uses the pin definition in the beginning of the code as the input. If it senses a rising edge it runs the crankISR function. Void loop runs over and over until the system is shut down. In the loop the different functions are called to watch the crank and camshaft, calculate the PWM signal to the solenoids and display the results. The display updates every 50 ms (20 Hz).

```

1 // =====
2 // MAIN SETUP & LOOP
3 // =====
4 void setup() {
5     Serial.begin(115200);
6
7     pinMode(CRANK_PIN, INPUT);
8     pinMode(TRIGG_PIN, INPUT);
9     pinMode(CAM_IN_PIN, INPUT);
10    pinMode(CAM_EX_PIN, INPUT);
11
12    pinMode(SOLENOID_IN_PIN, OUTPUT);
13    pinMode(SOLENOID_EX_PIN, OUTPUT);
14
15    pinMode(ENC_IN_BTN, INPUT_PULLUP);
16    pinMode(ENC_EX_BTN, INPUT_PULLUP);
17
18    analogWrite(SOLENOID_IN_PIN, 0);
19    analogWrite(SOLENOID_EX_PIN, 0);
20
21    analogWriteResolution(12); //12-bit to allow greater than 1/255
22                                accuracy PWM
23
24    u8g2.begin();
25
26    attachInterrupt(digitalPinToInterrupt(CRANK_PIN), crankISR,
27                    RISING);
28    attachInterrupt(digitalPinToInterrupt(TRIGG_PIN), triggISR,
29                    FALLING);
30    attachInterrupt(digitalPinToInterrupt(CAM_IN_PIN), camInISR,
31                    RISING);
32    attachInterrupt(digitalPinToInterrupt(CAM_EX_PIN), camExISR,
33                    RISING);
34 }
35
36 void loop() {
37     checkCrankWatchdog();
38     handleInputs();
39     calculatePID();
40
41     static uint32_t lastDisplay = 0;
42     if (millis() - lastDisplay >= 50) {
43         lastDisplay = millis();
44         updateDisplay();
45     }
46 }

```


6

Conclusion

The product works as intended with the testing that has been able to be performed. The two main tests to prove the functionality of the product has been to determine the cam phaser offset and driving the solenoids. The final test would have been to actually test if the system successfully can turn the cam phaser. Due to supply issues for the camshaft gears with phasing capability it was not possible to test that. The test for reading the signals and calculating the offset was successful. During the learning phase the camshaft teeth are found at the estimated positions. During operation, the actual offset hovers around 0° which it should since the cam shaft gears are fixed. There were slight irregularities on one side which was likely caused by purely physical variations. As the timing belt stretches, the belt tensioner oscillates and the valve spring push back on the system, it is expected to see mechanical noise. To solve this EMA filtering was used. This results in a smoother but less responsive operation.

Powering the MOSFETs had initially been troublesome and were very sensitive to mistakes. The pins are close to each other and as fairly high current were going through the connections, it was preferable to have direct connections between the wire and the MOSFET legs, causing large solder joints. This combination of large solder joints and tight space meant that any debris would short the MOSFET causing it to heat up and destroy the thin metal oxide layer inside, rendering it useless. But with proper precautions and the addition of the gate driver resulted in the MOSFETs barely heating up at all during operation.

What type of hardware requirements are there when creating a control system for VVT?

Firstly the microcontroller or the electric platform used is the most important part, it can be seen as the brain of regulating the VVT solenoids. The controller needs to be deterministic. Any system that uses a software on top of the processor will not reliably work as it is too slow to keep up with the engine. A great example is Raspberry Pi which on paper has a much more powerful processor, but it most commonly uses a Linux operating system, and code would need to be written inside the operating system. Platforms that use an operating system also have drawbacks such as boot up times and a susceptibility to data corruption if an sudden power loss occurs during write operations.

In retrospect it might not be possible to deduce what microcontroller is fast enough to run the system. The earlier test software that was ran on the Arduino was far

inferior to the final software. However there are some physical limitations that suggest that a Arduino or similar microcontroller are to slow. The most demanding part about this application is the CDM pulses (3600 per revolution). At 4000 rpm which would be the absolute fastest the system needs to be able to read 240 000 pulses per second, leaving very little time between interrupts.

The controller also needs to have enough pins with hardware interrupt capability. These pins combined with the code can execute a function purely based on the hardware signal which is needed for the four input signals. Controllers like the Arduino R3 only has two of these pins.

Peripherals are stated in the Section 5.2. Many of these are interchangeable or can be adapted depending on use case. For example an automated system similarly to the ones found in a production vehicle has no input from the user, nor any way of displaying whats going on. These components can be seen as added value and is not a requirement for creating a VVT controller.

The rest of the electronics could possibly be solved in other ways but using MOSFETs with a gate driver is most likely the best way of doing it. MOSFETs have a very fast switching time especially when paired with a gate driver. Making that combo ideal for PWM control with low heat generation. In the end the included fan was not necessary.

What software requirements are there to control a hydraulically based VVT hardware?

The software is described in detail in Section 5.5. The main functions of the code are:

- handle and filter signal inputs
- handle inputs from the encoders
- use the crankshaft position as a very accurate clock
- calculate the offset of the camshaft at each tooth
- regulate the error (target offset - actual offset) using PID regulator, sending PWM signals to the MOSFETs
- display information.

What specific needs are there for a VVT system in a lab environment?

The identified user needs are presented in Chapter 3. Possibly the strongest need is the controllability of the system which has been a large factor in deciding to actually start this master thesis. There likely are “off the shelf” solutions that could be adapted to work with the lag engine but that would not have the possibility to easily change the camshaft offset. As previously mentioned a aftermarket ECU could be adapted to work with the lab engine and flashing the VVT angle table could fulfill the same function.

The product also needs to be easy to use. The lab engine already have multiple fairly complicated systems that needs to be learned, adding another difficult to use system would not be ideal. This included with the limited amount of staff that

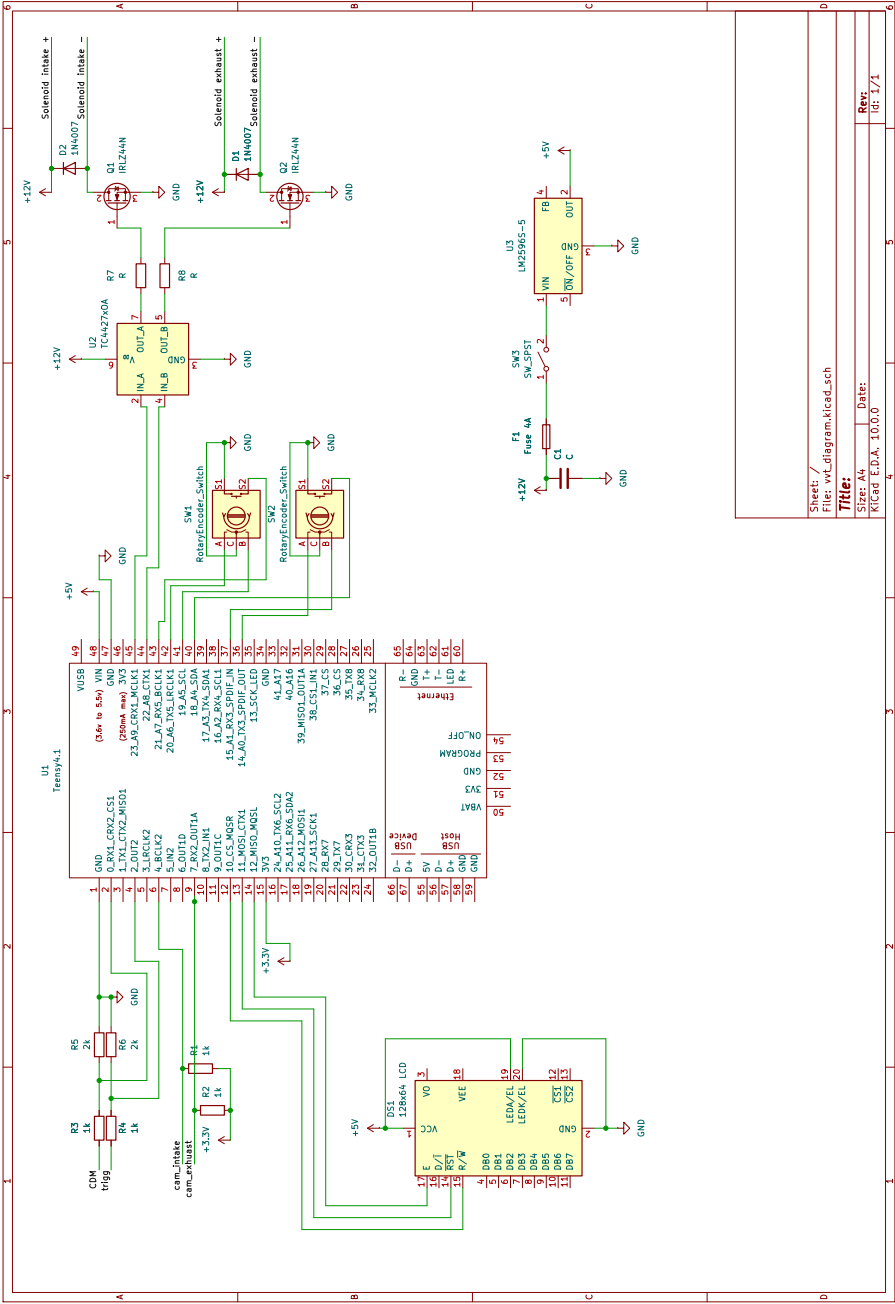
knows the system and the possibility of employee turnover does not help. With the final product the system can be on at all times, the on/off button on the back does not need to be used. The system is powered by an auxiliary port, that only supplies electricity when the test rig is on. After the learning phase the camshaft angles are displayed on the screen and halts all operations. Two options are presented on the screen, relearn or start which can be activated by pressing either the left or right encoder button. The user could relearn the camshaft positions due to the system not finding the correct positions which could be caused by acceleration or signal noise. Not pressing any button while on this screen would just not turn on the system, which could be used for some instances.

Bibliography

- AVL List GmbH. (2010). *High-precision 365C angle encoder set: Product description* [Technical Documentation]. AVL List GmbH. Graz, Austria.
- Digilent Inc. (2025). *Waveforms* (Version 3.25.1). <https://digilent.com/reference/software/waveforms/waveforms-3/start>
- Hattori, M., Inoue, T., Mashiki, Z., Takenaka, A., Urushihata, H., Morino, S., & Inohara, T. (2008). Development of variable valve timing system controlled by electric motor. *SAE International Journal of Engines*, 01(1), 985–990. <https://doi.org/10.4271/2008-01-1358>
- Heywood, J. (2019). *Internal combustion engine fundamentals 2e* (2nd edition). McGraw-Hill Education.
- National Research Council. (2015). *Cost, effectiveness, and deployment of fuel economy technologies for light-duty vehicles*. National Academies Press.
- Schiele, C. A., DeNagel, S. F., & Bennethum, J. E. (1974). Design and development of a variable valve timing (Vvt) camshaft. <https://doi.org/10.4271/740102>
- Stewart, J. (2024, May 28). *Ardu-stim*. Retrieved April 30, 2026, from <https://github.com/speeduino/Ardu-Stim>
- Wapcaplet. (2005, June 10). *Labeled diagram of a four-stroke dohc engine* [Wikimedia Commons, CC BY-SA 3.0]. Retrieved April 22, 2026, from <https://commons.wikimedia.org/w/index.php?curid=182044>

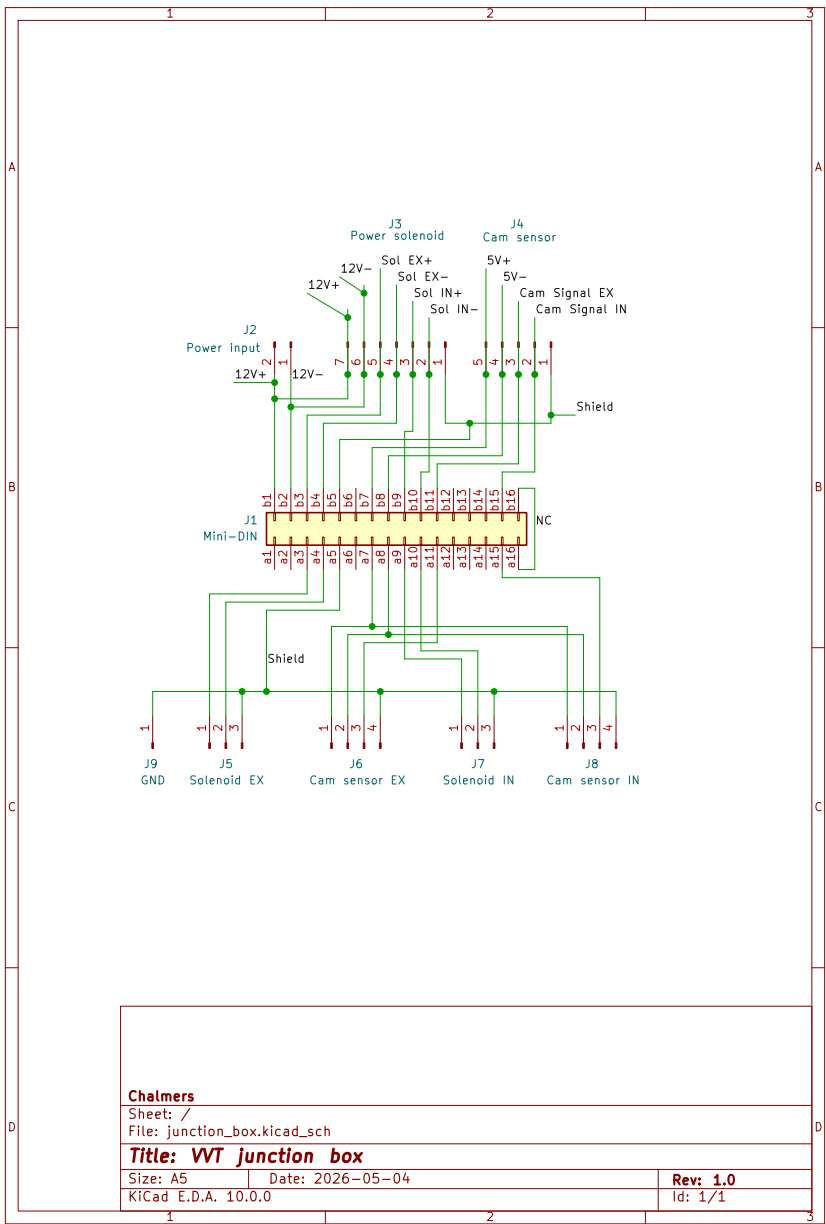
A

Schematic of Final Product



B

Schematic of Junction Box



DEPARTMENT OF MECHANICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se



CHALMERS
UNIVERSITY OF TECHNOLOGY