Design and Prototyping of an Electrostatic Discharge Machining (EDM) Device

Electrode, Dielectric Fluid and Structural Design

EE3L11: Bachelor Graduation Project

Rick Helmer
Tim Qualm
Sarhoz Mahmoud

Delft University of Technology

Design and Prototyping of an Electrostatic Discharge Machining (EDM) Device

Electrode, Dielectric Fluid and Structural Design

by

Rick Helmer
Tim Qualm
Sarhoz Mahmoud

Student Name	Student Number
Rick Helmer	5630177
Tim Qualm	5531144
Sarhoz Mahmoud	5643562

in partial fulfilment of the requirements for the degree of

Bachelor of Science
in Electrical Engineering

at the Delft University of Technology,
to be defended publicly on Tuesday, June 25, 2024, at 11:00 AM.

Supervisor:	Mohamad Ghaffarian Niasar
Advisors:	Dennis van der Born and Hani Vahedi
Thesis committee:	Ioan E. Lager, Mohamad Ghaffarian Niasar and Hani Vahedi
Project Duration:	April, 2024 - June, 2024
Faculty:	Faculty of Electrical Engineering, Delft

Abstract

This paper presents the design and development of an Electrical Discharge Machining (EDM) device aimed at achieving precise hole creation in diverse metal materials. The EDM device was conceptualized and constructed based on specific requirements identified by the group. The design process encompassed four main stages: dielectric fluid selection, electrode design, control system development and power supply.

In the dielectric fluid stage, the importance and criteria for selecting an appropriate fluid were discussed, resulting in the choice of distilled water for its superior dielectric properties. The electrode design stage followed a similar methodology, leading to the selection of a copper rod as the optimal electrode material.

The control system stage detailed the development of an open-loop, manual, and closed-loop control system, emphasizing the utilization of Klipper software for precise electrode control.

The power supply section outlined three primary circuits: the power source, power amplification circuit, and square wave generator circuit. Detailed schematics, component justifications, and optimization values for key parameters were provided to enhance power supply efficiency.

Experimental evaluation demonstrated the capability of the EDM device to effectively create holes in various metals using an open-loop control system. Additional experiments focused on parameter variations within the power supply setup further illustrated their impact on machining performance.

The discussion highlights the challenges encountered throughout the project, particularly the constraints imposed by limited time, which prevented the realization of all initially set requirements. Despite these challenges, the EDM device successfully met most of the specified objectives, showcasing promising results for future refinements and applications in precision machining.

Preface & Acknowledgements

We delved into the world of Electrical Discharge Machining (EDM) over the course of two months. This thesis documents the process of creating the EDM and the results of the experiments. The project was done with another subgroup of three other ambitious students who were also interested in creating an EDM [1].

As the people responsible for the electrode, dielectric fluid, and structural design, we found the practical experience immensely valuable and fulfilling. The high-voltage lab at Delft University of Technology allowed us to work independently with the actual tools (hardware) needed for the project.

We would like to express our sincere thanks to our supervisor, Mohamad Ghaffarian Niasar, who was invaluable in sharing his knowledge and always had time to assist us.

Secondly, we are grateful to Hani Vahedi and Dennis van der Born for their help in clarifying complex topics and for their innovative suggestions to improve various concepts.

We are very thankful to Wim Termorshuizen for his kind assistance when the supervisor was unavailable in the lab and for his practical help with the design.

Finally, we extend our deepest gratitude to our families for their unwavering support throughout the past three years of our bachelor's studies in Electrical Engineering. Their encouragement and patience provided us with the opportunity and time to pursue this path. Without their support, completing this graduation project would not have been possible.

Sarhoz Mahmoud, Rick Helmer & Tim Qualm
Delft, June 2024

Contents

1
Introduction

In this chapter, key concepts used in this project will be presented. After this, the goal of the entire project will be defined. The division of the project into subgroups will be explained, and the goal of the subgroup of this thesis will be specified. Lastly, the structure of this thesis will be described.

1.1. Principle of Electrical Discharge Machining

Electrical Discharge Machining (EDM) is a widely used non-conventional material removal process that uses electrical energy to generate controlled sparks between the tool and a workpiece. This process mainly relies on thermal energy produced by the discharge sparks to erode material from the workpiece, creating a replica of the tool's form on the workpiece. An EDM is particularly effective for machining hard-to-machine material and intricate shapes.

Fundemental Mechanism

EDM operates on the principle of thermal erosion, where a series of high-frequency electrical discharges occur between the tool and the workpiece. The tool, typically the cathode, and the workpiece, the anode, are both immersed in a dielectric medium. A potential difference is applied between them, generating an electric field across the gap. This concept is visualised in figure 1.1. As the tool approaches the workpiece, dielectric breakdown occurs, forming a plasma channel.

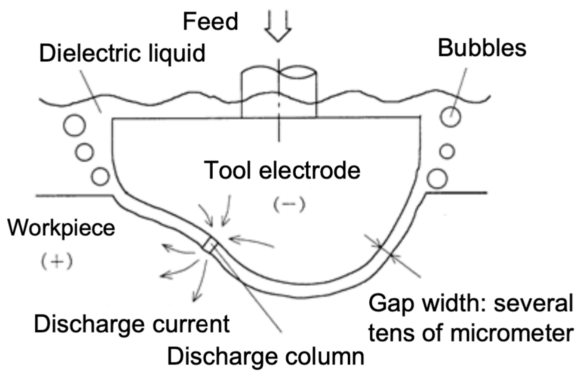


Figure 1.1: Basic concept of Electric Discharge Machining [2].

Dielectric Medium and Plasma Channel

The dielectric medium, usually a liquid with high breakdown strength, low viscosity, and effective cooling capabilities, plays a crucial role in the EDM process [3]. When the electrical field is established, free electrons are accelerated from the tool towards the workpiece, causing collisions and ionization of the dielectric molecules. This ionization creates a plasma channel with low electrical resistance, allowing a significant current to flow between the tool and the workpiece [4]. This sudden flow of electrons and ions is visually observed as a spark.

Thermal Energy and Material Removal
The kinetic energy of the accelerated electrons and ions is converted into thermal energy upon impact with the workpiece and tool surface. This intense localized heat flux, which can exceed temperatures of 10.000 °C, results in the melting and partial vaporization of the workpiece material [4]. The molten material is then expelled from the crater by the dielectric fluid, with some solidifying as a recast layer. the size of the crater, and consequently the material removal rate, is controlled by the discharge energy, which can be adjusted by varying the discharge current and duration [3].

Advantages and Applications
One key advantage of using an EDM is the lack of mechanical contact between the tool and workpiece, eliminating issues related to mechanical stresses, chatter, and vibration during machining. This makes it particularly suitable for machining difficult-to-machine materials and high-strength, temperature-resistant alloys. The process is extensively used to produce complex shapes, such as injection moulds, punch dies, and intricate cavities in hard materials [4].

Summary
In summary, EDM is a highly effective and versatile machining process that uses controlled electrical discharges to erode material from a workpiece. Its ability to machine complex shapes in hard materials with high precision and surface finish makes it an essential technique in modern manufacturing. The principle of EDM lies in its thermal erosion mechanism, facilitated by forming a plasma channel in a dielectric medium, resulting in localized melting and material removal.

1.2. The Goal of the Project

Current drilling technologies often struggle to create precise holes in a wide range of metals, particularly when high accuracy is required. Traditional methods can be effective in some contexts, but they often fall short when dealing with harder or more delicate materials, leading to inaccuracies and potential damage to the workpiece. In industrial applications, such imperfections can result in significant problems, including increased material waste and higher production costs. Additionally, precise hole creation is critical in aerospace, medical devices, and precision engineering, where minor deviations can have substantial repercussions.

To address these challenges, our project aims to develop a method for creating precise holes in various metals using advanced EDM techniques. By leveraging the unique properties of EDM, which allows for non-contact material removal and high precision, we seek to achieve greater accuracy and consistency compared to the drilling process.

The goal of this project is to utilize the EDM process to produce highly accurate holes in a range of metals, including those that are difficult to machine using conventional methods. This involves optimizing the EDM parameters and ensuring the process can be applied to different metal types with minimal adjustment. By focusing on these areas, we aim to enhance the overall efficiency and reliability of the drilling process.

Ultimately, our project aims to establish a robust and versatile drilling method that can be applied across multiple industries, providing a reliable solution for creating precise holes in metals. This method's successful implementation will improve manufacturing accuracy and reduce material waste and production costs, leading to significant advancements in various high-precision fields.

1.3. Problem Definition

Conventional drilling methods for creating holes in metals face numerous challenges. The most notable issue is the significant pressure these methods exert on the metals, which complicates machining fragile metal shapes, such as making holes in thin surfaces with limited drilling space. To address these problems, the group will utilize a non-conventional technology called Electrical Discharge Machining (EDM) to create an EDM device. This EDM device will enable precise hole creation in various metals and metal surfaces, possibly achieving an exceptional precision in the order of micrometers [5]. Thus, the group aims to develop an EDM device to overcome the limitations of traditional drilling methods.

Multiple factors must be considered when creating the EDM device. The most crucial aspect is that the group has only about nine weeks to study the technology and build the EDM device. To achieve the goal, the group will be divided into subgroups. This paper will focus on the ”Electrode, Dielectric Fluid, and Structural Design” subgroup. As the name suggests, this subgroup will focus on selecting the most optimal electrode and dielectric fluid. Most of the effort will be dedicated to the structural design, which includes setting up the EDM device and developing its mechanical movement system. The subgroup will also pay attention to the power supply needed to conduct experiments that test the device.

The EDM device has numerous design requirements, detailed in Chapter 2. The most significant limitations are the availability of materials at the university and the cost of certain equipment needed to create the device. These challenges have been addressed by selecting smart designs that utilize resources available at the university.

The EDM device’s performance must be measured throughout the project. Many studies on EDM have evaluated performance using Material Removal Rate (MRR) and Tool Wear Rate (TWR). Since this paper focuses on creating holes, the primary performance measure is the time required to penetrate the material. Another key performance measure is the surface finish of the metal, which indicates the precision of the hole created.

In summary, conventional drilling methods face significant challenges, particularly in applying excessive pressure on metals, complicating the machining of delicate shapes and thin surfaces. To overcome these hurdles, the group is adopting Electrical Discharge Machining (EDM) technology to develop a precise EDM device with exceptional precision tolerance. Despite a tight nine-week timeline, the group’s efforts are organized into subgroups, focusing on crucial aspects such as electrode and dielectric fluid selection, structural design, and power supply considerations. Material availability and cost constraints are carefully managed to ensure project feasibility. Performance measurement throughout the project will center on time efficiency in material penetration and the quality of surface finishing, key indicators of the EDM device’s effectiveness in surpassing conventional drilling limitations.

1.4. Structure of Thesis

The thesis is structured as follows. Chapter 2 outlines the program of requirements for this project. Chapter 3 details the design for this subgroup, including an overview and a detailed description. Chapter 4 presents the evaluation of the design through experiments and their results. Chapter 5 discusses the project’s progression. Finally, Chapter 6 provides the thesis conclusion and offers recommendations for future work.

2
Program of Requirements

The aim of this thesis is to develop a method for creating precise holes in a variety of metals, including but not limited to steel, aluminium, titanium, and their alloys. This will be achieved by designing an Electrical Discharge Machine (EDM) capable of producing accurate holes. Given the numerous EDM design variations, specific requirements have been established for the overall system. The requirements are split into two sections. The first section covers the technical specifications for the electrode, dielectric fluid, control system, accuracy and precision, machine design, and power supply. This is crucial for the design segment of the thesis since the components must be designed according to these specifications. The second section addresses operational requirements, concentrating on the core functionality of the entire EDM device and the necessary safety features for its operation. The requirements are as follows:

1. **Technical Requirements**
   a) **Electrode Specifications**
      - The electrode must be readily accessible; therefore, rare materials should not be utilized as electrodes.
      - The electrode size should be between 1 and 3 mm.

   b) **Dielectric Fluid Specification**
      - The dielectric fluid must not have a flash point (to avoid fire hazards).
      - The dielectric fluid must not emit toxic fumes.
      - The dielectric fluid should have a low viscosity to ensure good flushing of debris.
      - The dielectric fluid should be electrically non-conductive to avoid short circuits.

   c) **Control system**
      - The control system must have the possibility of manual control.
      - The control system shall have the possibility of open-loop control.
      - The control system should have the possibility of closed-loop control.

   d) **Accuracy and Precision**
      - The system must achieve an accuracy of ± 0.005 mm.

e) Machine Design

- The machine design must be compact. It should be suitable for placement on a desk or similar surface.
- The machine design shall include flushing.
- The machine design should be able to operate along all (x, y, z) dimensions.

f) Power Supply

- The power supply must use a limited power source of 100 [W].
- The power supply must be capable of delivering a variable peak current between 0 to 15 [A] to the workpiece.
- The power supply shall be capable of delivering a variable voltage between 40 and 150 [V] to the workpiece.
- The power supply should be capable of delivering a spark frequency range of 1 to 50 [kHz].
- The power supply should be able to sense the gap voltage between the tool and the workpiece.
- The power supply should preferably quantify the duration of the voltage gap between the tool and the workpiece.

2. **Operational Requirements**

a) **Core Functionality**

- The machine must absolutely make a hole inside a workpiece.
- The system should exhibit a minimum material removal rate (MRR) of 1.0 mm^3/min in Aluminium.

b) **Safety Features**

- The machine shall have an emergency stop button that automatically shuts down all electricity through the machine.
- The machine should have proper shielding to protect the operator from sparks and dielectric splashes.
- The machine should have built-in sensors to detect any malfunction and automatically stop the operation.

3 Design Description

3.1. Overview

The most general overview of the EDM device is presented in Figure 3.1. The mechanical part of the device consists of the electrode, dielectric fluid, 3D printer, flushing system, and general (3D-printable) parts, which make the operation of the device simple and stable. The goal of the mechanical structure is to support the EDM operation and allow for easy and precise processing of conductive materials. The controller is responsible for the movement of the electrode and workpiece in the x-, y-, and z-direction. The EDM device is primarily controlled by the use of an open-loop system, but a closed-loop system is also investigated. The sensing, which is part of the closed-loop system, is done primarily by our colleagues from the power supply subgroup [1]. In the case of a closed-loop system, the input of the controller is the voltage wave that is applied across the electrode and workpiece. Based on the waveform, the controller can determine whether a short circuit is present and adjust the movements of the 3D printer accordingly.
The required voltage and current waveforms are supplied by the power supply. The power supply provides a square voltage wave of which the duty cycle, frequency, voltage level and short circuit current are adjustable. This allows for the possibility to optimise these parameters so that optimal results can be obtained for different workpieces, dielectric fluids, and electrode materials.

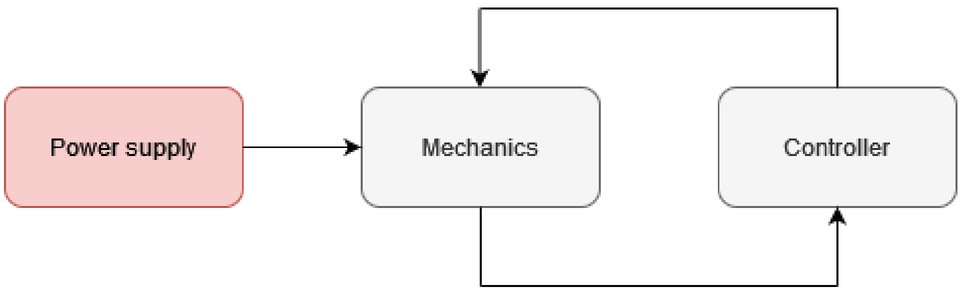
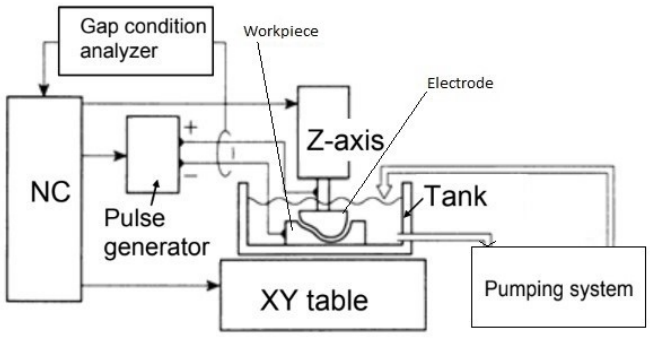


Figure 3.1: General diagram of EDM machine.

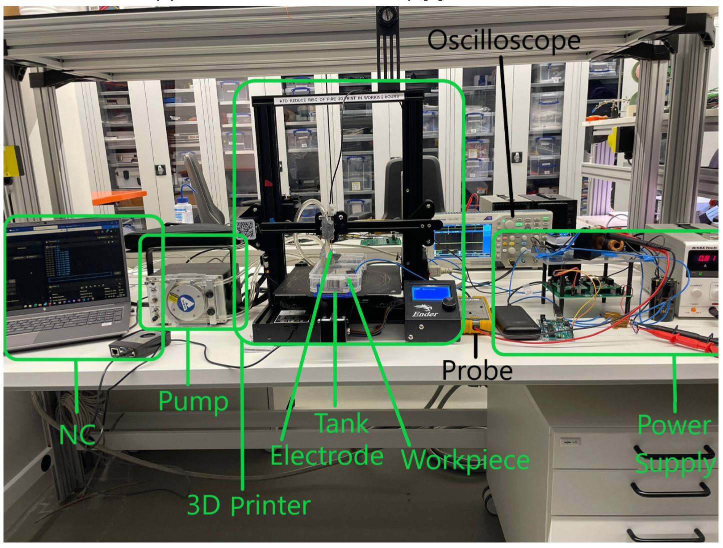
3.2. Mechanics

3.2.1. Overview

The mechanical component of the EDM device is crucial for maintaining stable operation and achieving optimal performance. This section initially presents a comparative analysis of the dielectric medium and electrode. Following that, the key features of the 3D printer are examined. Additionally, the design of a 3D-printable electrode mount is briefly outlined, along with a brief discussion on the flushing system used in the EDM machine. A comprehensive view of how these elements are integrated into a single setup is provided in Figure 3.2. The NC stands for numerical control, where the 3D printer (its motor) and closed-loop control (Raspberry Pi Pico and Raspberry Pi 5) are included.



(a) Schematic of the EDM setup [2].



(b) Real-life implementation of the EDM setup.

Figure 3.2: The EDM setup.

3.2.2. Dielectric Fluid

Dielectrics have a major influence on the performance of an EDM and should, therefore, be chosen carefully. Under normal conditions, the dielectric material acts as an insulator but ionizes into positive and negative ions when a high electric field is applied. This process causes a bridge to form between the electrode and workpiece for current to flow. The heat formed in this process vaporizes part of the workpiece and electrode. A carefully chosen dielectric fluid enhances cooling, ensures effective debris flushing from the spark gap, and influences overall machining efficiency [6]. In this section, different dielectrics are compared in different aspects, such as time, availability, cost, and performance.

Hydrocarbon oils

Hydrocarbon oils, such as transformer oil, mineral oil and kerosene, are the most widely used dielectrics today in the case of die-sinking EDM [7]. Especially kerosene has been used extensively in EDM due to its high dielectric strength, low conductivity, and high breakdown strength [8]. Other EDM types, such as wire EDM, micro-EDM and fast hole drilling EDM, mostly work with deionized water [9]. According to [10], hydrocarbon oils are preferred over water-based dielectrics in terms of performance. The disadvantages of hydrocarbon oils are risks of fire hazards and the release of toxic fumes [8], making them one of the main sources of pollution according to [9]. Another drawback is that flushing using jets, which have been found to increase performance, cannot be used due to an increased risk of fire hazards [9].

Biodiesel

Biodiesel has the same characteristics as conventional dielectrics, but less harmful substances are emitted during processing, and the waste is processed more easily [11]. P. Sadagopan and B. Mouliprasanth [12] investigated the use of biodiesel as dielectric against kerosene and transformer oil. It was found that a higher MRR and lower EWR can be achieved with biodiesel. The same conclusion was found by P. S. Ng et al. [11] when using canola and sunflower biodiesel.

Water-based

A. Erden and D. Temel [13] investigated the performance of tap water, distilled water, salted water, and kerosene. They found that only distilled water performed reasonably well if a brass tool was used. For distilled water, smoother surfaces were obtained, and fire hazards and logistical problems were eliminated. On the other hand, S. Tariq Jilani and P.C. Pandey [14] found that better operating performance could be achieved under specific circumstances with tap water. They suggested that machining instability and short circuits may be the cause of the lower performance achieved by Erden and Temel [13]. S.Tariq Jilani and P.C. Pandey [14] also found that the tool wear rate was not that different between distilled water and tap water. They found that the use of tap water is preferred at low currents and low pulse duration. A pulse duration of 100 μs was found to be optimal.

Current dielectric research

Even higher performance can be achieved by using powders in water-based dielectrics [9]. This is outside of the scope of this report. Other dielectrics, such as gases, are also being investigated. Gaseous dielectrics can have advantages, such as low tool wear and no environmental harm, for certain EDM operations. However, it still faces big challenges due to arcing and debris reattachment on the workpiece [7]. This technique should first be optimised more to be economically viable [9].

Selection of dielectric

The dielectric medium in an Electrical Discharge Machining (EDM) setup should possess several key characteristics that make it suitable for usage. Dielectric strength is crucial as it determines the maximum electric field a medium can withstand before breakdown occurs, thus affecting the time after which a breakdown happens when a potential is applied between the electrode and the workpiece [15]. Under normal conditions, a dielectric should exhibit insulating behaviour. Other important characteristics include a high flash point, good wetting behaviour, appropriate viscosity, and thermal conductivity [16].

The advantages and disadvantages of various dielectrics are summarized in Table 3.2 and Table 3.3. Based on easy accessibility, low cost, and environmentally friendly properties, the most suitable dielectric for the EDM device is deionized water. Additionally, the literature supports the potential of deionized water, and its properties can be enhanced by adding various materials, such as powders. In reality, distilled water was used as it is almost identical to deionized water and was obtainable more easily.

Table 3.1 compares the material properties of deionized water to kerosene. The state of a dielectric can either be liquid or gas, affecting its application and performance. Dielectric strength measures the maximum electric field a material can withstand without breaking down, with higher values indicating better insulation. The dielectric constant shows a material's ability to store electrical energy, whereas higher constants imply better energy storage. Electric conductivity reflects how easily electric current can pass through a material, with lower values indicating better insulation. Viscosity, both kinematic and dynamic, influences the flow and cooling characteristics of the dielectric, with lower viscosity promoting better flow and cooling. Specific heat measures the heat required to raise the material's temperature, with higher specific heat signifying better heat absorption. Thermal conductivity indicates how well a material conducts heat, where higher values mean better heat dissipation. Density affects the weight and buoyancy of the dielectric, making different densities suitable for different applications. Lastly, the boiling point is the temperature at which a material changes from liquid to gas, where higher boiling points prevent vaporization during EDM.

Property	**Hydrocarbon oil (kerosene)** [17] [18]	**Deionized Water** [17]
State	Liquid	Liquid
Dielectric Strength (MV/m)	14-22	13
Dielectric Constant	1.8	80.4
Electric Conductivity (S/cm)	0.015	1.33
Kinematic Viscosity (cm/s)	1.16 $\cdot 10^{-2}$	$0.852 \cdot 10^{-2}$
Dynamic viscosity (g/ms)	1.64	0.92
Specific Heat (J/kgC)	2100	4200
Thermal Conductivity (W/mk)	0.149	0.606
Density (kg/m)	860	1000
Boiling Point (C)	200	100

Table 3.1: Overview of physical properties for dielectrics used in EDM.

Advantages Hydrocarbon	**Disadvantages Hydrocarbon**	**Advantages Deionized water**	**Disadvantages Deionized water**
General good performance	Toxic fumes	Environment-friendly	General lower performance
	Risk of fire hazards	Good performance for specific cases	Energy need for deionization
	Polluted material waste	Easily obtainable	

Table 3.2: Advantages and disadvantages of Hydrocarbon and Deionized water

Advantages Tap water	**Disadvantages Tap water**	**Advantages Biodiesel**	**Disadvantages Biodiesel**
Low cost	General low performance	Good performance	More costly than water
Easily obtainable			Less convienent to work with

Table 3.3: Advantages and disadvantages of tap water and biodiesel

3.2.3. Electrode

Functionality of Electrode for Electrical Discharge Machining (EDM)

The functionality of an electrode in Electrical Discharge Machining (EDM) is to serve as a tool that helps create the desired shape or feature on the workpiece through the process of electrical discharge. The electrode is typically made of conductive materials such as copper, graphite, or tungsten, and it is used to generate sparks that erode the workpiece material. The electrode is connected to a power supply that generates electrical pulses, creating a spark gap between the electrode and the workpiece. As the sparks discharge, material is removed from the workpiece, allowing for precise machining and shaping. The choice of electrode material and design plays a crucial role in the efficiency and accuracy of the EDM process.

Selection of an electrode for Electrical Discharge Machining (EDM)

Several decisions need to be made to select the optimal electrode for the project. Initially, it is important to consider the necessary properties of an electrode to perform effectively in Electrical Discharge Machining (EDM). The two key properties of the electrode material are a high melting point and electrical conductivity. High electrical conductivity is vital for efficient material removal in EDM. Additionally, the electrode's high melting point is necessary to withstand the intense heat produced during EDM.

Other advantageous properties of the electrode include high thermal conductivity, increased density (to reduce tool wear), ease of manufacturing, and cost-effectiveness. High thermal conductivity plays a crucial role in Electrical Discharge Machining (EDM) by aiding in the dissipation of heat generated during the process. EDM involves the use of electrical discharges to eliminate material from a workpiece, resulting in significant heat production. If the material utilized in EDM possesses high thermal conductivity, it can effectively transfer this heat away from the cutting area, preventing overheating and ensuring consistent and precise machining. This leads to enhanced machining efficiency, improved surface finish, and prolonged tool life. On the other hand, increased density is vital for EDM as it can enhance material removal rates and machining effectiveness. Materials with higher density typically exhibit superior mechanical properties, including increased strength and wear resistance. This enables more efficient material removal during the EDM process, leading to quicker machining speeds and reduced tool wear. Moreover, materials with higher density often boast better thermal conductivity, aiding in the dissipation of heat generated during EDM and resulting in more consistent and accurate machining outcomes.

After reviewing multiple research papers on Electrical Discharge Machining (EDM), it was found that a small number of electrodes, namely Copper, Tungsten, Aluminium, and Graphite, were frequently discussed and utilized [19]–[24]. As a result, these electrodes will be examined in this thesis for potential use in the final EDM process. The key characteristics influencing the performance of Electrical Discharge Machining (EDM) are outlined in Table 3.4.

	Electrical conductivity at 20° (S/m)	Liquefaction point (°C)	Heat conductivity $((W/m)*K)$	Density (kg/m^3)
Copper	5.96×10^7	1084	398	8960
Brass	1.67×10^7	930	111	8500
Tungsten	1.79×10^7	3420	164	19250
Graphite	2 to 3×10^5	3600	168	641

Table 3.4: Properties of candidates for the electrode.

It is also beneficial to examine the output parameters of the EDM. This can aid in selecting the most suitable electrode. According to various research papers, the two key performance parameters are Material Removal Rate (MRR) and Tool Wear Rate (TWR) [19], [20], [22]–[24]. These performance metrics are significantly influenced by the input parameters of the EDM. The Material Removal Rate is particularly influenced by the peak current input parameter, followed by pulse duration [23]. On the other hand, the Tool Wear Rate is greatly impacted by the properties of the electrode. Since the outcomes are closely tied to input parameters, the pros and cons of each electrode can be found in research papers. These pros and cons of each electrode are depicted in tables 3.5 and 3.6.

Upon evaluating the advantages and disadvantages of various electrodes, it is clear that copper and graphite are the most effective choices. This trend is also observed in modern EDM practices within large corporations [25]. Ultimately, the choice between graphite and copper electrodes should be determined by the specific needs of the EDM operation, considering factors such as workpiece material, desired machining outcomes, cost, dielectric fluid selection, and overall machining efficiency. The choice of graphite is mainly gone because of the non-compatibility between the graphite and dielectric fluid. Consequently, copper emerges as the best electrode option for this project.

Electrode Shape Design Die-sinking EDM can form various shapes, including those intended to be imprinted onto the workpiece. The project aims to create a hole in a workpiece with a circular shape selected as specified in the requirements. Research into various EDM devices revealed that drilling

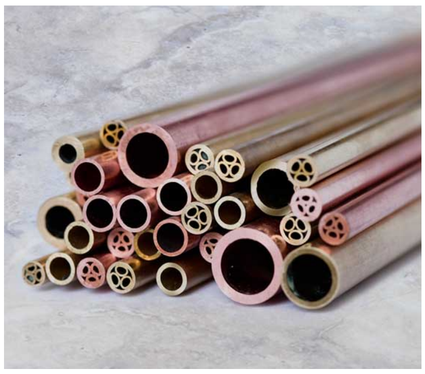
Advantages Copper	Disadvantages Copper	Advantages Graphite	Disadvantages Graphite
Good wear resistance	Low wear resistance compared to Graphite	Cheap Material for long duration EDM applications	Vulnerable to breakage
Highly Conductive	Expensive Material for long duration EDM applications	Good wear resistance	Low Conductivity
Strong metal	Limited Machining Speed	Withstand high temperatures	Limited compatibility with certain dielectric fluids
Resistance to Corrosion	Limited Hardness	Environmentally friendly	Exhibit dimensional changes under high heat
High MMR Large Craters	Poor surface finish	Easy to handle	X

Table 3.5: Advantages and disadvantages of Copper and Graphite.

Advantages Tungsten	Disadvantages Tungsten	Advantages Aluminium	Disadvantages Aluminium
Good Wear Rate	High amount of Brittleness	High MMR Large Craters	Bad wear resistance
Highly Conductive	Difficult to Machine	X	Poor surface finish
Resistant to Erosion	Difficulties with Fine Detailing	X	Low melting point
Good Surface Finish	High Initial Costs	X	Chemical reactive with the dielectric fluid
Consistent in Performance	Poor Combability with certain EDM Machines	X	challenging to machine to precise shapes

Table 3.6: Advantages and disadvantages of Tungsten and Aluminium.

EDM shapes was the most effective method. This shape, illustrated in Figure 3.3a, features a hollow center in the electrode to facilitate the flushing process, where dielectric fluid is injected through the electrode's hole. This electrode shape requires a powerful pump, which was neither readily available nor cost-effective. Therefore, it was opted for a cylindrical electrode taken from copper wires. A possibility is to sharpen the electrode as depicted in Figure 3.3b. The sharp point at the end of the electrode is essential for localised energy concentration. The sharp point concentrates electrical discharges in a small area, leading to more efficient energy use and faster material removal in that localized region. In the immediate area of the point, the MRR can be higher due to the concentrated energy [26]. Eventually, a flat-bottomed electrode was used, because a pointy electrode would loose its effect after erosion from machining.


(a) The drilling electrode shape design [27].

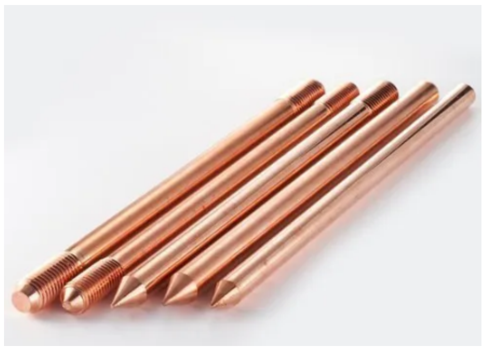

(b) The selected electrode shape design [28].

Figure 3.3: The electrode shape designs that have been considered.

3.2.4. 3D printer

The 3D printer used is the Ender3 Pro from Creality. It uses the 32-bit Creality 4.2.2 board with a GD32F103 chip and NEMA 17 stepper motors to facilitate movement in the x-, y-, and z directions. Movement in the x—and z-direction is achieved by movement of the electrode, while movement of the printer bed controls movement in the y-direction.

The stepper motor rotates with a fixed angle for each received pulse, called the step angle [29]. The step angle for NEMA 17 stepper motors is 1.8° [30]. The addition of a lead screw and the possibility of micro-stepping, which is a feature of stepper motors, provide extra precision in the z-direction. The precision of the movement of the 3D printer, particularly in the z-direction, is of great importance because the spark gap between the electrode and the workpiece should be narrow.

The minimum theoretical step size in the z-axis is calculated using Equation 3.1 and follows the principles of stepper motors.

$$step\ size = \frac{lead\ screw\ pitch}{steps\ per\ revolution \cdot microsteps} \tag{3.1}$$

For the Ender3 Pro, the lead screw pitch is 8 [mm] [31], the steps per revolution are 360°/1.8° = 200, and the amount of microsteps is equal to 16 [32]. Therefore, the theoretical minimum step size in the z-axis is 0.0025 [mm], which is sufficiently accurate. The minimum x- and y-axis step size is 0.0125 [mm].

For the EDM device in this report, it is essential to have low operating speeds since the material removal of EDM machines is generally slow. Stepper motors are controlled by a PWM (Pulse Width Modulated) signal [33]. The greater the density of high pulses, the faster the rotation of the motor. This feature of stepper motors allows for extremely slow machining speeds, as the time between pulses can be made infinitely large.

3.2.5. 3D-printable electrode mount

At the position where the nozzle of the extrusion system was originally located, a 3D-printable mount was designed to hold electrodes up to a diameter of 3 [mm]. The mount was designed in Fusion 360 and is shown in Figure 3.4. The electrode can be pinned down by inserting a screw in the front holes.

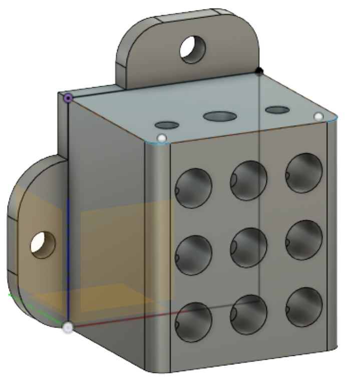


Figure 3.4: Electrode mount drawing in Fusion 360.

3.2.6. Flushing System

The flushing system in Electrical Discharge Machining (EDM) plays a critical role in maintaining stable machining. Flushing serves primarily to evacuate eroded debris particles from the electrode-workpiece gap, which, if not removed, can obstruct the electric discharge path, resulting in unstable machining conditions and increasing the likelihood of short circuits. Moreover, the flushing is boosted by lifting the electrode periodically and replacing the polluted fluid. A flushing angle of 15 degrees relative to the tool axis has been established for minimizing debris accumulation and ensuring the stability of the EDM process [34]. The placement of the pump can be seen in Figure 3.2b.

3.3. Control system

3.3.1. Overview

The movement of the electrode with respect to the workpiece should be controlled accurately and implemented by an external control system. This allows for better control over the 3D printer and the possibility to implement feedback and set up a remote connection to a laptop. A schematic overview of the control system is shown in Figure 3.5.

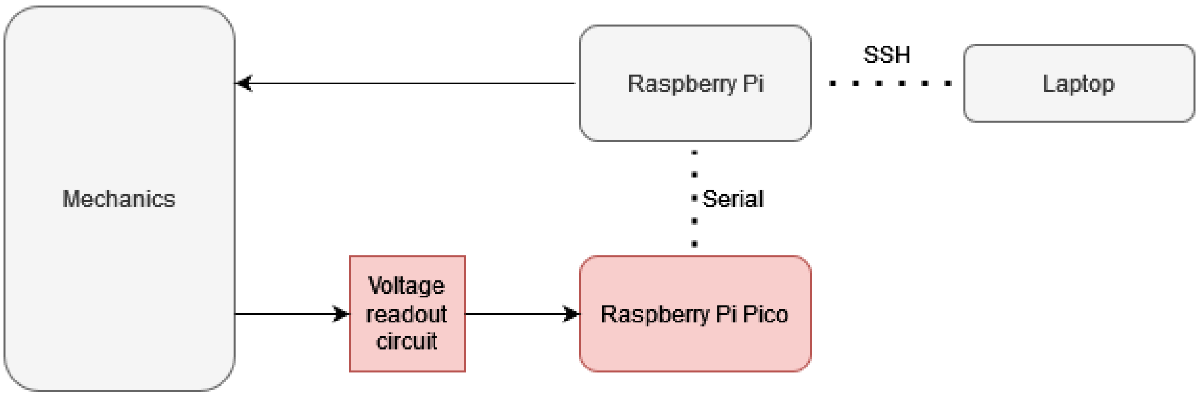


Figure 3.5: General schematic of EDM control system.

The control system consists of a Raspberry Pi 5, Raspberry Pi Pico, and a voltage scaling circuit. The system is remotely connected to a laptop via an SSH connection that allows for remote and manual input. The input, which could be a singular command or complete file, is send to the Raspberry Pi 5 that decodes these commands and sends them to the 3D printer, which translates the commands into mechanical movement.

Closed-loop feedback is implemented by a Raspberry Pi Pico microcontroller and some additional circuitry that is responsible for sensing and scaling the voltage waveform produced by the power supply. Based on the waveform, the Raspberry Pi Pico can detect short circuits, which can be used to alter the movement of the electrode via the feedback network. In order to communicate with the Raspberry Pi, a webserver is hosted on the Raspberry Pi Pico that allows for wireless serial communication. Besides short circuit detection, the Raspberry Pi Pico is also used to control the duty cycle and frequency of the square voltage wave that is applied across the electrode and workpiece. The duty cycle and frequency can also be set via the laptop connected to the Raspberry Pi. These parameters are send to the Raspberry Pi Pico via the bidirectional serial connection. Implementation of the voltage readout circuit and Raspberry Pi Pico are done by our colleagues from the power supply group [1].

This section starts by describing the Raspberry Pi 5 and its connections to the Raspberry Pi Pico, laptop, and 3D printer. Furthermore, installation of Klipper, which is the firmware used for the 3D printer control, is explained. This is followed by a brief overview of the most relevant Klipper source code and any adjustments that have been made to improve the usability of the 3D printer and implement closed-loop feedback. Finally, the usage of the control system is discussed.

3.3.2. Raspberry Pi 5

The Raspberry Pi 5 is a small single-board computer developed by the Raspberry Pi Foundation. Two versions are available with either 4GB or 8GB of memory. In order to operate the device, a micro-SD card should be inserted on which the Klipper software is stored. Some relevant features of the Pi are the availability of 40 I/O pins, 802.11ac Wi-Fi and several USB ports. Figure 3.6 shows a general schematic of the Raspberry Pi and corresponding GPIO pins.

Communication with the Raspberry Pi Pico is done via I2C, which is a communication protocol that uses two buses, namely SDA and SCL. The connection is made by connecting the Pi Pico and Pi 5 with jumper wires. Pin 3 and 5 from Figure 3.6 are used to connect the I2C inputs and pin 6 is used for the ground reference. The connection with the 3D printer is established via a USB port. Power is delivered to the board with a 27W charger that is also provided by Raspberry.

The Raspberry Pi 5 4GB is used to run Klipper, which is the firmware responsible for controlling the 3D printer. Even though other methods might be available, the Raspberry Pi is the recommended host for Klipper, and plenty of information is available about the installment [35]. Next section will delve deeper into the software of the control system.

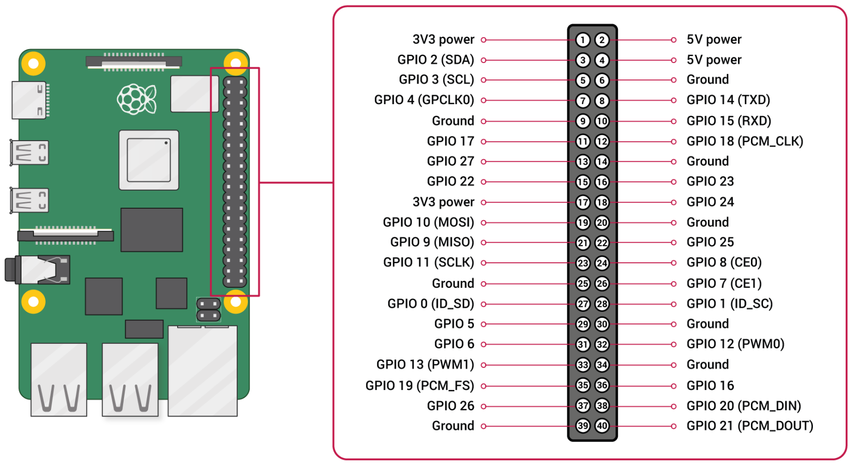


Figure 3.6: GPIO Pinout diagram of Raspberry Pi 5 [36]

3.3.3. Klipper Firmware

The standard Marlin firmware on the printer is unsuitable for external control system use. Instead, the Marlin firmware is replaced with the open-source Klipper firmware, which is fully adjustable to specific needs. The Klipper firmware must be installed on the 3D printer and the Raspberry Pi 5. By setting up a web server on the Raspberry Pi 5, the 3D printer can be controlled remotely via a laptop or desktop. The Moonraker web server API and Fluidd web server interface are also installed to facilitate this. KIAUH is the Klipper installer used for this project and also facilitated the instalment of Moonraker and Fluidd.

Klipper Software Installation

A generic but detailed guide on how to install Klipper is provided by the YouTube channel Vector 3D [37]. This tutorial is used as a basis for installing Klipper on the Raspberry Pi 5. Since the exact setup of the EDM device does not completely correspond with that shown in the video, some general steps and additional information are provided here.

The Raspberry Pi OS Imager is used to install the Raspberry Pi OS on a micro-SD card by using a USB to micro-SD adapter. After installation of the imager, some advanced settings were configured by entering the control, shift, and x keyboard commands. In the settings, SSH should be enabled for remote access, a username and password should be set, and the credentials for a Wi-Fi connection should be entered. Instead of a regular Wi-Fi connection, a hotspot was hosted from a laptop. After editing the advanced settings, the Raspberry Pi OS (32-bit) operating system was flashed onto the micro-SD card, which was then inserted into the Raspberry Pi.

To connect with the Raspberry Pi, the following command should be entered in a command terminal, where the IP address can be found in the settings of the mobile hotspot: **ssh pi@<ip address>**. Logging in uses the username and password entered during the Raspberry Pi OS installation. The following commands are used to install git, install KIAUH, and open KIAUH, respectively [37]:

- **sudo apt-get install git -y**
- **git clone https://github.com/th33xitus/kiauh.git**

- **./kiauh/kiauh.sh**

Installation of Klipper, Moonraker, and Fluidd on the Raspberry Pi via KIAUH is straightforward and done in the same manner as shown in [37], and thus not repeated here. In order for Klipper to work, a config (.cfg) file should be supplied to Klipper that contains information about the specific printer microcontroller board. The configuration file corresponding to the Creality Ender3 Pro is printer-creality-ender3pro-2020.cfg, which is found at [38] and should be renamed to printer.cfg. This file can be supplied to Klipper by uploading it to the configuration section in the Fluidd web interface, which can be accessed by typing the IP address of the Raspberry Pi 5 in a web browser. To finalize, the command *[include fluidd.cf]* should be added to the printer.cfg file [37].

Before the operation of the printer can start, the right configuration should be set up. The configuration of Klipper with KIAUH and Fluidd in [37] is done for the SKR Mini E3 printer board. Due to the difference in 3D printers, the correct configurations for the Ender3 Pro are listed below. With these settings the installation of Klipper on the Raspberry Pi 5 is completed.

- The board type is Creality V4.2.2
- The micro-controller architecture is 'STMicroelectronics STM32'.
- The processor model is STM32F103.
- Disable SWD at startup.
- Bootloader offset is 28 KiB.
- Communication interface is USART1 PA10/PA9.
- The GPIO startup pin does not have to be set.

Installation of Klipper on the printer is simply done by inserting a micro-SD card in the printer micro-controller board (MCU). The micro-SD card should contain the Klipper.bin file from the Klipper GitHub repository [38].

Klipper Code Overview

G-codes are used as commands to specify the movement of the printer and regulate other functionalities, such as movement speed, and fan speed. An extensive list of supported G-code commands by Klipper and their functionalities can be found at [39]. These commands can be inserted in the Fluidd web server interface and are sent via an SSH connection to the Raspberry Pi 5, which hosts the Klipper firmware. Controlling the printer from the Fluidd web interface allows for different methods to send G-code commands to the printer. These are manual controls that involve clicking buttons, sending G-code via a command line, and sending a complete file with numerous G-codes.

Klipper's main function is receiving, processing, and executing G-code commands. Extra functionality can be added by creating additional macros in the printer config file or by directly editing the Klipper source code. Adding macros is a great method that allows for easy implementation of automated tasks, but they have limited flexibility. Therefore, the main approach has been to edit the Klipper source code instead. The Klipper firmware can be edited directly on the Raspberry Pi 5 from the connected laptop. Due to the extensiveness of the Klipper software, only a brief overview will be given about the most important files and functionalities because most code does not have to be altered nor understood for correct operation of the EDM device. The official Klipper source code can be found in the Klipper github repository [38].

The main loop that runs the Klipper program can be found in *klippy.py*. Besides the main loop, the file also takes care of all initializations needed before the operation can start. The *reactor.py* file implements polling mechanisms, which are used to check inputs from the Fluidd webserver interface, for example.

The schematic in Figure 3.7 shows a simplified schematic of the code execution in Klipper. Inputs regarding console commands, restarting Klipper or other inputs from the Fluidd web server are received in the *webhooks.py* file. If inputs are detected by the polling mechanism, then the command is analysed and send to appropriate handler functions. Each handler function contains the code responsible for handling a certain command. For example, the emergency stop command is handled by the *_handle_estop_request* function and gcode command directly typed into the Fluidd console are handled by the *_handle_script* function.

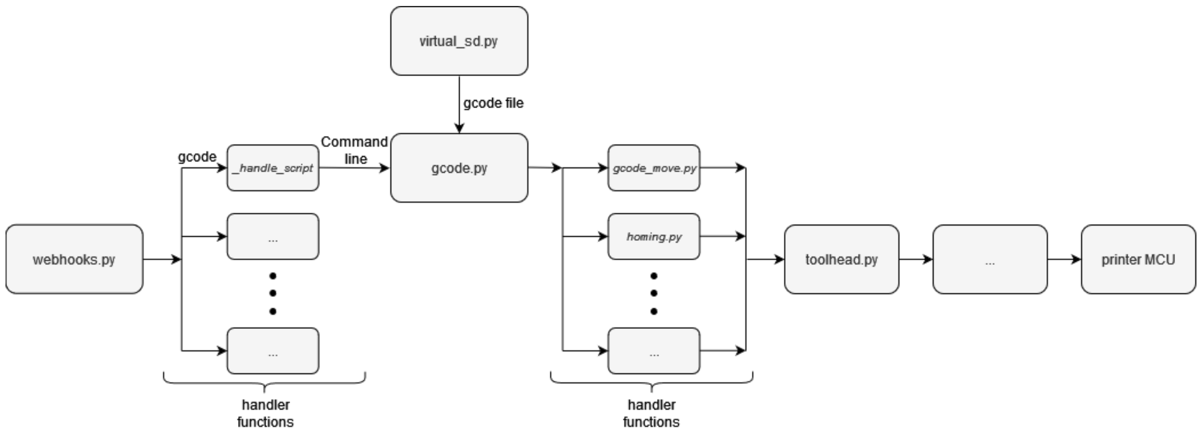


Figure 3.7: Simplified schematic of code execution within the Klipper firmware.

G-code files are processed separately by the *virtual_sd.py* file. G-codes from either the command line or g-code file input are send to *gcode.py*. In this file, the commands are dissected and directed to the appropriate handler function. These handler functions can be found in various files such as *gcode_move.py*, *homing.py*, and *gcode.py* file itself. The G1 command, which is used to control the movement of the printer nozzle (in this case electrode), is processed further in the *gcode_move.py*, while the G28 command is handled in the *homing.py* file. The G28 command is responsible for moving the electrode to its origin position. This code should be executed before any program to help the program calibrate its position.

The *toolhead.py* file tracks the printer movements and calculates acceleration and movement speed by looking at future commands. In the file, a buffer of g-codes is implemented which are used to predict the future movement of the printer for smoother motions. From *toolhead.py*, other files are invoked that further process g-code commands. Based on this, signals are send to the printer microcontroller (MCU), which translates inputs to stepper movements.

Klipper Test Setup

Homing the printer to its origin before each operation is inconvenient, especially during tests. Therefore, the necessity for this command is overwritten by adding additional g-codes to the Klipper software. The G30 and G31 commands are used to set an override variable to 'True' or 'False', respectively. In case the variable is set true, the check for whether the printer is homed before operation is not performed.

The handler function for the G30 and G31 commands are implemented in the *homing.py* file. New command should be registered in order for the code to know what G-code commands can be used. The homing check and override are performed in *cartesian.py*. The modified code can be found in Appendix A.

Closed-loop Feedback

The closed-loop feedback is based on the principle shown in Figure 3.8. If a voltage is applied across the sparkgap, then after a certain delay, the discharge occurs. During the discharge, a bridge is formed between the electrode and workpiece for current to flow. Consequently, the gap voltage decreases significantly. The time it takes for a discharge to occur after a gap voltage is applied is the ignition delay and is a measure of the width of the sparkgap [2]. As shown in Figure 3.8, a short delay corresponds to a small gap width, while a longer delay indicates a large gap. By measuring the voltage waveform across the electrode and workpiece and comparing it to a threshold value, it is determined whether the sparkgap should be larger or smaller. The electrode movement can be adjusted based on this decision. This theory is supported by measurement of a discharge as shown in Figure 4.4 of section 4.3

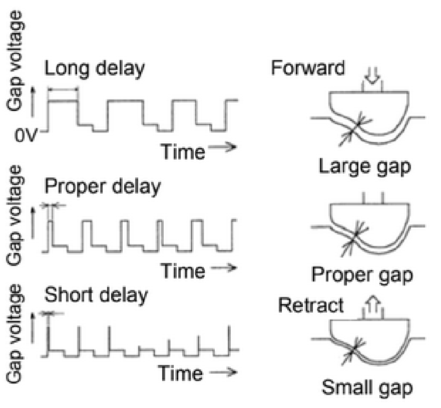


Figure 3.8: Approximate voltage and current profiles during an EDM discharge as documented by [2].

A simpler implementation of this principle is to only detect short circuits or discharges, without including time measurements. This form of closed-loop feedback control is used for the EDM device discussed here, but implementing a more complex control system based on the principle discussed above is also possible. Sensing of the discharge delay is done by the power supply group [1]. By hosting a web server, the Raspberry Pi Pico sends the measurement result to the Raspberry Pi 5 over a wireless serial communication. The Raspberry Pi Pico should be connected to the hotspot set up by the laptop, as discussed in subsection 3.3.3. Some modifications of the Klipper code were made to ensure adequate responds to these inputs.

The *Printer* class in *klippy.py* is modified to contain functions for setting up and closing the connection to the Raspberry Pi Pico as well as sending and reading data. Setting up a connection with the Raspberry Pi Pico is done by invoking the *MakeConnection* function. Inside this function, the IP address of the Raspberry Pi Pico should be entered as well as the port number, which is set by the code on the Raspberry Pi Pico. The *writeCommand* function is used to send data to the Raspberry Pi Pico. Sending data to the Raspberry Pi Pico is not required for implementing the closed-loop feedback, but has been implemented so that the frequency and duty cycle of the voltage wave can be set with a single G-code command, which greatly improves testability and adaptability of the system. The functions *readCommand* and *CloseConnection* are used for reading data and closing the connection, respectively.

Each of the functions regarding the connection between the Raspberry Pi Pico and Raspberry Pi 5, except for the *readCommand* function, are invoked by newly specified G-code command. These commands are implemented in the *gcode.py* file where other commands are handled as well. The M60 command is implemented to invoke the *MakeConnection* function. Likewise, M61 and M62 invoke the *readCommand* and *writeCommand* functions, respectively.

The actual response mechanism for the closed-loop feedback is implemented in *virtual_sd.py*. By opening a new thread, the communication can be regularly polled without interrupting the normal code execution. The communication is polled every hundredth of a second by invoking the *readCommand*, which is discussed above. If a short circuit is detected, a G-code command is added to the already existing G-code command list. This command can insert a pause in case multiple discharges are measured or a command that enables upward movement of the electrode in case of short circuit detection.

Finally, to reduce delays between short circuit or discharge detection and execution of the modified electrode path the G-code command buffer in the *toolhead.py* file should be reduced. The buffer is implemented to allow for better prediction of future printer movements and thus smoother operation. The standard settings introduce a delay of approximately 2-3 [s], which is undesired for precise control of the EDM device. Therefore, the buffer length is shortened.

To conclude, the Klipper software modifications discussed above regarding the closed-loop feedback integration have been implemented succesfully. The modified code parts can be found in Appendix B. The location of the added code could still be improved for better code structure.

Operation

Klipper handles all G-code commands sequentially, which poses a challenge on creating an optimal control system. Pausing the printer by inserting a pause command requires the last processed G-code command to be completely executed first. The related time delay can be minimised to insert G-code files where each movement is as big as the smallest step size, which is 0.0025 [mm] in the z-direction. By implementing this strategy, a constant downward movement of the printer can be achieved with quick adjustments in case either short circuits or multiple discharges are detected.

3.4. Power Supply

3.4.1. Overview

The power supply delivers the required voltage waveform to the electrode and workpiece. The required voltage waveform is a square wave, of which the duty cycle, frequency, voltage, and current should be controllable. Controlling these parameters is necessary to find the optimal set of parameters to achieve optimal performance.

The power supply comprises of three distinct electronic circuits: a DC power source, a power amplification circuit, and a square wave generator. The power amplification circuit is designed to provide a substantial amount of DC voltage and current essential for operation. The square wave generator is responsible for producing the waveform required by the EDM device. The block diagram presented in Figure 3.9 displays the interconnections among all components. The real-life implementation is visible in Figure 3.10. The board visible with the blue tape is the power amplification circuit. The board below is the square wave generator circuit. These circuits are all explained in detail throughout this chapter.

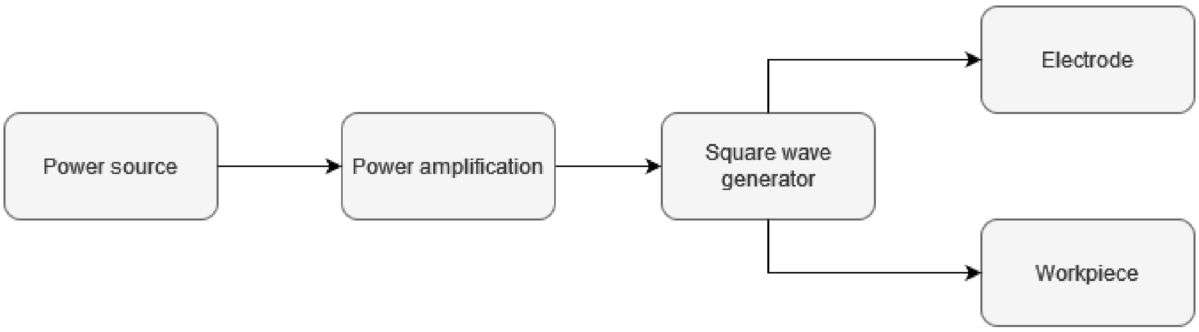


Figure 3.9: General schematic of Power Supply

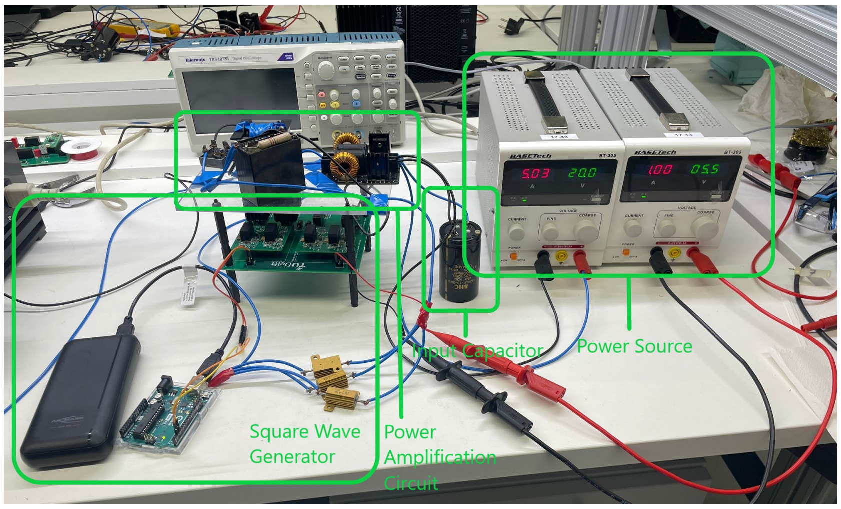


Figure 3.10: Real-life implementation of the power supply.

3.4.2. Power Source

To avoid unnecessary complexity, a pre-existing voltage— and current-controllable power source, the BoseTech 305, is used. This source can provide a DC voltage ranging from 0 to 30 [V] and a DC current ranging from 0 to 5 [A]. As per the requirements, the maximum voltage that was used during operation is 20 [V] to ensure a maximum power of 100 [W] consumed by the system. Also, increasing the voltage further introduced unpleasant sounds in the transformer, which is discussed later.

The power source circuit is depicted in Figure 3.11. This figure shows that inside of the BoseTech 305, a capacitor is present. However, this capacitor seemed not to deliver enough energy to the circuit as it would consistently limit the current to the workpiece. Besides, a built-in short circuit detection circuit required the power source to be turned off and on, thus preventing the automatic operation of the EDM. Thus, an additional (polarised) capacitor was added. This capacitor is relatively large and is also used to stabilise the power source at low voltages and prevent short currents inside the power source. Additional benefits of the capacitor are:

- **Voltage Smoothing:** The capacitor smooths out voltage fluctuations, essential for maintaining a stable output voltage. This is achieved as the capacitor charges during peaks and discharges during dips in the input voltage, formulated as:

  $$V_{\text{out}} = \frac{1}{RC}\int V_{\text{in}}(t)dt$$

  where $V_{\text{in}}(t)$ is the input voltage as a function of time, and RC is the time constant of the circuit.

- **Energy Storage:** A substantial capacitance is necessary to store sufficient energy, which is calculated using the formula:

  $$E = \frac{1}{2}CV^2$$

  Due to a relatively low voltage from the power source, a large capacitor is required to store enough energy, so that the short circuit protection is not triggered in case large currents are required at the output.

- **Pulse Handling:** The capacitor can quickly release stored energy, supporting circuits that require high power pulses. This capability is especially useful in applications needing quick, substantial energy releases, where the discharge current is given by:

  $$I = C\frac{dV}{dt}$$

  where dV/dt is the rate of voltage change across the capacitor.

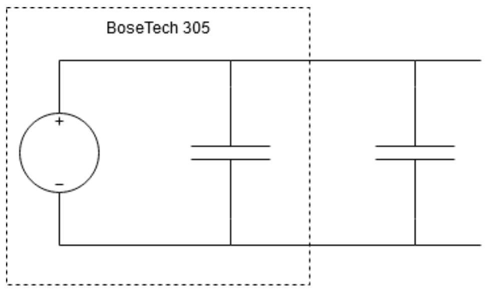


Figure 3.11: Schematic of power source plus additional capacitor.

The capacitor within the power source is approximately 2.2 [μF], and the capacitor connected to the power source is around 1.5 [mF]. The value was chosen such that no limitations were experienced by the power source. Furthermore, an even larger capacitor was not used due to safety concerns, as using an excessively large capacitor can be extremely hazardous.

3.4.3. Power Amplification Circuit

As the power source's specifications fall short of the necessary requirements, a power amplification circuit has been constructed that allows for short-duration and high-energy pulses during which the current far exceeds the values delivered by the DC power source. Besides, higher voltages are reached as well, because that is required to set the dielectric breakdown phenomenon in motion.

The exact breakdown voltage level of distilled water is unknown for the specific set-up used in this report because it depends on many factors, such as electrode geometry and material and the purity of the water, among others [40]. The optimal value is best found experimentally, as also done in [41]. The output voltage of the EDM design discussed here is based on literature and observations in the experiments, which are discussed in chapter 4. Values that are often used are around 50 to 100+ [V] [41]–[43]. A value of approximately 140 [V] has been used in the EDM device discussed here, but the optimal value remains unknown and should be found through experiments. The output current is determined based on the power amplification circuit and a limiting resistor, which is discussed together with the square wave generation circuit in subsection 3.4.4.

The power amplification circuit includes several key components: a DC-to-AC converter, a transformer, a rectifier, and a capacitor with a resistor in parallel. The schematic of the circuit is visible in Figure 3.12. The input of this circuit is connected to the output of the power source, and the output of this circuit is connected to the input of the square wave generation circuit. This circuit will explained in detail by going through each component separately.

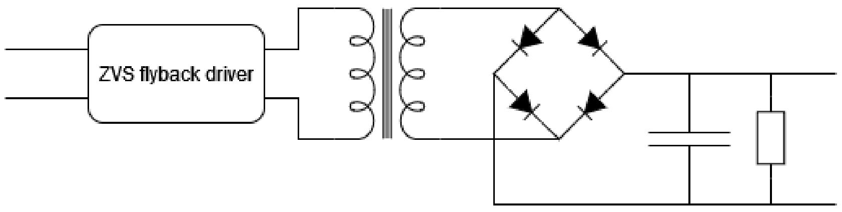


Figure 3.12: Power amplification schematic

The first component, the ZVS high-voltage flyback driver board, serves as a DC-to-AC converter. It operates within a voltage range of 10-30 [V] and produces an AC peak-to-peak voltage equal to the input DC voltage multiplied by pi.

The second component, a custom-made transformer that raises the voltage to 140[V], requires the AC waveform. The transformer features 4 primary and 10 secondary windings, which amplify the AC voltage by a factor of 2.5 and concurrently reduce the current by the same factor.

The third component, known as the KBPC5010 rectifier, is notable for its capability to handle voltages of up to 1000V and currents of 50A. Its primary function within the circuit is to convert the AC voltage and current back to their DC counterparts. Given the circuit's requirement for DC voltage and current delivery, this component assumes significant importance.

The last stage of the power amplification circuit consists of an RC network. The main function of the capacitor is to quickly provide the energy needed during discharges, essentially performing the same function as the capacitor that was placed after the power source. Also, the additional benefits of a capacitor mentioned before in subsection 3.4.2 are also applicable here. The parallel resistance allows for faster discharge after the operation has finished and is thus mostly implemented for safety reasons. Based on the requirement that the power source cannot deliver more than 5 [A], the capacitor was fitted to meet this requirement and is chosen to be 40 [μF]. The requirements imposed on the resistor are that it should allow for faster discharging after operation, but it must not influence the output current much during operation. A value of 1.2 [MΩ] for the resistor was found to be sufficient. The complete calculation and test on the discharge times are found in Appendix C.4.

3.4.4. Square Wave Generation Circuit

The square wave generation circuit employs a main board with an H-bridge (shown in Figure C.6) configuration that consists of four gate drivers (shown in Figure C.5) and four MOSFETs, controlled by an Arduino Uno. A high-level overview of the square wave generation circuit is shown in Figure 3.13. In the figure, V_{dd} corresponds to the positive output voltage of the power amplification circuit and GND to the negative output voltage.

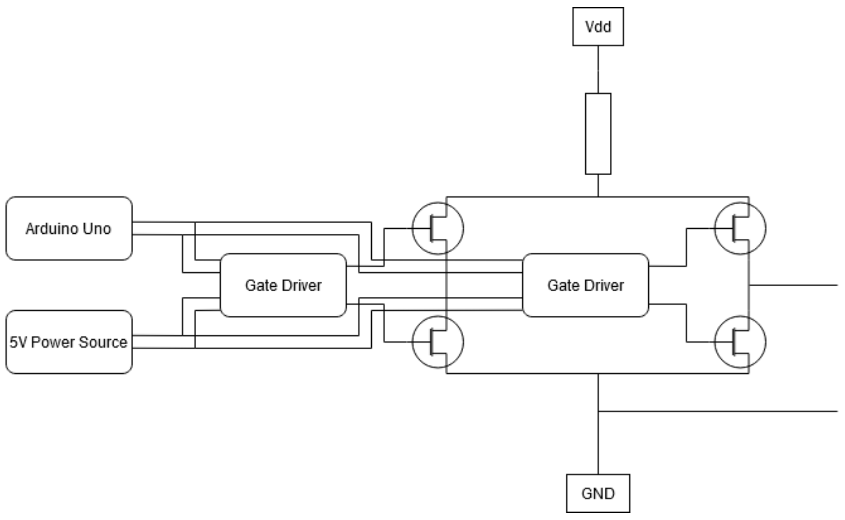


Figure 3.13: Square Wave Generation schematic

The Arduino is programmed to precisely control the timings for turning the MOSFETs on and off, denoted as t_{on} and t_{off}, outputting two 5 [V] signals that correspond to these timings. The gate drivers receive the 5 [V] signals from the Arduino and amplify them to 15 [V] square waves, maintaining the same t_{on} and t_{off} timings set by the Arduino. These enhanced signals are then used to control the gates of the MOSFETs. From the four MOSFETs of the H-bridge, only two are required for the EDM. That is because two MOSFETs are responsible for a positive pulse and two for negative pulses. In the EDM device, only the positive pulses are used, while in the t_{off} period, the output should be zero. Additionally, to ensure controlled operation and to prevent excessive current flow, which could damage the components, a current limiting resistor is integrated into the circuit to limit the maximum current passing through the MOSFETs, thereby safeguarding the circuit from overcurrent conditions. The power resistor is positioned between V_{dd} and the MOSFETs. Its value is crucial because it must limit the current to prevent damaging the MOSFETs, yet it needs to be as low as possible to maximize the current delivered to the workpiece. This current is significant because it affects the energy of the sparks and, consequently, the material removal rate [43]. The optimal value for the power resistor is 2.4 [Ω] and has been determined through experiments, which are found in section 4.4.

As becomes clear from the experiments (see Figure 4.4 in section 4.3), the voltage over the gap showed pronounced oscillations during switching activities. The presence of these oscillations results mostly from the MOSFETs and stray inductances, as was investigated by [1]. The influence of these oscillations on the sparks remains unknown.

3.4.5. Energy efficiency

During the experiments, attention was given to find a proper duty cycle to achieve a balance between power efficiency and effective machining. The aim was to establish an energy balance where the high-voltage side consistently supplied a controlled voltage across a current-limiting resistor and the spark gap.

The energy per pulse E_{pulse} can be calculated as follows, assuming the voltage across the gap when it is conducting:

$$E_{\text{pulse}} = \frac{V^2}{R} \cdot t_{\text{on}}$$

where V is the applied voltage, R is the resistance of the current limiter and the spark gap, and t_{on} is the duration of the "on" time. This calculation indicates that the energy per pulse is directly proportional to the square of the voltage and the duration of the on-time while being inversely proportional to the resistance.

The input energy to the system, provided by the power source over one complete cycle, is expressed as:

$$E_{\text{input}} = V_{\text{power source}} \times I_{\text{eff}} \times (t_{\text{on}} + t_{\text{off}})$$

Where $V_{\text{power source}}$ is the voltage of the power source and I_{eff} is the effective current, accounting for total current supplied minus current lost due to consumption by various components. For system equilibrium, the energy supplied per cycle by the power source should match the energy utilized in sparking:

$$\frac{V^2}{R} \cdot t_{\text{on}} = V_{\text{power source}} \times I_{\text{eff}} \times (t_{\text{on}} + t_{\text{off}})$$

From experimental tests it was concluded that adjustments of the duty cycle and frequency parameters are crucial for enhancing the EDM process efficiency. By fine-tuning these settings, a significant influence on the machining performance, specifically in terms of spark consistency and material removal rates, is achieved. Thus, a well-calibrated duty cycle and frequency ensure optimal power usage and enhance the EDM operation's overall stability and reliability.

As discussed by the authors in [44], the material removal rate (MRR) is significantly influenced by the pulse duration during the EDM process. Longer pulse durations lead to a spread in the diameter of the discharge column, which lowers the energy density at the discharge spot on the workpiece surface. This insufficient electrical energy density fails to effectively melt and vaporize the material, resulting in a decrease in MRR. Conversely, excessively short pulse durations do not provide enough electrical discharge energy to the machining gap. This leads to minimal material removal, especially for materials like cemented tungsten carbides with high melting points. Therefore, optimizing the energy density is crucial for increasing MRR.

Considering these insights, a pulse duration of 10 [μs] with an off time of 500 [μs] was found to balance the energy density effectively, aiming to optimize the MRR for materials with high melting points. More extensive analysis should be done to further optimize these values, but operation using the values above have been validated to be satisfactory in experiment 3 of section 4.4.

4 Experimental Evaluation

4.1. Overview

The Electrical Discharge Machining (EDM) process involves numerous components that require thorough testing. To ensure comprehensive evaluation, a series of experiments is conducted. The first experiment serves as the baseline, providing essential data. Subsequent experiments build upon the findings of the previous ones, progressively refining the process. This iterative experimentation is designed to reach the point of developing the final version of the EDM system, which is the ultimate objective of this project. By systematically enhancing each aspect of the EDM through careful testing and analysis, the project aims to achieve optimal performance and reliability.

4.2. Experiment 1: Observation of Discharge Spark

The primary objective of the first experiment is to observe a discharge spark between the tool and the workpiece. This involves creating a power supply and establishing the basic setup of the EDM system. Notably, this initial configuration lacks both flushing mechanisms and mechanical control systems. The focus at this stage is on confirming the basic functionality of spark generation, which will serve as a foundation for further experimentation.

4.2.1. Materials and Equipment

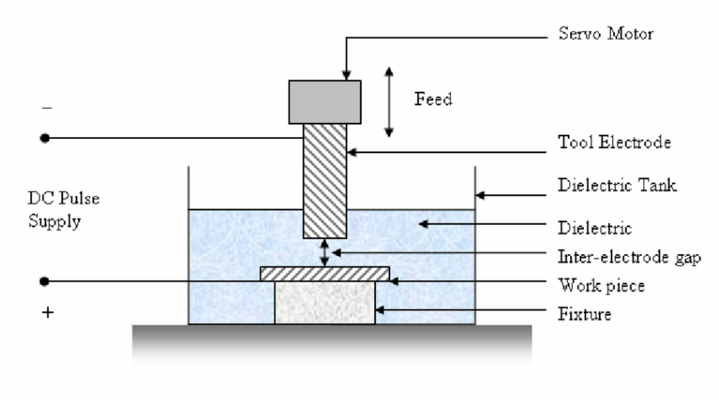
A variety of materials are required to set up the experiment. These materials have been chosen based on their quality and equipment availability at the university. The necessary materials and equipment for the experiment include:

- Deionized Water
- Copper Rod
- Aluminum Plate
- Plastic Bucket
- Manual load frame
- Power Supply
- Power Source
- Connection Cables
- Arduino
- Laptop
- Oscilloscope
- Differential Probes

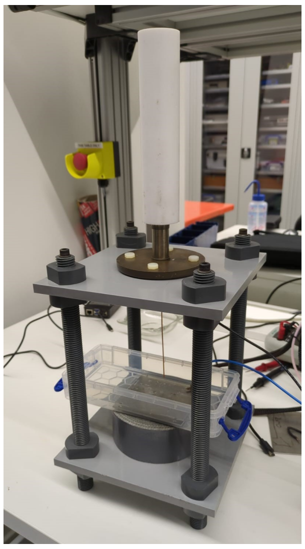
The deionized water functions as the dielectric fluid, the aluminium plate is the workpiece, and the tungsten rod is the electrode. The manual load frame has the efficiency of moving the electrode towards the workpiece in micrometers. The laptop is needed to program the Arduino, creating the wanted square pulses. Finally, the differential probes will be connected to nodes inside the circuit to view the performance of the power supply.

4.2.2. Experiment Setup

The experiment setup is divided into two sections. The first section is the EDM setup, which includes the manual load frame, electrode, and a bucket filled with dielectric fluid, as illustrated in Figure 4.1. The second section is the power supply, which needs to be connected to the workpiece and electrode. The Arduino of the power supply is connected to the computer. By manually rotating the manual load frame, the electrode is moved towards the workpiece with micrometer precision, until the spark gap is sufficiently small to create discharges.



(a) Experiment 1: EDM setup schematic [45].



(b) Experiment 1: Real-Life implementation of the EDM setup.

Figure 4.1: The EDM setup used for experiment 1.

4.2.3. Results

The results of this experiment are largely qualitative, as the primary objective was to observe the occurrence of sparks. This objective was met, and various types of sparks, ranging from very weak to strong, were observed. These variations were influenced by changing the parameters of the square wave used. These variations were mostly on the duty cycle of the square wave made by the power supply.

The hole created in the workpiece was not satisfactory because the manual load frame did not keep the electrode straight, allowing it to move in small circles. The outcome of the hole in the workpiece is depicted in Figure 4.2.

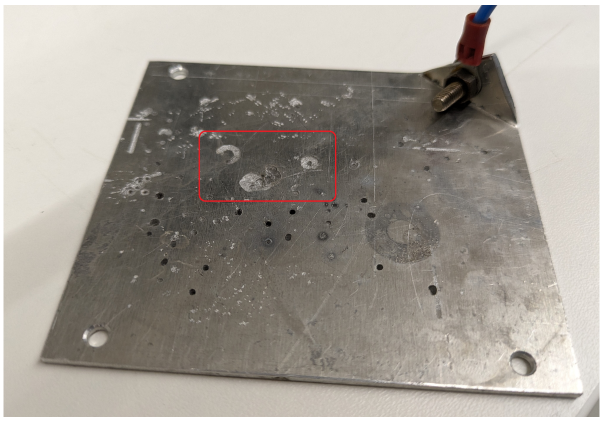


Figure 4.2: Experiment 1: The created hole inside the workpiece encircled by the red square.

4.3. Experiment 2: Incorporation of 3D printer and Flushing

The main goal of this experiment is to complete the setup of the EDM device with the integration of a 3D printer. This addition is expected to enhance the control of the electrode movement, both open-loop and closed-loop. These controls facilitate precise spark generation between the tool and the workpiece, thus effectively creating a hole in the workpiece. Given the important role of effective debris removal in EDM processes, integrating a flushing system into the setup is also deemed essential.

4.3.1. Materials and Equipment

This experiment builds upon the setup from experiment 1 but introduces modifications to accomplish more accuracy with an Ender3 Pro 3D printer. While many materials from the initial experiment are reused, additional components for the control and flushing for this experiment are as follows:

- Raspberry Pi 5
- Raspberry Pi Pico
- 20K potentiometer
- Aalborg TPU AD
- Ender 3 Pro
- Electrode mount on the 3D printer

4.3.2. Experiment Setup

The configuration closely mirrors that of experiment 1. However, there are some notable changes. The manual frame has been substituted with the Ender3 Pro 3D printer. Consequently, the electrode is now secured to the 3D printer via an electrode mount. The 3D printer is linked to the Raspberry Pi 5 and a laptop. Additionally, a connection must be established between the 20K potentiometer that is mounted on the Raspberry Pi Pico and the Raspberry Pi 5. Also, the Aalborg TPU AD water pump is connected to the EDM setup. This pump, connected through the electrode mount, ensures flushing occurs at approximately a 15-degree angle into the gap between the tool and the workpiece. The schematic and real-life implementation of the setup is shown in Figure 3.2.

4.3.3. Results

In this experiment, the incorporation of the 3D printer allowed for precise control over the electrode movements in the EDM setup. This capability resulted in the successful creation of a hole in an aluminium workpiece within 4 minutes. The impact of the flushing mechanism on the process efficiency and accuracy has yet to be assessed. The hole can be seen in figure 4.3.

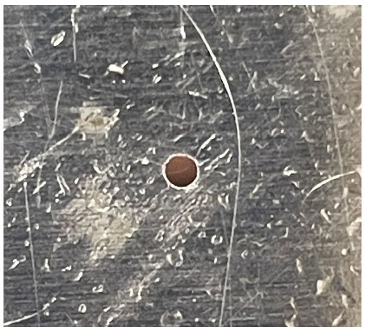


Figure 4.3: Experiment 2: The created hole inside the workpiece.

For the control mechanisms, the open-loop operation was managed by sending multiple G-code commands directly to the 3D printer. Even though accurate holes could be made, there were still challenges regarding short circuits. The lack of an automated closed-loop control system made it difficult to deal with the prevention of short circuits between tool and electrode, except for manually altering the electrode heights or setting the movement speed of the electrode extremely low. Closed-loop control was implemented through a combination of sending G-commands and visually monitoring the status of the discharge. Observations included the current of the power source, the frequency of discharges, and the movement of the electrode. Adjusting the movement speed via a potentiometer, as explained in subsection 3.3.3, worked as intended, except for a relatively large delay between input and output. This was, however, not tested in combination with creating discharges.

Another important conclusion could be drawn for the closed-loop feedback system. The phenomena on which the theoretical control system discussed in subsection 3.3.3 was based could be measured on the oscilloscope. Figure 4.4 shows that the discharge happens after approximately 4 [μs], after which the current rises and voltage drops.

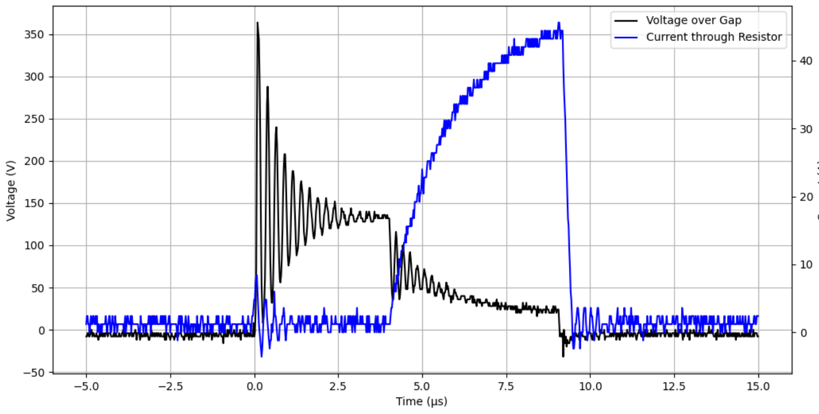


Figure 4.4: Measured voltage and current waveform during a discharge. The discharge can be seen after about 4 [μs].

4.4. Experiment 3: Increasing Discharge Energy

To further improve the performance of the EDM, the influence of the current limiting power resistor is examined. This is important because the power source only delivers 100 [W] at maximum. The initial resistance value of 3.33 [Ω] is compared to resistor values of 2.4 [Ω] and 1.6 [Ω]. This was assessed by measuring the current through the resistances with a special current probe. Also, some qualitative tests were performed to see the impact of discharges on the capacitor charge.

Except for the resistor values, the materials and measurement setup was the same as in experiment 2, with the power supply set to constant parameters: 140 [V] DC and a square wave with a t_{on} of 10 [μs] and t_{off} of 500 [μs].

In case of a short circuit between the electrode and the workpiece, overcurrent must not happen. Therefore, by ignoring the inductance that is present in the power resistor and assuming only the resistance of the current-limiting power resistor, a rough estimation could be made about the maximum current. The current should be below the rated pulse current of the MOSFETs. For 3.33 [Ω], this would be 140/3.3 = 42.4 [A], and for 2.4 [Ω] and 1.6 [Ω], 58.3 [A] and 87.5 [A], respectively. The 140 [V] is the maximum voltage across the output capacitor. From the datasheet of the MOSFET, the voltage-current relation can be found for pulsed signals. The corresponding figure can be found in Appendix D. For a t_{on} of 10 [μs], no data is available, so the limits were tested experimentally using the estimations above.

4.4.1. Results

The current measurements for resistances of 2.4 [Ω] and 3.3 [Ω] are shown in Figure 4.5. The corresponding current levels were 36 [A] and 47 [A], respectively. The discrepancy between the theoretical and experimental currents is due to the parasitic inductance of the power resistors. This parasitic inductance increases the impedance, leading to a lower current than predicted by theoretical calculations. Inserting a resistance of 1.6 [Ω] resulted in a current of approximately 70 [A], which could not be recorded because the MOSFET was broken down as a result.

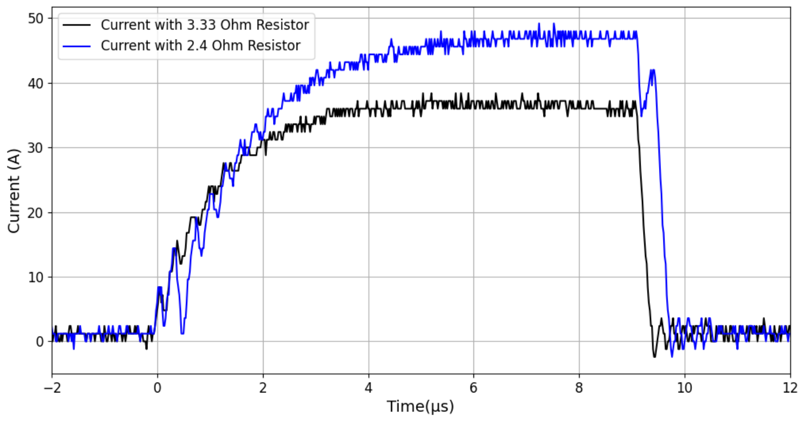


Figure 4.5: Current measurements for a current limiting resistance of 3.33 [Ω] and 2.4 [Ω].

In the plot shown in Figure 4.6, the voltage across the capacitor demonstrated a decrease during short circuit events, indicating that energy stored in the capacitor was being released. After multiple discharges, the voltage across the capacitor stabilized, suggesting that equilibrium was reached where the energy utilized by the resistor and discharge equalled the energy supplied by the power source. This observation supports the theoretical principles concerning the discharge timing parameters t_{on} and t_{off} (subsection 3.4.5).

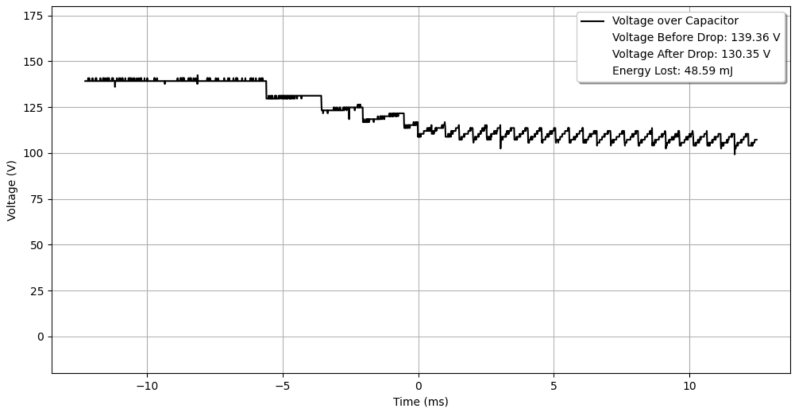


Figure 4.6: Voltage over capacitor during discharges, illustrating the energy dynamics and stabilization.

To enhance the material removal rate (MRR) and discharge energy, one potential approach involves reducing the resistance of the current-limiting resistors. This modification would increase the current and subsequently the energy across the discharge gap. However, the current MOSFETs in use cannot handle these increased current levels.

4.5. Experiment 4: Integration of Power Supply

Following the observation of an enhanced MRR with increased current, a new experiment was initiated using electronic components from another subgroup [1]. This group employed an IPW65R037C6 MOSFET, maintaining a turn-on time (t_{on}) of 10 microseconds. According to the data sheet, this MOSFET can handle a maximum current of approximately 297A at a voltage range of 30-90V. (see Appendix E). Consequently, the transformer was removed and a current limiting resistor of approximately 0.25 [Ω] was installed. This adjustment theoretically allowed for the possibility of reaching the maximum current of 300A. However, in practical scenarios, achieving this current is unlikely due to other circuit resistances.

The integration of the power supply from [1] also enabled the ability to test and implement the closed-loop feedback system.

4.6. Results

There was a noticeable improvement in MRR based on visual inspection, and the circuit components remained intact without any signs of failure or overheating. Unfortunately, due to time constraints, quantitative measurements could not be recorded. One significant issue encountered was the loss of connectivity between the Raspberry Pi and the Creality microcontroller, caused by high current discharges. This necessitated manual operation of the experiments. Despite extensive grounding measures, including the 3D printer, a metallic tank under the workpiece, and surrounding the cables and electronics with aluminum foil as shown in Figure 4.7, the issue persisted. Despite these challenges, a hole was quickly created in the workpiece by manually moving the workpiece to the electrode, leading to the experiment being deemed successful. The outcome of the hole in the workpiece is depicted in Figure 4.8. The removal of the indicated metal part was done in a time scale of approximately 300 [s].

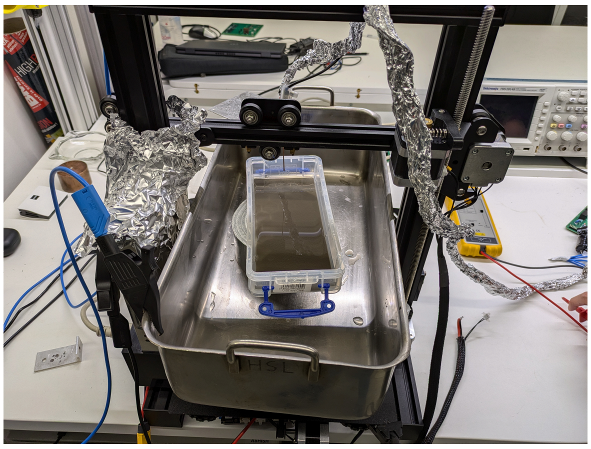


Figure 4.7: 3D printer setup with measures that unsuccessfully prevent problems regarding interference.

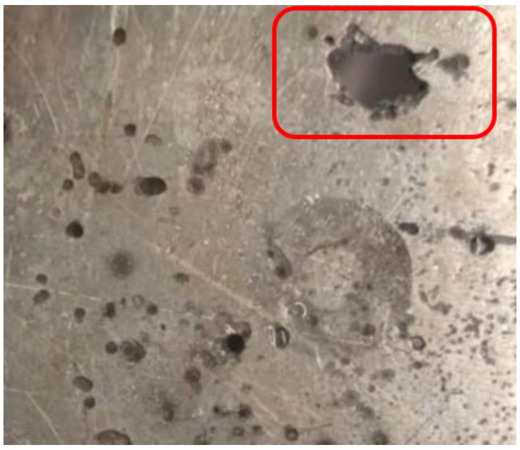


Figure 4.8: The created hole inside the workpiece encircled by the red square.

Loss of connectivity due to interference also obstructed comprehensive testing of the closed-loop feedback system. Even though automatic control of the printer was mostly not possible, adjusted movement of the printer based on a detected discharge was observed. Also, changing the frequency and duty cycle of the voltage signal by inserting G-codes was found to be successful.

5 Discussion

5.1. Challenges

The project progressed at an exceptionally rapid pace, from the literature review to the development of the EDM device and the creation of the first holes. Despite the enjoyment, the group faced numerous difficulties and technical challenges. The initial problem involved developing a power supply for testing. While the power supply group focused entirely on creating a dedicated power supply, our group had to develop one quickly to test critical aspects such as the mechanical system and the overall EDM system design. This task consumed a lot of valuable time, although it provided us with some valuable insights.

The initial circuit had several issues, primarily due to missing components essential for its functionality. The first issue was the exclusion of current-limiting power resistors, which led to the failure of transistors and gate drivers. Once this was addressed, the second problem was the breakdown of the transformer. Figure 5.1 shows the damaged transformer, which was not immediately apparent and likely resulted from excessive heat. The third issue involved the square wave generation; the circuit was supposed to operate at 5 [V] but required at least 5.3 [V] to function correctly. Numerous issues that arose each took some time to resolve.

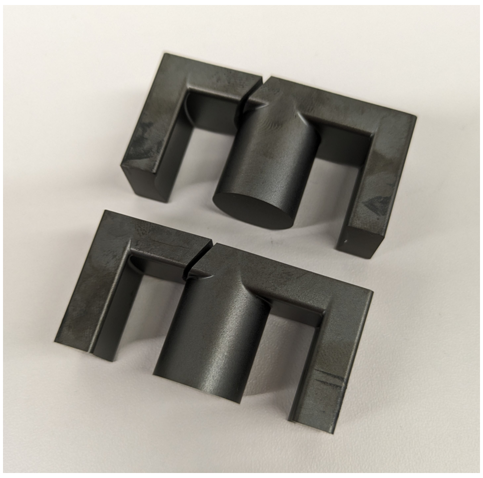


Figure 5.1: Broken transformer, likely due to excessive heat.

Significant obstruction of the project was due to the problem of interference between the applied power and the microcontroller board of the 3D printer. When the applied current was limited at the beginning of the project, this could be prevented by simply moving the electrode and workpiece away from the microcontroller. However, after integration with the power supply group, the current was increased significantly. Consequently, the interference became irrepressible, even after numerous attempts to minimize it. For example, packing the wires in grounded aluminium foil and adding a grounded metallic pan did not prevent interference, and it still remains an unsolved problem. As a result of this obstruction, the ability to test the closed-loop feedback system was also limited.

Furthermore, the EDM device faces some small mechanical challenges. Small vibrations can already negatively affect the EDM operation due to short circuits between the tool and electrode. This problem was primarily caused by the lack of a proper mechanical structure to support the bath with workpiece and dielectric fluid.

The final and most critical challenge was the limited time available. With numerous interconnected parameters to test, the group could not develop the optimal EDM device. Achieving this would require extensive testing and continuous improvements. Each week brought new changes or enhancements, making it an ongoing challenge. Additionally, issues with the power supply further delayed progress and made consistent weekly testing difficult, as these problems had to be resolved before any new tests could be conducted. Despite these obstacles, the group made significant strides, although the time constraints prevented reaching the desired level of optimization.

5.2. Validation of Requirements

Reflecting back at the requirements that were introduced in chapter 2, it can be stated that the requirements for the electrode, dielectric fluid, and accuracy and precision were met successfully. The requirements for the control system are not fully satisfied, although the most important requirements regarding manual control and open-loop control are considered to be achieved. The ability of closed-loop control was investigated and partially tested but not physically implemented. In the end, open-loop control was not executable due to inference, but it has been observed to work before.

The requirements regarding machine design were only partly met. Even though compact packaging of the used components is not yet done, it is possible to place and operate the device on a desk or similar surface. Operation in the (x, y, z) dimensions has been observed to be possible but is not further investigated in this report. The inclusion of flushing has been briefly tested as well, but too little has been reported to report significant results. Eventually, flushing has been left out because other tasks were prioritized.

The power supply requirements were met by our self-created power supply and the power supply designed by [1]. With only extracting a maximum of 100 [W] from the grid, fast metal removal has been achieved by manual control. The upper limit of the current requirement was not correct since much higher currents are required for satisfactory operation. Achieving a spark frequency of up to 50 [kHz] is theoretically possible but has not been tested. The power requirement and limitations caused by charging the capacitors in the power supply are causes that could limit the achievable frequency. Because a frequency of about 2 [kHz] has been found to be working reasonably well, it can be concluded that the requirement regarding the spark frequency was too stringent for this project. Sensing the gap voltage and quantifying its duration was eventually done by the power supply group.

Operational requirements were largely fulfilled, demonstrating core functionality as the EDM successfully created holes in aluminium and steel workpieces. Exact measurements for the MRR have not yet been performed, but it can be stated that the requirement regarding the material removal rate (MRR) has not been met consistently. The fastest time to create a (messy) 1.5[mm] diameter hole in a 2[mm] thick workpiece was measured to be approximately 30 [s].

While progress was made in integrating safety features, full compliance was not achieved. The emergency stop function in the Fluidd web server or the power switch could be used to stop the movement of the 3D printer. After shutting off the power supply, some delay is present due to the capacitors discharging before it becomes safe to touch the electronic circuits. Additionally, time constraints prevented the development of a built-in sensor.

6 Conclusion

6.1. Conclusion

This paper investigated the development of an Electrical Discharge Machining (EDM) device that is designed to create precise holes or shapes in various metals. The development of the EDM was guided by limitations and requirements detailed in Chapter 2. The fulfilment of the requirements is addressed in Section 5. A short literature review was performed, which resulted in the choice to use distilled water as dielectric fluid and copper as the electrode. Furthermore, research has been performed to control a 3D printer optimally. This research involved investigations into the specifications of the 3D printer, the 3D print design of an electrode mount, and the setting up of an external control system. Additionally, a power supply has been created to allow for testing of the EDM machine. During the last week of this project, attention has been given to the integration of the power supply and the voltage sensing designed by [1].

The EDM device discussed in this report does not yet fully operate as desired, but a solid foundation has been provided, which can be used as a basis for future work.

6.2. Recommendation and Future Work

The recommendations for future work are based on utilizing the other group's square wave generation circuit from the power supply, as our group's square wave generation circuit has several potential improvements that could not be implemented in time. Therefore, it is strongly advised to use the other group's square wave generation circuit from their power supply [1]. Moving forward, there are several key areas that need focused attention to improve the performance and functionality of the EDM device.

Firstly, while the groundwork has been laid for a closed-loop control system, its implementation was not feasible within the current time constraints. Future efforts should prioritize the development and integration of a closed-loop system. This advancement will improve the device's precision and operational stability by enabling real-time adjustments based on feedback.

Secondly, improvements to the mechanical structure could be explored. A proper bath for the dielectric fluid combined with mounts to hold the workpiece tightly is necessary for optimal performance. Moreover, addressing electromagnetic interference, particularly with the microcontroller board, remains a significant challenge. Future designs should explore more effective shielding methods and spatial configurations to mitigate these issues.

Additionally, certain components warrant further attention, which were not covered in this report. Specifically, the flushing and emergency stop buttons are critical elements highlighted in various literature but were not explored due to time constraints.

Lastly, comprehensive experimentation with operational parameters is essential for optimizing the EDM's performance. These operational parameters include gap voltage, current, duty cycle, and frequency. Future studies should systematically vary and refine these parameters to determine the most effective settings for different machining tasks.

Addressing these areas in future research and development cycles can lead to significant advancements in the capabilities and efficiency of the EDM device, leading to improved outcomes in manufacturing and prototyping applications.

References

[1] T. vd. Akker, J. Cheung, and B. Pieper, “Electrical discharge machining, design of a power supply,” Technical University of Delft, Tech. Rep., 2024.

[2] M. Kunieda, B. Lauwers, K. Rajurkar, and B. Schumacher, “Advancing edm through fundamental insight into the process,” *CIRP Annals*, vol. 54, no. 2, pp. 64–87, 2005, ISSN: 0007-8506. DOI: `https://doi.org/10.1016/S0007-8506(07)60020-1`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0007850607600201`.

[3] R. K. Shastri, C. P. Mohanty, S. Dash, K. M. P. Gopal, A. R. Annamalai, and C.-P. Jen, “Reviewing performance measures of the die-sinking electrical discharge machining process: Challenges and future scopes,” *Nanomaterials*, vol. 12, no. 3, 2022, ISSN: 2079-4991. DOI: `10.3390/nano12030384`. [Online]. Available: `https://www.mdpi.com/2079-4991/12/3/384`.

[4] A. K. S. Azhar Equbal, “Electrical discharge machining: An overview on various areas of research,” 2014. [Online]. Available: `https://www.researchgate.net/publication/287399233_Electrical_Discharge_Machining_An_Overview_on_Various_Areas_of_Research`.

[5] E. I. Solutions, “Fine hole edm drilling.”

[6] J. Tao and U. Michigan, “Investigation of dry and near-dry electrical discharge milling processes,” 2008. [Online]. Available: `https://deepblue.lib.umich.edu/handle/2027.42/60725`.

[7] H. M. et al., “State of the art in powder mixed dielectric for edm applications,” 2016. [Online]. Available: `https://www.sciencedirect.com/science/article/abs/pii/S0141635916300691`.

[8] J. Sai Ram, S. Jeavudeen, P. A. Mouda, and N. Ahamed, “The role of various dielectrics used in edm process and their environmental, health, and safety issues,” *Materials Today: Proceedings*, 2023, ISSN: 2214-7853. DOI: `https://doi.org/10.1016/j.matpr.2023.05.137`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S2214785323027530`.

[9] F. N. Leão and I. R. Pashby, “A review on the use of environmentally-friendly dielectric fluids in electrical discharge machining,” *Journal of Materials Processing Technology*, vol. 149, no. 1, pp. 341–346, 2004, 14th Interntaional Symposium on Electromachining (ISEM XIV), ISSN: 0924-0136. DOI: `https://doi.org/10.1016/j.jmatprotec.2003.10.043`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0924013604001773`.

[10] W. König and L. Jörres, “Aqueous solutions of organic compounds as dielectrics for edm sinking,” *CIRP Annals*, vol. 36, no. 1, pp. 105–109, 1987, ISSN: 0007-8506. DOI: `https://doi.org/10.1016/S0007-8506(07)62564-5`. [Online]. Available: `https://www.sciencedirect.com/science/article/pii/S0007850607625645`.

[11] P. S. Ng, S. A. Kong, and S. H. Yeo, “Investigation of biodiesel dielectric in sustainable electrical discharge machining,” *The International Journal of Advanced Manufacturing Technology*, vol. 90, pp. 2549–2556, 2017, ISSN: 1433-3015. DOI: `https://doi.org/10.1007/s00170-016-9572-6`. [Online]. Available: `https://link.springer.com/article/10.1007/s00170-016-9572-6#citeas`.

[12] P. Sadagopan and B. Mouliprasanth, “Investigation on the influence of different types of dielectrics in electrical discharge machining,” *The International Journal of Advanced Manufacturing Technology*, vol. 92, pp. 277–291, 2017, ISSN: 1433-3015. DOI: `https://doi.org/10.1007/s00170-017-0039-1`.

[13] A. Erden and D. Temel, “Investigation on the use of water as a dielectric liquid in e.d.m.,” in *Proceedings of the Twenty-second International Machine Tool Design and Research Conference*, B. J. Davies, Ed. London: Macmillan Education UK, 1982, pp. 437–440, ISBN: 978-1-349-06281-2. DOI: `10.1007/978-1-349-06281-2_54`. [Online]. Available: `https://doi.org/10.1007/978-1-349-06281-2_54`.

[14] S. Jilani and P. Pandey, "Experimetnal investigations into the performance of water as dielectric in edm," *International Journal of Machine Tool Design and Research*, vol. 24, no. 1, pp. 31–43, 1984, ISSN: 0020-7357. DOI: https://doi.org/10.1016/0020-7357(84)90044-1. [Online]. Available: https://www.sciencedirect.com/science/article/pii/0020735784900441.

[15] P. Chaudhury and S. Samantaray, "A comparative study of different dielectric medium for sustainable edm of non-conductive material by electro-thermal modelling," *Materials Today: Proceedings*, vol. 41, pp. 437–444, 2021, International Conference on Recent Advances in Mechanical Engineering Research and Development (ICRAMERD-20), ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2020.10.162. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785320377439.

[16] W. e. a. Ming, "Green manufacturing: A comparative study of renewable dielectrics in the edm process," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 44, p. 580, 2022, ISSN: 1806-3691. DOI: https://doi.org/10.1007/s40430-022-03867-3.

[17] A. Shih and J. Ni, "Experimental study of the dry and near-dry electrical discharge milling processes," *Journal of Manufacturing Science and Engineering-transactions of The Asme - J MANUF SCI ENG*, vol. 130, Feb. 2008. DOI: 10.1115/1.2784276.

[18] S. Mullya, G. Karthikeyan, and V. Ganachari, "Electric discharge milling: A state-of-the-art review," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 43, Sep. 2021. DOI: 10.1007/s40430-021-03146-7.

[19] A. K. S. Rahul Mahajan Hare Krishna and R. K. Ghadai, "A review on copper and its alloys used as electrode in edm," 2018. [Online]. Available: https://iopscience.iop.org/article/10.1088/1757-899X/377/1/012183.

[20] T. Czelusniak, C. Higa, R. Torres, *et al.*, "Materials used for sinking edm electrodes: A review," *Journal of the Brazilian Society of Mechanical Sciences and Engineering*, vol. 41, Jan. 2019. DOI: 10.1007/s40430-018-1520-y.

[21] D.V.Lohar, "Literature review of electrodes used in edm," 2018. [Online]. Available: https://www.ijream.org/papers/ICRTET0108.pdf.

[22] R. Rakshaskar and C. Kannan, "Influence of electrode type and dielectric during edm machining of ti-6al-4v," *Materials Today: Proceedings*, 2023, ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2023.06.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785323033850.

[23] H. Singh, J. Singh, S. Sharma, and J. Singh Chohan, "Parametric optimization of mrr & twr of the al6061/sic mmcs processed during die-sinking edm using different electrodes," *Materials Today: Proceedings*, vol. 48, pp. 1001–1008, 2022, SCPINM-2021, ISSN: 2214-7853. DOI: https://doi.org/10.1016/j.matpr.2021.06.323. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2214785321047143.

[24] N. Ahmed, "Machining and wear rates in edm of d2 steel: A comparative study of electrode designs and materials," *Journal of Materials Research and Technology*, vol. 30, pp. 1978–1991, 2024, ISSN: 2238-7854. DOI: https://doi.org/10.1016/j.jmrt.2024.03.242. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2238785424007725.

[25] J. Mercer, "Graphite vs. copper electrode material for edm applications, part 1," *MoldMaking Technology*, 2014.

[26] H. Lahbishi, I. Husain, and S. Aghdeab, "Licensed under creative commons attribution cc by influence of electrode geometry in parameters of electrical discharge machining process of aisid2 tool steel," vol. 9, pp. 883–889, Jan. 2018. DOI: 10.21275/SR20305140618.

[27] S. Industries, "Small hole electrodes." [Online]. Available: https://saturnedm.com/small-hole-electrode.

[28] R. N. Sell, "Vone copper earthing 3 meter copper electrode." [Online]. Available: https://www.racknsell.com/product/productdetails/304253-vone-vone-copper-earthing-3-meter-copper-electrode-dia-50-mm-2-bags-25-kg-pvc-chamber-cover.

[29] C. Globe. "Stepper motor." (), [Online]. Available: https://circuitglobe.com/stepper-motor.html (visited on 05/29/2024).

[30] Components101. “Nema 17 stepper motor.” (2019), [Online]. Available: https://components101.com/motors/nema17-stepper-motor (visited on 05/29/2024).

[31] S. O. M.
bibinitperiod Electronics. “Lead screw tr8x8-300.” (2017), [Online]. Available: https://www.omc-stepperonline.com/download/Tr8x8-300.pdf (visited on 05/29/2024).

[32] “Printer-creality-ender3pro-2020.cfg.” (2022), [Online]. Available: https://github.com/Klipper3d/klipper/blob/master/config/printer-creality-ender3pro-2020.cfg (visited on 05/29/2024).

[33] Aspina. “How are stepper motors controlled? - speed control of stepper motors.” (2021), [Online]. Available: https://us.aspina-group.com/en/learning-zone/columns/what-is/017/ (visited on 06/05/2024).

[34] T. R. Ablyaz, E. S. Shlykov, and K. R. Muratov, “Modeling of edm process flushing mechanism,” *Materials*, vol. 16, no. 11, 2023, ISSN: 1996-1944. DOI: 10.3390/ma16114158. [Online]. Available: https://www.mdpi.com/1996-1944/16/11/4158.

[35] Klipper. “Can i run klipper on something other than a raspberry pi 3?” (), [Online]. Available: https://www.klipper3d.org/FAQ.html?h=home (visited on 06/05/2024).

[36] R. P. Foundation. “Raspberry pi documentation.” (), [Online]. Available: https://www.raspberrypi.com/documentation/computers/raspberry-pi.html (visited on 06/05/2024).

[37] V. 3D. “How to upgrade to klipper on any ender 3 for high performance.” (2023), [Online]. Available: https://www.youtube.com/watch?v=N41JY1Gukuk (visited on 06/08/2024).

[38] K. et al., *Klipper3d*, https://github.com/Klipper3d/klipper, 2024.

[39] Klipper. “G-codes.” (), [Online]. Available: https://www.klipper3d.org/G-Codes.html (visited on 06/08/2024).

[40] V. G. Sree, “Breakdown study of water with different conductivities,” 2015. [Online]. Available: https://api.semanticscholar.org/CorpusID:211837801.

[41] Y.-f. Tzeng and F.-c. Chen, “A simple approach for robust design of high-speed electrical-discharge machining technology,” *International Journal of Machine Tools and Manufacture*, vol. 43, no. 3, pp. 217–227, 2003, ISSN: 0890-6955. DOI: https://doi.org/10.1016/S0890-6955(02)00261-4. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0890695502002614.

[42] B. M. Schumacher, “After 60 years of edm the discharge process remains still disputed,” *Journal of Materials Processing Technology*, vol. 149, no. 1, pp. 376–381, 2004, 14th Interntaional Symposium on Electromachining (ISEM XIV), ISSN: 0924-0136. DOI: https://doi.org/10.1016/j.jmatprotec.2003.11.060. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0924013604001839.

[43] J. E. A. Qudeiri, A. Zaiout, A.-H. I. Mourad, M. H. Abidi, and A. Elkaseer, “Principles and characteristics of different edm processes in machining tool and die steels,” *Applied Sciences*, vol. 10, no. 6, 2020, ISSN: 2076-3417. DOI: 10.3390/app10062082. [Online]. Available: https://www.mdpi.com/2076-3417/10/6/2082.

[44] C.-T. L. Yan-Cherng Lin Yuan-Feng Chen and H.-J. Tzeng, “Electrical discharge machining (edm) characteristics associated with electrical discharge energy on machining of cemented tungsten carbide,” *Materials and Manufacturing Processes*, vol. 23, no. 4, pp. 391–399, 2008. DOI: 10.1080/10426910801938577. eprint: https://doi.org/10.1080/10426910801938577. [Online]. Available: https://doi.org/10.1080/10426910801938577.

[45] S. Saha, “Experimental investigation of the dry electric discharge machining (dry edm) process,” Ph.D. dissertation, Jan. 2008.

[46] *Mosfet strongirfet*^{TM}, IRF300P226, V2.2, Infineon Technologies, 2020. [Online]. Available: https://www.infineon.com/dgdl/Infineon-IRF300P226-DataSheet-v02_01-EN.pdf?fileId=5546d462602a9dc8016038801939199d.

[47] *Mosfet strongirfet*^{TM}, IPW65R037C6, Infineon Technologies, 2011. [Online]. Available: https://www.infineon.com/dgdl/Infineon-IPW65R037C6-DS-v02_00-en.pdf?fileId=db3a3043337a914d0133877a719210a9.

A

Klipper Source Code Modifications 1

Listing A.1: PrinterHoming class of the homing.py file with modifications

```
# Helper code for implementing homing operations
#
# Copyright (C) 2016-2021  Kevin O'Connor <kevin@koconnor.net>
#
# This file may be distributed under the terms of the GNU GPLv3 license.

class PrinterHoming:
    #----------------- Added code ------------
    override = 0 # Added code
    #-----------------------------------------
    def __init__(self, config):
        self.printer = config.get_printer()
        # Register g-code commands
        gcode = self.printer.lookup_object('gcode')
        gcode.register_command('G28', self.cmd_G28)
        gcode.register_command('G30', self.cmd_G30)
    def manual_home(self, toolhead, endstops, pos, speed,
                    triggered, check_triggered):
        hmove = HomingMove(self.printer, endstops, toolhead)
        try:
            hmove.homing_move(pos, speed, triggered=triggered,
                              check_triggered=check_triggered)
        except self.printer.command_error:
            if self.printer.is_shutdown():
                raise self.printer.command_error(
                    "Homing␣failed␣due␣to␣printer␣shutdown")
            raise
    def probing_move(self, mcu_probe, pos, speed):
        endstops = [(mcu_probe, "probe")]
        hmove = HomingMove(self.printer, endstops)
        try:
            epos = hmove.homing_move(pos, speed, probe_pos=True)
        except self.printer.command_error:
            if self.printer.is_shutdown():
                raise self.printer.command_error(
                    "Probing␣failed␣due␣to␣printer␣shutdown")
            raise
        if hmove.check_no_movement() is not None:
            raise self.printer.command_error(
                "Probe␣triggered␣prior␣to␣movement")
        return epos

    #----------------- Added code ------------
    @staticmethod
    def override_state():
        return PrinterHoming.override
    def cmd_G30(self, gcmd):
        PrinterHoming.override = 1
    def cmd_G31(self, gcmd):
        PrinterHoming.override = 0
    #-----------------------------------------

    def cmd_G28(self, gcmd):
        logging.info('G28␣homing.py')
        # Move to origin
        axes = []
        for pos, axis in enumerate('XYZ'):
            if gcmd.get(axis, None) is not None:
                axes.append(pos)
        if not axes:
            axes = [0, 1, 2]
        homing_state = Homing(self.printer)
        homing_state.set_axes(axes)
        kin = self.printer.lookup_object('toolhead').get_kinematics()
        try:
            kin.home(homing_state)
        except self.printer.command_error:
            if self.printer.is_shutdown():
                raise self.printer.command_error(
                    "Homing␣failed␣due␣to␣printer␣shutdown")
            self.printer.lookup_object('stepper_enable').motor_off()
            raise
```

Listing A.2: CartKinematics class of the cartesian.py file with modifications

```
# Code for handling the kinematics of cartesian robots
#
# Copyright (C) 2016-2021  Kevin O'Connor <kevin@koconnor.net>
#
# This file may be distributed under the terms of the GNU GPLv3 license.
import logging
import stepper
from . import idex_modes
#----------------- Added code ------------
import extras.homing as homing
#-----------------------------------------

class CartKinematics:
    def __init__(self, toolhead, config):
        self.printer = config.get_printer()
        # Home override state
        self.override = 0
        # Setup axis rails
        self.dual_carriage_axis = None
        self.dual_carriage_rails = []
        self.rails = [stepper.LookupMultiRail(config.getsection('stepper_' + n))
                      for n in 'xyz']
        for rail, axis in zip(self.rails, 'xyz'):
            rail.setup_itersolve('cartesian_stepper_alloc', axis.encode())
        ranges = [r.get_range() for r in self.rails]
        self.axes_min = toolhead.Coord(*[r[0] for r in ranges], e=0.)
        self.axes_max = toolhead.Coord(*[r[1] for r in ranges], e=0.)
        self.dc_module = None
        if config.has_section('dual_carriage'):
            dc_config = config.getsection('dual_carriage')
            dc_axis = dc_config.getchoice('axis', {'x': 'x', 'y': 'y'})
            self.dual_carriage_axis = {'x': 0, 'y': 1}[dc_axis]
            # setup second dual carriage rail
            self.rails.append(stepper.LookupMultiRail(dc_config))
            self.rails[3].setup_itersolve('cartesian_stepper_alloc',
                                          dc_axis.encode())
            dc_rail_0 = idex_modes.DualCarriagesRail(
                    self.rails[self.dual_carriage_axis],
                    axis=self.dual_carriage_axis, active=True)
            dc_rail_1 = idex_modes.DualCarriagesRail(
                    self.rails[3], axis=self.dual_carriage_axis, active=False)
            self.dc_module = idex_modes.DualCarriages(
                    dc_config, dc_rail_0, dc_rail_1,
                    axis=self.dual_carriage_axis)
        for s in self.get_steppers():
            s.set_trapq(toolhead.get_trapq())
            toolhead.register_step_generator(s.generate_steps)
        self.printer.register_event_handler("stepper_enable:motor_off",
                                            self._motor_off)
        # Setup boundary checks
        max_velocity, max_accel = toolhead.get_max_velocity()
        self.max_z_velocity = config.getfloat('max_z_velocity', max_velocity,
                                              above=0., maxval=max_velocity)
        self.max_z_accel = config.getfloat('max_z_accel', max_accel,
                                           above=0., maxval=max_accel)
        self.limits = [(1.0, -1.0)] * 3
    def get_steppers(self):
        return [s for rail in self.rails for s in rail.get_steppers()]
    def calc_position(self, stepper_positions):
        return [stepper_positions[rail.get_name()] for rail in self.rails]
    def update_limits(self, i, range):
        l, h = self.limits[i]
        # Only update limits if this axis was already homed,
        # otherwise leave in un-homed state.
        if l <= h:
            self.limits[i] = range
    def override_rail(self, i, rail):
        self.rails[i] = rail
    def set_position(self, newpos, homing_axes):
        for i, rail in enumerate(self.rails):
            rail.set_position(newpos)
            if i in homing_axes:
                self.limits[i] = rail.get_range()
    def note_z_not_homed(self):
        # Helper for Safe Z Home
        self.limits[2] = (1.0, -1.0)
    def home_axis(self, homing_state, axis, rail):
        # Determine movement
        position_min, position_max = rail.get_range()
        hi = rail.get_homing_info()
        homepos = [None, None, None, None]
        homepos[axis] = hi.position_endstop
        forcepos = list(homepos)
        if hi.positive_dir:
            forcepos[axis] -= 1.5 * (hi.position_endstop - position_min)
        else:
            forcepos[axis] += 1.5 * (position_max - hi.position_endstop)
        # Perform homing
        homing_state.home_rails([rail], forcepos, homepos)
    def home(self, homing_state):
        # Each axis is homed independently and in order
        for axis in homing_state.get_axes():
            if self.dc_module is not None and axis == self.dual_carriage_axis:
                self.dc_module.home(homing_state)
            else:
                self.home_axis(homing_state, axis, self.rails[axis])
    def _motor_off(self, print_time):
        self.limits = [(1.0, -1.0)] * 3
    def _check_endstops(self, move):
        end_pos = move.end_pos
        #----------------- Added code ------------
        override_state = homing.PrinterHoming.override_state()
        #-----------------------------------------
        if override_state == 0:
            for i in (0, 1, 2):
                if (move.axes_d[i]
                    and (end_pos[i] < self.limits[i][0]
                        or end_pos[i] > self.limits[i][1])):
                    if self.limits[i][0] > self.limits[i][1]:
                        raise move.move_error("Must home axis first")
                    raise move.move_error()
    def check_move(self, move):
        limits = self.limits
        xpos, ypos = move.end_pos[:2]
        if (xpos < limits[0][0] or xpos > limits[0][1]
            or ypos < limits[1][0] or ypos > limits[1][1]):
            self._check_endstops(move)
        if not move.axes_d[2]:
            # Normal XY move - use defaults
            return
        # Move with Z - update velocity and accel for slower Z axis
        self._check_endstops(move)
        z_ratio = move.move_d / abs(move.axes_d[2])
        move.limit_speed(
            self.max_z_velocity * z_ratio, self.max_z_accel * z_ratio)
    def get_status(self, eventtime):
        axes = [a for a, (l, h) in zip("xyz", self.limits) if l <= h]
        return {
            'homed_axes': "".join(axes),
            'axis_minimum': self.axes_min,
            'axis_maximum': self.axes_max,
        }

def load_kinematics(toolhead, config):
    return CartKinematics(toolhead, config)
```

B
Klipper Source Code Modifications 2

Listing B.1: Printer class in the klippy.py file with modifications

```
#----------------- Added code ------------
import socket
#-----------------------------------------

class Printer:
    config_error = configfile.error
    command_error = gcode.CommandError
    def __init__(self, main_reactor, bglogger, start_args):
        #----------------- Added code ------------
        self.SERVER_ADDR = 0
        self.SERVER_PORT = 0
        self.sock = 0
        self.addr = 0
        #-----------------------------------------

        self.bglogger = bglogger
        self.start_args = start_args
        self.reactor = main_reactor
        self.reactor.register_callback(self._connect)
        self.state_message = message_startup
        self.in_shutdown_state = False
        self.run_result = None
        self.event_handlers = {}
        self.objects = collections.OrderedDict()
        # Init printer components that must be setup prior to config
        for m in [gcode, webhooks]:
            m.add_early_printer_objects(self)
    #----------------- Added code ------------
    # set up pico connection
    def MakeConnection(self):
        self.SERVER_ADDR = '192.168.137.117'
        self.SERVER_PORT = 4242
        self.sock = socket.socket()
        self.addr = (self.SERVER_ADDR, self.SERVER_PORT)
        self.sock.connect(self.addr)
    def readCommand(self):
        buf = self.sock.recv(1)
        return buf
    def writeCommand(self, data):
        logging.info('Data:␣%s', data)
        data_to_bytes = bytes(data, 'utf-8')
        logging.info('Data_to_bytes:␣%s', data_to_bytes)
        self.sock.send(data_to_bytes)
    def CloseConnection(self):
        self.sock.close()
    #-----------------------------------------
    def get_start_args(self):
        return self.start_args
```

```python
    def get_reactor(self):
        return self.reactor
    def get_state_message(self):
        if self.state_message == message_ready:
            category = "ready"
        elif self.state_message == message_startup:
            category = "startup"
        elif self.in_shutdown_state:
            category = "shutdown"
        else:
            category = "error"
        return self.state_message, category
    def is_shutdown(self):
        return self.in_shutdown_state
    def _set_state(self, msg):
        if self.state_message in (message_ready, message_startup):
            self.state_message = msg
        if (msg != message_ready
            and self.start_args.get('debuginput') is not None):
            self.request_exit('error_exit')
    def add_object(self, name, obj):
        if name in self.objects:
            raise self.config_error(
                "Printer␣object␣'%s'␣already␣created" % (name,))
        self.objects[name] = obj
    def lookup_object(self, name, default=configfile.sentinel):
        if name in self.objects:
            return self.objects[name]
        if default is configfile.sentinel:
            raise self.config_error("Unknown␣config␣object␣'%s'" % (name,))
        return default
    def lookup_objects(self, module=None):
        if module is None:
            return list(self.objects.items())
        prefix = module + '␣'
        objs = [(n, self.objects[n])
                for n in self.objects if n.startswith(prefix)]
        if module in self.objects:
            return [(module, self.objects[module])] + objs
        return objs
    def load_object(self, config, section, default=configfile.sentinel):
        if section in self.objects:
            return self.objects[section]
        module_parts = section.split()
        module_name = module_parts[0]
        py_name = os.path.join(os.path.dirname(__file__),
                               'extras', module_name + '.py')
        py_dirname = os.path.join(os.path.dirname(__file__),
                                  'extras', module_name, '__init__.py')
        if not os.path.exists(py_name) and not os.path.exists(py_dirname):
            if default is not configfile.sentinel:
                return default
            raise self.config_error("Unable␣to␣load␣module␣'%s'" % (section,))
        mod = importlib.import_module('extras.' + module_name)
        init_func = 'load_config'
        if len(module_parts) > 1:
            init_func = 'load_config_prefix'
        init_func = getattr(mod, init_func, None)
        if init_func is None:
            if default is not configfile.sentinel:
                return default
            raise self.config_error("Unable␣to␣load␣module␣'%s'" % (section,))
        self.objects[section] = init_func(config.getsection(section))
        return self.objects[section]
    def _read_config(self):
        self.objects['configfile'] = pconfig = configfile.PrinterConfig(self)
        config = pconfig.read_main_config()
        if self.bglogger is not None:
            pconfig.log_config(config)
        # Create printer components
        for m in [pins, mcu]:
            m.add_printer_objects(config)
        for section_config in config.get_prefix_sections(''):
            self.load_object(config, section_config.get_name(), None)
        for m in [toolhead]:
            m.add_printer_objects(config)
        # Validate that there are no undefined parameters in the config file
        pconfig.check_unused_options(config)
    def _build_protocol_error_message(self, e):
        host_version = self.start_args['software_version']
        msg_update = []
        msg_updated = []
        for mcu_name, mcu in self.lookup_objects('mcu'):
            try:
                mcu_version = mcu.get_status()['mcu_version']
            except:
                logging.exception("Unable␣to␣retrieve␣mcu_version␣from␣mcu")
                continue
            if mcu_version != host_version:
                msg_update.append("%s:␣Current␣version␣%s"
                                  % (mcu_name.split()[-1], mcu_version))
            else:
                msg_updated.append("%s:␣Current␣version␣%s"
                                   % (mcu_name.split()[-1], mcu_version))
        if not msg_update:
            msg_update.append("<none>")
        if not msg_updated:
            msg_updated.append("<none>")
        msg = ["MCU␣Protocol␣error",
               message_protocol_error1,
               "Your␣Klipper␣version␣is:␣%s" % (host_version,),
               "MCU(s)␣which␣should␣be␣updated:"]
        msg += msg_update + ["Up-to-date␣MCU(s):"] + msg_updated
        msg += [message_protocol_error2, str(e)]
        return "\n".join(msg)
    def _connect(self, eventtime):
        try:
            self._read_config()
            self.send_event("klippy:mcu_identify")
            for cb in self.event_handlers.get("klippy:connect", []):
                if self.state_message is not message_startup:
                    return
                cb()
        except (self.config_error, pins.error) as e:
            logging.exception("Config␣error")
            self._set_state("%s\n%s" % (str(e), message_restart))
            return
        except msgproto.error as e:
            logging.exception("Protocol␣error")
            self._set_state(self._build_protocol_error_message(e))
            util.dump_mcu_build()
            return
        except mcu.error as e:
            logging.exception("MCU␣error␣during␣connect")
            self._set_state("%s%s" % (str(e), message_mcu_connect_error))
            util.dump_mcu_build()
            return
        except Exception as e:
            logging.exception("Unhandled␣exception␣during␣connect")
            self._set_state("Internal␣error␣during␣connect:␣%s\n%s"
                            % (str(e), message_restart,))
            return
        try:
            self._set_state(message_ready)
            for cb in self.event_handlers.get("klippy:ready", []):
                if self.state_message is not message_ready:
                    return
                cb()
        except Exception as e:
            logging.exception("Unhandled␣exception␣during␣ready␣callback")
            self.invoke_shutdown("Internal␣error␣during␣ready␣callback:␣%s"
                                 % (str(e),))
    def run(self):
        systime = time.time()
        monotime = self.reactor.monotonic()
        logging.info("Start␣printer␣at␣%s␣(%.1f␣%.1f)",
                     time.asctime(time.localtime(systime)), systime, monotime)
        # Enter main reactor loop
        try:
            self.reactor.run()
        except:
            msg = "Unhandled␣exception␣during␣run"
            logging.exception(msg)
            # Exception from a reactor callback - try to shutdown
            try:
                self.reactor.register_callback((lambda e:
                                                self.invoke_shutdown(msg)))
                self.reactor.run()
            except:
                logging.exception("Repeat␣unhandled␣exception␣during␣run")
                # Another exception - try to exit
                self.run_result = "error_exit"
        # Check restart flags
        run_result = self.run_result
        try:
            if run_result == 'firmware_restart':
                self.send_event("klippy:firmware_restart")
            self.send_event("klippy:disconnect")
        except:
            logging.exception("Unhandled␣exception␣during␣post␣run")
        return run_result
    def set_rollover_info(self, name, info, log=True):
        if log:
            i = 1
            # logging.info(info)
        if self.bglogger is not None:
            self.bglogger.set_rollover_info(name, info)
    def invoke_shutdown(self, msg):
        if self.in_shutdown_state:
            return
        logging.error("Transition␣to␣shutdown␣state:␣%s", msg)
        self.in_shutdown_state = True
        self._set_state("%s%s" % (msg, message_shutdown))
        for cb in self.event_handlers.get("klippy:shutdown", []):
            try:
                cb()
            except:
                logging.exception("Exception␣during␣shutdown␣handler")
        logging.info("Reactor␣garbage␣collection:␣%s",
                     self.reactor.get_gc_stats())
    def invoke_async_shutdown(self, msg):
        self.reactor.register_async_callback(
            (lambda e: self.invoke_shutdown(msg)))
    def register_event_handler(self, event, callback):
        self.event_handlers.setdefault(event, []).append(callback)
    def send_event(self, event, *params):
        return [cb(*params) for cb in self.event_handlers.get(event, [])]
    def request_exit(self, result):
        if self.run_result is None:
            self.run_result = result
        self.reactor.end()
```

Listing B.2: GCodeDispatch class in the gcode.py file with modifications

```
# Parse and dispatch G-Code commands
class GCodeDispatch:
    error = CommandError
    Coord = Coord
    def __init__(self, printer):
        self.printer = printer
        self.is_fileinput = not not printer.get_start_args().get("debuginput")
        printer.register_event_handler("klippy:ready", self._handle_ready)
        printer.register_event_handler("klippy:shutdown", self._handle_shutdown)
        printer.register_event_handler("klippy:disconnect",
                                        self._handle_disconnect)
        # Command handling
        self.is_printer_ready = False
        self.mutex = printer.get_reactor().mutex()
        self.output_callbacks = []
        self.base_gcode_handlers = self.gcode_handlers = {}
        self.ready_gcode_handlers = {}
        self.mux_commands = {}
        self.gcode_help = {}
        self.status_commands = {}
        self.delay = 0
        # Signal parameters
        self.frequency = 2000
        self.duty_cycle = 0.5
        # Register commands needed before config file is loaded
        handlers = ['M110', 'M112', 'M115',
                    #----------------- Added code ------------
                    'M60', 'M61', 'M62',
                    #-----------------------------------------
                    'RESTART', 'FIRMWARE_RESTART', 'ECHO', 'STATUS', 'HELP']
        for cmd in handlers:
            func = getattr(self, 'cmd_' + cmd)
            desc = getattr(self, 'cmd_' + cmd + '_help', None)
            self.register_command(cmd, func, True, desc)
    def is_traditional_gcode(self, cmd):
        # A "traditional" g-code command is a letter and followed by a number
        try:
            cmd = cmd.upper().split()[0]
            val = float(cmd[1:])

            return cmd[0].isupper() and cmd[1].isdigit()
        except:
            return False
    def register_command(self, cmd, func, when_not_ready=False, desc=None):
        # logging.info("CMD: %s FUNC: %s", cmd, func)
        if func is None:
            old_cmd = self.ready_gcode_handlers.get(cmd)
            if cmd in self.ready_gcode_handlers:
                del self.ready_gcode_handlers[cmd]
            if cmd in self.base_gcode_handlers:
                del self.base_gcode_handlers[cmd]
            self._build_status_commands()
            return old_cmd
        if cmd in self.ready_gcode_handlers:
            raise self.printer.config_error(
                "gcode␣command␣%s␣already␣registered" % (cmd,))
        if not self.is_traditional_gcode(cmd):
            origfunc = func
            func = lambda params: origfunc(self._get_extended_params(params))
        self.ready_gcode_handlers[cmd] = func
        if when_not_ready:
            self.base_gcode_handlers[cmd] = func
        if desc is not None:
            self.gcode_help[cmd] = desc
        self._build_status_commands()
    def register_mux_command(self, cmd, key, value, func, desc=None):
        prev = self.mux_commands.get(cmd)
        if prev is None:
            handler = lambda gcmd: self._cmd_mux(cmd, gcmd)
            self.register_command(cmd, handler, desc=desc)
            self.mux_commands[cmd] = prev = (key, {})
        prev_key, prev_values = prev
        if prev_key != key:
            raise self.printer.config_error(
                "mux␣command␣%s␣%s␣%s␣may␣have␣only␣one␣key␣(%s)" % (
                    cmd, key, value, prev_key))
        if value in prev_values:
            raise self.printer.config_error(
                "mux␣command␣%s␣%s␣%s␣already␣registered␣(%s)" % (
                    cmd, key, value, prev_values))
        prev_values[value] = func
    def get_command_help(self):
        return dict(self.gcode_help)
    def get_status(self, eventtime):
        return {'commands': self.status_commands}
    def _build_status_commands(self):
        commands = {cmd: {} for cmd in self.gcode_handlers}
        for cmd in self.gcode_help:
            if cmd in commands:
                commands[cmd]['help'] = self.gcode_help[cmd]
        self.status_commands = commands
    def register_output_handler(self, cb):
        self.output_callbacks.append(cb)
    def _handle_shutdown(self):
        if not self.is_printer_ready:
            return
        self.is_printer_ready = False
        self.gcode_handlers = self.base_gcode_handlers
        self._build_status_commands()
        self._respond_state("Shutdown")
    def _handle_disconnect(self):
        self._respond_state("Disconnect")
    def _handle_ready(self):
        self.is_printer_ready = True
        self.gcode_handlers = self.ready_gcode_handlers
        self._build_status_commands()
        self._respond_state("Ready")
    # Parse input into commands
    args_r = re.compile('([A-Z_]+|[A-Z*/])')
    def _process_commands(self, commands, need_ack=False):
        for line in commands:
            # Ignore comments and leading/trailing spaces
            line = origline = line.strip()
            cpos = line.find(';')
            if cpos >= 0:
                line = line[:cpos]
            # Break line into parts and determine command
            parts = self.args_r.split(line.upper())
            numparts = len(parts)
            cmd = ""
            if numparts >= 3 and parts[1] != 'N':
                cmd = parts[1] + parts[2].strip()
            elif numparts >= 5 and parts[1] == 'N':
                # Skip line number at start of command
                cmd = parts[3] + parts[4].strip()
            # Build gcode "params" dictionary
            params = { parts[i]: parts[i+1].strip()
                       for i in range(1, numparts, 2) }
            gcmd = GCodeCommand(self, cmd, origline, params, need_ack)
            # Invoke handler for command
            handler = self.gcode_handlers.get(cmd, self.cmd_default)
            try:
                handler(gcmd)
            except self.error as e:
                self._respond_error(str(e))
                self.printer.send_event("gcode:command_error")
                if not need_ack:
                    raise
            except:
                msg = 'Internal␣error␣on␣command:"%s"' % (cmd,)
                logging.exception(msg)
                self.printer.invoke_shutdown(msg)
                self._respond_error(msg)
                if not need_ack:
                    raise
            gcmd.ack()
    def run_script_from_command(self, script):
        self._process_commands(script.split('\n'), need_ack=False)
    def run_script(self, script, data):
        with self.mutex:
            self._process_commands(script.split('\n'), need_ack=False)
    def get_mutex(self):
        return self.mutex
    def create_gcode_command(self, command, commandline, params):
        return GCodeCommand(self, command, commandline, params, False)
    # Response handling
    def respond_raw(self, msg):
        for cb in self.output_callbacks:
            cb(msg)
    def respond_info(self, msg, log=True):
        if log:
            logging.info(msg)
        lines = [l.strip() for l in msg.strip().split('\n')]
        self.respond_raw("// " + "\n// ".join(lines))
    def _respond_error(self, msg):
        logging.warning(msg)
        lines = msg.strip().split('\n')
        if len(lines) > 1:
            self.respond_info("\n".join(lines), log=False)
        self.respond_raw('!! %s' % (lines[0].strip(),))
        if self.is_fileinput:
            self.printer.request_exit('error_exit')
    def _respond_state(self, state):
        self.respond_info("Klipper state: %s" % (state,), log=False)
    # Parameter parsing helpers
    extended_r = re.compile(
        r'^\s*(?:N[0-9]+\s*)?'
        r'(?P<cmd>[a-zA-Z_][a-zA-Z0-9_]+)(?:\s+|$)'
        r'(?P<args>[^#*;]*?)'
        r'\s*(?:[#*;].*)?$')
    def _get_extended_params(self, gcmd):
        m = self.extended_r.match(gcmd.get_commandline())
        if m is None:
            raise self.error("Malformed command '%s'"
                             % (gcmd.get_commandline(),))
        eargs = m.group('args')
        try:
            eparams = [earg.split('=', 1) for earg in shlex.split(eargs)]
            eparams = { k.upper(): v for k, v in eparams }
            gcmd._params.clear()
            gcmd._params.update(eparams)
            return gcmd
        except ValueError as e:
            raise self.error("Malformed command '%s'"
                             % (gcmd.get_commandline(),))
    # G-Code special command handlers
    def cmd_default(self, gcmd):
        cmd = gcmd.get_command()
        if cmd == 'M105':
            # Don't warn about temperature requests when not ready
            gcmd.ack("T:0")
            return
        if cmd == 'M21':
            # Don't warn about sd card init when not ready
            return
        if not self.is_printer_ready:
            raise gcmd.error(self.printer.get_state_message()[0])
            return
        if not cmd:
            cmdline = gcmd.get_commandline()
            if cmdline:
                logging.debug(cmdline)
            return
        if cmd.startswith("M117 ") or cmd.startswith("M118 "):
            # Handle M117/M118 gcode with numeric and special characters
            handler = self.gcode_handlers.get(cmd[:4], None)
            if handler is not None:
                handler(gcmd)
                return
        elif cmd in ['M140', 'M104'] and not gcmd.get_float('S', 0.):
            # Don't warn about requests to turn off heaters when not present
            return
        elif cmd == 'M107' or (cmd == 'M106' and (
                not gcmd.get_float('S', 1.) or self.is_fileinput)):
            # Don't warn about requests to turn off fan when fan not present
            return
        gcmd.respond_info('Unknown␣command:"%s"' % (cmd,))
    def _cmd_mux(self, command, gcmd):
        key, values = self.mux_commands[command]
        if None in values:
            key_param = gcmd.get(key, None)
        else:
            key_param = gcmd.get(key)
        if key_param not in values:
            raise gcmd.error("The␣value␣'%s'␣is␣not␣valid␣for␣%s"
                             % (key_param, key))
        values[key_param](gcmd)
    #----------------- Added code ------------
    # send data to raspberry pico
    def cmd_M60(self, gcmd):
        # Start connection with pico
        try:
            self.printer.MakeConnection()
        except:
            raise self.error('No␣connection␣established')
    def cmd_M61(self, gcmd):
        # Write signal parameters to pico
        params = gcmd.get_command_parameters()
        if 'F' in params:
            # check for change in frequency
            freq = int(params['F'])
            if freq >= 100 and freq <= 100000:
                send_freq = freq
                self.frequency = send_freq
        else:
            send_freq = self.frequency
        if 'D' in params:
            # check for change in duty cycle
            duty_cycle = float(params['D'])
            if duty_cycle > 0 and duty_cycle <= 1:
                send_duty_cycle = duty_cycle
                self.duty_cycle = send_duty_cycle
        else:
            send_duty_cycle = self.duty_cycle
        try:
            self.printer.writeCommand(f'freq={send_freq},duty={send_duty_cycle};\n')
        except:
            raise self.error('Could␣not␣write␣to␣Raspberry␣Pi␣Pico')
    def cmd_M62(self, gcmd):
        # Close connection with pico
        try:
            self.printer.CloseConnection()
        except:
            raise self.error('Connection␣not␣closed')
    #----------------------------------------
    # Low-level G-Code commands that are needed before the config file is loaded
    def cmd_M110(self, gcmd):
        # Set Current Line Number
        pass
    def cmd_M112(self, gcmd):
        # Emergency Stop
        self.printer.invoke_shutdown("Shutdown␣due␣to␣M112␣command")
    def cmd_M115(self, gcmd):
        # Get Firmware Version and Capabilities
        software_version = self.printer.get_start_args().get('software_version')
        kw = {"FIRMWARE_NAME": "Klipper", "FIRMWARE_VERSION": software_version}
        msg = "␣".join(["%s:%s" % (k, v) for k, v in kw.items()])
        did_ack = gcmd.ack(msg)
        if not did_ack:
            gcmd.respond_info(msg)
    def request_restart(self, result):
        if self.is_printer_ready:
            toolhead = self.printer.lookup_object('toolhead')
            print_time = toolhead.get_last_move_time()
            if result == 'exit':
                logging.info("Exiting␣(print␣time␣%.3fs)" % (print_time,))
            self.printer.send_event("gcode:request_restart", print_time)
            toolhead.dwell(0.500)
            toolhead.wait_moves()
        self.printer.request_exit(result)
    cmd_RESTART_help = "Reload␣config␣file␣and␣restart␣host␣software"
    def cmd_RESTART(self, gcmd):
        self.request_restart('restart')
    cmd_FIRMWARE_RESTART_help = "Restart␣firmware,␣host,␣and␣reload␣config"
    def cmd_FIRMWARE_RESTART(self, gcmd):
        self.request_restart('firmware_restart')
    def cmd_ECHO(self, gcmd):
        gcmd.respond_info(gcmd.get_commandline(), log=False)
    cmd_STATUS_help = "Report␣the␣printer␣status"
    def cmd_STATUS(self, gcmd):
        if self.is_printer_ready:
            self._respond_state("Ready")
            return
        msg = self.printer.get_state_message()[0]
        msg = msg.rstrip() + "\nKlipper␣state:␣Not␣ready"
        raise gcmd.error(msg)
    cmd_HELP_help = "Report␣the␣list␣of␣available␣extended␣G-Code␣commands"
    def cmd_HELP(self, gcmd):
        cmdhelp = []
        if not self.is_printer_ready:
            cmdhelp.append("Printer␣is␣not␣ready␣-␣not␣all␣commands␣available.")
        cmdhelp.append("Available␣extended␣commands:")
        for cmd in sorted(self.gcode_handlers):
            if cmd in self.gcode_help:
                cmdhelp.append("%-10s:␣%s" % (cmd, self.gcode_help[cmd]))
        gcmd.respond_info("\n".join(cmdhelp), log=False)
```

Listing B.3: VirtualSD class in the virtual_sd.py file with modifications

```
class VirtualSD:
    def __init__(self, config):
        self.printer = config.get_printer()
        self.printer.register_event_handler("klippy:shutdown",
                                            self.handle_shutdown)
        #----------------- Added code ------------
        self.data = 0
        self.data_i2c = 255
        self.data_prev = 0

        self.set_break = False
        self.pause = False
        #-----------------------------------------

        # sdcard state
        sd = config.get('path')
        self.sdcard_dirname = os.path.normpath(os.path.expanduser(sd))
        self.current_file = None
        self.file_position = self.file_size = 0
        # Print Stat Tracking
        self.print_stats = self.printer.load_object(config, 'print_stats')
        # Work timer
        self.reactor = self.printer.get_reactor()
        self.must_pause_work = self.cmd_from_sd = False
        self.next_file_position = 0
        self.work_timer = None
        # Error handling
        gcode_macro = self.printer.load_object(config, 'gcode_macro')
        self.on_error_gcode = gcode_macro.load_template(
            config, 'on_error_gcode', DEFAULT_ERROR_GCODE)
        # Register commands
        self.gcode = self.printer.lookup_object('gcode')
        for cmd in ['M20', 'M21', 'M23', 'M24', 'M25', 'M26', 'M27']:
            self.gcode.register_command(cmd, getattr(self, 'cmd_' + cmd))
        for cmd in ['M28', 'M29', 'M30']:
            self.gcode.register_command(cmd, self.cmd_error)
        self.gcode.register_command(
            "SDCARD_RESET_FILE", self.cmd_SDCARD_RESET_FILE,
            desc=self.cmd_SDCARD_RESET_FILE_help)
        self.gcode.register_command(
            "SDCARD_PRINT_FILE", self.cmd_SDCARD_PRINT_FILE,
            desc=self.cmd_SDCARD_PRINT_FILE_help)
    def handle_shutdown(self):
        if self.work_timer is not None:
            self.must_pause_work = True
            try:
                readpos = max(self.file_position - 1024, 0)
                readcount = self.file_position - readpos
                self.current_file.seek(readpos)
                data = self.current_file.read(readcount + 128)
            except:
                logging.exception("virtual␣sdcard␣shutdown␣read")
                return
            logging.info("Virtual␣sdcard␣(%d):␣%s\nUpcoming␣(%d):␣%s",
                         readpos, repr(data[:readcount]),
                         self.file_position, repr(data[readcount:]))
    def stats(self, eventtime):
        if self.work_timer is None:
            return False, ""
        return True, "sd_pos=%d" % (self.file_position,)
    def get_file_list(self, check_subdirs=False):
        if check_subdirs:
            flist = []
            for root, dirs, files in os.walk(
                    self.sdcard_dirname, followlinks=True):
                for name in files:
                    ext = name[name.rfind('.')+1:]
                    if ext not in VALID_GCODE_EXTS:
                        continue
                    full_path = os.path.join(root, name)
                    r_path = full_path[len(self.sdcard_dirname) + 1:]
                    size = os.path.getsize(full_path)
                    flist.append((r_path, size))
            return sorted(flist, key=lambda f: f[0].lower())
        else:
            dname = self.sdcard_dirname
            try:
                filenames = os.listdir(self.sdcard_dirname)
                return [(fname, os.path.getsize(os.path.join(dname, fname)))
                        for fname in sorted(filenames, key=str.lower)
                        if not fname.startswith('.')
                        and os.path.isfile((os.path.join(dname, fname)))]
            except:
                logging.exception("virtual_sdcard␣get_file_list")
                raise self.gcode.error("Unable␣to␣get␣file␣list")
    def get_status(self, eventtime):
        return {
            'file_path': self.file_path(),
            'progress': self.progress(),
            'is_active': self.is_active(),
            'file_position': self.file_position,
            'file_size': self.file_size,
        }
    def file_path(self):
        if self.current_file:
            return self.current_file.name
        return None
    def progress(self):
        if self.file_size:
            return float(self.file_position) / self.file_size
        else:
            return 0.
    def is_active(self):
        return self.work_timer is not None
    def do_pause(self):
        if self.work_timer is not None:
            self.must_pause_work = True
            while self.work_timer is not None and not self.cmd_from_sd:
                self.reactor.pause(self.reactor.monotonic() + .001)
    def do_resume(self):
        if self.work_timer is not None:
            raise self.gcode.error("SD␣busy")
        self.must_pause_work = False
        self.work_timer = self.reactor.register_timer(
            self.work_handler, self.reactor.NOW)
    def do_cancel(self):
        if self.current_file is not None:
            self.do_pause()
            self.current_file.close()
            self.current_file = None
            self.print_stats.note_cancel()
        self.file_position = self.file_size = 0
    # G-Code commands
    def cmd_error(self, gcmd):
        raise gcmd.error("SD␣write␣not␣supported")
    def _reset_file(self):
        if self.current_file is not None:
            self.do_pause()
            self.current_file.close()
            self.current_file = None
        self.file_position = self.file_size = 0
        self.print_stats.reset()
        self.printer.send_event("virtual_sdcard:reset_file")
    cmd_SDCARD_RESET_FILE_help = "Clears␣a␣loaded␣SD␣File.␣Stops␣the␣print␣"\
        "if␣necessary"
    def cmd_SDCARD_RESET_FILE(self, gcmd):
        if self.cmd_from_sd:
            raise gcmd.error(
                "SDCARD_RESET_FILE␣cannot␣be␣run␣from␣the␣sdcard")
        self._reset_file()
    cmd_SDCARD_PRINT_FILE_help = "Loads␣a␣SD␣file␣and␣starts␣the␣print.␣␣May␣"\
        "include␣files␣in␣subdirectories."
    def cmd_SDCARD_PRINT_FILE(self, gcmd):
        if self.work_timer is not None:
            raise gcmd.error("SD␣busy")
        self._reset_file()
        filename = gcmd.get("FILENAME")
        if filename[0] == '/':
            filename = filename[1:]
        self._load_file(gcmd, filename, check_subdirs=True)
        self.do_resume()
    def cmd_M20(self, gcmd):
        # List SD card
        files = self.get_file_list()
        gcmd.respond_raw("Begin␣file␣list")
        for fname, fsize in files:
            gcmd.respond_raw("%s␣%d" % (fname, fsize))
        gcmd.respond_raw("End␣file␣list")
    def cmd_M21(self, gcmd):
        # Initialize SD card
        gcmd.respond_raw("SD␣card␣ok")
    def cmd_M23(self, gcmd):
        # Select SD file
        if self.work_timer is not None:
            raise gcmd.error("SD␣busy")
        self._reset_file()
        filename = gcmd.get_raw_command_parameters().strip()
        if filename.startswith('/'):
            filename = filename[1:]
        self._load_file(gcmd, filename)
    def _load_file(self, gcmd, filename, check_subdirs=False):
        files = self.get_file_list(check_subdirs)
        flist = [f[0] for f in files]
        files_by_lower = { fname.lower(): fname for fname, fsize in files }
        fname = filename
        try:
            if fname not in flist:
                fname = files_by_lower[fname.lower()]
            fname = os.path.join(self.sdcard_dirname, fname)
            f = io.open(fname, 'r', newline='')
            f.seek(0, os.SEEK_END)
            fsize = f.tell()
            f.seek(0)
        except:
            logging.exception("virtual_sdcard␣file␣open")
            raise gcmd.error("Unable␣to␣open␣file")
        gcmd.respond_raw("File␣opened:%s␣Size:%d" % (filename, fsize))
        gcmd.respond_raw("File␣selected")
        self.current_file = f
        self.file_position = 0
        self.file_size = fsize
        self.print_stats.set_current_file(filename)
    def cmd_M24(self, gcmd):
        # Start/resume SD print
        self.do_resume()
    def cmd_M25(self, gcmd):
        # Pause SD print
        self.do_pause()
    def cmd_M26(self, gcmd):
        # Set SD position
        if self.work_timer is not None:
            raise gcmd.error("SD␣busy")
        pos = gcmd.get_int('S', minval=0)
        self.file_position = pos
    def cmd_M27(self, gcmd):
        # Report SD print status
        if self.current_file is None:
            gcmd.respond_raw("Not␣SD␣printing.")
            return
        gcmd.respond_raw("SD␣printing␣byte␣%d/%d"
                         % (self.file_position, self.file_size))
    def get_file_position(self):
        return self.next_file_position
    def set_file_position(self, pos):
        self.next_file_position = pos
    def is_cmd_from_sd(self):
        return self.cmd_from_sd
    #------------------ Added code ------------
    # Regulate printer control based on communication input
    def _read_data(self):
        while True:
            buf = self.printer.readCommand()
            self.data_prev = self.data
            self.data = int.from_bytes(buf, byteorder='big')
            logging.info('Short␣Circuit␣Present:␣%s', buf)
            time.sleep(0.1)

            self.pause = True
            if self.set_break:
                break
        logging.info('Program␣Completed')
    #-----------------------------------------
    # Background work timer
    def work_handler(self, eventtime):
        logging.info("Starting␣SD␣card␣print␣(position␣%d)", self.file_position)
        self.reactor.unregister_timer(self.work_timer)
        try:
            self.current_file.seek(self.file_position)
        except:
            logging.exception("virtual_sdcard␣seek")
            self.work_timer = None
            return self.reactor.NEVER
        self.print_stats.note_start()
        gcode_mutex = self.gcode.get_mutex()
        partial_input = ""
        lines = []
        error_message = None

```

```python
        #----------------- Added code ------------
        listener_thread = threading.Thread(target=self._read_data, daemon=True)
        listener_thread.start()
        #-----------------------------------------
        while not self.must_pause_work:
            if not lines:
                # Read more data
                try:
                    data = self.current_file.read(8192)
                except:
                    logging.exception("virtual_sdcard␣read")
                    #----------------- Added code ------------
                    self.set_break = True
                    #-----------------------------------------
                    break
                if not data:
                    # End of file
                    self.current_file.close()
                    self.current_file = None
                    logging.info("Finished␣SD␣card␣print")
                    self.gcode.respond_raw("Done␣printing␣file")
                    break
                lines = data.split('\n')
                lines[0] = partial_input + lines[0]
                partial_input = lines.pop()
                lines.reverse()
                self.reactor.pause(self.reactor.NOW)
                continue
            #----------------- Added code ------------
            if self.data == 1 and self.pause == True:
                lines.append('G1␣Z-0.1␣F10\r') # add movement for short circuit
                lines.append('G1␣Z0.1␣F10\r')
                # lines.append('G4 P100') # add 0.1 [s] pause for multiple discharge detection
                self.pause = False
            #-----------------------------------------
            # Pause if any other request is pending in the gcode class
            if gcode_mutex.test():
                self.reactor.pause(self.reactor.monotonic() + 0.100)
                continue
            # Dispatch command
            self.cmd_from_sd = True
            line = lines.pop()
            if sys.version_info.major >= 3:
                next_file_position = self.file_position + len(line.encode()) + 1
            else:
                next_file_position = self.file_position + len(line) + 1
            self.next_file_position = next_file_position
            try:
                self.gcode.run_script(line, self.data)
            except self.gcode.error as e:
                error_message = str(e)
                try:
                    self.gcode.run_script(self.on_error_gcode.render(), self.data)
                except:
                    logging.exception("virtual_sdcard␣on_error")
                break
            except:
                logging.exception("virtual_sdcard␣dispatch")
                break
            self.cmd_from_sd = False
            self.file_position = self.next_file_position
            # Do we need to skip around?
            if self.next_file_position != next_file_position:
                try:
                    self.current_file.seek(self.file_position)
                except:
                    logging.exception("virtual_sdcard␣seek")
                    self.work_timer = None
                    return self.reactor.NEVER
                lines = []
                partial_input = ""

        logging.info("Exiting␣SD␣card␣print␣(position␣%d)", self.file_position)
        self.work_timer = None
        self.cmd_from_sd = False
        if error_message is not None:
            self.print_stats.note_error(error_message)
        elif self.current_file is not None:
            self.print_stats.note_pause()
        else:
            self.print_stats.note_complete()

        #----------------- Added code ------------
        listener_thread.join()
        #-----------------------------------------
        return self.reactor.NEVER
        # except KeyboardInterrupt:
        #     pass
        # finally:
        #     device.stop()

def load_config(config):
    return VirtualSD(config)
```

C
Power Supply

The power supply comprises of three primary components: the power source, power amplification, and square wave generator. Consequently, a comprehensive examination of numerous components is required. This chapter will present figures depicting the outputs of these components to assess their functionality.

C.1. DC to AC Converter

Figure C.1 displays the output of the DC to AC converter, commonly referred to as the ZVS Flyback Driver. This figure confirms the successful completion of the DC to AC conversion process as intended. Additionally, it highlights the amplification of pi, which is evident in the higher voltages present in the AC lines.

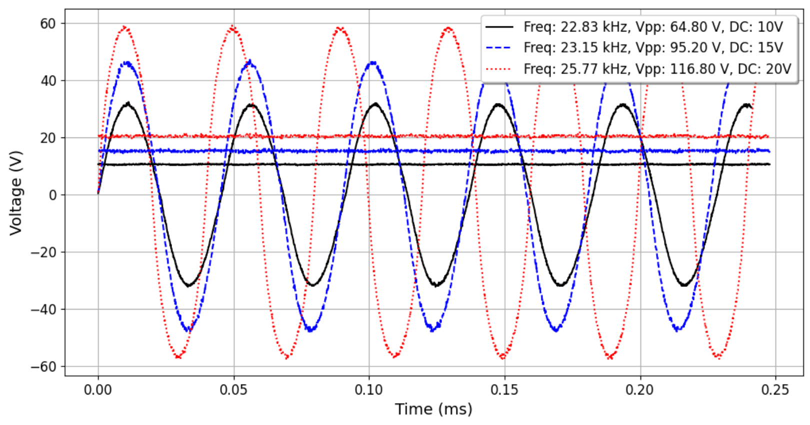


Figure C.1: Input and output voltage of the DC to AC converter.

C.2. Transformer

The output of the transformer is also important to measure, as it should deliver the desired voltage amplification. The amplification is shown in Figure C.2. The transformer has achieved the goal of amplifying the voltage by 2.5.

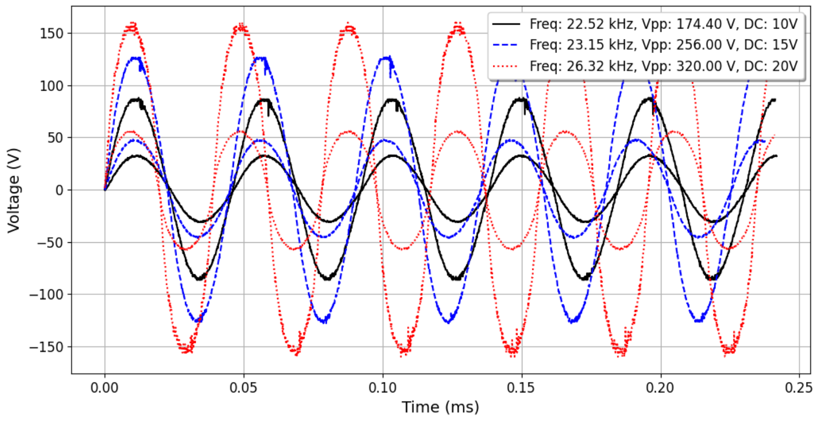


Figure C.2: Input and output voltage of the transformer.

C.3. Rectifier and Capacitor

The conversion of the AC to DC voltage is shown in Figure C.3. This figure shows the output of the rectifier and capacitor as the DC voltage.

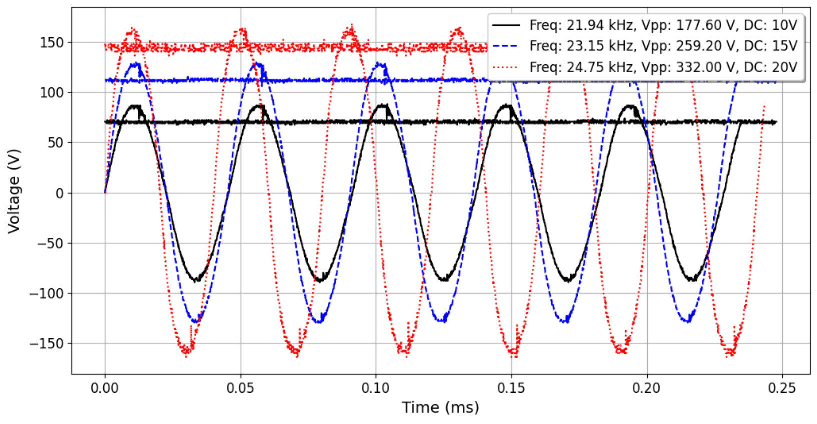


Figure C.3: Input and output voltage of the rectifier and capacitor.

C.4. Discharge Time of Circuit

To stabilize the voltage during discharge phases, the capacitance C calculated based on the charge Q discharged over 10 microseconds (the typical duration of a spark) with a 50A current, taking into account a voltage fluctuation ΔV of 20V:

$$Q = 50\text{ A} \times 10 \times 10^{-6}\text{s} = 500 \times 10^{-6}\text{C},$$

Thus:

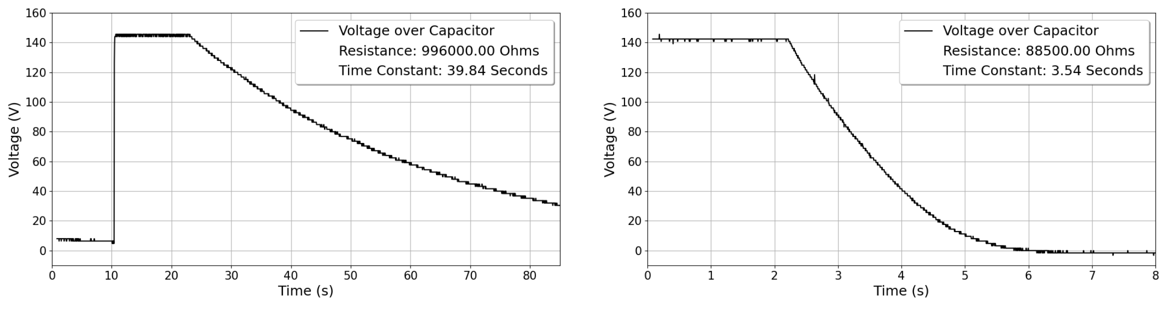
$$C = \frac{500 \times 10^{-6}\text{C}}{20\text{V}} = 25\mu F.$$

The choice of a 40 μF capacitor offers additional margin to ensure reliable operation under varying conditions and provides enhanced capability for energy storage and pulse handling, calculated as:

The resistor is placed in parallel with the capacitor for safety reasons. The resistor was chosen based on availability. The value of the chosen resistor is 1.2 MΩ. The resulting discharge time constant is calculated as follows:

$$\tau = R \times C = 1.2 \times 10^{6} \times 40 \times 10^{-6} = 48\text{s}$$

The experiments yielded the results depicted in Figures C.4, illustrating the variations in discharge time depending on whether the square wave is active or inactive. The theoretical RC calculation was fairly accurate when the square wave circuit was off. The discrepancy arises from the additional capacitances at the power source, which reduce the circuit's RC value. When the square wave circuit is on, the circuit exhibits a much lower RC value due to the significant influence of MOSFETs and other resistances, which accelerate the discharge process.

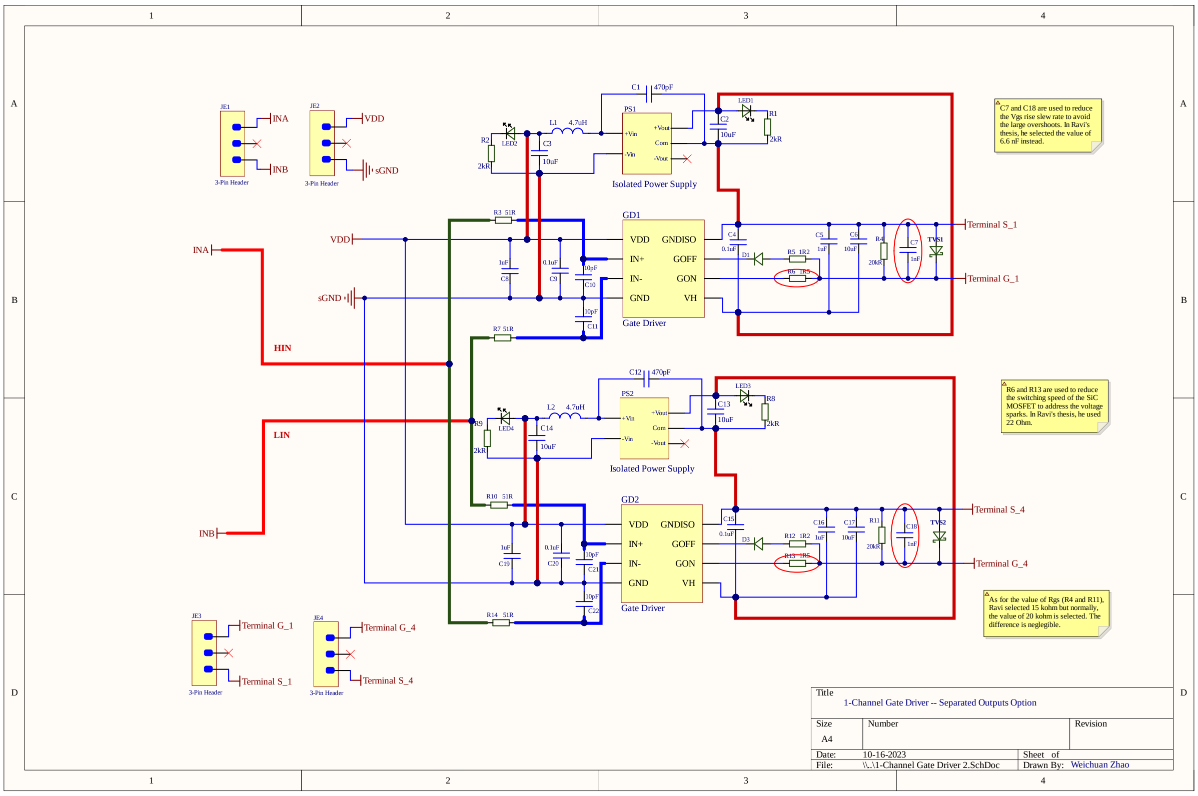


(a) The discharge time when the square wave circuit is turned off. **(b)** The discharge time when the square wave circuit is turned on.

Figure C.4: The discharge time of the power amplification circuit.

C.5. Square Wave Generator

The complete schematic of the square wave generator that had been given to us is visible in Figures C.5 and C.6.



C7 and C18 are used to reduce the Vgs rise slew rate to avoid the large overshoots. In Ravi's thesis, he selected the value of 6.6 nF instead.

R6 and R13 are used to reduce the switching speed of the SiC MOSFET to address the voltage sparks. In Ravi's thesis, he used 22 Ohm.

As for the value of Rgs (R4 and R11), Ravi selected 15 kohm but normally, the value of 20 kohm is selected. The difference is neglegible.

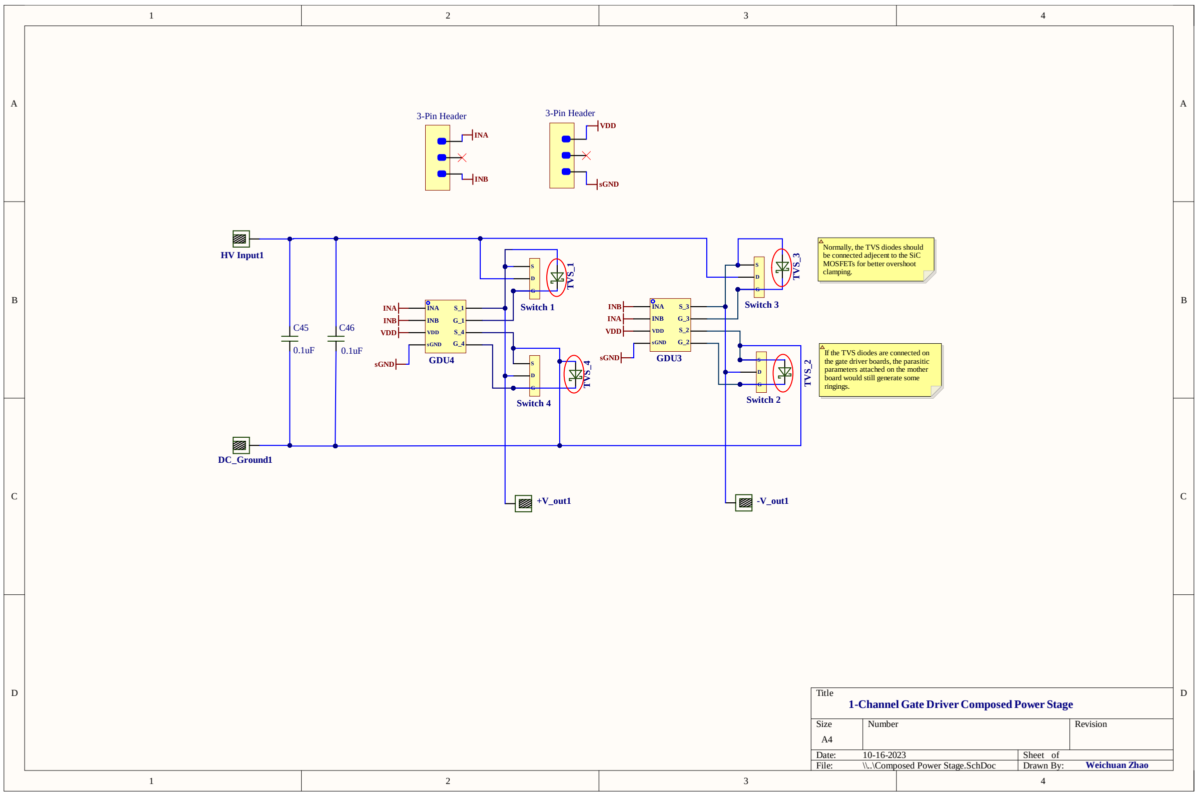
Title 1-Channel Gate Driver -- Separated Outputs Option

Size	Number	Revision
A4		

Date: 10-16-2023 Sheet of

File: \\..\1-Channel Gate Driver 2.SchDoc Drawn By: Weichuan Zhao

Figure C.5: Schematic of Gate Driver.



Normally, the TVS diodes should be connected adjecent to the SiC MOSFETs for better overshoot clamping.

If the TVS diodes are connected on the gate driver boards, the parasitic parameters attached on the mother board would still generate some ringings.

Title 1-Channel Gate Driver Composed Power Stage			
Size A4	Number		Revision
Date:	10-16-2023	Sheet of	
File:	\\..\Composed Power Stage.SchDoc	Drawn By:	Weichuan Zhao

Figure C.6: Schematic of Main Board with gate drivers.

D

IRF300P226 MOSFET voltage-current limits

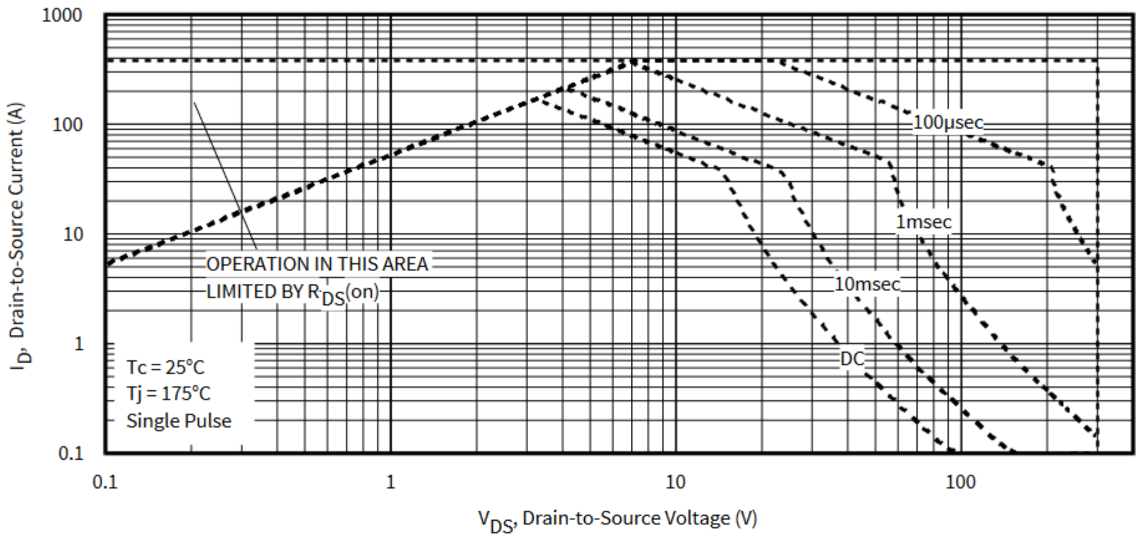


Figure D.1: IRF300P226 MOSFET voltage-current limits [46].

E
IPW65R037C6 MOSFET voltage-current limits

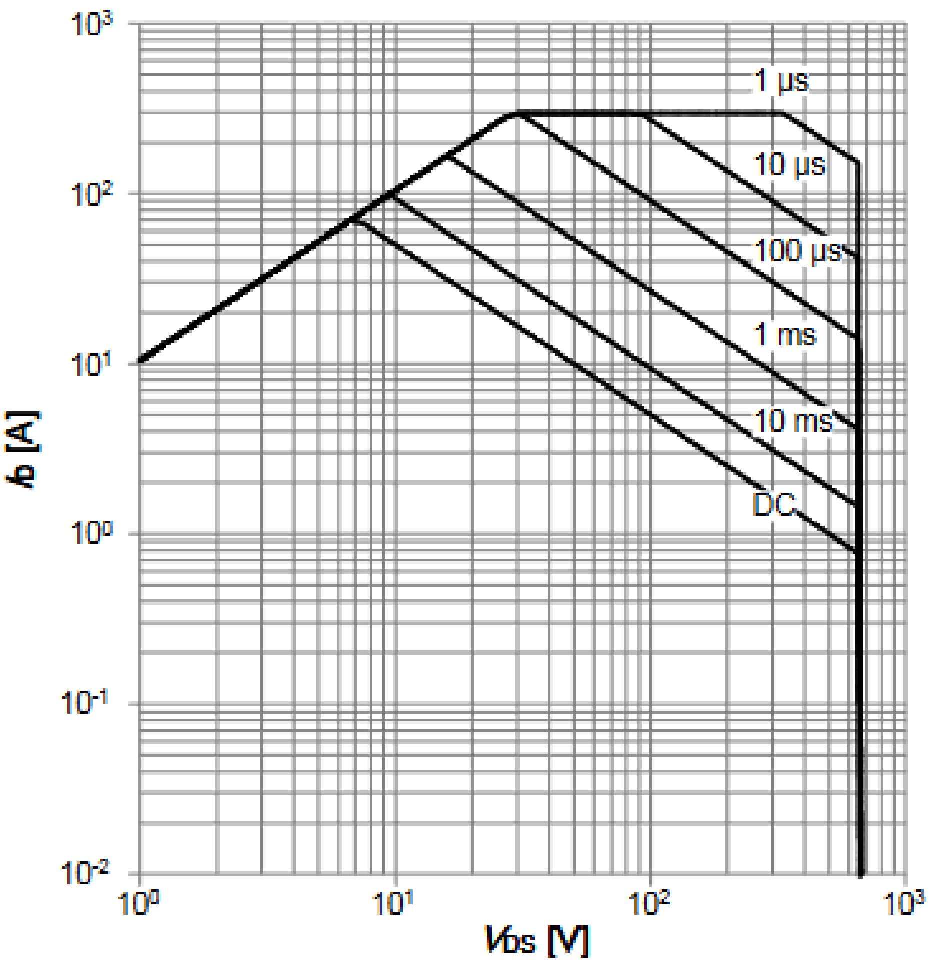


Figure E.1: IPW65R037C6 MOSFET voltage-current limits [47].