UPPSALA
UNIVERSITET

Universal PID-control and control of an electron gun vacuum system

Project in Embedded Systems

Simon Gollbo

2017-03-17

Supervisors: Uwe Zimmermann, Ping Wu

Abstract

The purpose of this project is to design a control system for a vacuum system in the Ångström laboratory clean room. This entails interfacing a microcontroller to industrial grade electronics and a touch screen interface for user input. A universal PID-controller is also designed, to which the user can input feedback gains using the touch screen. The project was prototyped on a breadboard after which a PCB was designed, ordered and hand-mounted. The resulting PCB, after some minor adjustments, worked to specifications.

Contents

1. Introduction 3
  1.1. Background 3
  1.2. Purpose of the project 3
  1.3. Project specifications 4
  1.4. Project planning 4
    1.4.1. Hardware 4
    1.4.2. Software 5
  1.5. Grading criteria 5
2. Working principles 7
  2.1. Control theory 7
  2.2. Introduction to PID-control 8
  2.3. PID-control in discrete time 9
  2.4. Touch screen 10
  2.5. Microcontroller 11
3. Implementation 11
  3.1. Overview of the system 12
  3.2. Hardware and components 12
    3.2.1. Microcontroller 12
    3.2.2. PCB 12
    3.2.3. Relays 12
    3.2.4. Optocouplers 12
    3.2.5. Touch screen 12
    3.2.6. Power supply 12
    3.2.7. USBasp programmer 13
  3.3. Integrated Development Environment (IDE) 13
  3.4. Development tools 13
    3.4.1. Sublime text 13
    3.4.2. Git 13
    3.4.3. AVRDUDESS 13
    3.4.4. KiCad 13
  3.5. Implementation 13

3.5.1. Hardware 13
3.5.2. Software 17
4. Results and discussion 22
5. Conclusions 23
6. References 24

1. Introduction

1.1. Background

In the clean room of the Ångström Laboratory there is an electron gun system which is used to deposit thin films of metal onto a substrate. The electron gun uses an electric field to accelerate electrons which form an electron beam of high power. The electron beam is then directed using magnetic fields into a crucible that holds the metal to be deposited. The high energy of the electron beam vaporizes the metal and the metal vapor shoots upwards toward the substrate target.

To avoid oxidation and contamination of the samples, the process needs to be carried out in a vacuum chamber. The vacuum chamber for the system at the Ångström Laboratory utilizes two pumps, an air pump and a cryo pump. The air pump is able to bring the chamber from atmospheric pressure down to $10^{-2}\ mbar$. At this pressure, the air pump is unable to lower the pressure further and the cryo pump is used. The cryo pump works by moving a portion of gas from the chamber to a separate chamber. The separate chamber is then cooled down using liquid helium, which compresses the gas inside. The compressed gas is then moved to a storage tank. The cryo pump is able to bring the pressure in the vacuum chamber down to $10^{-8}\ mbar$. The air pump is required due to the limited volume of the cryo pump storage tank. The cryo pump storage tank fills quickly if the cryo pump is used at atmospheric pressure. An overview of the system is shown in Fig. 1.1.

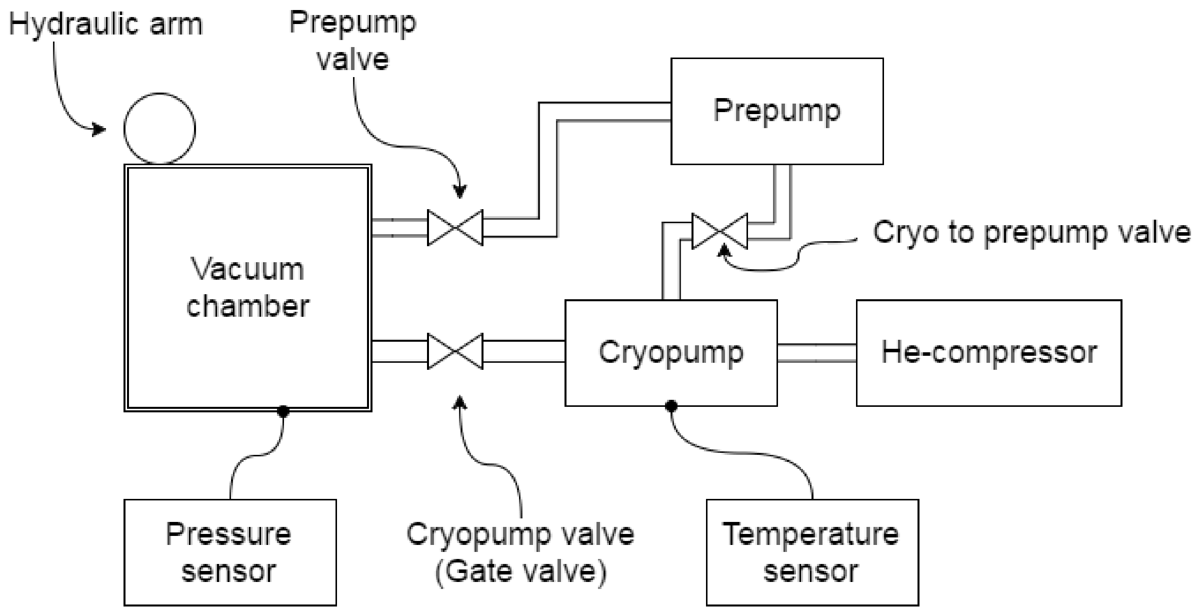


Fig. 1.1: Overview of the vacuum system which contains the electron gun.

In its current implementation, the system is manually controlled with each valve, pump et cetera having a separate control interface. Some valves are electronically controlled while others are manually screwed open or closed, but the system is operable in its current implementation.

1.2. Purpose of the project

The purpose for this project is to upgrade the control system for the vacuum system. Currently, there is no mechanism for ensuring that the user performs the correct sequence of actions to utilize the vacuum system. An example of erroneous usage is incorrectly turning on the cryo pump at atmospheric pressure which leads to the cryo pump storage tank filling up. If this were to happen, the cooling of the system has to be turned off and the gas has to be allowed to heat up before being ventilated. The system then has to cool back down. This entire process takes several days to complete during which time the electron gun cannot be used.

Also, the current control system is spread out over several controls. This project also aims to gather all controls in a single control device.

The project also aims to implement a universal PID-controller. The scope of the project does not include implementing the PID-controller in the vacuum system, but control of the metal crucible could be controlled using this type of controller. As such, the universal PID-controller enables further work on the control system in the future.

1.3. Project specifications

The control system is to use a touch screen interface with a graphical user interface which displays the current state of the system. The various valves and pumps are to be controlled via the touch screen. The touch screen is run at 3.3V.

The main logic of the control system is implemented on a microcontroller. The microcontroller is interfaced with the touch screen and reads where the user is touching the screen as well as communicating what the screen is to display.

The valves and pumps have existing interfaces for control, however they operate at 24V. This means that an interface is required between the logic levels of the microcontroller, which is run at 3.3V, and the external 24V controls. To protect the microcontroller and touch screen, some form of insulation with regards to the external interfaces is required.

The system should disallow or warn the user when a selected action is not appropriate, for example if the user should attempt to open an erroneous combination of valves.

The universal PID-controller should also be controlled via the touch screen interface. The feedback gains of the PID-controller should be customizable, as well as the reference signal level.

1.4. Project planning

1.4.1. Hardware

The first objective to be achieved is to get the microcontroller online and programmable on a breadboard for prototyping and testing purposes. Related to this is also establishing correct procedure for communication between the microcontroller and the touch screen and getting the screen to display basic objects. Also, the touch interface should be made to output values when pressed.

Next, the touch interface coordinates should be calibrated such that the coordinates read for a press on the touch interface corresponds to the same position on the graphics display.

To insulate the microcontroller from the digital output side of the system mechanical relays are to be used. These should be tested. On the digital input side of the system, optocouplers should be used. Similarly, these should be tested.

When the components of the system have been prototyped and tested, a PCB is to be designed and ordered along with components. The components are to be soldered by hand to the PCB. When the PCB has been mounted, it is to be tested.

1.4.2. Software

The first software objective is to program a hello world for the microcontroller to ensure functionality.

Functions for interfacing with the touch screen should be written, essentially an application programming interface (API). A set of functions that provide functionality for communicating with the screen, a set of functions that allow objects and text to be written on the screen and a set of functions for interfacing with the touch interface, for example to retrieve touch coordinates, form the main components of the API.

Using these functions, graphical user interfaces (GUIs) should be designed for the vacuum control system and the universal PID-controller respectively. The vacuum control system GUI should display the current state of the system and visualize a system overview. It should also have menu screen for the various components of the system. The universal PID-controller GUI should have a main screen displaying the current feedback gains as well as the current reference level. It should also contain numerical input menu screens for changing the feedback gains and the reference level.

A main routine is also to be written. This main routine should coordinate the subroutines that handle the touch interface, the drawing to the screen as well as the input and output logic. It should poll the touch interface for user input and draw the various GUI screens on the screen depending on what the user inputs.

1.5. Grading criteria

The project will be graded using the U, 3, 4, 5 scale. Grade U is non-passing while the numbered grades are all passing grades with higher numbers corresponding to higher grades and grade 5 being the highest grade achievable. The following criteria will be used when grading the project:

Grade 3: To achieve grade 3, a user should be able to interact with the touch screen and change some digital output. Basic 2-level control should be implemented. The project should be implemented on a prototyping breadboard.

Grade 4: To achieve grade 4, the project should, in addition to what is required for grade 3, implement a functioning universal PID-controller. The PID-controller feedback gains should be modifiable by interacting with the touch screen interface. The project should be implemented on a prototyping breadboard.

Grade 5: To achieve grade 5, the project should, in addition to what is required for the lower grades, be implemented and functioning on a PCB which should be designed using an electronics CAD software.

2. Working principles

2.1. Control theory

Control theory in general describes how the output or outputs of some system can be controlled to behave according to a user using input or inputs to the system. The aim of control theory is to determine what inputs should be sent into a system for the system outputs to behave as a user defines. An example of a mechanism achieving this is a thermostat, which takes as input a user defined reference temperature and controls some thermal element such that the measured temperature in for example a room maintains the user defined level.

A system can be described with the block diagram shown in Fig. 2.1 where u is the input control signal, y is the output signal or measurement signal and v is some unknown disturbance signal. (Glad & Ljung)

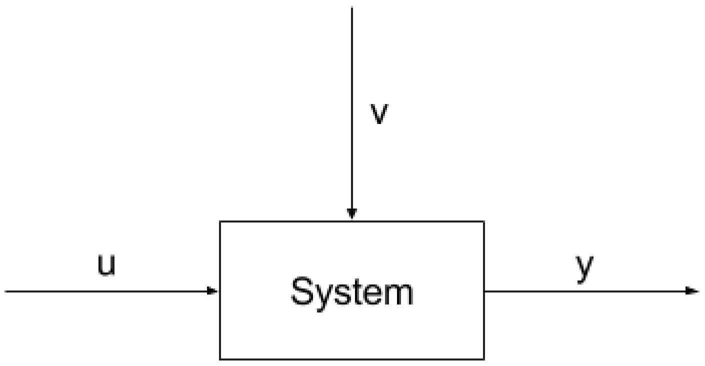


Fig. 2.1: Block diagram of system where u is the input control signal, y is the output signal or measurement signal and v is some unknown disturbance signal. (Glad & Ljung)

The input signal u can be either scalar or a vector in the case where the system accepts multiple inputs. Likewise, the output or measurement signal y and the disturbance signal v can also be either scalars or vectors depending on what model of the system is used.

Perhaps the most simple system studied in control theory is a system that has a scalar input and a scalar output, a so-called single-input-single-output (SISO) system. SISO systems are very well studied in control theory and there are a multitude of different approaches on controlling SISO-systems. Some of these approaches are generalizable to systems that have multiple inputs and/or multiple outputs.

A common theme in control theory is to base the control signal u on an error signal e, which is created by subtracting the output y from the user defined reference level r. This is called *feedback control.* The mechanism that produces the control signal u attempts to drive the error signal e to zero, thereby making the system output equal to the input reference level. This mechanism is called a *controller*. A block diagram of feedback control is shown in Fig. 2.2.

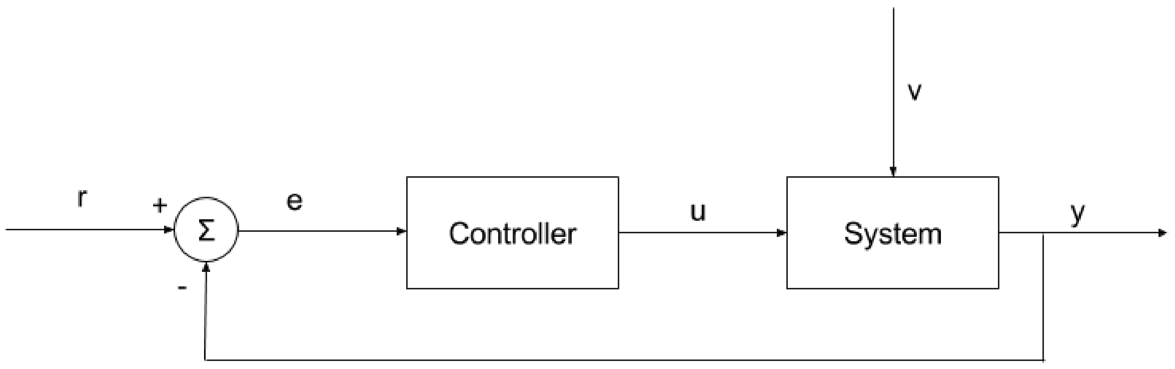


Fig. 2.2: Block diagram of feedback control system.

Depending on what is known about the system and the disturbance, various modifications can be made to the feedback control scheme shown in Fig. 2.2 to make use of the additional information. In general, if additional information about the system dynamics is available, the control scheme can be made more precise and tailored to the specific system which is being controlled. This leads to better control performance, typically meaning that the system is able to achieve a stable output in a shorter amount of time, without too much rippling in the output signal.

For a control system to be usable in a universal manner, the extent of consideration for the internal system dynamics of the system being controlled should be minimized. Some demands for the controllability of the system will have to be imposed, but further demands should be kept to a minimum. A controller structure which does not require information about the internal system dynamics is the so-called Proportional-Integral-Derivative (PID) controller. This type of controller is described in more detail in the following section.

2.2. Introduction to PID-control

The PID-type controller is a feedback control scheme which consists of three parts, as the name suggests, a proportional feedback, an integrating feedback and a differentiating feedback. These three parts each have a feedback gain assigned to them, essentially controlling how much each of the three parts affects the control signal.

Instead of providing additional information about the internal dynamics of the system being controlled, the 3 feedback gains are used to tune the controller to achieve satisfactory control performance of the PID-controller.

The inner workings of a PID-controller is shown in block diagram form in Fig. 2.3, where Kp is the proportional feedback gain, Ki is the integral feedback gain and Kd is the differentiating feedback gain.

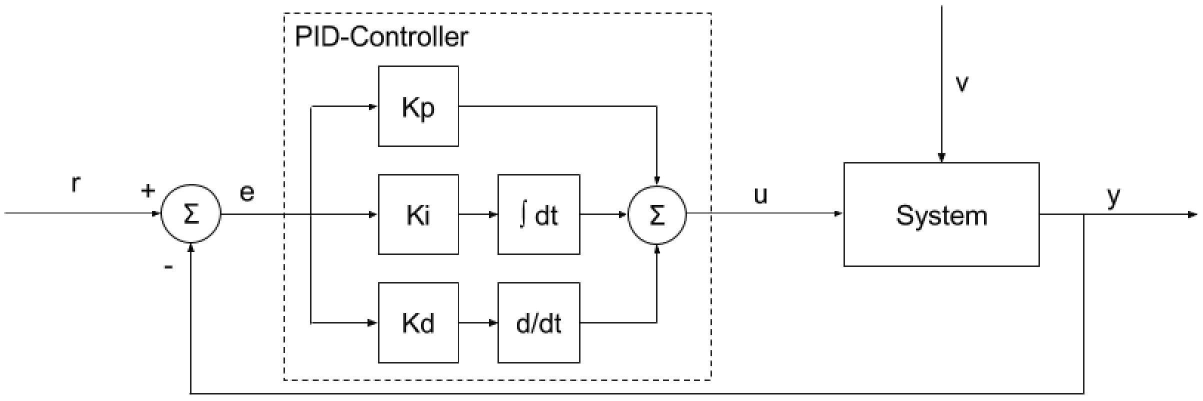


Fig. 2.3: Block diagram that shows the internal dynamics of a PID-controller. Kp is the proportional feedback gain, Ki is the integral feedback gain and Kd is the differentiating feedback gain.

The workings of the PID-controller can also be described mathematically. Eq. (2.1) describes the relationship between the generated control signal u and the error signal e when using an ideal PID-controller in continuous time.

$$u(t) = Kp * e(t) + Ki * \int_{t_0}^{t} e(\tau)d\tau + Kd * \frac{de(t)}{dt} \tag{2.1}$$

The proportional feedback is directly proportional to the control error e. The proportional feedback is direct in its acting and when the error signal e is large it provides a large contribution to the control signal u. It essentially constantly tells the system to move in the direction of the reference signal, given correct signage.

The integrating feedback keeps track of the control error over time. When properly designed, it ensures that the system, if stable, is controlled to control error zero in the steady state. Compared to the proportional feedback, the integrating feedback is slower in its acting but is able to compensate for small errors given enough time, something the proportional feedback may not be able to.

The differentiating feedback generates a large contribution to the control signal u whenever the control error e changes rapidly. If the system suddenly jerks, the differentiating feedback is able to provide the control signal with an extra boost to compensate for the quick change in system output. Another example is if the user makes a large sudden change in the reference level, then the differentiating feedback helps kick start the system to quickly move the system to the new reference level.

2.3. PID-control in discrete time

When controllers are implemented in digital form, they have to be ported from a continuous time domain to a discrete time domain. The error signal e is sampled with some periodicity T_s.

In Fig. 2.4 the control circuit is shown in a block diagram form with a digital microcontroller (MC). The output or measurement signal y is converted to digital form yd using an analog-to-digital converter (ADC). The ADC samples y and converts it to a digital signal to some number of bits precision and sends a digital packet to the MC containing these bits. The MC can then perform the necessary control calculations before outputting a digital control signal ud to the digital-to-analog converter (DAC) which converts the digital control signal into an analog control signal u.

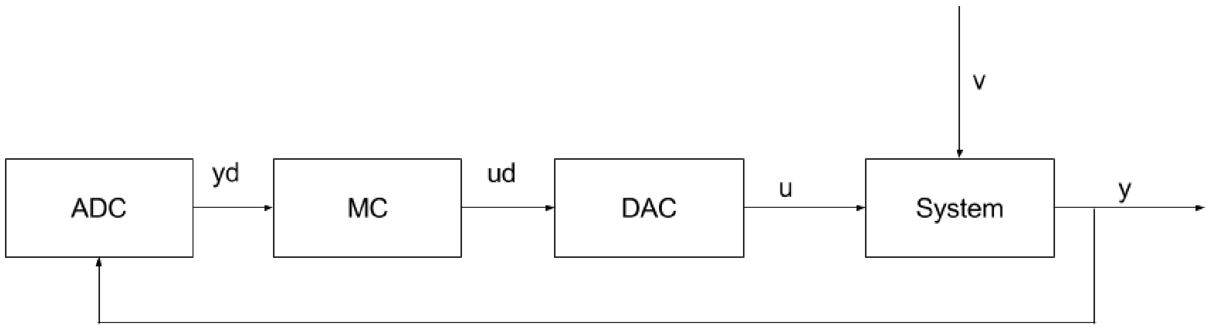


Fig. 2.4: Control circuit with digital MC. The output or measurement signal y is converted to digital form yd using an ADC. The MC outputs a digital control signal ud which is converted to an analog control signal u by using a DAC.

For the MC to be able to perform PID-control, Eq. (2.1) has to be discretized. This discretization can be performed according to different schemes with respect to differentiation. One such discretization scheme is shown in Eqs. (2.2) and (2.3), where the Euler forward method of numerical differentiation is used.

$$u[t] = Kp * e[t] + Ki * integralError[t] * \Delta t + Kd * \frac{(e[t] - e[t-1])}{\Delta t} \tag{2.2}$$

$$integralError[t] = e[t] + integralError[t-1] \tag{2.3}$$

The proportional feedback is essentially unaffected by the discretization of time. The integrating feedback is transformed to a summation of all error terms up to the current time t.

The factor Δt is the time taken between loops, the time for the controller to return to the calculation of the discrete control signal $u[t]$. If the sampling interval $T_s \gg calculation\ time\ for\ u[t]$ then $\Delta t \approx T_s$.

To scale the integration and compensate for varying execution times, the factor Δt is multiplied into the integrating feedback. This essentially turns the discrete time-point approximation into a continuous time approximation where the error signal e during the time period Δt is taken as $e[t]$.

In a similar fashion, the differentiating feedback is divided by this same factor Δt. This approximates the derivative at time t, by finding the slope between the line that connects $e[t]$ and $e[t-1]$.

2.4. Touch screen

The touch screen to be used in the project is a so-called 4-wire resistive touch screen. This type of screen consists of two sheets layered on top of each other, with the top layer being flexible. An example of this configuration is shown in Fig. 2.5, where the bottom sheet consists of glass, to form a stable base, and the top sheet consists of a flexible plastic film.

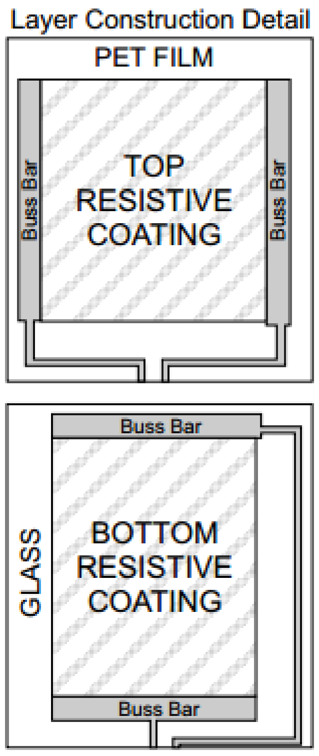


Fig. 2.5: Example of a 4-wire touch screen configuration. (Sparkfun)

The two sheets are coated in a resistive material. One layer is outfitted with connection terminals along either the north and south sides of the layer or the east and west sides of the layer. The second layer is outfitted with terminals in whichever configuration was not applied to the first layer. (Sparkfun)

To operate the screen, a voltage is applied across the north terminal to the south terminal or across the east terminal to the west terminal. When the sheets are brought together they start conducting essentially forming a voltage divider with one resistor on the top sheet and a second resistor on the bottom sheet. By reading the voltage on one of the unused terminals, the ratio of the read voltage value and the full applied voltage value yields the ratio of where the screen was pressed, in a single dimension. The procedure can then be repeated by applying the voltage across whichever north and south or east and west configuration was not used. The resulting pair of measurements corresponds to where the screen was pressed. (Sparkfun)

2.5. Microcontroller

A microcontroller is a system-on-chip consisting of a microprocessor with additional peripherals such as general purpose input/output (GPIO) ports, timer modules, memory modules, analog to digital converter (ADC) modules et cetera.

The microcontroller is programmed by uploading compiled code to its memory. As the microcontroller boots up, it reads the program memory and starts executing instructions.

3. Implementation

3.1. Overview of the system

The system consists of a microcontroller which can be interfaced to industrial grade electronics. The system state can be controlled by a user via a touch screen interface which also displays a graphical user interface. The system is also able to read analog voltage signals as well as being able to output pulse width modulated signals that can be filtered to achieve outputting of analog voltage signals.

3.2. Hardware and components

3.2.1. Microcontroller

For prototyping purposes the ATMega328 in a PDIP capsule was used on a breadboard. For the PCB the ATMega644 in a QFP capsule was chosen for its additional GPIO ports to allow for additional flexibility in the finished project.

3.2.2. PCB

The PCB was designed using a single layer for components and two copper layers. Power nets and ground nets are supplied via polygons and stitched with vias to ensure stable voltage levels. The polygons do not flow under the external input and output components to further isolate the microcontroller from externalities. The traces connecting the output channels were made thicker to be able to withstand higher currents.

3.2.3. Relays

To output digital signals to external systems running at higher voltages and/or currents than the microcontroller can withstand, mechanical relays are used. A mechanical relay typically has three connector leads, one common, one normally closed and one normally open. Additionally a mechanical relay has coil leads. The coil acts as an electromagnet, mechanically switching a contact between the normally closed lead and the normally open lead. The microcontroller output port drives the base of a bipolar transistor which can switch the relay coil current and toggle the relay. This enables the microcontroller to send an output to an external device without having to interface with it directly.

3.2.4. Optocouplers

To input digital signals from external systems running at higher voltages and/or current than the microcontroller can withstand, optocouplers are used. Optocouplers are optically isolated and contain an LED on the input side and a phototransistor on the output side. When an input switches the LED on, the phototransistor begins to conduct. This enables the microcontroller to detect that an input has been switched on without it having to interface with the external system directly.

3.2.5. Touch screen

The touch screen used is a 4-wire resistive touch panel with a 320 by 240 pixel LCD display. It has a variety of interfaces for communicating with the screen driver. The Serial Peripheral Interface (SPI) was chosen for this purpose.

3.2.6. Power supply

During prototyping, power was provided via the USB-port of the programming computer. USB outputs a maximum of 5V which was not sufficient to drive the mechanical relay coils. To test these, an external variable power supply was used, which could provide up to 20 V.

Since the mechanical relays need a higher voltage than the microcontroller, the PCB was outfitted with a switched voltage regulator. As input, the board is supplied with 24V, which is regulated down to 3.3V for the microcontroller and touch screen.

3.2.7. USBasp programmer

To transfer the compiled program from the computer to the microcontroller memory, a USBasp programmer clone was used.

3.3. Integrated Development Environment (IDE)

The software for the microcontroller was written in C. To compile the code, AtmelStudio was used. AtmelStudio provides microcontroller specific project creation where necessary definitions are automatically included.

The text editor in AtmelStudio is decent, but I opted to use Sublime Text, detailed below, for writing the software. AtmelStudio was quick to recognize whenever the file being edited in Sublime Text was saved and was immediately ready to compile.

3.4. Development tools

3.4.1. Sublime text

As mentioned above, Sublime Text was used to write the code. Sublime Text has many handy features, such as multi-selection editing and auto completion, which made it my primary choice. The program is responsive and feels lightweight which was also found advantageous.

3.4.2. Git

Git was used to backup and version code. It was also handy when working from different locations, being able to synchronize files.

3.4.3. AVRDUDESS

The software used to communicate with the USBasp programmer was AVRDUDESS, which is a graphical user interface for the programming API AVRDude. It has a built in library of presets for various different programmers and is able to detect what type of microcontroller it is connected to.

3.4.4. KiCad

To design the PCB, the open source program KiCad was used. KiCad comes equipped with tools to draw circuit schematics, footprints and PCB schematics as well as some additional tools.

3.5. Implementation

3.5.1. Hardware

The project hardware had two main phases, a prototype phase and a PCB phase. The prototype consisted of an ATMega328 in a PDIP package mounted on a breadboard. The ATMega328 was connected to the programmer and the touch screen using external cables. An image of the setup is shown in Fig. 3.1. The figure shows the programmer circuit in the red and white rectangle and the microcontroller under the wires in the blue and white rectangle. To the right in the figure, the touch screen is shown. In the top right, the power supply that was used for testing the mechanical relays is shown.

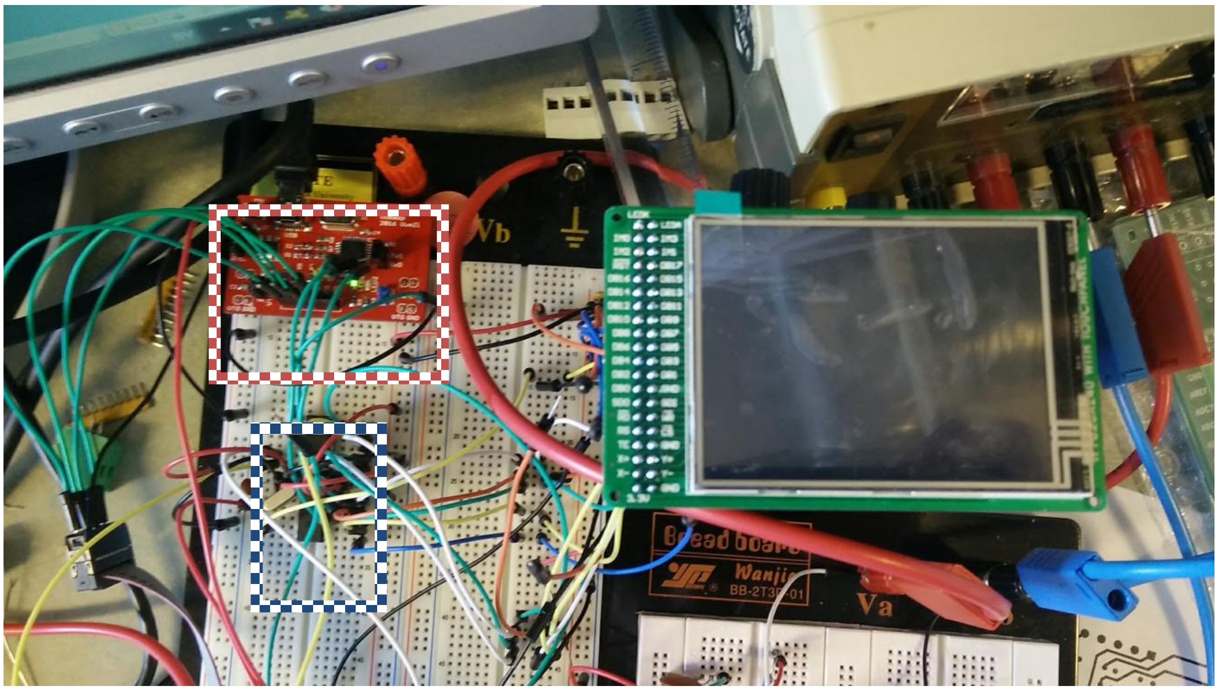


Fig. 3.1: Prototype breadboard setup. The programmer is shown inside the red and white rectangle in the top left. The ATMega328 microcontroller is under the wires in the blue and white rectangle below the programmer.

During design of the PCB, flexibility of the project was taken into account. As a result of this, all pins not used by the microcontroller to communicate to the screen, relays and optocouplers were connected to terminal connectors on the sides of the board. This allows a user a certain amount of flexibility in the application of the project and it could be adapted to service other purposes than the originally intended one by simple reprogramming the microcontroller.

Flexibility was also considered when designing the output ports. The relays, as covered in section 3.2.3, have 3 connector leads. These 3 leads are connected to terminals on the PCB side which let the user choose what kind of signals the relay should be used for. This allows, for example, switching AC current or, as previously mentioned, higher voltages than the microcontroller can handle.

Standard PCB design considerations were taken into account when designing the PCB. Decoupling capacitors were placed close to the pins they decouple. As mentioned in section 3.2.2, power nets and ground nets are supplied via polygons with via stitching.

The voltage regulator on the PCB is a switched regulator. Switched regulators can introduce quite a bit of noise in the circuit. To prevent disturbances on the analog ports of the microcontroller, the voltage regulator was placed away from them. The microcontroller also has an analog ground pin, this was connected to the rest of the ground net via ferrite bead. The ferrite bead is able to suppress high frequency noise.

When the PCB design was finished and the PCBs arrived, work was started to mount components on the board. In Fig. 3.2 the PCBs without mounted components is shown.

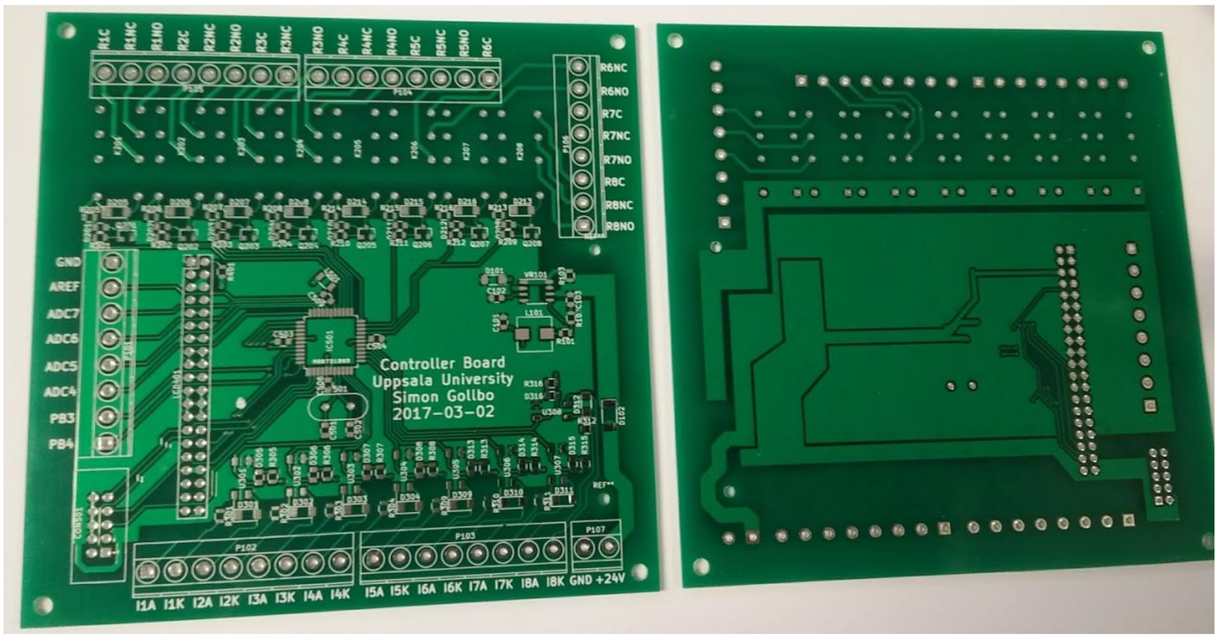


Fig. 3.2: Project PCBs without components mounted.

As mounting progressed, it was discovered that an inductor had been misplaced in the design of the power supply unit. Due to time constraints it was decided against ordering new PCBs for the moment and instead rewiring the affected pins by cutting away connections that were incorrect and soldering Teflon insulated wires to make the missing correct connections. Also, it was opted to use a larger capacitor between the ground plane and the 24V feed line than was in the original design. To mount this, the plastic layer was scraped off and the capacitor was soldered directly to the exposed copper. Fig. 3.3 shows the resulting power supply unit on the PCB.

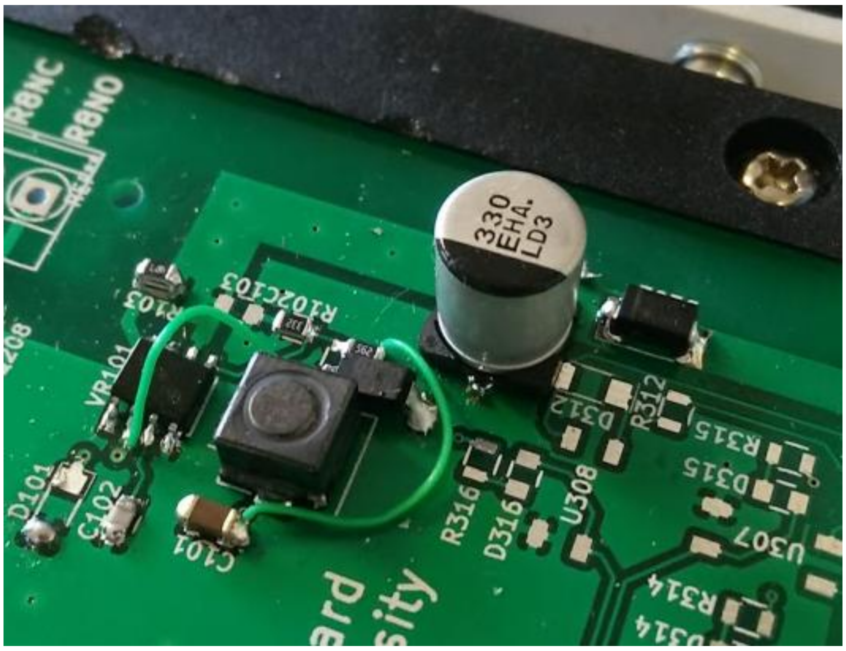


Fig. 3.3: The power supply circuit after alteration to correct for the erroneous placement of the inductor.

No further errors were discovered during mounting. The finished PCB is shown without the screen mounted in Fig. 3.4 and with the screen mounted in Fig. 3.5.

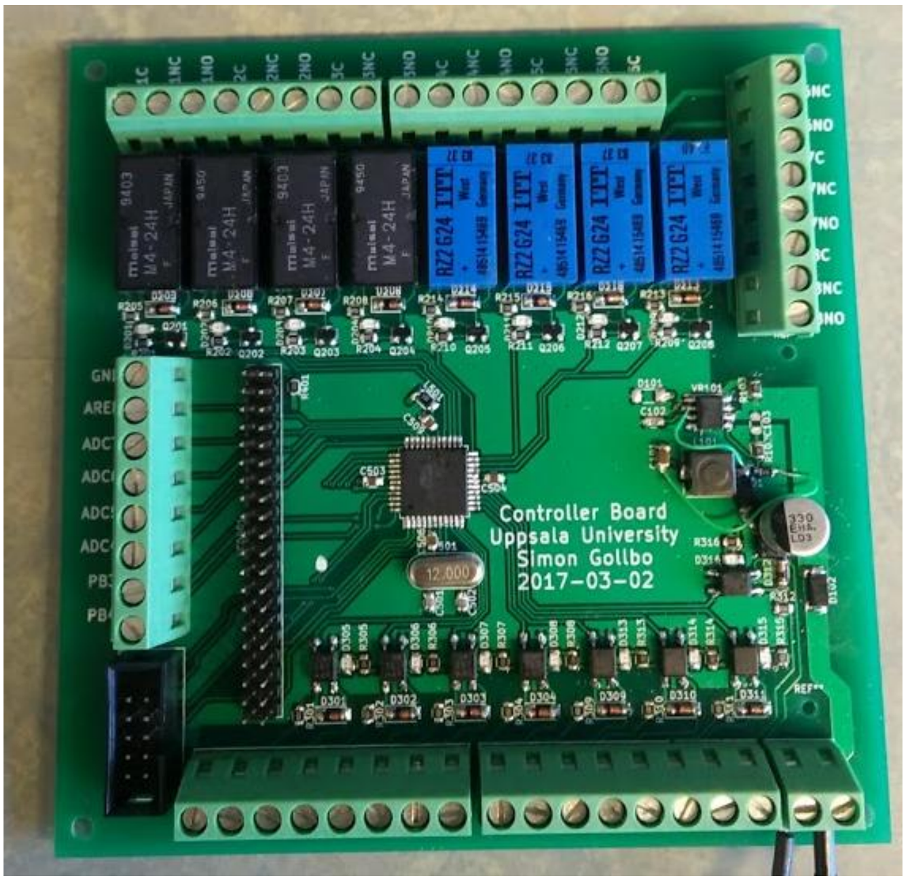


Fig. 3.4: The PCB without the touch screen mounted

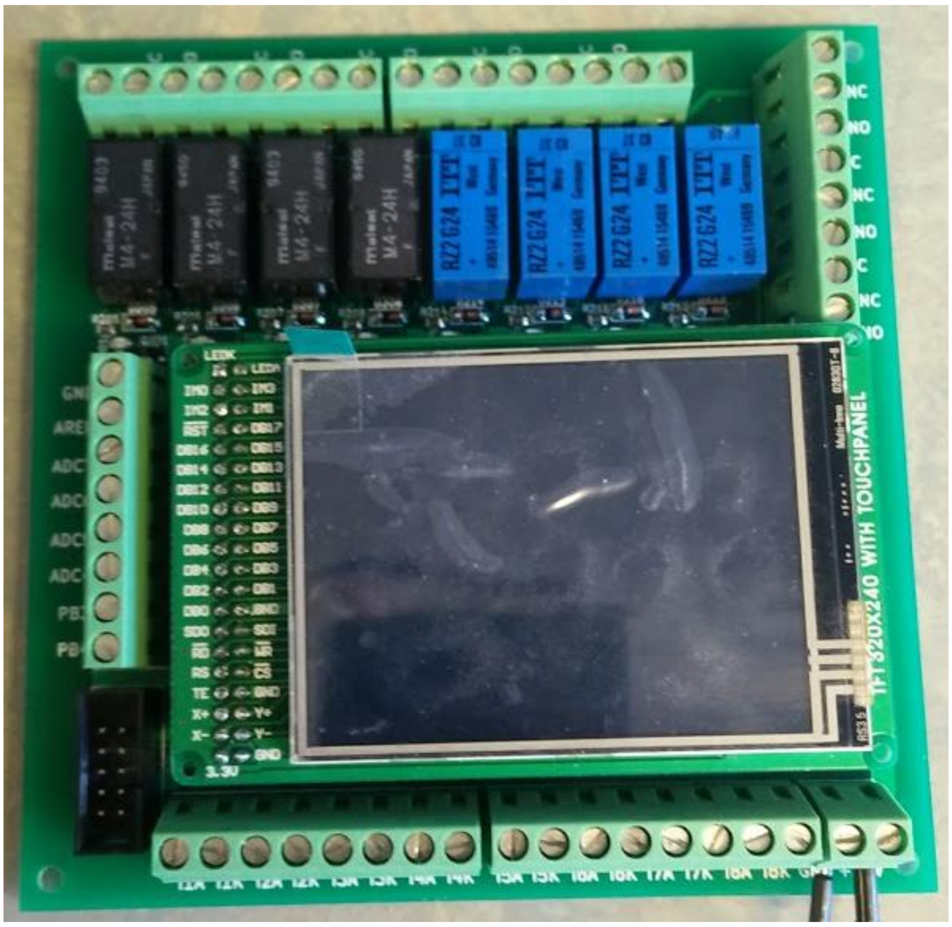
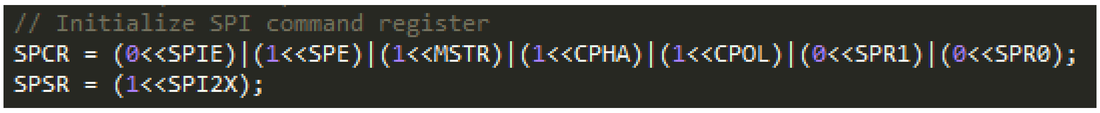


Fig. 3.5: The PCB with the touch screen mounted

3.5.2. *Software*

The software was implemented in C.

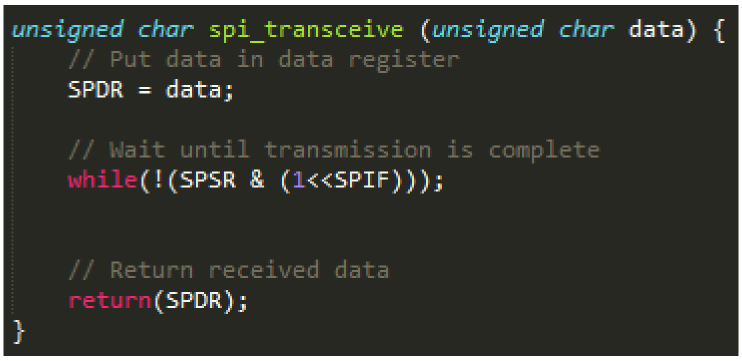
To communicate with the touch screen, as mentioned in section 3.2.5, the SPI mode was chosen. The ATMega microcontrollers used in this project both come equipped with hardware SPI modules. In Fig. 3.6, a code snippet is shown which sets the appropriate values in the SPI command registers to enable the hardware SPI module.



```
// Initialize SPI command register
SPCR = (0<<SPIE)|(1<<SPE)|(1<<MSTR)|(1<<CPHA)|(1<<CPOL)|(0<<SPR1)|(0<<SPR0);
SPSR = (1<<SPI2X);
```

Fig. 3.6: Setting the command registers to initialize the SPI hardware module

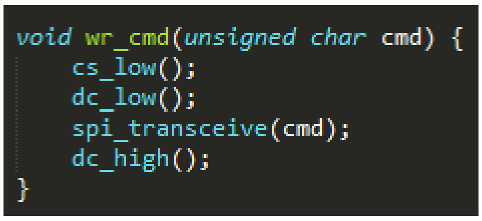
To send data using the hardware SPI module, the data is put into the SPI data register. When new data is put into the SPI data register, the hardware module automatically starts transmitting. In Fig. 3.7, a simple function for transceiving data using the hardware SPI is shown. If there was any data sent back to the microcontroller it is stored in the SPI data register after completing the transmission.



```c
unsigned char spi_transceive (unsigned char data) {
    // Put data in data register
    SPDR = data;

    // Wait until transmission is complete
    while(!(SPSR & (1<<SPIF)));


    // Return received data
    return(SPDR);
}
```

Fig. 3.7: Simple function for transceiving data using the hardware SPI module

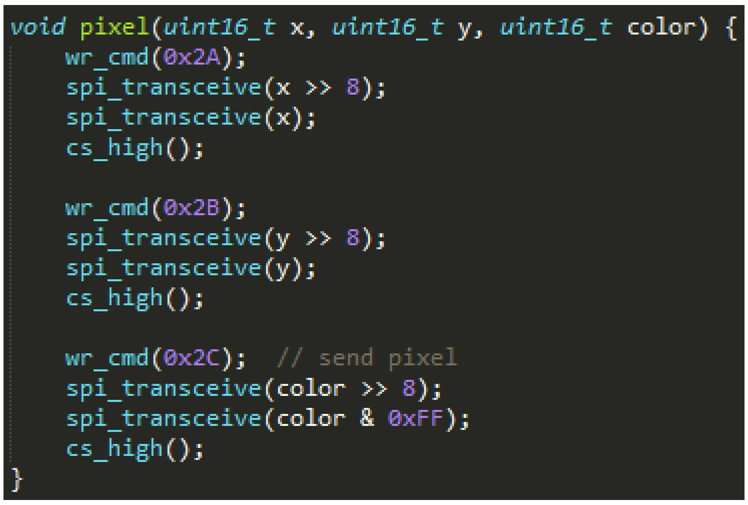
To control the LCD panel of the touch screen, command codes are sent along with data. The screen also has two additional pins connected that are used to signal when there is a command signal being sent, a data signal being sent or that the transmission is complete. In Fig. 3.8, code for sending a command code to the screen using SPI is shown. The *cs* and *dc* pins are the signaling pins previously mentioned.



```c
void wr_cmd(unsigned char cmd) {
    cs_low();
    dc_low();
    spi_transceive(cmd);
    dc_high();
}
```

Fig. 3.8: Function for sending command codes to the touch screen

To alter a single pixel the screen is sent an x-coordinate, a y-coordinate and a 16-bit color value. A function for changing the color of a single pixel is shown in 3.9.

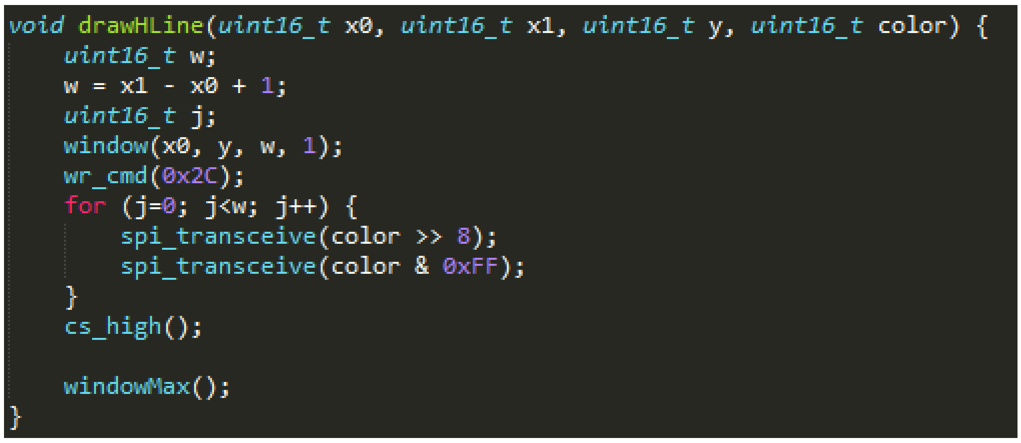


```c
void pixel(uint16_t x, uint16_t y, uint16_t color) {
    wr_cmd(0x2A);
    spi_transceive(x >> 8);
    spi_transceive(x);
    cs_high();

    wr_cmd(0x2B);
    spi_transceive(y >> 8);
    spi_transceive(y);
    cs_high();

    wr_cmd(0x2C);  // send pixel
    spi_transceive(color >> 8);
    spi_transceive(color & 0xFF);
    cs_high();
}
```

Fig. 3.9: Function for altering a single pixel

Instead of sending every pixel using x- and y-coordinates, the screen also supports defining windows. A window has x- and y-coordinate as well as a width and a height. After defining a window, pixel color values can be sent in a stream. The screen will then fill the window row by row, wrapping to the next row when the width is reached.

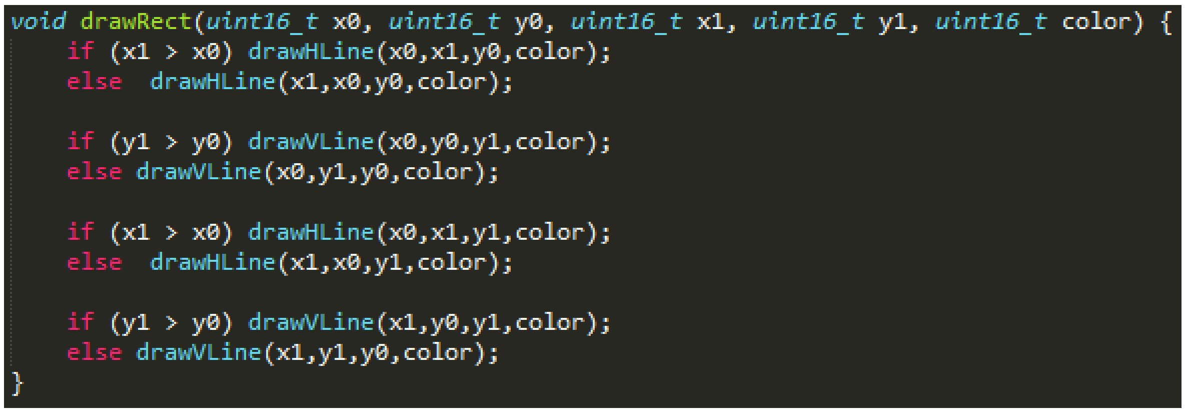
Using the pixel and window primitives, additional graphical functionality was implemented. An example of one such function is the $drawHLine$ function which draws a horizontal line on the screen. The code for $drawHLine$ is shown in Fig. 3.10. Similarly, there is also a $drawVLine$ for drawing vertical lines on the screen.



```
void drawHLine(uint16_t x0, uint16_t x1, uint16_t y, uint16_t color) {
    uint16_t w;
    w = x1 - x0 + 1;
    uint16_t j;
    window(x0, y, w, 1);
    wr_cmd(0x2C);
    for (j=0; j<w; j++) {
        spi_transceive(color >> 8);
        spi_transceive(color & 0xFF);
    }
    cs_high();

    windowMax();
}
```

Fig. 3.10: Function for drawing a horizontal line on the screen

Iterating the process and using these line drawing functions, further more advanced drawing functionality was implemented. An example that illustrates this is the $drawRect$ function, which draws a non-filled rectangle on the screen. The code for $drawRect$ is shown in Fig. 3.11.



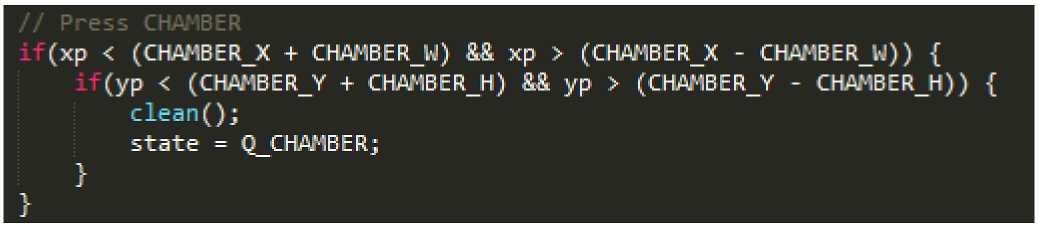
```
void drawRect(uint16_t x0, uint16_t y0, uint16_t x1, uint16_t y1, uint16_t color) {
    if (x1 > x0) drawHLine(x0,x1,y0,color);
    else  drawHLine(x1,x0,y0,color);

    if (y1 > y0) drawVLine(x0,y0,y1,color);
    else drawVLine(x0,y1,y0,color);

    if (x1 > x0) drawHLine(x0,x1,y1,color);
    else  drawHLine(x1,x0,y1,color);

    if (y1 > y0) drawVLine(x1,y0,y1,color);
    else drawVLine(x1,y1,y0,color);
}
```

Fig. 3.11: Code for drawing a non-filled rectangle onto the screen

Additionally, code for drawing text strings and numbers was implemented.

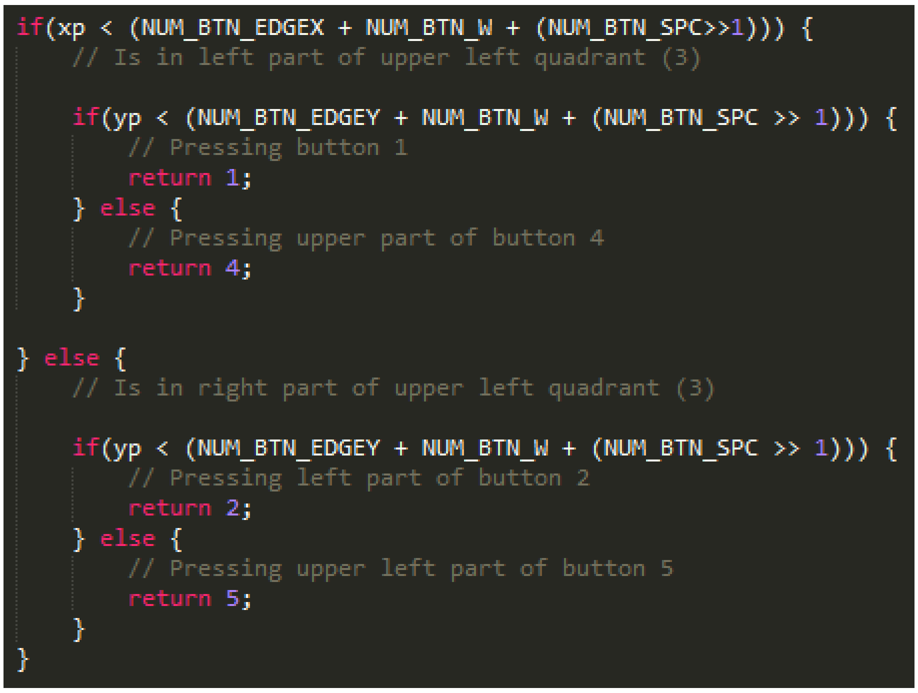
To determine where the user is touching the screen, first some calibrations had to be made. Code was written to write onto the screen what the ADC were registering when a user was touching the screen. Documenting these values along with where the user was pressing on the screen enabled the calibration procedure which essentially consisted of a two dimensional linear system of equations. After the calibration, the coordinates read from a user pressing the screen coincided with the pixel coordinates of that location.

After calibrating the screen and using the previously outlined functionality for drawing onto the screen, a GUI was implemented. To determine if a user is pressing a button on the display, different methods were used. Either a direct if-statement check whether or not the pressed coordinates corresponded to a button, shown in Fig. 3.12, or a binary search, shown in Fig. 3.13, was used. The binary search is on average more efficient, but the direct check is more flexible when more features might want to be added on later.



```
// Press CHAMBER
if(xp < (CHAMBER_X + CHAMBER_W) && xp > (CHAMBER_X - CHAMBER_W)) {
    if(yp < (CHAMBER_Y + CHAMBER_H) && yp > (CHAMBER_Y - CHAMBER_H)) {
        clean();
        state = Q_CHAMBER;
    }
}
```

Fig. 3.12: Code sample of a direct check of whether the x- (xp) and y-coordinates (yp) are inside a button

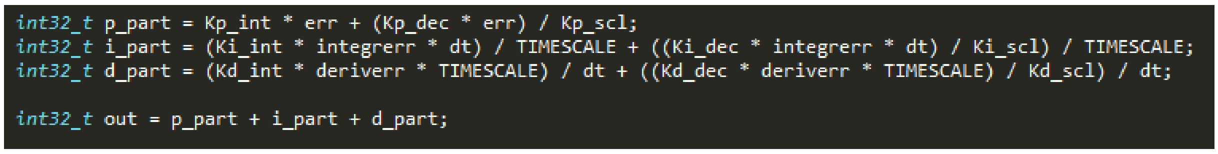


```
if(xp < (NUM_BTN_EDGEX + NUM_BTN_W + (NUM_BTN_SPC>>1))) {
    // Is in left part of upper left quadrant (3)

    if(yp < (NUM_BTN_EDGEY + NUM_BTN_W + (NUM_BTN_SPC >> 1))) {
        // Pressing button 1
        return 1;
    } else {
        // Pressing upper part of button 4
        return 4;
    }

} else {
    // Is in right part of upper left quadrant (3)

    if(yp < (NUM_BTN_EDGEY + NUM_BTN_W + (NUM_BTN_SPC >> 1))) {
        // Pressing left part of button 2
        return 2;
    } else {
        // Pressing upper left part of button 5
        return 5;
    }
}
```

Fig. 3.13: Code sample of a binary search of where the user is pressing

As is hinted in Fig. 3.12 the menu system was implemented using states. When certain buttons were pressed on the screen, the state would change and the screen would be redrawn. To save bandwidth, instead of redrawing the entire screen, only the active (non-background color) pixels of the specific menu that was just visited were deactivated (redrawn as background color). This allowed for much faster transitions between menus, compared to doing a full screen redraw between transitions.

PID-control is most often implemented allowing decimal values for the feedback gains. The ATMega microcontrollers do not come equipped with a float-point hardware unit, it was therefore chosen to implement fixed point decimal handling for the PID-control. The PID-feedback code is very similar to Eqs. (2.2) and (2.3) in section 2.3 and is shown in Fig. 3.14.



```
int32_t p_part = Kp_int * err + (Kp_dec * err) / Kp_scl;
int32_t i_part = (Ki_int * integrerr * dt) / TIMESCALE + ((Ki_dec * integrerr * dt) / Ki_scl) / TIMESCALE;
int32_t d_part = (Kd_int * deriverr * TIMESCALE) / dt + ((Kd_dec * deriverr * TIMESCALE) / Kd_scl) / dt;

int32_t out = p_part + i_part + d_part;
```

Fig. 3.14: Implementation of PID-feedback

The dt in the code in Fig. 3.14 corresponds to the Δt. In this code snippet it also becomes apparent how the fixed point decimals are handled. The $TIMESCALE$ variable was used for calibration and testing purposes. To determine dt, a hardware timer was used to keep track of the time taken between calculations of the out variable.

Functions for reading a sensor value, averaging a number of sensor values before performing the control on it as well as functions to output a control signal using pulse width modulation are also implemented.

It should be noted that this type of touch screen is commonly used together with Arduino systems and sample code for interacting with the screen could be found written in C++. The C++ code was used as inspiration when writing the same functionality in C, with some adjustments where appropriate.(adafruit)

The code in its entirety is attached as separate files.

4. Results and discussion

The aim of the project was to design a control system that is able to interface with industrial grade electronics. Using mechanical relays and optocouplers, the interface to industrial grade electronics is achieved without exposing the microcontroller to high voltage levels.

The system uses a touch screen that a user can use to interface with the system. This also enables the user to customize the PID-controller gains, in accordance with the project specification.

Throughout the project, various problems have been encountered. Debugging using an oscilloscope to find out determine whether some unexpected behavior is a result of hardware or software has been very useful. Hardware problems have usually taken the form of something being incorrectly connected. Software problems have usually taken the form of either typos not caught by the compiler or flawed program logic.

Perhaps the most difficult to debug problem was related to the fuse settings of the ATMega644. By default, some of the pins on the ATMega644 are reserved for a JTAG interface that enables programming of the device, which overrides their regular GPIO functionality. Fortunately, Uwe had already encountered this behavior and the problem was easily solved by changing the fuse settings.

A problem encountered with the PCB design was an incorrect connection of an inductor in the power supply. The error was discovered when mounting components on the PCB and due to time constraints it was not an option to redesign the PCB and order new ones. The problem was solved by cutting into the PCB, severing the incorrect connections, and adding insulated wires to connect the correct pins.

A motor with a drive-shaft mounted potentiometer was used to test the PID-control. The control was applied to the motor position. Using the GUI to select decent feedback gains enabled the motor position to be controlled successfully.

Comparing the work that has been done to the grading criteria, it can be established that the grade level 5 has been achieved. The project is implemented on a functioning PCB and implements a universal PID-controller with customizable feedback gains using the touch screen GUI.

5. Conclusions

The control system performs according to the specifications set and the project can be considered successful.

My previous experience with microcontrollers was highly useful along previous general experience with coding and specifically coding in C. Even though I have never written C++, my previous programming experience allowed me to understand C++ code written for the Arduino which made coding the touch screen API easier.

For the PCB design, my previous experience with Altium Design was highly useful. Transitioning from Altium Design to KiCad required only minor adjustments and while KiCad is not an as fully fledged electronic CAD environment as Altium Design, it was fully sufficient for this project.

6. References

(n.d.). Retrieved March 16, 2017, from Sparkfun: https://www.sparkfun.com/datasheets/LCD/HOW%20DOES%20IT%20WORK.pdf

adafruit. (n.d.). Retrieved March 17, 2017, from Github: https://github.com/adafruit/Touch-Screen-Library/blob/master/TouchScreen.cpp

Glad, T., & Ljung, L. *Reglerteknik, Grundläggande teori.*