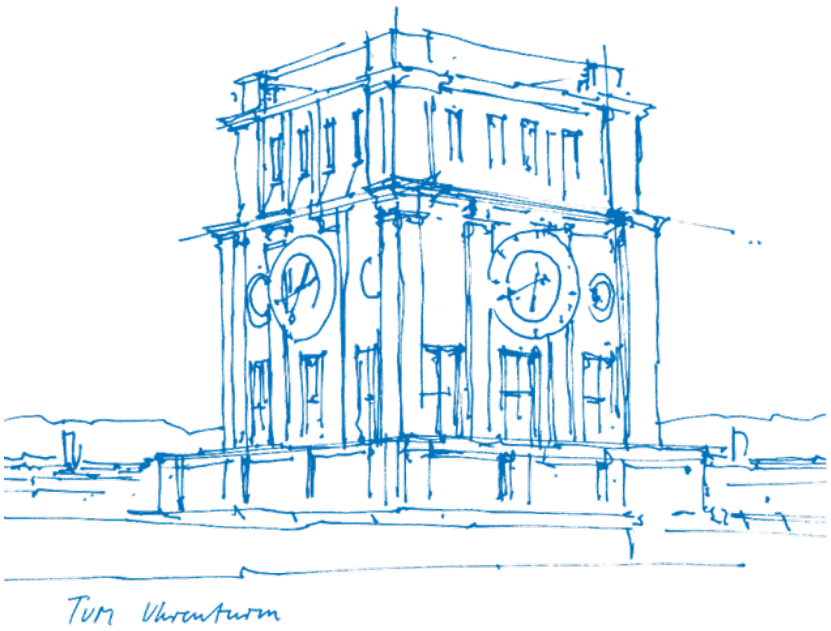
Chair of Robotics, Artificial Intelligence and Real-time Systems
Technical University of Munich

Micromouse Practical Course

Final Report

Summer Semester 2019

Supervised by
Prof. Alexander Lenz
Team Members
Steffen Berhorst
Natalia Paredes
Gyri Reiersen
Adrian Schultz
Submitted
München, 13.09.2019



Contents

1 Introduction

This project is organised as a practical course at the Technical University Munich for the students at the department of informatics. The purpose is for the students to build an electrical micro mouse from scratch and program it to explore a mace and find the fastest way to a goal.

Building the mouse included designing the PCB, selecting all components and soldering them on the board. Initially, we started working with the microcontroller on a bread board to test out and learn the basic programming that was needed and prototype our own micro mouse design. The motors were already given, but it was needed to design mounts and connect them to the microcontroller to later on be able to program the movement of the mouse.

In the weeks where we were waiting for the board and components to arrive we were given a prototype mouse from previous years to start testing our algorithms. This lead to a bit of changes as we were using a different microcontroller, but was very helpful to the debug and further develop the motor control and the mace algorithms.

In the following report we have summarized the technical specifications and the parts of the development of the micro mouse project.

2 Board Design

In order to have a reliable circuit to develop the algorithms of the maze, we designed a board that is mechanically stable and which circuit is efficient and flexible enough to be able to provide additional features in the future. Our first approach was to design a board that would include only the electronics part and that would be assembled to an additional wood board that would hold the wheels and the sensor. Even though this approach provides a smaller board size, the design of an additional board would require the use of unnecessary material and the size that would be saved in the PCB would not be significant.

The final version of our board includes all the components of the maze in only one board. Most of the components were chosen to have an SMD compatible package and the size of the motors, sensors and Bluetooth controller was taken into account to provide a stable and easy to mount circuit. In the following sections, we will explain into detail the designed circuit, the selection of the components and the PCB design of our board.

2.1 Circuit and components

The circuit design was divided into functional sections. Each part of the circuit was positioned in the PCB in a way that the mechanical interface, for example the connection to the motors and sensors, was as close and simple as possible. The full schematics of the final circuit can be found in Appendix A, figure A.1.

2.1.1 Power Control: DC/DC converters

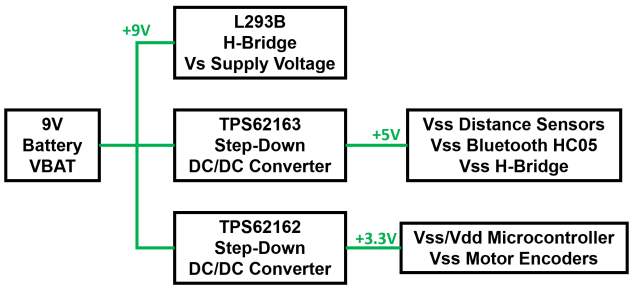


Figure 2.1 Circuit Power Tree

Three power rails are required for the circuit. A 9v rail that powers the H-Bridge drives the motors, the 5v rail powers the distance sensors, the logic supply of the H-Bridge and the bluetooth module and finally a 3.3v rail powers the microcontroller and the motor encoders. The power source for the whole circuit is a 9V battery. In our design we decided to connect the H-Bridge power supply directly to the battery because there are no requirements of voltage ripple for the H-Bridge we are using and because it can be powered by a wide range of voltage (5V - 36V) [1].

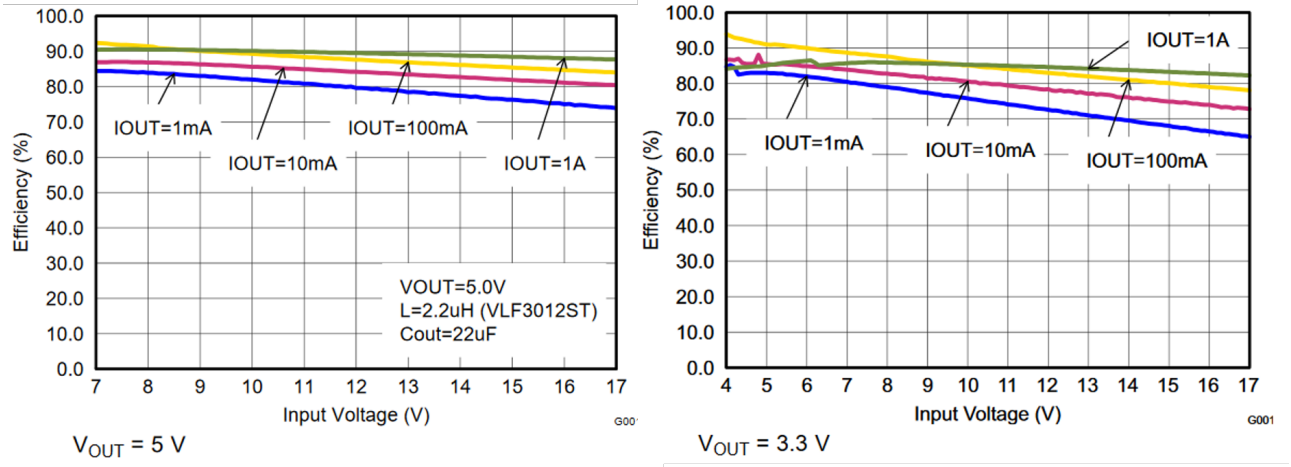


Figure 2.2 Efficiency charts for Vout = 5V and Vout = 3.3V [2]

One of the big challenges for our circuit is the power consumption of our micromouse. The power that a 9V battery can deliver should be used as efficient as possible, for this reason we decided to use the fixed output versions of the TPS62160 DC/DC Step-down converter instead of linear regulators to supply the 5V and 3.3V rails as figure 2.1 shows. The main reason to prefer DC/DC converters over linear regulators is power efficiency. For example if we were to use linear regulators the maximum efficiency we would get, assuming zero losses, is:

$$\text{Efficiency} = \frac{\text{Output Power}}{\text{Input Power}} = \frac{V_{out} \cdot I_{out}}{V_{in} \cdot I_{in}}$$

$$\text{Efficiency}_{5V} = \frac{5}{9} = 50\%$$

$$\text{Efficiency}_{3.3V} = \frac{3.3}{9} = 36.7\%$$

Whereas using DC/DC converters we would have in the worst case for a $V_{in} = 9v$ an efficiency of minimum 82% in the case of an output current of $1A$ (see Figure 2.2). In order to know if we were under that output current, we measured the input current of the circuit when the motors are not running. The input current is of $110mA$, which in the worst case would translate for the 5v and 3.3v rail to $I_{out} = 180mA$ and $I_{out} = 273mA$, respectively. This increment in efficiency directly translates in less power consumption and a longer battery life. Furthermore, the total input current when both motors are operating in their maximum torque is $300mA$, which indicates that the maximum instant power of our micromouse is 2.7 Watts.

However, DC/DC converters have a drawback, their fast switching introduces noise in the output voltage. In order to make sure that this effect is not big enough to affect our system, we measured the voltage output ripple without the motors functioning as seen in Figure 2.3. As we can see, there are 30mV peaks at a frequency of about 1.6kHz which would still be under a 1% ripple of the output, a voltage ripple of less than 1 % is a reasonable amount of noise that would not affect the system.

On the other hand, when the motors are working, there is significant noise added to the voltage supplies. Figure 2.4 shows one of the worst case scenarios for the voltage output in both rails when the motors are working. In this case, this signal is obtained when the motors break while exploring the maze. We can see that the ripple is about 60mV at a frequency of 20kHz. In future

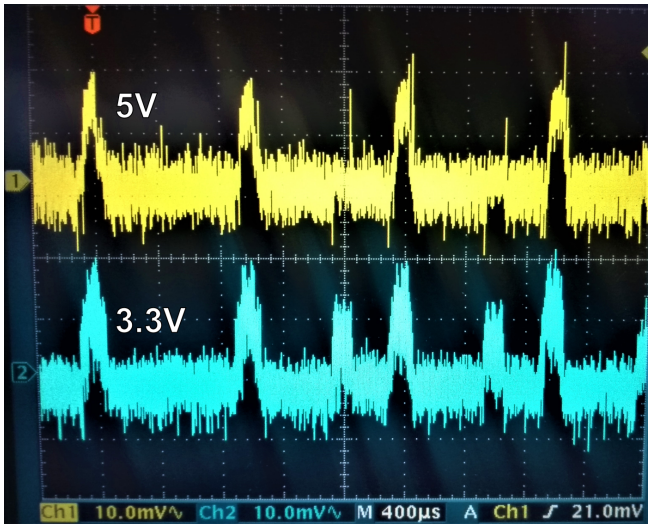


Figure 2.3 Voltage output ripple DC/DC converters without motors

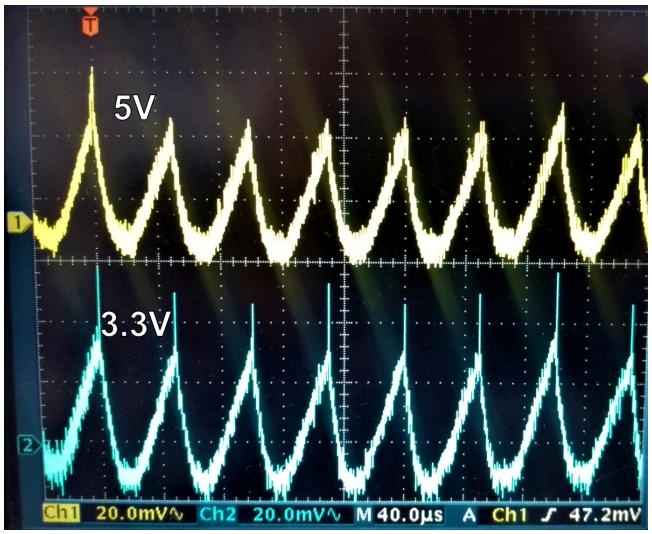


Figure 2.4 Voltage output ripple DC/DC converters when the motors break

design versions of our micromouse it must be taken into account how much does this noise would affect the functioning of the circuit and if an additional filter stage or the isolation of the supply voltage for the motor could be beneficial to the system.

The final design of the circuit for the power stage is taken directly from the recommended circuit of the TPS6216x datasheet [2]. Because the fixed output voltage versions of the converters are being used, there is no need of a feedback resistor divider. Figure 2.5 shows the detailed circuit schematic of the power stage, J1 is the connector of the 9V battery that powers the whole system (VBAT). The pair of pins JP4 and JP5 provide access to the 5v and 3.3v rail output in case measurements or additional powering is needed. If additional resources are going to be powered through these pins, it must always be taken into account that the output current of every converter must not exceed 1A. Table 2.1 shows the component details used in the circuit of figure 2.5.

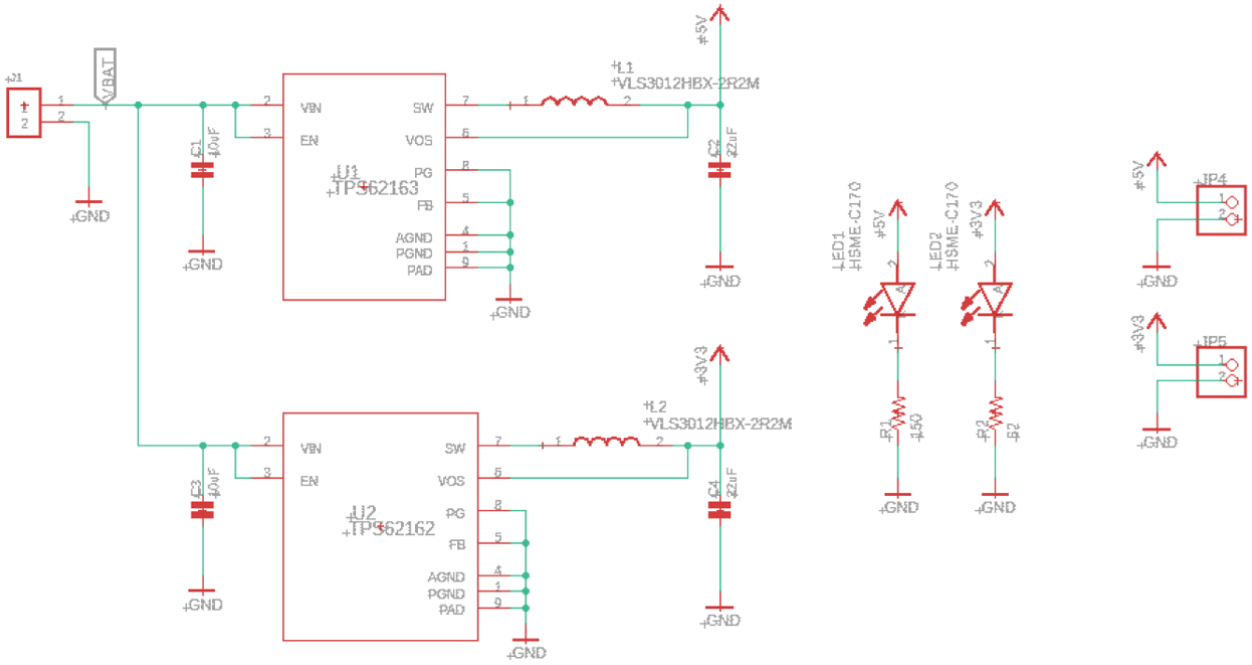


Figure 2.5 Power stage circuit schematics

Reference	Description	Manufacturer
C1, C3	10 μF, 25 V, Ceramic, 0805	Standard
C2, C4	22 μF, 25 V, Ceramic, 0805	Standard
L1, L2	2.2 μH, 1.4 A, 3 mm x 3 mm x 1.2 mm	VLS3012HBX-2R2M, TDK
R1	150 Ω, Chip, 0805, 1/16W, 1%	Standard
R2	62 Ω, Chip, 0805, 1/16W, 1%	Standard
LED1, LED2	LED, SMD, GREEN, HSME-C170	Broadcom
U1	17 V, 1 A Step-Down Converter, WSON, 5V fixed output	TPS62163DSGR, Texas Instruments
U2	17 V, 1 A Step-Down Converter, WSON, 3.3V fixed output	TPS62162DSGR, Texas Instruments

Table 2.1 List of components Power Stage

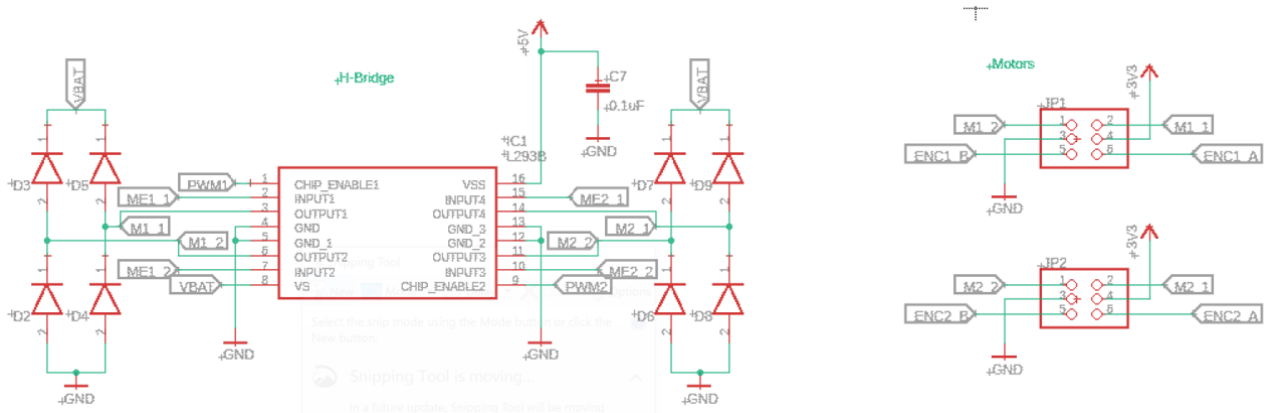


Figure 2.6 Motor drivers circuit schematics

Reference	Description	Manufacturer
IC1	H-Bridge, Push-Pull 4 channel drivers L293B	STMicroelectronics
D2, D3, D4, D5, D6, D7, D8, D9	1N5819HW Diode Schottky 1A 40V SOD123	Diodes Inc.
C7	0.1 μF, 25 V, Ceramic, 0805	Standard
Left and right Motors	DC-Gearmotors Series 2619 006SR	Faulhaber

Table 2.2 List of components Motor Control

2.1.2 Motor Control

Figure 2.6 shows the circuit schematic of the motor control part. The left side shows the connections between the H-Bridge, the microcontroller and the motors; and the right part shows two pairs of pins JP1 and JP2 that are the connection pins where the motors are connected.

We chose the H-Bridge motor controller L293B [1] and the recommended circuit connection that is shown in its datasheet. We chose this H-Bridge because we worked with it since the beginning of the Micromouse course and its specification was easy to integrate to our design. However, as we explained in the last section, the induced noise of the power supply can be caused by the H-Bridge. In future designs it would be important to reconsider the choice of an H-Bridge that can compensate for this noise or a filter that helps its performance.

Table 2.2 shows the list of components used in this stage, the capacitor C7 is a decoupling capacitor for the digital supply of the H-Bridge. JP1 and JP2 are connected according to the datasheet specification of the DC motors [3]. It is important to notice that the H-Bridge has a Through-Hole package so it is recommended to solder a 16-pin IC base to the PCB instead of directly soldering the H-Bridge, in this way changing the IC can be easily done.

2.1.3 Bluetooth driver

In order to communicate with the mouse and be able to see the explored tiles, we used the ready-to-use Bluetooth module HC-05 [4]. This Bluetooth module is configured through UART and therefore only needs an UART line to communicate with the microcontroller and a +5V voltage supply to function, as seen in the circuit of Figure 2.7.

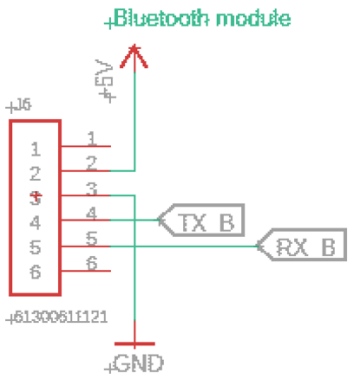


Figure 2.7 Bluetooth module circuit schematics

2.1.4 Distance sensors

For this design we are using the analog distance sensor GP2Y0A51SK0F [5] that has a sense range from 2 to 15 cm which is in the range of the distance we need to sense between our micromouse and the walls of the maze.

Figure 2.8 shows the connectors J3, J4 and J5 for the three sensors that were used in our design where each is positioned in the front, left and right part of the mouse. The sensor output is an analog signal between 0V and 2.5V that is connected with the ADC interface of the microcontroller.

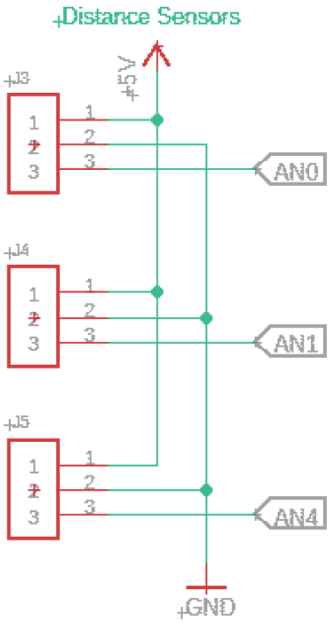


Figure 2.8 Distance sensors circuit schematics

2.1.5 Microcontroller

The central unit of control of our micromouse is the microcontroller, therefore its circuit design had to be done very carefully and taking into account the functionalities of our system and the future possible problems that we could encounter with our first prototype.

The microcontroller we used was the dsPIC33FJ64MC 804 by Microchip [6] with its TQFP package version. Figure 2.9 shows the IC of our microcontroller, the capacitors C8 - C11 are decoupling capacitors necessary for the correct functioning of the circuit according to its datasheet.

Figure 2.10 shows the clocking part of the microcontroller, we are using a 16MHz crystal that needs two ceramic capacitors of about 20pF as shows in its schematic. In order to program our microcontroller, we are using the PICkit 3 programmer [7]. Figure 2.11 shows the recommended circuit according to the programmer datasheet, the set of pins J2 is where the programmer is connected and S1 is a switch implemented to work as a hard reset for the whole system.

Figure 2.9 Microcontroller IC circuit schematics

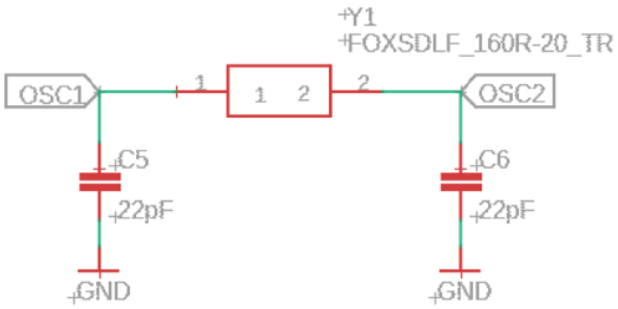


Figure 2.10 Microcontroller Clock circuit schematics

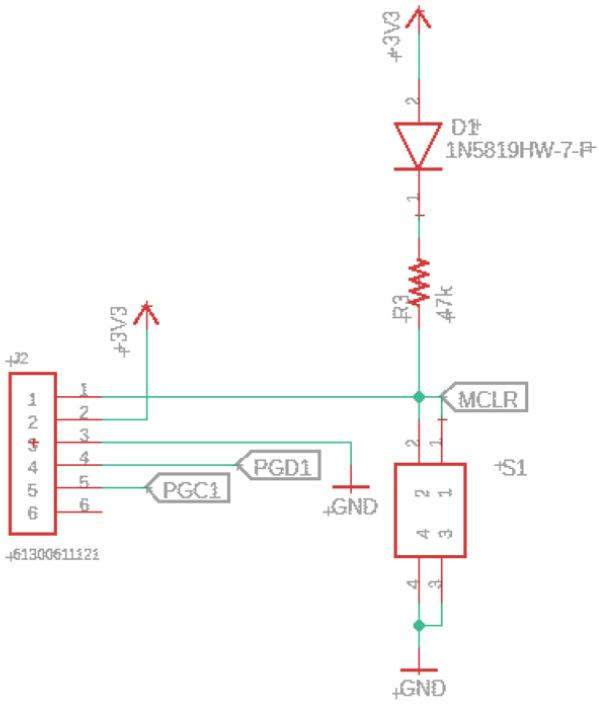


Figure 2.11 Microcontroller Programmer circuit schematics

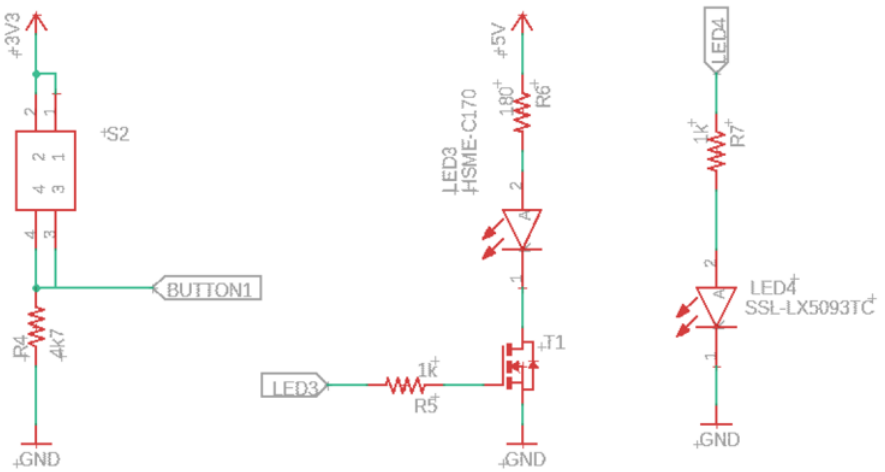


Figure 2.12 Input/Output accesible Microcontroller pins circuit schematics

After making the necessary connections for our circuit, we made two additional output and one input pins for debugging purposes. Figure 2.12 shows the circuit of this pins. S2 is a programmable input digital signal and LED3 and LED4 are a programmable output digital signals. The main difference between LED3 and LED4 is that LED3 is an SMD LED driven by a Mosfet T1 because its current needed its higher than the one the microcontroller can output. On the other hand, LED4 is a low current 5mm through-hole LED that serves as the front base wheel of the mouse.

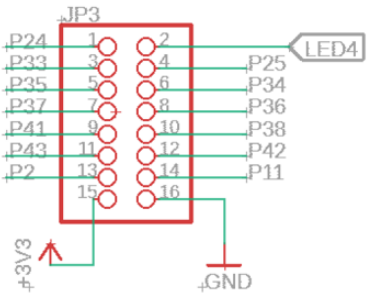


Figure 2.13 Additional pins of Microcontroller circuit schematics

Finally, one of the last features of our system is the possibility to have access to the pins that were not used as seen in figure 2.13. JP3 provides the connection to 13 PINS in the microcontroller that were left and also provides access to the LED4 pin, ground and the +3.3V power rail. The final list of the components used in this stage can be found in table 2.3.

Reference	Description	Manufacturer
IC2	dsPIC33FJ64MC804-H/PT Digital Signal Processors & Controllers 16 bit	Microchip
C9, C10, C11	0.1 μF, 25 V, Ceramic, 0805	Standard
C8	10 μF, 25 V, Tantalum, ESR = 1800 mOhms	Standard
C5, C6	22 pF, 25 V, Ceramic, 0805	Standard
Y1	Crystals 16MHz 20pF, FOXSDLF/160R-20/TR	Fox
D1	1N5819HW Diode Schottky 1A 40V SOD123	Diodes Inc.
S1, S2	Switches 50mA 12VDC	Standard
R4	4.7 kΩ, Chip, 0805, 1/16W, 1%	Standard
R5, R7	1 kΩ, Chip, 0805, 1/16W, 1%	Standard
R6	180 Ω, Chip, 0805, 1/16W, 1%	Standard
LED3	LED, SMD, Green, HSME-C170	Broadcom
LED4	LED, 5mm Through Hole Package	Standard
T1	NMOS, SOT-23, FDN359BN	ON Semiconductor

Table 2.3 List of components Microcontroller section

2.2 PCB design

Figure 2.14 shows the 2-layer PCB that was designed in the EAGLE software, every part of the circuit schematic explained in the last section was positioned so that the Motors and Sensors would have easy access to its connectors. The board circuit after soldering is shown in figure 2.15.

The board was designed with enough space for the bluetooth module, however the pins that connect to the programmer (J2) do not have enough space to directly connect the programmer and additional cables must be used. This does not represent a significant drawback in the design because the programmer can be disconnected once the program is downloaded to the microcontroller, however, in future designs it can be beneficial to include additional space for the programmer.

The motor mounts of the PCB were 3D printed to use as less material as possible and still provide a robust base for the motors. Figure 2.16 shows the motor mounts after printing them (left) and the positioning in the PCB (right). Finally, it is important to notice that the diameter of the wheels must be taken into account in order to have wheels that can lift the circuit to an admissible height and that at the same time they do not interfere with the positioning of the distance sensors.

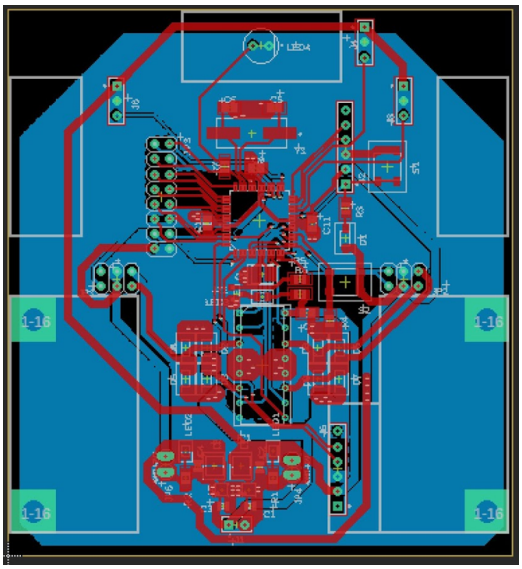


Figure 2.14 PCB circuit design of the micromouse

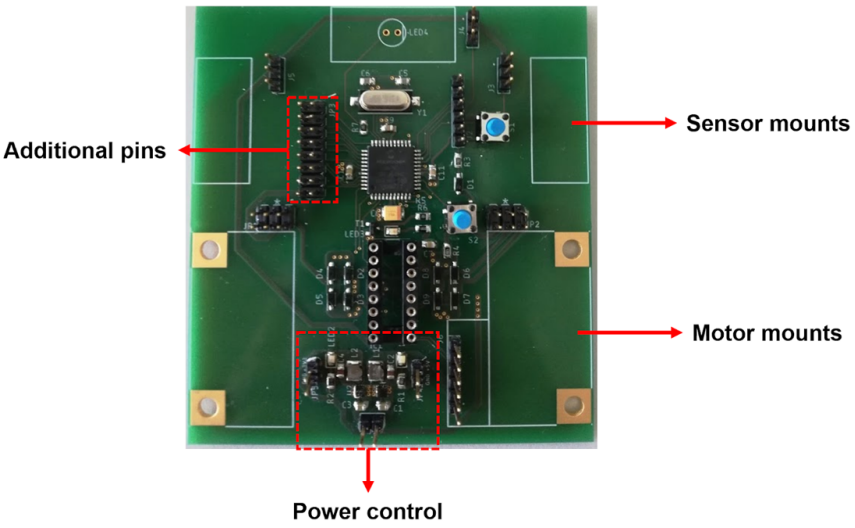


Figure 2.15 PCB sections

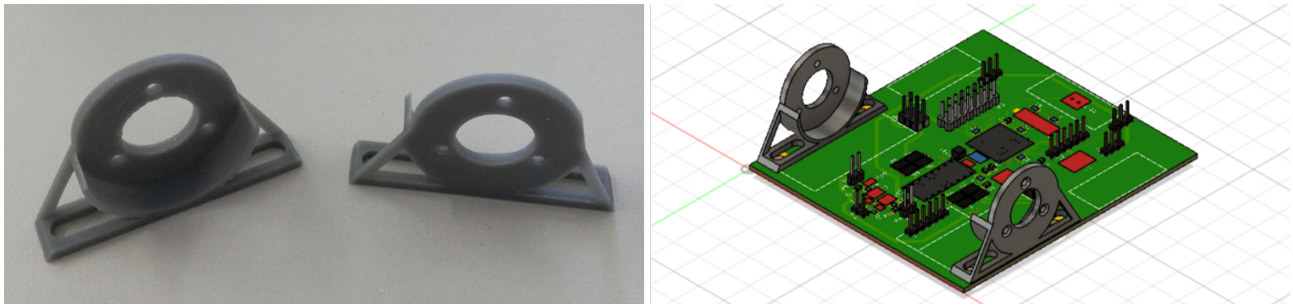


Figure 2.16 Motor mounts for the PCB

3 Communication

Section 2.1.3 explained the circuit used for the HC-05 module [4]. The HC-05 module transmits and receives through Bluetooth the data sent and received to it via UART. For this reason it was necessary to implement an UART module for our microcontroller. In addition to this, we developed a Java application that visualizes the data received from the micromouse, in this way we can visualize which tiles the micromouse has explored and what is the most efficient path that was calculated.

3.1 UART

The files that control the UART module are uart.c and uart.h. These files were written based on the UART manual provided by Microchip to configure the UART module in its microcontrollers [8]. Special care must be taken into account when selecting the baudrate in uart.h, this baudrate needs to match the baudrate of the HC-05, the datasheet of the bluetooth module [4] provides the instructions to configure its baudrate.

3.1.1 Sending

The essential part of th UART is to send data to anther device. This especially helps to understand more complex behaviors as the concrete values for certain distances returned from the sensors or the misbehavior of the encoders, which sometimes occurred, as well as just enables a external visualization of internal data.
This can be done in three ways. The easiest and most basic approach is to just send char by char to the register U1TXREG of the micro-controller, which is inconvenient. A better way is to implement a method to send Strings. This can be done by repeatedly send a character to the same register as before, but with checking the UTXBF register, which tells if the sending buffer is full, this can be done automatically in a loop. The third and most convenient way is to just use the printf method provided by C, which writes to the standard output and as this is configured to the UART it works like just prompting an output on a normal console.

3.1.2 Receiving

Receiving is done in a similar manner. The data has to be read from U1RXREG in the corresponding interrupt method. It is important to write that buffer explicitly to a variable otherwise it won't be freed.
The receiving of data was very important in the beginning of the driving phase to easily change speeds, stop the motors or just test, whether we can drive completely straight, move for 90 degrees etc or if we have to improve slightly.

3.2 Java Application

For visualization purposes, we started developing a small Java application, which is able to either connect to the HC-05 Bluetooth module automatically or any other Bluetooth device with serial communication manually. This is implemented with the use of bluecove, which provides a good set of tools to establish bluetooth connections using Java.

After connecting to the Bluetooth module, the position of the robot in the maze and its direction its facing are visualized. All other messages send from the robot are prompted to the text-area on the bottom of the application.

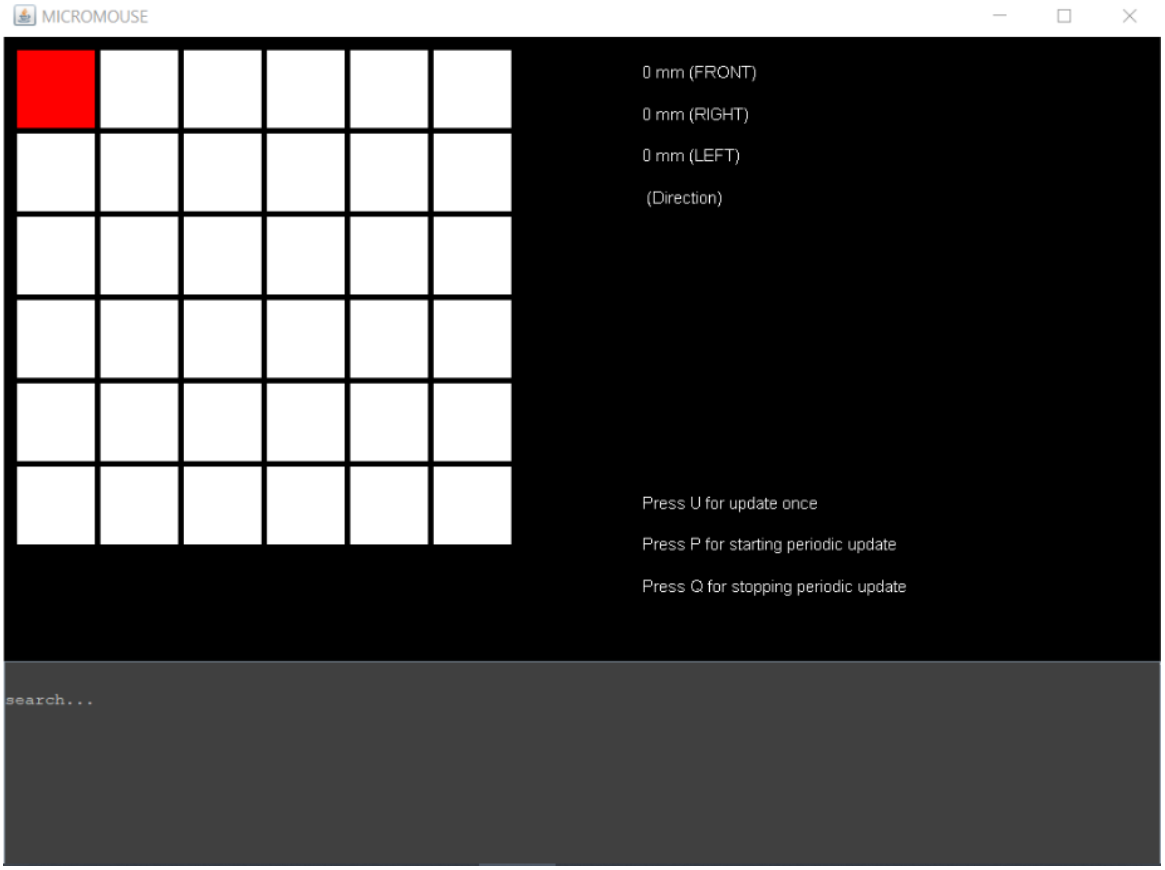


Figure 3.1 GUI of the Java Application

4 QEI - Quadratic Encoder Interface

4.1 The QEI module

The encoders are suitable for monitoring and regulating the speed, position and direction of the rotation of the drive shaft. The DC-motors have an optical encoder with two output channels, A and B. A code wheel on the shafts is optically captured and further processed. At the encoder outputs two 90deg phase shifted rectangular signals are available with 16 impulses per motor revolution.

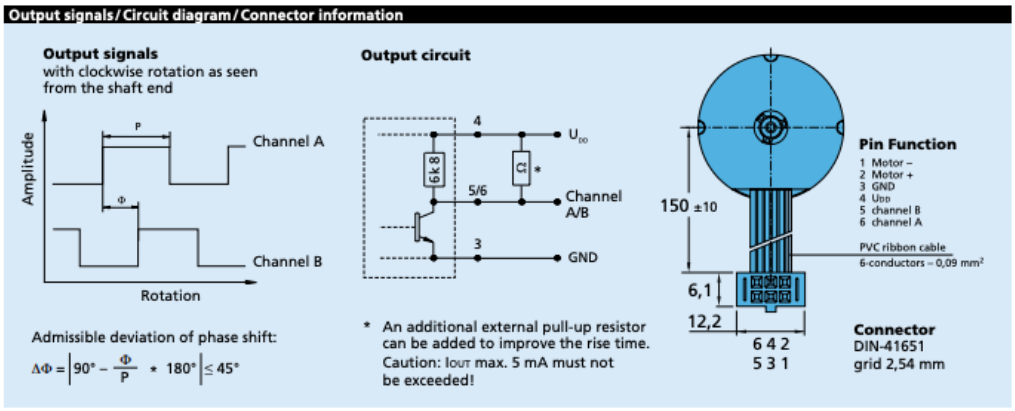


Figure 4.1 A description of the qei module of the microcontroller

It has two channels at pin 5 (channel A) and 6 (channel B) on the motor connector. The encoders also needs a ground in pin 3 and Udd reference of 3.3V in pin 4 in order to work. The 3.3V is to make the digital signal readable for the micro controller.

Our micro controller *dsPIC33FJ128MC804* has a flexible trigger for ADC conversions and configurations of the PWM for motor control. The micro controller does most of the work for determining the directions, whether channel A or B has its edge first. There is a 16 lines and impulses per revolution, which means that the controller counts 16x4 (two edges for two channels) for one revolution. It is an option to add noise filters to the encoders, but we decided not to as the accuracy doesn't need to be very high.

4.2 Mapping of the pins

In the micro controller data sheet [9] the two channels are called *ENC1A* and *ENC1B* for the two channels and respectively *ENC1A/B* for the two motors.

The encoder outputs are directly connected with the micro controller through this schematic:

- ENC1A = pin 26 - RP17
- ENC1B = pin 27 - RP18

- ENC2A = pin 4 - RP24
- ENC2B = pin 5 - RP25

There are many pins and references to go through to make sure that the correct pins are identified and mapped correctly. This is the process to find the QEI pins:

1. checked the Family Reference Manual (FRM) section on peripherals
2. defined the configuration bits QE(A/B)(1/2) to the registers RPINR14 and 16, for R<4:0>
3. set the pins to digital mode (by default) in GPIO
4. making sure that the oscillator is working as it should
5. decided to map the pins with the available pins of the mc ex. RPINR14bits.QEB1R = 00010 (channel B of motor 1 is set to pin RP2 aka pin 6 on the microchip)

4.3 Initialization and interrupts

Depending on the operating mode, the QEI module generates interrupts for the following events:

- In Reset On Match mode (QEIM<2:0> = 111 and 101), an interrupt occurs on position counter rollover/underflow.
- In Reset On Index mode (QEIM<2:0> = 110 and 100), an interrupt occurs on detection of index pulse and optionally when the CNTERR bit (QEIxCON<15>) is set
- When operating as a timer/counter (QEIM<2:0> = 001), an interrupt occurs on a period match event or a timer-gate falling-edge event when TQGATE = 1

The interrupt priority level bits (QEIxIP<2:0>) was set to 6. As the encoders are necessary for the mouse to function properly. We decided to go for a Match mode 111 where the interrupt occurs on a position counter overflow or underflow. This is defined in the QEI interrupt initialization function.

4.4 Calculating the position and speed

The encoders allows the actual velocity of the shaft (speed and direction of rotation) to be compared to the desired velocity and then used in a controller.

Facts about the encoders: 16 pulses per motor rotations x 2 channels + 2 ups/downs (positive and negative edge) x 33motor rounds per wheel turns maxcount is 65.0000. This gives a high resolution, and is needed to be able to drive slowly. To get the most out of the counter position count variable needs to be a long.

For developing the position and speed calculation we followed this example [10]. Instead of calculating the angular position, we converted it into a linear position by taking the diameter of the wheels into account. This was only possible as we know which wheels we were using, but is a risk when the wheels might be changed. As one motor is opposite the other, we also had to make sure that the signs for forward and backward was correct.

To increase the speed of the calculation and to reduce the risk of errors, we introduced macros such as *GETENCODERVALUE1* in both functions and defined them in the header file.

5 Motors and Control

5.1 PWM

5.1.1 Initialization

The employed microcontroller features 4 separate duty cycle generators contained in two PWM modules. Each duty cycle generator feeds into a high (active low) and a low (active high) channel. In all cases the low channels were chosen, as an active high channel behaviour was considered to be more intuitive. The corresponding high channels were set to be usable as standard I/O pins as they were not needed. In the following, lowercase "x" indicates the number of the PWM module and lowercase "y" stands for the number of the duty cycle generator of that module.

PWMxCON1bits.PENyH = 0; // PWMxHy (high) of PWM module x is controlled by GPIO module
PWMxCON1bits.PENyL = 1; // PWMxLy (low) of PWM module x is controlled by PWM module

The PWM modules were set to independent mode because no inverter loads were driven in the setup [11].

PWMxCON1bits.PMODy = 1; // Define independent mode for PWM output pair y of module x

All duty cycle generators were run in free-running mode, leading to an edge-aligned PWM output mode (described in the following section).

PxTCONbits.PTMOD = 0; // Define free running mode for PWM module x

Pre- and postscaler of the PWM counter were left at the default 1:1 ratio. Accordingly, the 40MHz microprocessor clock is the PWM counter clock as well. The PWM period value (PTPER) was set to 2000, leading to a duty cycle period length of 50μs.

PxTPERbits.PTPER = 2000; // Setting the period value of PWM module x to 2000

5.1.2 Using the PWM

The PWM module increments its counter register by 1 every clock cycle, until it reaches its period value (PTPER), which causes a reset to 0.
A dutycycle for a specific output channel (PWMxLy) is defined by the corresponding duty cycle register (PxDCy). At the start of every period, all output channels are set active. The channels remain so until the counter reaches the value stored in the duty cycle register, when the respective channel is switched inactive. The whole procedure is illustrated in figure 5.1.

PxDCy = 1000; // Setting duty cycle register y of module x to 1000.

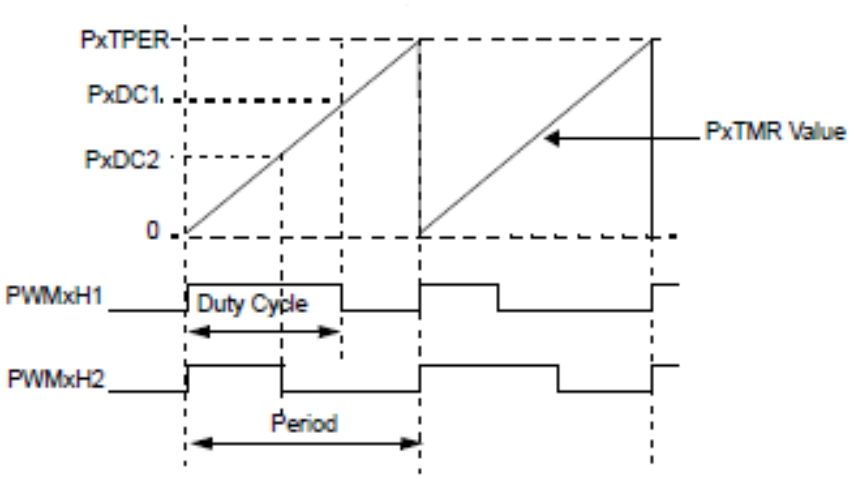


Figure 5.1 Edge-aligned PWM signal generation in free-running mode. PxDC1 and PxDC2 represent two duty cycle register values and PWMxH1/PWMxH2 their corresponding output channels. The top part of the image (above the 0 line) depicts the PWM counter values, counting up until being reset when reaching PxTPER. The bottom part of the image depicts the resulting channel values and duty cycles. Here, the x stands for the number of a PWM module. In this particular example the channels are high channels. An active phase of the dutycycle would therefore output 0s to the channel. Figure adapted from figure 14-6 in [11]

In this case, PWMxLy will be active until the counter reaches 1000, when it becomes inactive. It is reactivated when the counter is reset upon reaching the period value PTPER.

The timer period register is a 15 bit register, while the duty cycle register has 16 bits. The duty cycle register consequently has a half-cycle resolution, allowing a finer duty cycle adjustment. The correct formula to calculate the duty cycle register value needed for 100% dutycycle is:

$$DC = 2 \times (PTPER + 1)$$

The "+ 1" results from the counter register starting its count at 0. For any wanted duty cycle, the number to be written to the duty cycle register is the corresponding faction of the calculated 100% DC-number.

5.1.3 Evaluation

Using the oscilloscope, the resulting PWM signals where checked both in terms of dutycycles and resulting voltage at the motors. The checks were done for desired duty cycles of 25% and 50% with a 9V power source. All measurements resulted in the desired and expected duty cycle and voltage values. The PWM modules showed no unexpected or faulty behaviour.

5.2 Control

5.2.1 Initialization

A default controller was defined as a struct describing a PID controller. Each struct contains several variables: Firstly, there are the variables neccessary for the PID computation itself. An example use for these variables is to store the last error or the integrated past errors. They are vital for the computation, but only describe a basic PID procedure and are thus not further discussed here. Secondly, there are variables that define the PID controller itself:

kp: The multiplier of the proportional portion of the error
ki: The multiplier of the integral portion of the error
kd: The multiplier of the derivative portion of the error
top_lim: An upper bound to the PID controllers output
bot_lim: A lower bound to the PID controllers output
int_lim: An upper bound to the PID controllers integral portion of the error

The upper and lower bounds on the PID controllers outputs were added to ensure sane and safe output values. Especially in the testing phase, it was vital to ensure a programming error could not lead to a motor speed that might harm the robot.
The upper bound on the integral of the PID is neccessary to restrain the integral from growing to extremely high values, for example when the wheels are blocked. In such a case, the boundless growth of the integral portion would lead to the motors running uncontrollably at full speed for some time after the blocking is resolved, as the massive integral portion overpowers any other control mechanism.

The choice to standardize controllers as a struct was made for two main reasons: Firstly, the evaluation and calculation of a PID controller can be standardized and covered by a single function for any and all controllers as well. The second reason is that the creation of additional controllers, should the need arise at some point, is simple and straightforward.
Additionally, the struct allows to use only part of the PID controller without alteration of the surrounding process. Simply setting any of the multipliers to 0 effectively removes them from the calculaton.

5.2.2 Development and result

Figure 5.2 gives an overview of the development process of the control cascade. For educational purposes a description of problems at each step is given, and finally a more detailed description of the final outcome.
First tests were carried out with no position control and a simple P-controller for velocity (figure 5.2 top row). Movement instructions were given as target velocitis with defined durations. This setup showed to be largely insufficient for manoeuvring in the maze. As only velocity was controlled, no positional errors resulting from the blocking of a wheel, however short, or inertia could be corrected. Effectively, the robot was unable to move in a straight line in any except absolutely ideal environments with no disturbances.
Thus, in the second iteration position control was added as an additional P-controller (figure 5.2 middle row). Here, movement instructions were positional targets of each wheel. This setup already proved largely able to carry out basic movement patterns even under disturbance. Still,

it had major weaknesses. Due to the nature of P-controllers, target velocities calculated by the position control were never reached, making the control cascade and the robot itself less effective than it could be. Also, the control relied solely on odometry without including any sensory data, which lead to the uncorrected accumulation of positioning errors. In a maze, it could only operate for few cells at a time before crashing.

Figure 5.2 Schematics of the motor control at 3 different stages of development. The large blue X indicate a section of control being nonexistent in a certain stage. The large rectangles labelled with "P" or "PI" indicate P- and PI-controllers. The small rectangles indicate a limiting element. The line labelled with "d/dx" signifies a first derivative of the values taken. The dotted line in the top left of bottommost diagram signifies input from the sensors. In this particular case, only a general point of influence is illustrated, without depicting mathematical operations for reasons of clarity.

The final control cascade consists of a P-controller for position control and a PI-controller for velocity control. The position controller gets a target in form of a reachable position of the controlled wheel. This target is compared to the wheels position counter provided by the QEI module to calculate the position error. The position controllers configuration is as follows:

kp = 1 [$\frac{1}{t}$]
ki = 0 [$\frac{1}{t}$]
kd = 0 [$\frac{1}{t}$]
top_lim = 30 [$\frac{d}{t}$]
bot_lim = -30 [$\frac{d}{t}$]

Here, t is the regular time passing between two consecutive position measurements (in our case 10ms) and d is the position difference (distance) measured in ticks of the QEI module. The output of the position controller becomes the velocity controllers input (see figure 5.2.
Velocity control is carried out by a PI-controller with following configuration:

kp = 50 [$\frac{t \times DC}{d}$]
ki = 5 [$\frac{t \times DC}{d}$]
kd = 0 [$\frac{t \times DC}{d}$]
top_lim = 2000 [DC]
bot_lim = -2000 [DC]
int_lim = 100 [$\frac{t \times DC}{d}$]

where t and d are defined as before and DC is the duty cycle relative to the PWM period cycle value. The resulting value from this controller is written into the corresponding duty cycle register of the PWM module. The *kp* value was chosen for a specific reason: The maximum input target velocity from position control is 30 $\frac{d}{t}$. Accordingly, the maximum error is 30 as well. Multiplied by the proportional error factor *kp*, this would lead to a PWM duty cycle value of 1500, which was empirically found to roughly correspond to said speed of 30. The chosen *kp* value of 50 thus theoretically allows the system to reach even the maximum target speed in only few iterations.
The velocity controller has an integral portion to ensure that a target velocity is not only approached (as a P-controller would) but actually reached. For the position control, momentum and inertia serve as a kind of integral function by itself, so that position control works with only a proportional portion.

5.2.3 Movement calculation

Exploring the maze boils down to a repetition of basic movement patterns: Going one cell forward, turn right 90 and turn left 90. All of these movements are implemented as functions that calculate position targets for both wheels and feed them into the control cascade. To calculate the values for a specific movement, several measurements were taken and calculated:

wheel diameter $d_w = 60mm$
axis diameter $d_a = 105mm$
encoder signals per wheel turn $s = 4 \times 16 \times 33 = 2112$
wheel circumference $c_w = \pi \times d_w = 188.5mm$

axis circumference $c_a = \pi \times d_a = 329.87mm$

To move forward a distance of x mm, each wheel has turn n signals forward:

$n = \frac{x}{c_w} \times s$

To move forward a cell, all one has to do is measure the length of said cell and insert it into the formula above. An additional function to move any number of cells forward was implemented to avoid stopping in between every cell and thus allow for potentially smoother and faster driving after exploration.

To achieve an y turn, the number of signals each wheel has to count if both wheels turn in opposite directions calculates as:

$n = \frac{c_a}{c_w} \times \frac{y}{360} \times s$

For a 90 turn, this calculates to 924 encoder signals.

5.2.4 Sensor based movement

While the robot moves, movement and odometry errors are accumulating and adding up over time. To account for this and correct the errors, sensor measurements are included into the control cascade to enhance robot positioning.

1. Centering between walls
When the robot moves forward between two walls, it is desirable to correct its position to keep the robot in the center between the walls. For this purpose, the distance measurement of the right sensor is subtracted from the left sensors measurement. The resulting value is multiplied with an empirically determined scaling factor of 0.8 and then added to (right wheel) or subtracted from (left wheel) the velocity controllers target. This leads to the robot turning away from walls more and more the closer it gets to them.

2. Approaching walls
To account for small accumulated errors in the travelled distance and to avoid crashes, the robot elicits a different behaviour when moving forward and sensing a wall in front of it. Instead of just continuing the planned movement, the robot approaches the wall up until 1.5cm distance.

3. Wall following
When moving forward and only one wall on one side is sensed, the robot tries to follow that wall in a distance of roughly 3cm. In a calibration step the sensor measurement corresponding to 3cm was determined to be approximately 1800. This number is subtracted from the actual sensor measurement and multiplied with a scaling factor of 0.2. The result is then subtracted from the velocity target of of the wheel on the side of the wall and added on the opposite side. This results in the robot turning away from the wall for sensor values under 1800 and vice versa.

5.2.5 Evaluation

The controller cascade was tested both for position and velocity control. All measurements concluded to reaching exactly the wanted positions and velocities. No statement on the duration to reach the desired values could be made due to problems with the Bluetooth modules sending behaviour.

The exact number of signals for moving one or several cells forward had to be adjusted several times. The actual trials in the maze were carried out with a battery, whose charge status seemed to have some impact on how much distance a certain number of signals actually covered.

The wall following behaviour often does not work well in the current state. The best results in the maze were achieved when no active wall following was attempted, but the robot just moving forward without accounting for sensor signals when not between walls.

6 Sensors

For our robot, we are using the analog distance sensor GP2Y0A51SK0F by Sharp to sense the walls of the maze. This sensor has a range from 2 to 15cm and senses with 50Hz, which is sufficient for its task. [5]

6.1 Setup

Our setup contains three of those. One is mounted on the left side of the PCB, one on the right and the third in front (see figure 2.15). That structure leads to an optimal coverage of the surroundings, with an minimal amount of hardware used.

6.2 Analog to digital conversion (ADC)

As described in the Chapter Board-design/Distance sensors we have connected the sensors directly to the ADC interface of the micro-controller via analog inputs 0, 1 and 4.
We chose that the result as an unsigned 12bit integer would be best because firstly negative values wouldn't make any sense in this scenario and secondly the ADC is build to work with analog inputs up to 3.3V [6] whereas the used sensors only produce up to 2.5V (Figure 6.1). Therefore 10bit would be to small to properly work on because due to the sensors specification we are only using $2.5V/3.3V = 75.7...\%$ of the possible range.
The ADC interface is also configured to directly pass its results to the direct memory access (DMA) unit of the microcontroller.

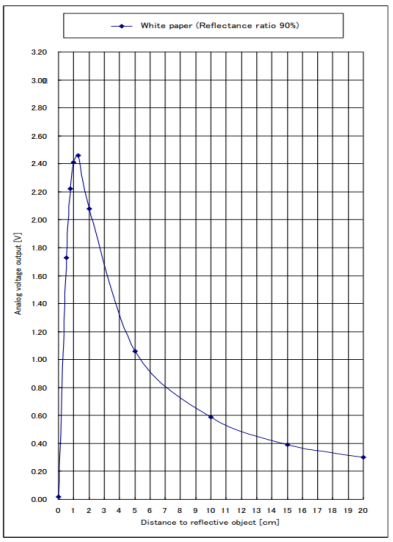


Figure 6.1 Analog sensor output [5]

6.3 Direct memory access (DMA)

The advantage of using the DMA in this setup is huge because the data from the ADC is (after correct initialisation) directly passed to a in the configuration defined register, which can be accessed like normal registers. There is no interrupt to manually handle and the data gets updated automatically.
The initial configuration is done via telling the DMA, which ADC unit it has to communicate with and declaring the start address of the wanted buffer in memory.

6.4 Evaluation

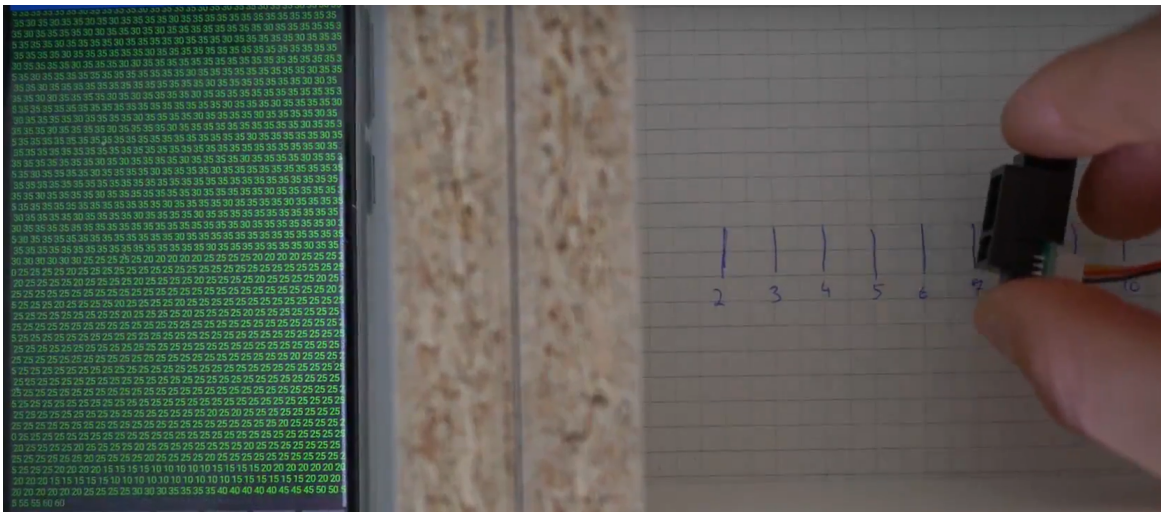


Figure 6.2 Analog sensor output parsed and send to phone

Due to the fact that the sensor values get lower, when the distance increased, we decided to implement an important helper method. This takes the values from the register, which is updated by the DMA and translates those values to distance in millimeters.

7 Exploration and shortest path

The exploration of a given maze and finding the shortest path to a certain cell is the is the most high-level task during the development of this robot. There are several maze sizes, but we decided to only solve six by six maze. Bigger mazes would need either a lot more time to solve or a way faster robot.
The robot has nearly unlimited time to explore the maze to check for walls and figure out all possible paths. So, this doesn't have to be the most optimized part in contrast to the final run to a certain cell, which will be timed. Therefore it is very important to use the shortest path.

7.1 Exploration of the maze

The starting position of the robot is always in the lower left corner and it's facing north. We developed an algorithm for the exploration task, which at first checks for walls in the current cell via the distance sensors and secondly drives to the next cell regarding the following rules:

- drive north, if there is no wall and the cell has not been visited yet
- drive east, if there is no wall and the cell has not been visited yet
- drive south, if there is no wall and the cell has not been visited yet
- drive west, if there is no wall and the cell has not been visited yet
- if non of the upper is possible, go back to the last cell

Following that algorithm step by step (for example see Figure 7.1 to 7.7), it is really easy to explore every given maze, but there is a lot of information to be stored for each cell:

- walls in each direction
- was that cell visited yet
- coordinates of the last visited cell before the current one

As we also want to calculate the shortest path later on, it is important to store the distance of each cell to the starting position.

7.2 Finding and executing shortest path

After the exploration phase is completed, the shortest path has to be calculated and performed. For that task, the distances of each cell to the starting position have to be updated. This is done in two steps again. The first is to check whether there the current cell has a reachable neighbor with a lower distance value or not and the second will update the distance value as well as updating the coordinates of the last visited cell before the current one to the one with the lower distance.

This has to be done for each cell n (dimension of the maze) times to get all cells updated. (see Figure 7.8)
As an advantage with this implementation, we have calculated shortest paths to reach every cell in the maze. After choosing one specific one, the sequence of movements for the robot to perform has to be extracted from the data-structure, which is straight forward due to the stored and updated last visited coordinates. This sequence will now follow the shortest path.(see Figure 7.9)

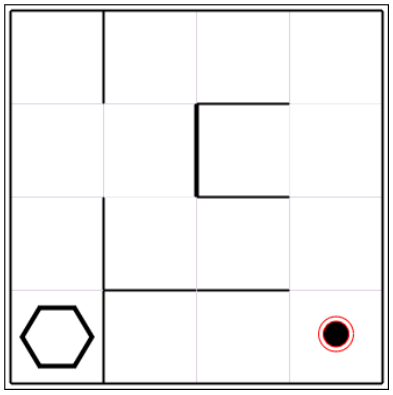


Figure 7.1 Example maze

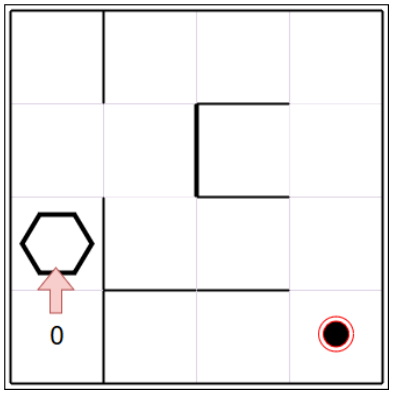


Figure 7.2 Step 1

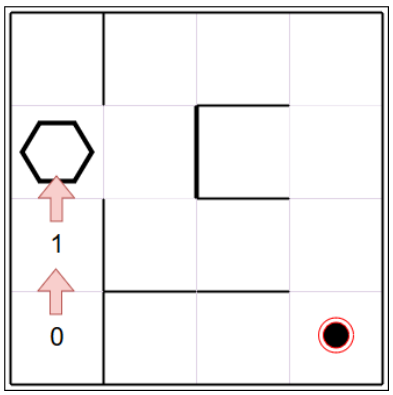


Figure 7.3 Step 2

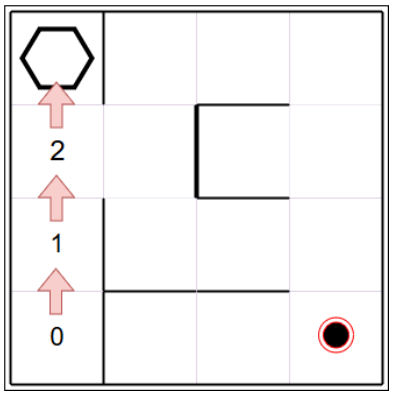


Figure 7.4 Step 3

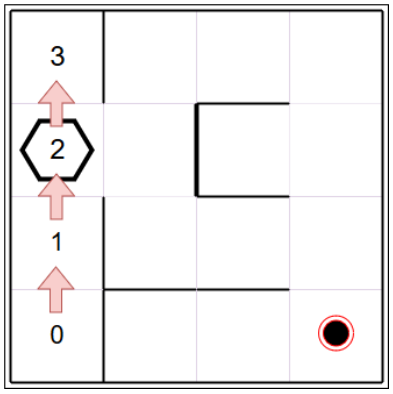


Figure 7.5 Step 4

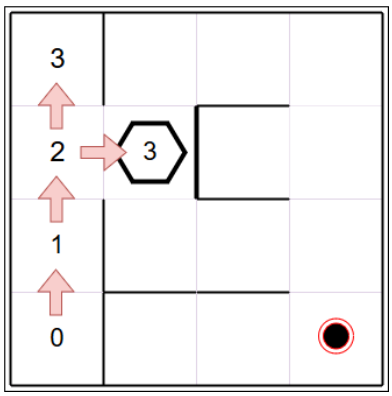


Figure 7.6 Step 5

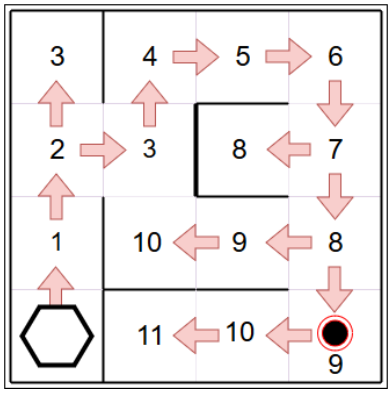


Figure 7.7 Done with exploration

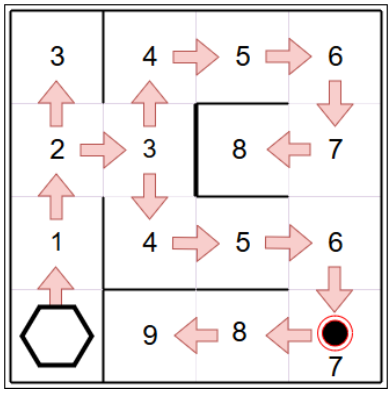


Figure 7.8 shortest Path calculation

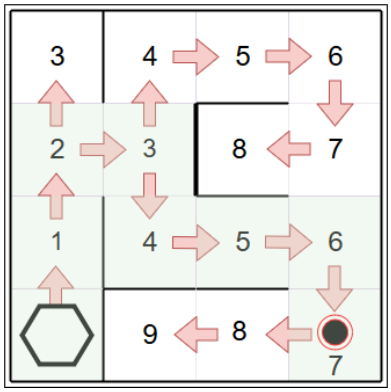


Figure 7.9 shortest Path to a certain cell

8 Easter Eggs and Challenges

8.1 Easter eggs and funny surprises

The project is mostly about learning and playing around building a mouse from scratch. Throughout the development, we used lights as debugging tools and found out that lights also bring life and joy to the project. That is why we also developed blinking lights on the back of the mouse. Even if the mouse is meant to be alone in the mace and in theory don't have the need (like cars) to signal in which direction it is headed, it is a nice feature to have.

To do this we made use of the unused additional pins that we had prepared for further development like this feature. Each LED was soldered with a resistance of 150k to avoid draining the current and covered with thermoplast to fit in and stand less out.

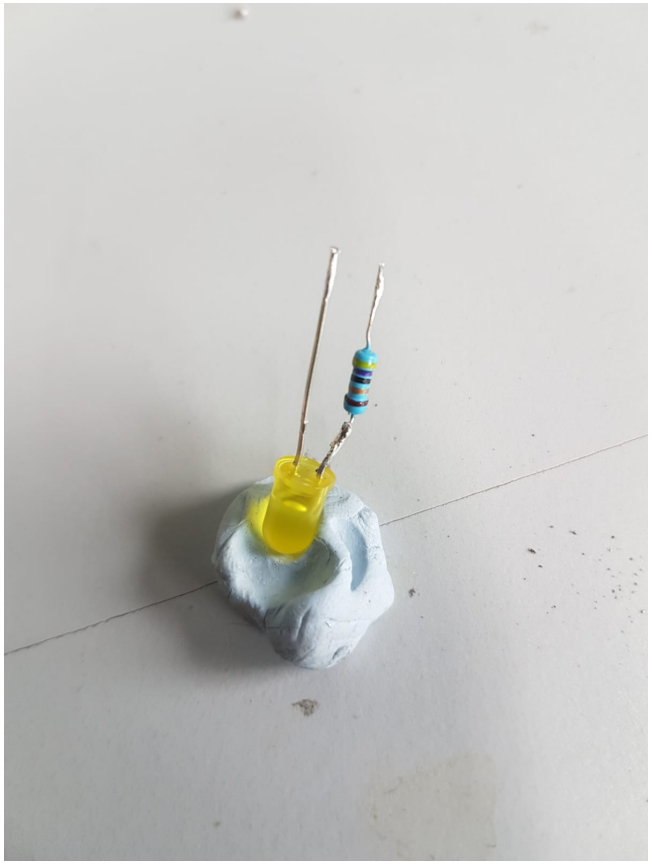


Figure 8.1 LEDs with soldered resistor

As solving the mace was a great success for us, we also wanted to illustrate that in adding a feature of joy in the mouse. When the mouse has arrived at the goal after first having explored the mace, she starts turning around and all lights are blinking. That turned into what we called "goal dancing and jazz hands".

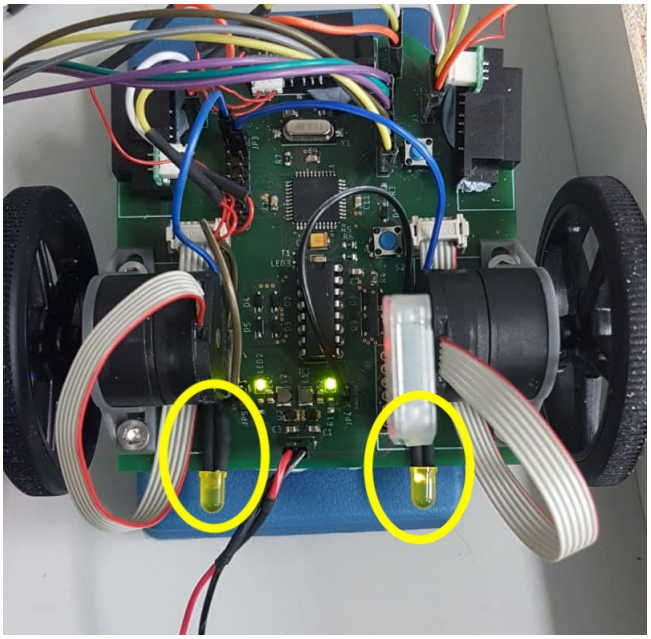


Figure 8.2 Blinking lights

8.2 Challenges and current problems

During the project we faced several challenges and problems. As the main challenge working with hardware is isolating errors and their causes. There are many different elements involved and that might cause the problem. Especially was the UART not sending or receiving, and sometimes not working at all a mystery for us. Although we at several occasions thought that we had isolated the cause, it started acting weird.

Our main concern, and a problem we still haven't solved is the behaviour of the encoders when starting up. When resetting the mouse, the encoders sometimes work, and then the two wheels are controlled (stopped) by the controller. However, this does not happen every time one resets. Sometimes only one wheel is stiff or controlled and the other is spinning with high speed and sometimes both wheels. By pressing the reset button often enough and checking the wheels in between each time, we get get to a reset where both wheels are controlled.

The last challenge and also point of improvement was the placement and position of the motor mounts. The motor could have been placed more in the middle of the board to be able to make the turn better. In addition, the reset button could have been placed somewhere more accessible.

Bibliography

[1] STMicroelectronics. L293b push-pull four channel drivers h-bridge. [Online]. Available: https://www.mouser.de/datasheet/2/389/l293b-954809.pdf

[2] Texas Instruments. Tps6216x3-v to 17-v, 1-a step-down converters with dcs-control. [Online]. Available: http://www.ti.com/lit/ds/symlink/tps62162.pdf

[3] Faulhaber. Dc-gearmotors series 2619 006sr. [Online]. Available: https://www.faulhaber.com/fileadmin/Import/Media/EN_2619_SR_IE2-16_DFF.pdf

[4] Hc-05 - bluetooth module. [Online]. Available: https://components101.com/wireless/hc-05-bluetooth-module

[5] P. R. . Electronics. Sharp gp2y0a51sk0f analog distance sensor 2-15cm. [Online]. Available: https://www.pololu.com/file/0J845/GP2Y0A41SK0F.pdf.pdf

[6] Microchip. dspic33fj64mc804, 16-bit dsc for precision motor control. [Online]. Available: https://www.microchip.com/wwwproducts/en/dsPIC33FJ64MC804

[7] ——. Pickit 3 in-circuit debugger. [Online]. Available: https://www.microchip.com/DevelopmentTools/ProductDetails/PartNO/PG164130

[8] ——. Universal asynchronous receiver transmitter (uart) for dspic. [Online]. Available: http://ww1.microchip.com/downloads/en/devicedoc/70000582e.pdf

[9] Microchip Technology Inc. dspic33f family data sheet. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70165d.pdf

[10] Jorge Zambada, Microchip Technology Inc. Measuring speed and position with the qei module. [Online]. Available: http://ww1.microchip.com/downloads/en/appnotes/93002a.pdf

[11] ——. Motor control pwm family reference manual. [Online]. Available: http://ww1.microchip.com/downloads/en/DeviceDoc/70187E.pdf

A Appendix: Images

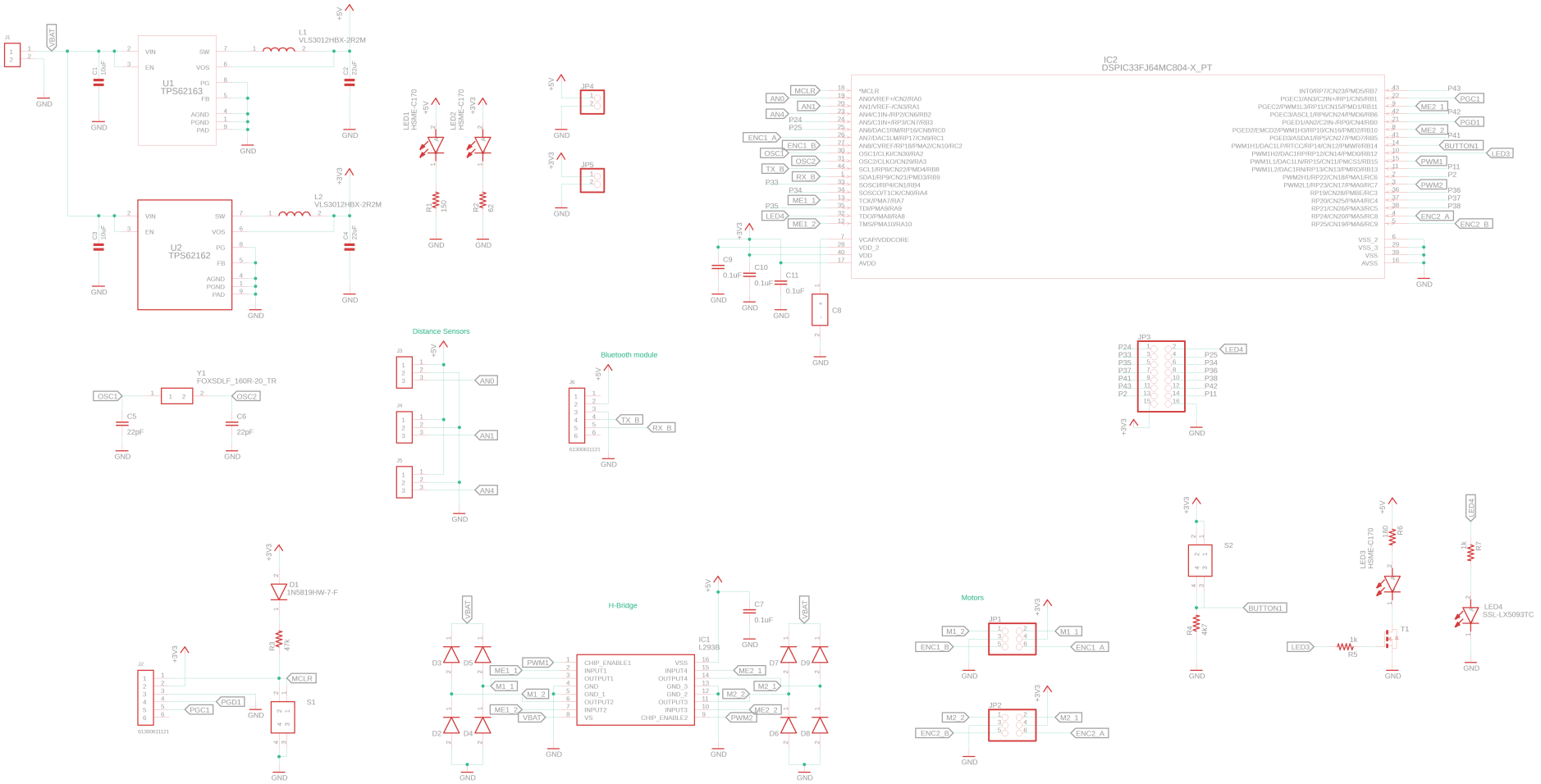


Figure A.1 Circuit Schematics