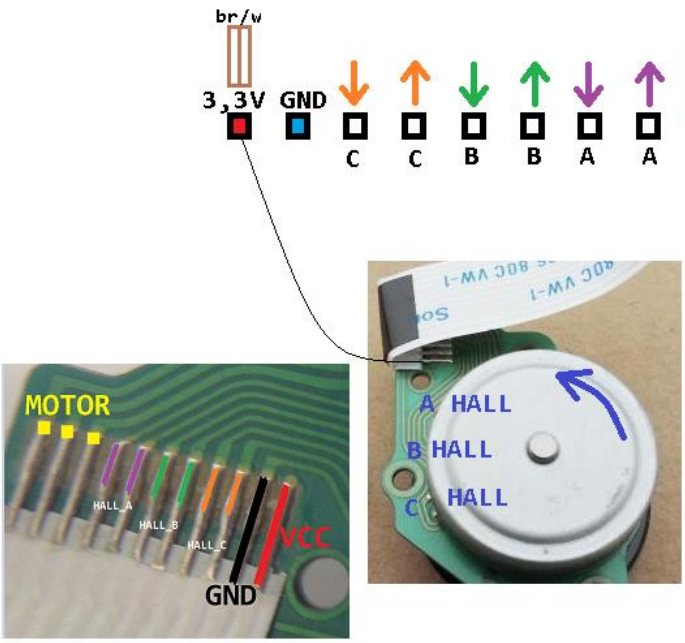
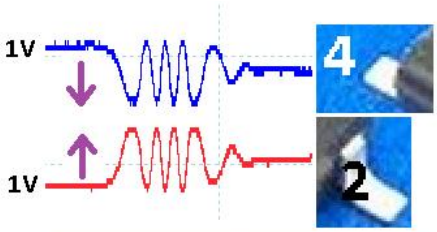
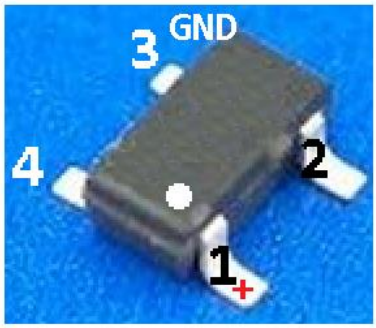
Three-Phase Motor Control with STM32F401RET6U

For a good animation of what we are doing look at http://elabz.com/brushless-dc-motor-with-arduino/

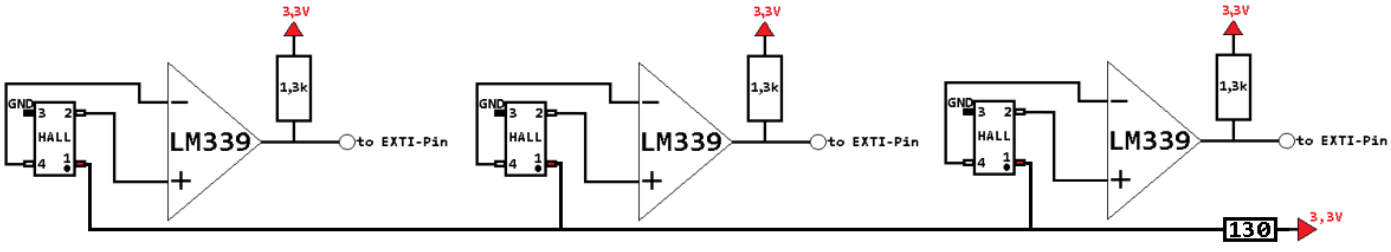
Our brushless direct current three-phase motor – **BLDC** – is a CDROM-motor and its connector is on the next pictures.
One can see two of the three HALL sensors:



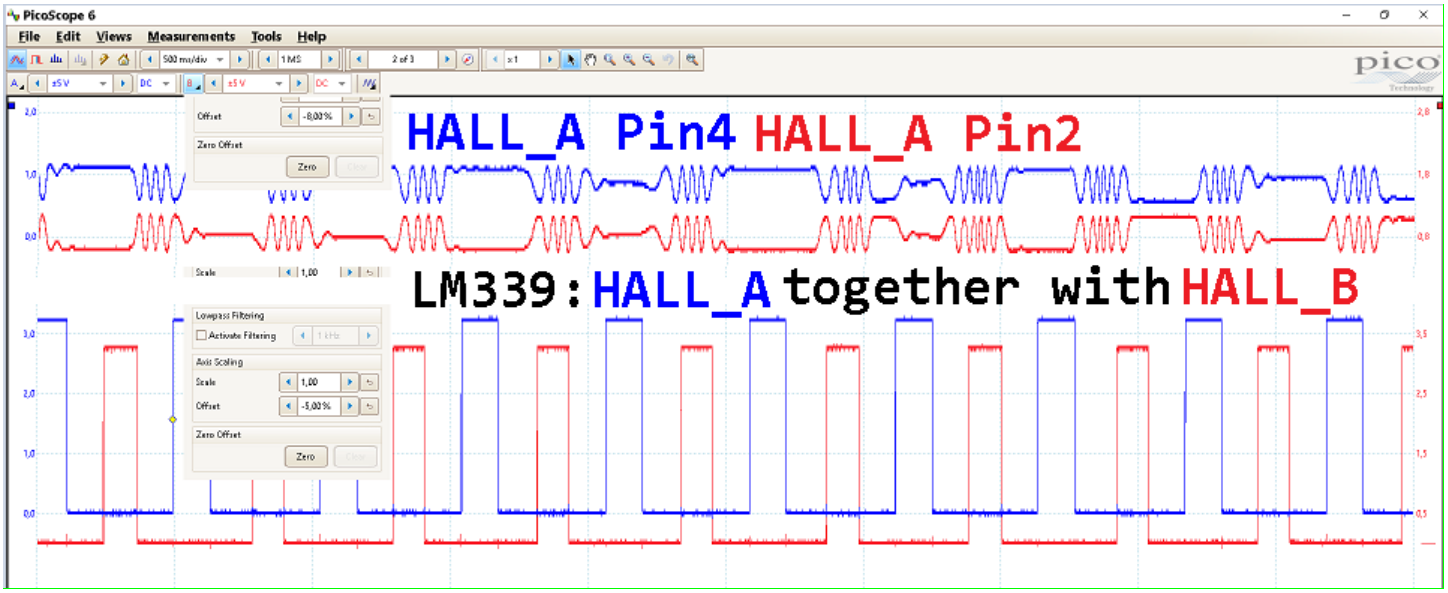




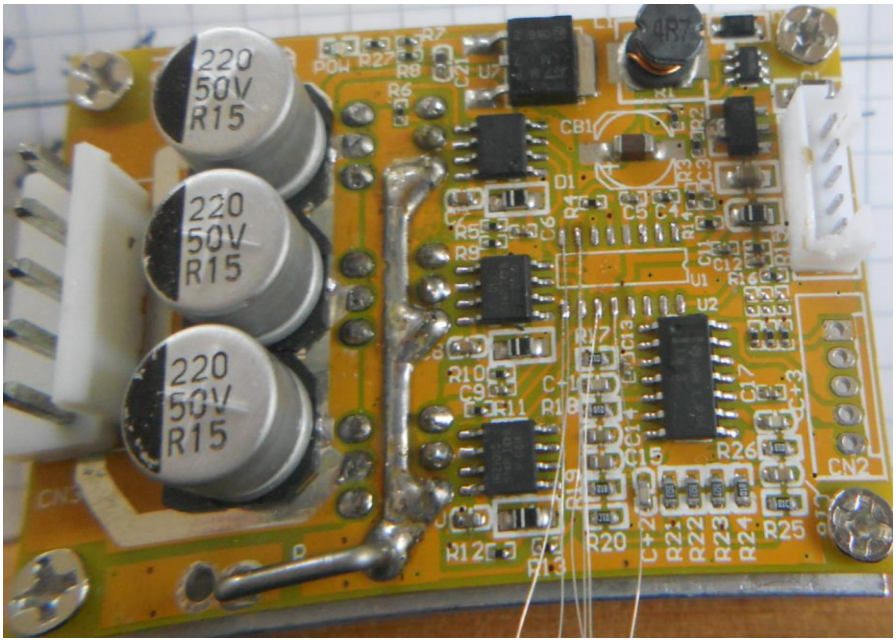
To get workable rectangular Hall sensor signals we use 3 of the 4 differential comparators of chip LM339N.



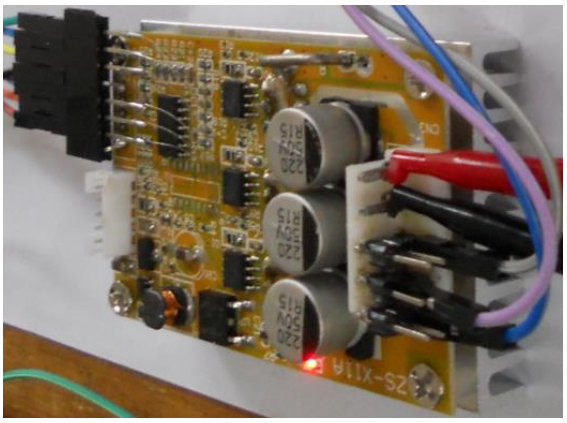
We will use both, the rising and falling edge of the rectangular signals.

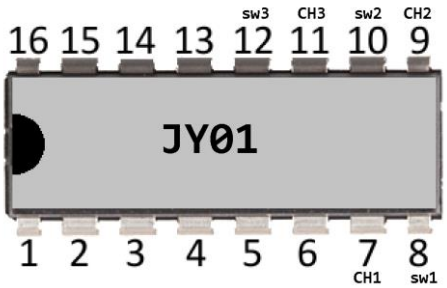


To conduct experiments with different three-phase BLDC-motors we need a suitable driver board.
We avoided a lot of soldering and bought some cheap 10€-boards in eBay and removed the JY01 BLDC-IC

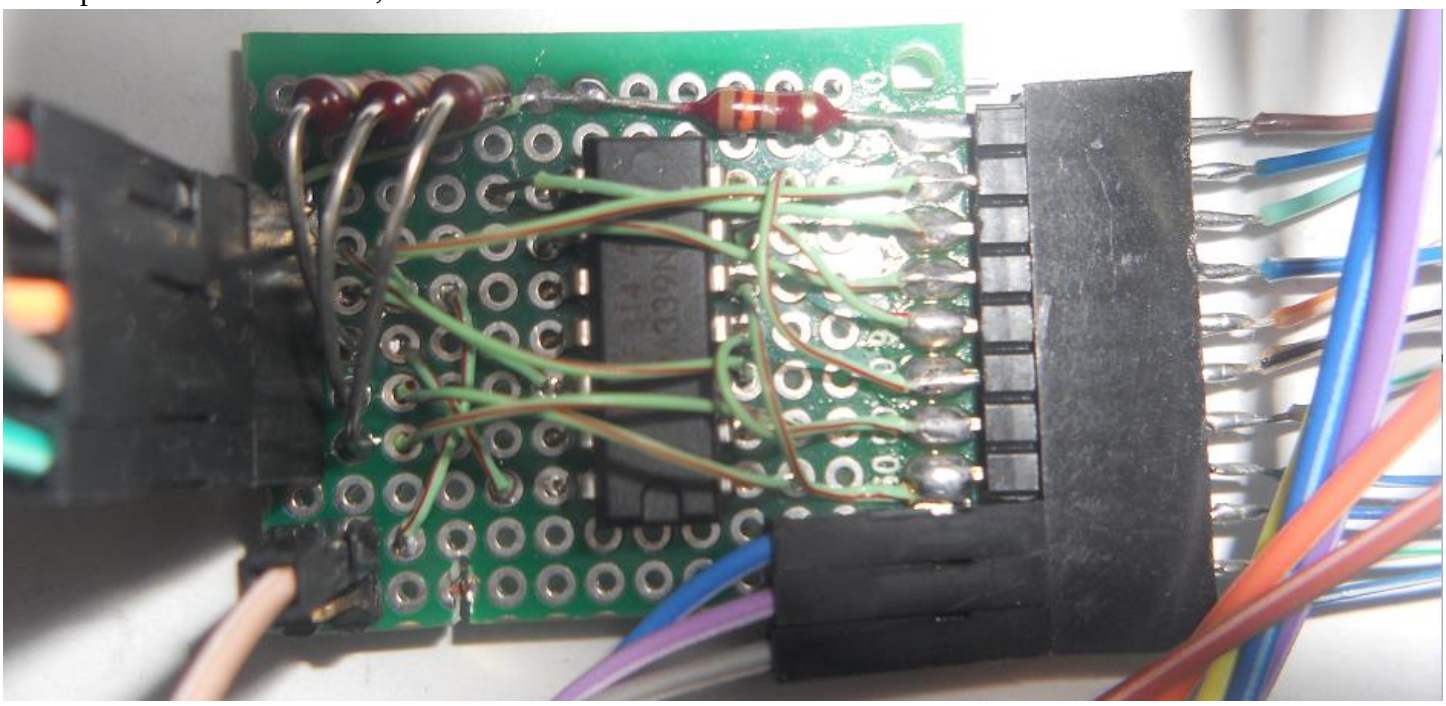


and ended up with:

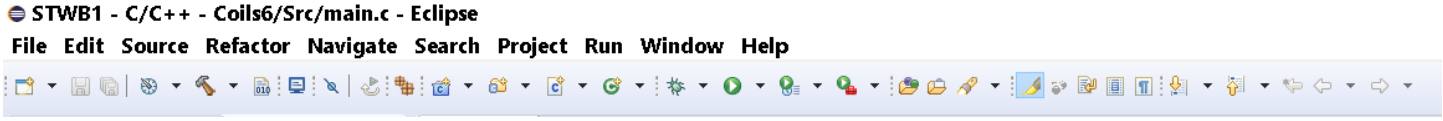




To switch the 6 MOSFETS we have to drive the 3 JY21L high speed power MOSFET drivers which are connected to the JY01-places of pin 7, 8 (first JY21L-IC), pin 9, 10 (second JY21L-IC) and pin 11, 12 (third JY21L-IC). To do this, we use the STM32 NUCLEO-F401RE prototype board and the port-pins CH1(PA6), CH2(PA7), CH3(PB0), sw1(PB6), sw2(PC7) and sw3(PC1). Some BLDC-motor driver boards JYQD_V6 have a comparator-IC too: LM339, that we do not use! Instead we use our own LM339N.

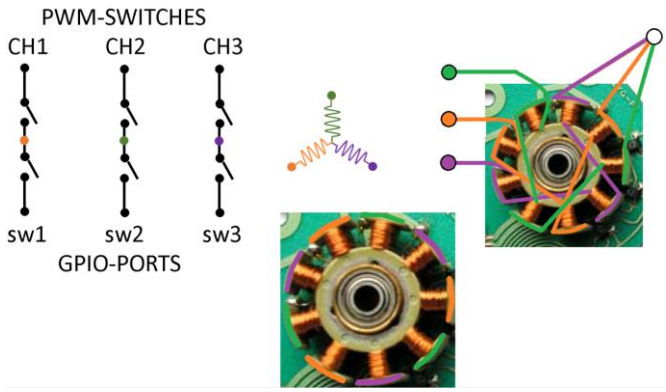


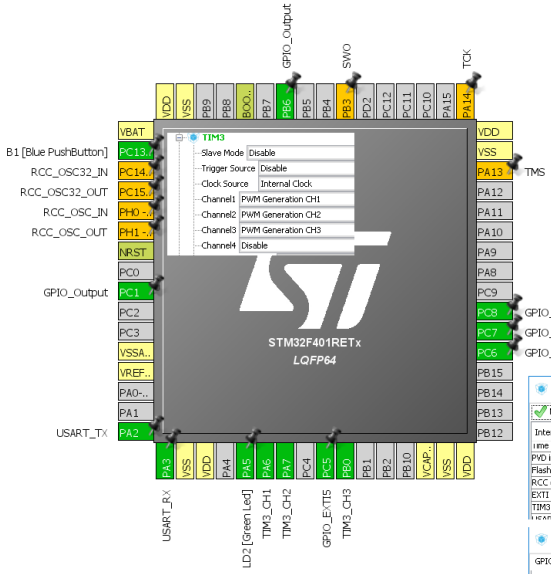
So much for the hardware. The software side consists of the SW4STM32 System Workbench http://www.st.com/en/development-tools/sw4stm32.html

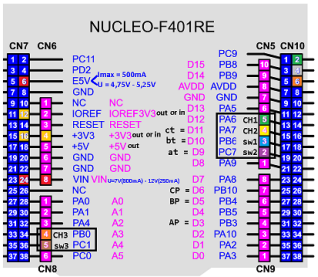


and the STM32CubeMX Workbench http://www.st.com/en/development-tools/stm32cubemx.html

CubeMX generates most of the code we use.
All of the PWM-switches are driven by Timer 3 and the three switches sw1, sw2 and sw3 are GPIO-ports in OUTPUT-mode.







TIM3 Configuration

Parameter Settings | User Constants | NVIC Settings | DMA Settings | GPIO Settings

Counter Settings	
Prescaler (PSC - 16 bits value)	1000
Counter Mode	Up
Counter Period (AutoReload Register - 16 bits value)	1000
Internal Clock Division (CKD)	No Division
Trigger Output (TRGO) Parameters	
Master/Slave Mode	Disable (no sync between this TIM (Master) and its S
Trigger Event Selection	Reset (UG bit from TIMx_EGR)
PWM Generation Channel 1	
Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 2	
Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High
PWM Generation Channel 3	
Mode	PWM mode 1
Pulse (16 bits value)	0
Fast Mode	Disable
CH Polarity	High

NVIC Configuration

NVIC | Code generation

Interrupt Table	Enabled	Preemption Priority	Sub Priority
time base: System tick timer	☑	0	0
PVD interrupt through EXTI line 16	☐	0	0
Flash global interrupt	☐	0	0
RCC global interrupt	☐	0	0
EXTI line[9:5] interrupts	☑	0	0
TIM3 global interrupt	☐	0	0

Pin Configuration

GPIO | Single Mapped Signals | TIM3 | USART2

Pin Name	Signal on Pin	GPIO output l…	GPIO mode	GPIO Pull-up/…	Maximum out…	U
PA5	n/a	Low	Output Push Pull	No pull-up and …	Low	LD2 [
PB6	n/a	Low	Output Push Pull	Pull-down	Low	
PC1	n/a	Low	Output Push Pull	Pull-down	Low	
PC5	n/a	n/a	External Interr…	No pull-up and …	n/a	
PC6	n/a	n/a	External Interr…	No pull-up and …	n/a	
PC7	n/a	Low	Output Push Pull	Pull-down	Low	
PC8	n/a	n/a	External Interr…	No pull-up and …	n/a	
PC13-ANTI_TA…	n/a	n/a	External Interr…	No pull-up and …	n/a	B1 [B

PC5#PC6#PC8 Configuration :

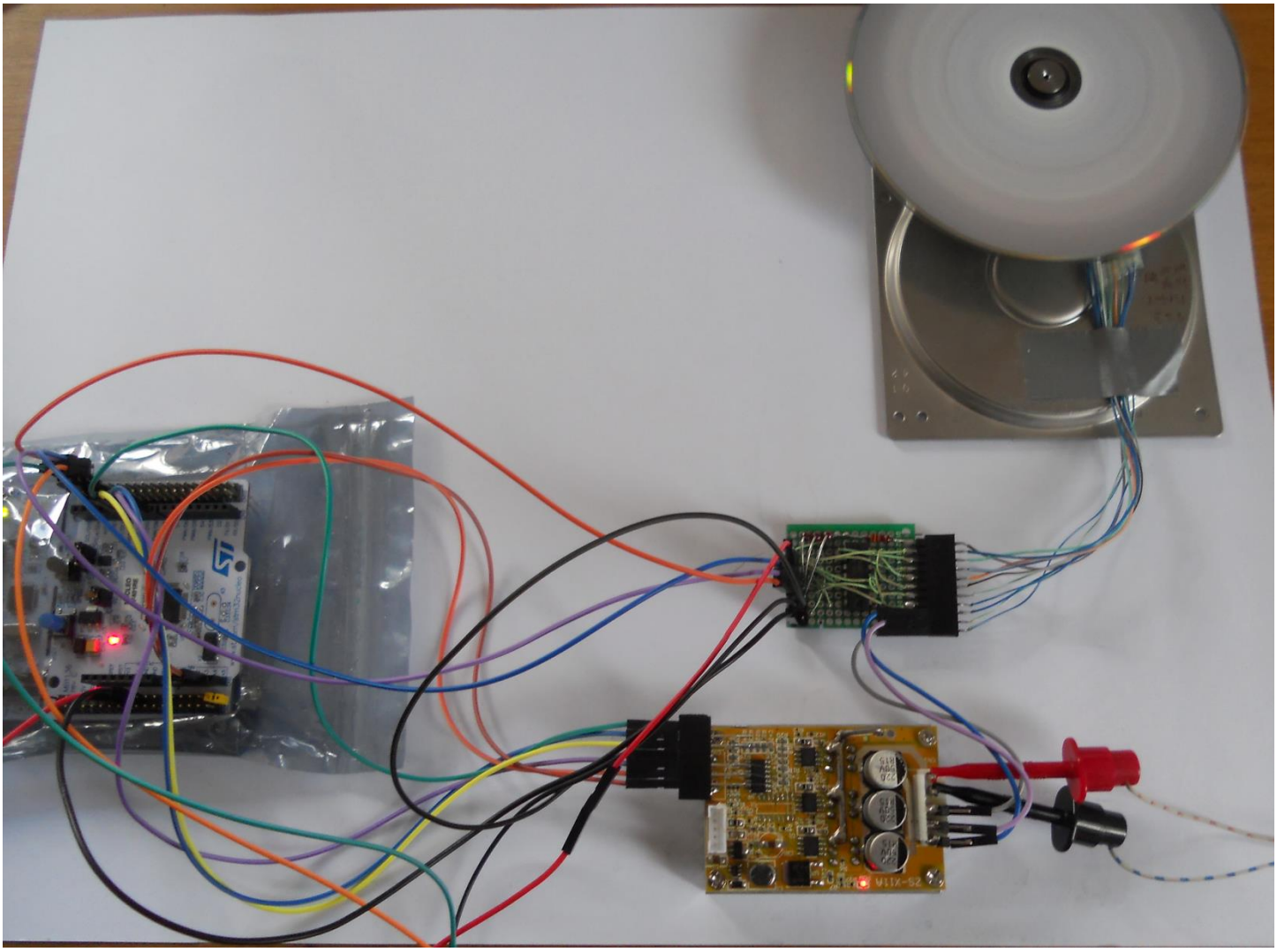
GPIO mode: External Interrupt Mode with Rising edge trigger detection

GPIO Pull-up/Pull-down: No pull-up and no pull-down

Pin Configuration

GPIO | Single Mapped Signals | TIM3 | USART2

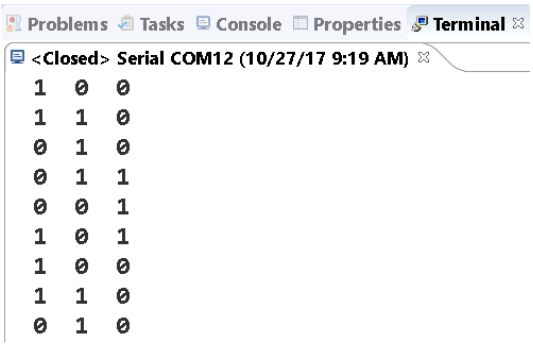
Pin Name	Signal on Pin	GPIO output l…	GPIO mode	GPIO Pull-up/…	Maximum out…	Use
PA5	n/a	Low	Output Push Pull	No pull-up and …	Low	LD2 [G
PB6	n/a	Low	Output Push Pull	Pull-down	Low	
PC1	n/a	Low	Output Push Pull	Pull-down	Low	
PC5	n/a	n/a	External Interr…	No pull-up and …	n/a	
PC6	n/a	n/a	External Interr…	No pull-up and …	n/a	
PC7	n/a	Low	Output Push Pull	Pull-down	Low	
PC8	n/a	n/a	External Interr…	No pull-up and …	n/a	
PC13-ANTI_TA…	n/a	n/a	External Interr…	No pull-up and …	n/a	B1 [Blu



The via LM339 rectangular HALL sensor signals are event sources and generate external interrupts on the NUCLEO board. We are interested in changing edges of the signals and do that with GPIO port pins PC5, PC6 and PC8 in EXTI mode. But what is the chronological order of these signals for one rotation of the CDROM BLDC-motor?

We get an answer by these lines of code and ***GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;*** :

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
uint8_t hall1 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_5);
uint8_t hall2 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6);
uint8_t hall3 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);
Info_3d(hall1, hall2, hall3, "  ");}
```

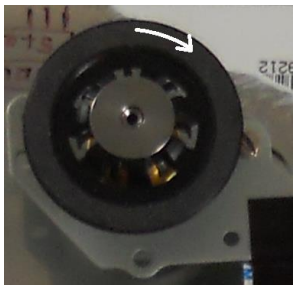


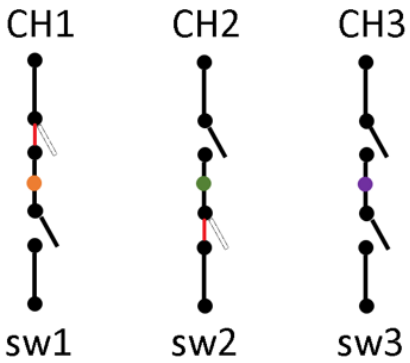
One clockwise rotation generates 6 times these sequence: 1 0 0 1 1 0 0 1 0 0 1 1 0 0 1 1 0 1

HALL_A, HALL_B, HALL_C or written as: hall1, hall2, hall3. In green step one is hall1=1, hall2=0, hall3=0, in yellow step two is hall1=1, hall2=1, hall3=0 and so on. As we can see we get 6 cases which we put in 6 different numbers: **hall1 = 100*hall1 + 10*hall2 + hall3;**

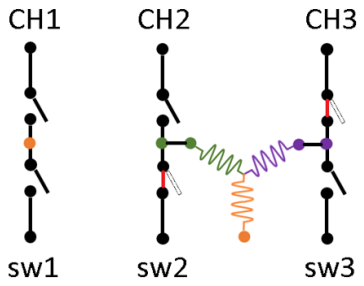
Each case calls for a different position of our switches of the JYQD_V6 driver board.

```
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  hall1 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_5);
  hall2 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6);
  hall3 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);
  new_step = 100*hall1 + 10*hall2 + hall3;
  switch (new_step){
    case 10://1,2
      TIM3->CCR2 = 0;
      TIM3->CCR3 = 0;
      GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
      GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
      TIM3->CCR1 = PWM_dn;
      GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
      break;
```

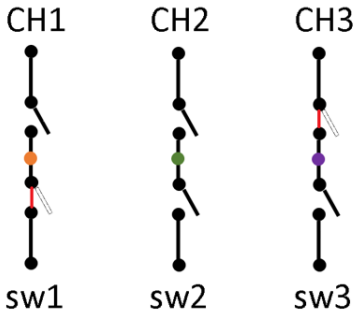




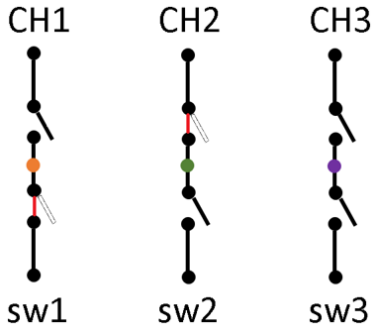
```
case 11://3,2
  TIM3->CCR1 = 0;
  TIM3->CCR2 = 0;
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR3 = PWM_dn;
  GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
  break;
```



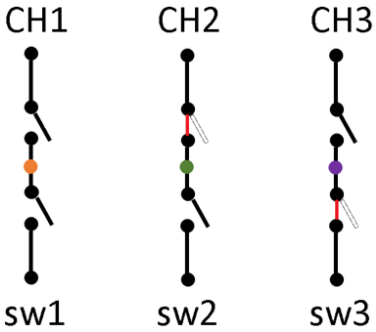

```
case 1://3,1
  TIM3->CCR1 = 0;
  TIM3->CCR2 = 0;
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR3 = PWM_dn;
  GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
  break;
```



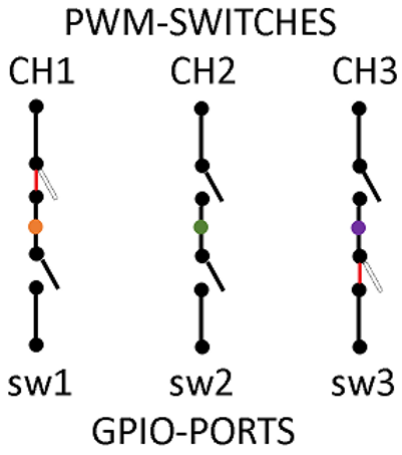

```
case 101://2,1
  TIM3->CCR1 = 0;
  TIM3->CCR3 = 0;
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR2 = PWM_dn;
  GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
  break;
```




```
case 100://2,3
  TIM3->CCR1 = 0;
  TIM3->CCR3 = 0;
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  TIM3->CCR2 = PWM_dn;
  GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
```



```
      break;
    case 110://1,3
      TIM3->CCR2 = 0;
      TIM3->CCR3 = 0;
      GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
      GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
      TIM3->CCR1 = PWM_dn;
      GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
      if(PWM_dn < 10000) PWM_dn += 1;
    break;
    default: //because of some noise
      TIM3->CCR1 = 0;
      TIM3->CCR2 = 0;
      TIM3->CCR3 = 0;
    }
}
```



In the code listed below CubeMX-code is black and our code is in red.

main.c

```
#include "main.h"
#include "stm32f4xx_hal.h"
#include <string.h> //strlen(str)
TIM_HandleTypeDef htim3;
UART_HandleTypeDef huart2;
uint8_t hall1, hall2, hall3, new_step, start = 1;
uint16_t PWM_dn = 5000;
void CH123_sw123_off(void);
char* my_utoa(unsigned, char*);
void Info_d(unsigned, char*);
void Info_3d(unsigned, unsigned, unsigned, char*);
char txt[20];
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_USART2_UART_Init(void);
static void MX_TIM3_Init(void);
void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);
int main(void)
{
  HAL_Init();
  SystemClock_Config();
  MX_GPIO_Init();
  MX_USART2_UART_Init();
  MX_TIM3_Init();
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_1); //PA6
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_2); //PA7
  HAL_TIM_PWM_Start(&htim3, TIM_CHANNEL_3); //PB0
  CH123_sw123_off();
  while(1){}
```

```
}
char* my_utoa(unsigned val, char *buffer)
{
char*cp = buffer;
unsigned v;
char c;
v = val;
do {
  v /= 10;
  cp++;
} while(v != 0);
*cp-- = 0;
do {
  c = val % 10;
  val /= 10;
  c += '0';
  *cp-- = c;
} while(val != 0);
  return  buffer;
}
void Info_d(unsigned n, char *str)
{
  my_utoa(n, txt);//size_t strlen(const char *str);
  HAL_UART_Transmit(&huart2, (uint8_t *) str, strlen(str), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t *) txt, strlen(txt), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t *) "\n\r", 2, HAL_MAX_DELAY);
}
void Info_3d(unsigned n1, unsigned n2, unsigned n3, char *str)
{
  my_utoa(n1, txt);//size_t strlen(const char *str);
  HAL_UART_Transmit(&huart2, (uint8_t *) str, strlen(str), HAL_MAX_DELAY);
```

```
  HAL_UART_Transmit(&huart2, (uint8_t *) txt, strlen(txt), HAL_MAX_DELAY);
  my_utoa(n2, txt);//size_t strlen(const char *str);
  HAL_UART_Transmit(&huart2, (uint8_t *) str, strlen(str), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t *) txt, strlen(txt), HAL_MAX_DELAY);
  my_utoa(n3, txt);//size_t strlen(const char *str);
  HAL_UART_Transmit(&huart2, (uint8_t *) str, strlen(str), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t *) txt, strlen(txt), HAL_MAX_DELAY);
  HAL_UART_Transmit(&huart2, (uint8_t *) "\n\r", 2, HAL_MAX_DELAY);
}
void CH123_sw123_off(void){
  //sConfigOC.Pulse = 0; means (A6)TIM3->CCR1 = 0; (A7)TIM3->CCR2 = 0; (B0)TIM3->CCR3 = 0;
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));//B6 off pwm_+_pol___/ _______/___GND
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));//C7 off +___/ _______/___GND
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));//C1 off +___/ _______/___GND
}
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  hall1 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_5);
  hall2 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6);
  hall3 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);
  new_step = 100*hall1 + 10*hall2 + hall3;
  switch (new_step){
    case 10://1,2
      TIM3->CCR2 = 0;
      TIM3->CCR3 = 0;
      GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
      GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
      TIM3->CCR1 = PWM_dn;
      GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
      break;
    case 11://3,2
      TIM3->CCR1 = 0;
```

```
  TIM3->CCR2 = 0;
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR3 = PWM_dn;
  GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
  break;
 case 1://3,1
  TIM3->CCR1 = 0;
  TIM3->CCR2 = 0;
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR3 = PWM_dn;
  GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
  break;
 case 101://2,1
  TIM3->CCR1 = 0;
  TIM3->CCR3 = 0;
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR2 = PWM_dn;
  GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
  break;
 case 100://2,3
  TIM3->CCR1 = 0;
  TIM3->CCR3 = 0;
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  TIM3->CCR2 = PWM_dn;
  GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
  break;
 case 110://1,3
```

```
    TIM3->CCR2 = 0;
    TIM3->CCR3 = 0;
    GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
    GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
    TIM3->CCR1 = PWM_dn;
    GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
    if(PWM_dn < 10000) PWM_dn += 1;//press black nucleo button for acceleration
   break;
   default: //because of some noise
    TIM3->CCR1 = 0;
    TIM3->CCR2 = 0;
    TIM3->CCR3 = 0;
   }
}
void SystemClock_Config(void)
{
  RCC_OscInitTypeDef RCC_OscInitStruct;
  RCC_ClkInitTypeDef RCC_ClkInitStruct;
  __HAL_RCC_PWR_CLK_ENABLE();
  __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE2);
  RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSI;
  RCC_OscInitStruct.HSIState = RCC_HSI_ON;
  RCC_OscInitStruct.HSICalibrationValue = 16;
  RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
  RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSI;
  RCC_OscInitStruct.PLL.PLLM = 16;
  RCC_OscInitStruct.PLL.PLLN = 336;
  RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV4;
  RCC_OscInitStruct.PLL.PLLQ = 7;
  if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK) _Error_Handler(__FILE__, __LINE__);
```

```
  RCC_ClkInitStruct.ClockType =
RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK|RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYP
E_PCLK2;

  RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;

  RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;

  RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV2;

  RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;

  if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
_Error_Handler(__FILE__, __LINE__);

  HAL_SYSTICK_Config(HAL_RCC_GetHCLKFreq()/1000);

  HAL_SYSTICK_CLKSourceConfig(SYSTICK_CLKSOURCE_HCLK);

  HAL_NVIC_SetPriority(SysTick_IRQn, 0, 0);

}

static void MX_TIM3_Init(void)

{

  TIM_ClockConfigTypeDef sClockSourceConfig;

  TIM_MasterConfigTypeDef sMasterConfig;

  TIM_OC_InitTypeDef sConfigOC;

  htim3.Instance = TIM3;

  htim3.Init.Prescaler = 1;//was 1;2 is good

  htim3.Init.CounterMode = TIM_COUNTERMODE_UP;

  htim3.Init.Period = 10000;

  htim3.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;//was TIM_CLOCKDIVISION_DIV1;

  if (HAL_TIM_Base_Init(&htim3) != HAL_OK) _Error_Handler(__FILE__, __LINE__);

  sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;

  if (HAL_TIM_ConfigClockSource(&htim3, &sClockSourceConfig) != HAL_OK) _Error_Handler(__FILE__,
__LINE__);

  if (HAL_TIM_PWM_Init(&htim3) != HAL_OK) _Error_Handler(__FILE__, __LINE__);

  sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;

  sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;

  if (HAL_TIMEx_MasterConfigSynchronization(&htim3, &sMasterConfig) != HAL_OK)
_Error_Handler(__FILE__, __LINE__);

  sConfigOC.OCMode = TIM_OCMODE_PWM1;
```

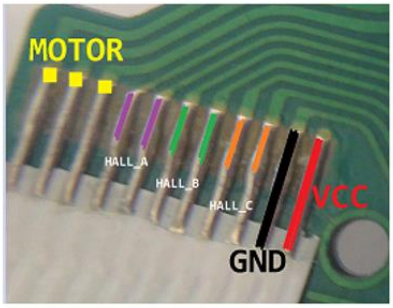
```
  sConfigOC.Pulse = 0;
  sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
  sConfigOC.OCFastMode = TIM_OCFAST_ENABLE;//was TIM_OCFAST_DISABLE;
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
_Error_Handler(__FILE__, __LINE__);
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_2) != HAL_OK)
_Error_Handler(__FILE__, __LINE__);
  if (HAL_TIM_PWM_ConfigChannel(&htim3, &sConfigOC, TIM_CHANNEL_3) != HAL_OK)
_Error_Handler(__FILE__, __LINE__);
  HAL_TIM_MspPostInit(&htim3);
}
static void MX_USART2_UART_Init(void)
{
  huart2.Instance = USART2;
  huart2.Init.BaudRate = 115200;
  huart2.Init.WordLength = UART_WORDLENGTH_8B;
  huart2.Init.StopBits = UART_STOPBITS_1;
  huart2.Init.Parity = UART_PARITY_NONE;
  huart2.Init.Mode = UART_MODE_TX_RX;
  huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
  huart2.Init.OverSampling = UART_OVERSAMPLING_16;
  if (HAL_UART_Init(&huart2) != HAL_OK) _Error_Handler(__FILE__, __LINE__);
}
static void MX_GPIO_Init(void)
{
  GPIO_InitTypeDef GPIO_InitStruct;
  __HAL_RCC_GPIOC_CLK_ENABLE();
  __HAL_RCC_GPIOH_CLK_ENABLE();
  __HAL_RCC_GPIOA_CLK_ENABLE();
  __HAL_RCC_GPIOB_CLK_ENABLE();
  HAL_GPIO_WritePin(LD2_GPIO_Port, LD2_Pin, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOB, GPIO_PIN_6, GPIO_PIN_RESET);
```

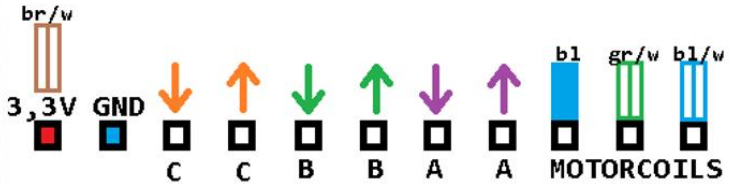
```
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_1, GPIO_PIN_RESET);
  HAL_GPIO_WritePin(GPIOC, GPIO_PIN_7, GPIO_PIN_RESET);
  GPIO_InitStruct.Pin = B1_Pin|GPIO_PIN_5|GPIO_PIN_6|GPIO_PIN_8;
  //GPIO_InitStruct.Mode = GPIO_MODE_IT_FALLING;
  //GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING;
  GPIO_InitStruct.Mode = GPIO_MODE_IT_RISING_FALLING;
  GPIO_InitStruct.Pull = GPIO_NOPULL;//GPIO_NOPULL;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = LD2_Pin;
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_NOPULL;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
  HAL_GPIO_Init(LD2_GPIO_Port, &GPIO_InitStruct);
  GPIO_InitStruct.Pin = GPIO_PIN_1|GPIO_PIN_7;//C1,C7
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);//C1,C7
  GPIO_InitStruct.Pin = GPIO_PIN_6;//B6
  GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
  GPIO_InitStruct.Pull = GPIO_PULLDOWN;
  GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_HIGH;
  HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);//B6
  HAL_NVIC_SetPriority(EXTI9_5_IRQn, 0, 0);
  HAL_NVIC_EnableIRQ(EXTI9_5_IRQn);
}
void _Error_Handler(char * file, int line)
{  while(1);  }
#ifdef USE_FULL_ASSERT
void assert_failed(uint8_t* file, uint32_t line){}
#endif
```

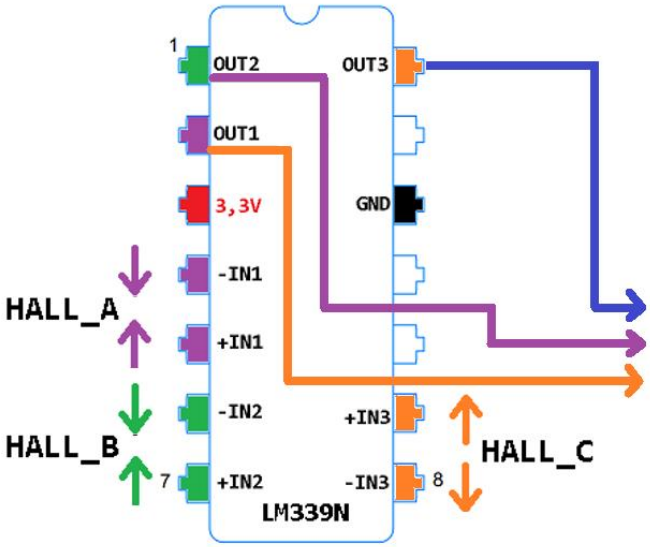
main.h

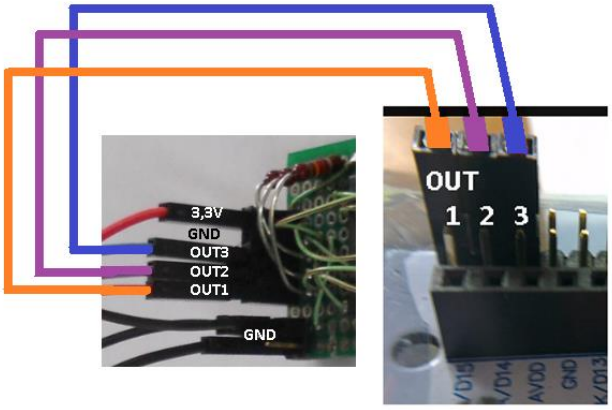
```
#ifndef __MAIN_H
#define __MAIN_H

#define B1_Pin GPIO_PIN_13
#define B1_GPIO_Port GPIOC
#define USART_TX_Pin GPIO_PIN_2
#define USART_TX_GPIO_Port GPIOA
#define USART_RX_Pin GPIO_PIN_3
#define USART_RX_GPIO_Port GPIOA
#define LD2_Pin GPIO_PIN_5
#define LD2_GPIO_Port GPIOA
#define TMS_Pin GPIO_PIN_13
#define TMS_GPIO_Port GPIOA
#define TCK_Pin GPIO_PIN_14
#define TCK_GPIO_Port GPIOA
#define SWO_Pin GPIO_PIN_3
#define SWO_GPIO_Port GPIOB
#define sw1 6 //switch1:GPIO_Port_B6 works with TIM3->CCR1
#define sw2 7 //switch2:GPIO_Port_C7 works with TIM3->CCR2
#define sw3 1 //switch3:GPIO_Port_C1 works with TIM3->CCR3

void _Error_Handler(char *, int);

#define Error_Handler() _Error_Handler(__FILE__, __LINE__)
#endif /* __MAIN_H */
```

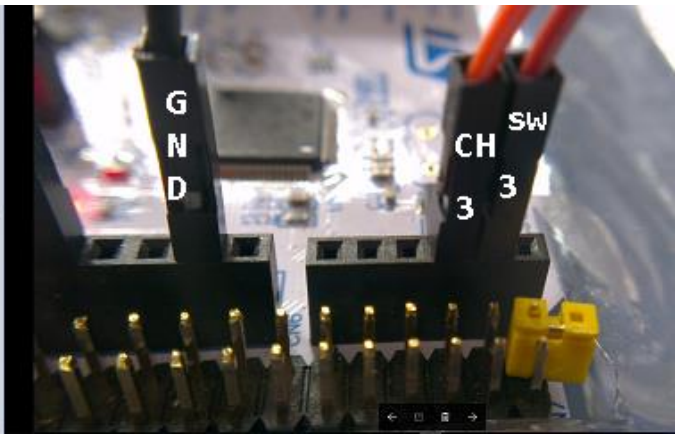
The whole wiring finally:

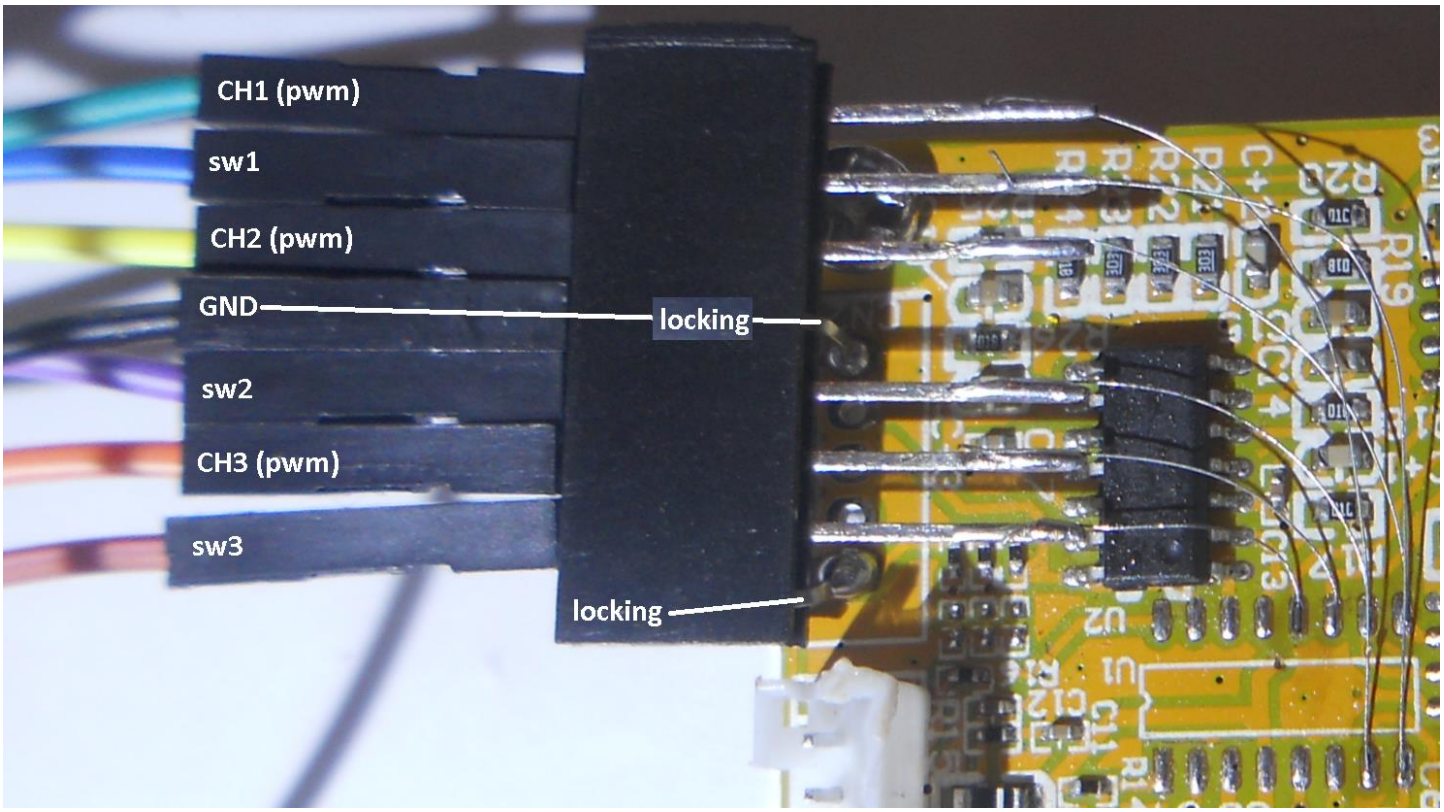


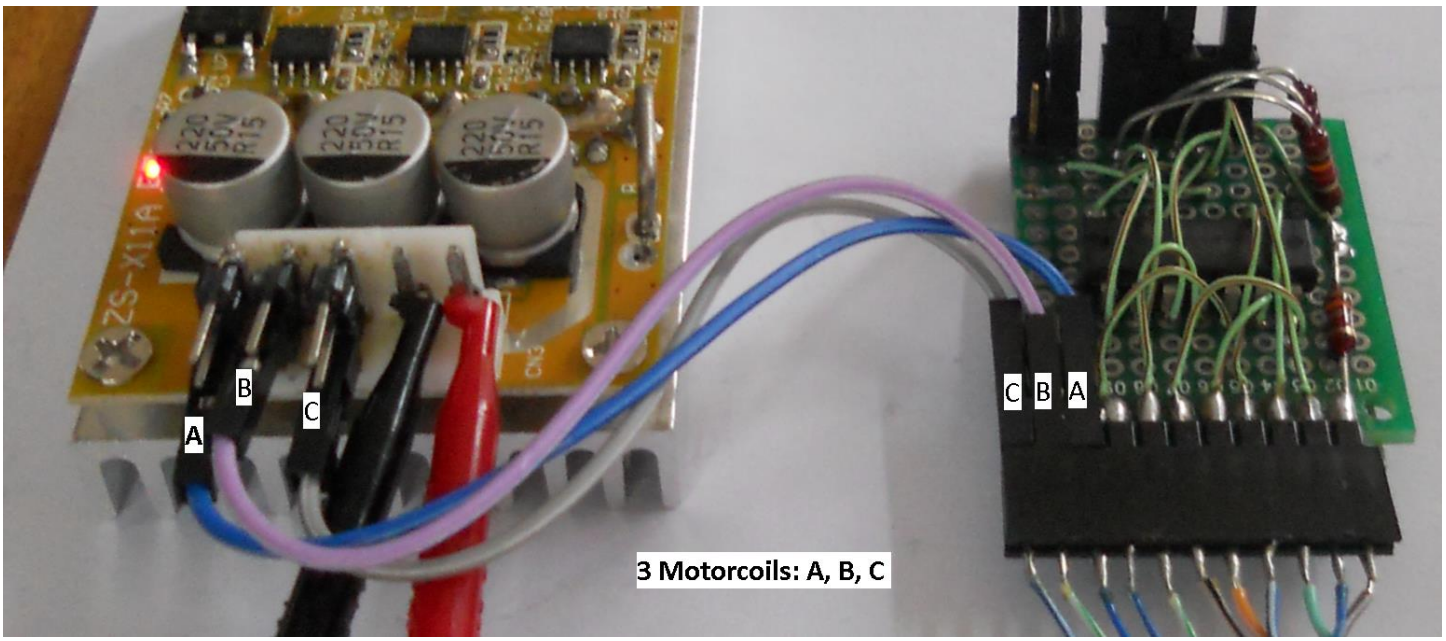








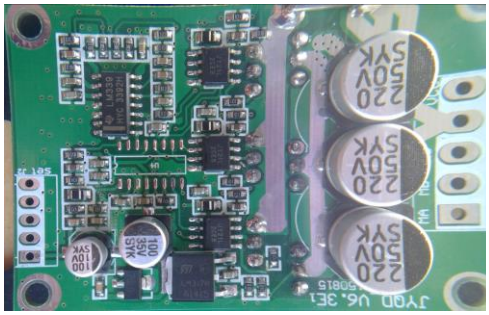

CH1 (pwm)
sw1
CH2 (pwm)
GND
locking
sw2
CH3 (pwm)
sw3
locking


A
B
C
C B A
3 Motorcoils: A, B, C

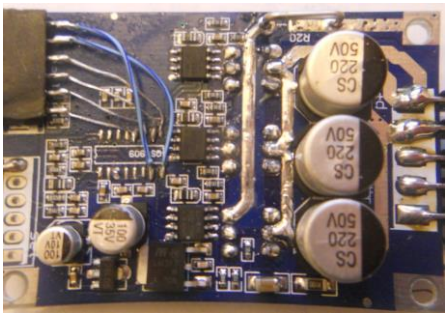
You can buy several driver-boards:

Works with U=3V and more

Works with U=5V and more

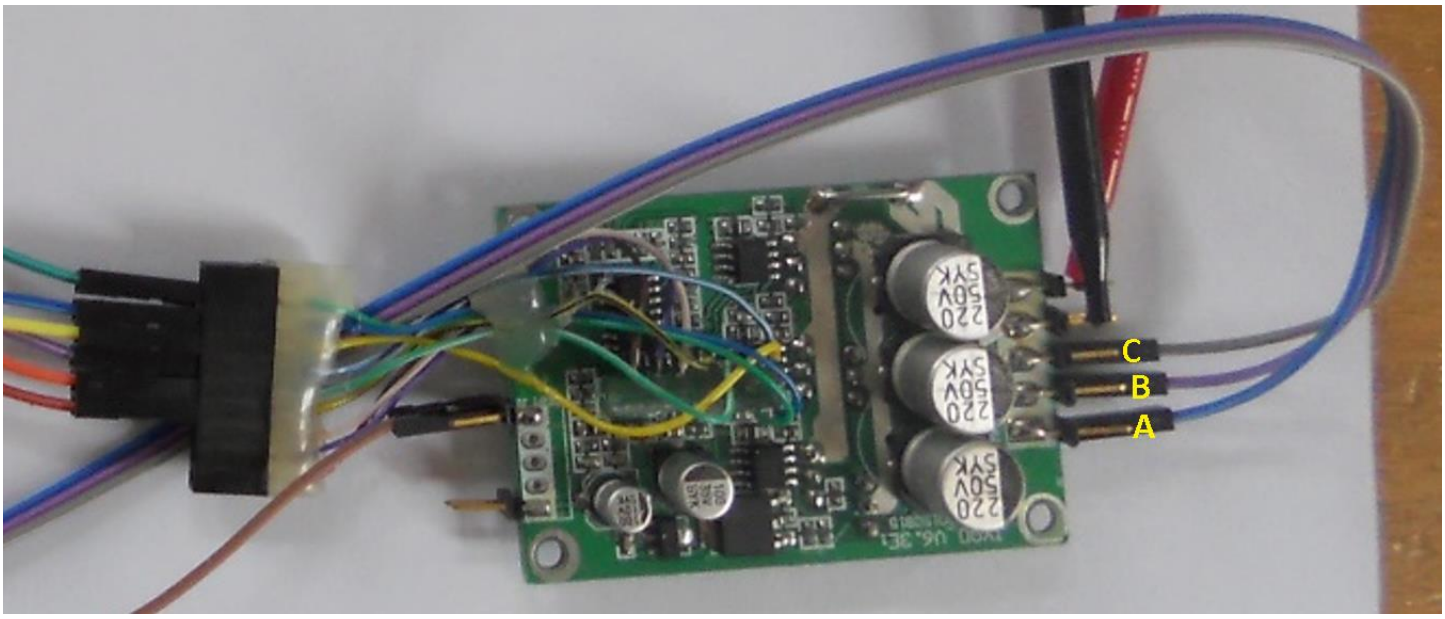


Works with U=11V and more

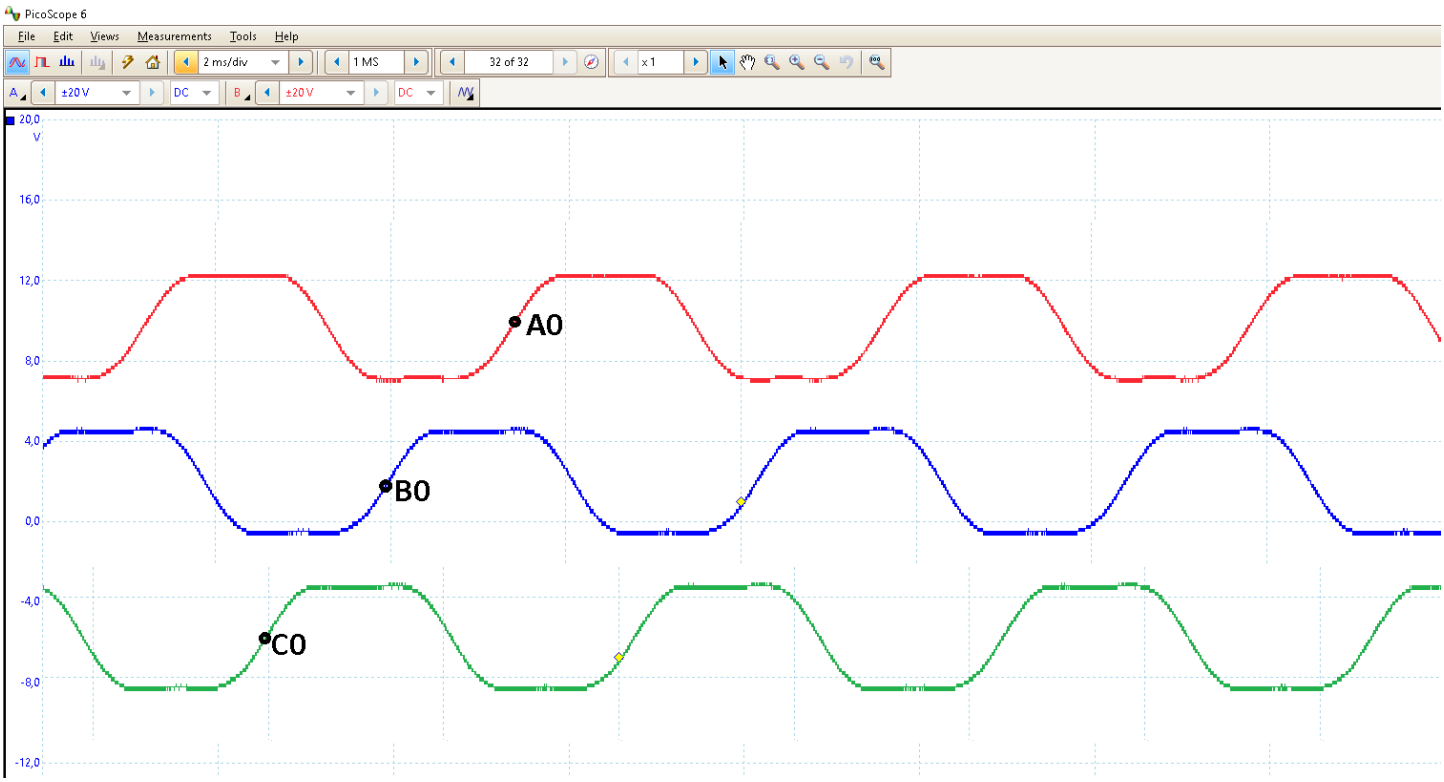


Three-Phase Motor Control without HALL sensors

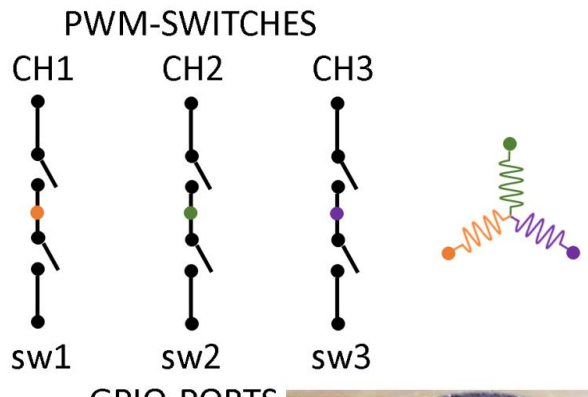
We used HALL sensors to detect the position of the rotor with its permanent magnets. A second way is to use induction to detect rotor positions.

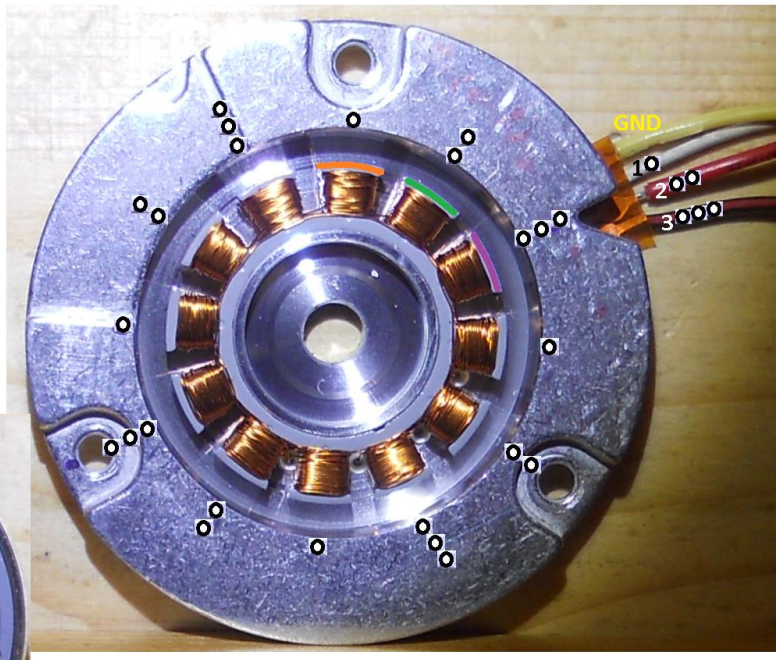


If we **switch off** the fast running CDROM-motor and measure the signals with respect to GND on pin A, B and C we get this picture:

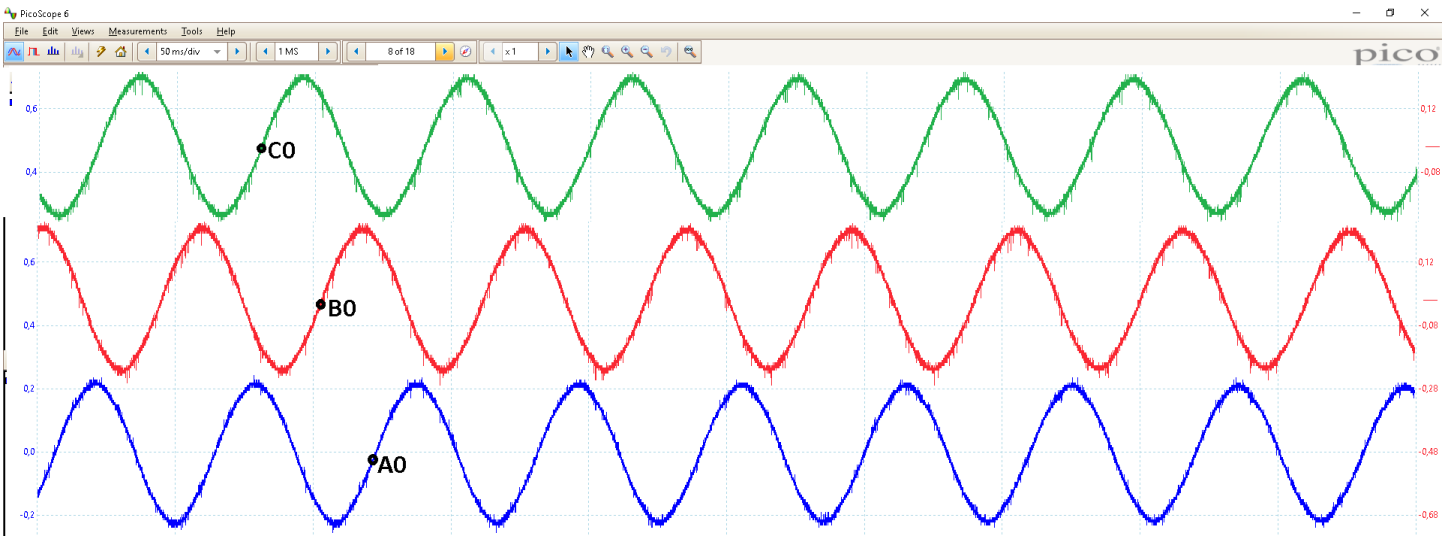


One can use many other BLDC-motors like HDD-motors. The principle to determine rotor positions without HALL-sensors will be always the same, and the oscilloscope diagrams too:

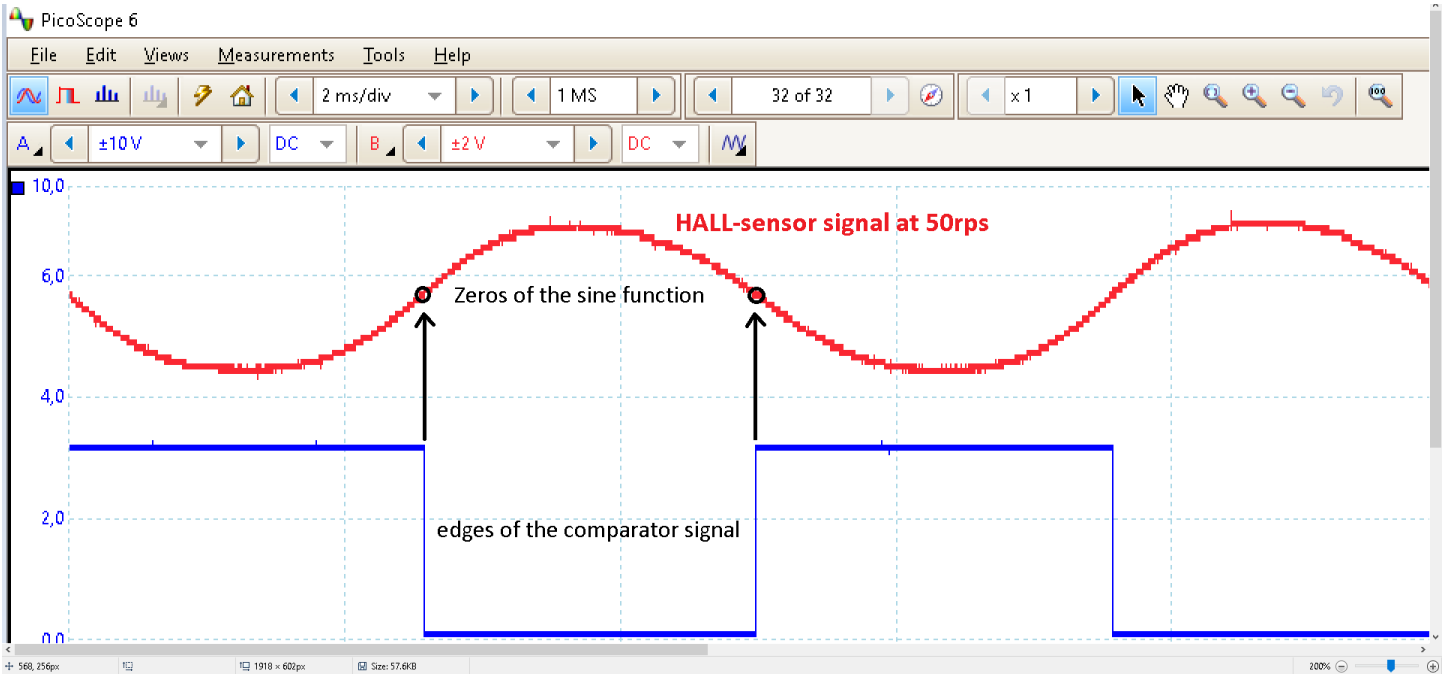




This HDD-motor has four times the three coils, and if we **switch off** the fast running HDD-motor and measure the signals between GND, 1 , GND, 2 and GND, 3 we get this picture:

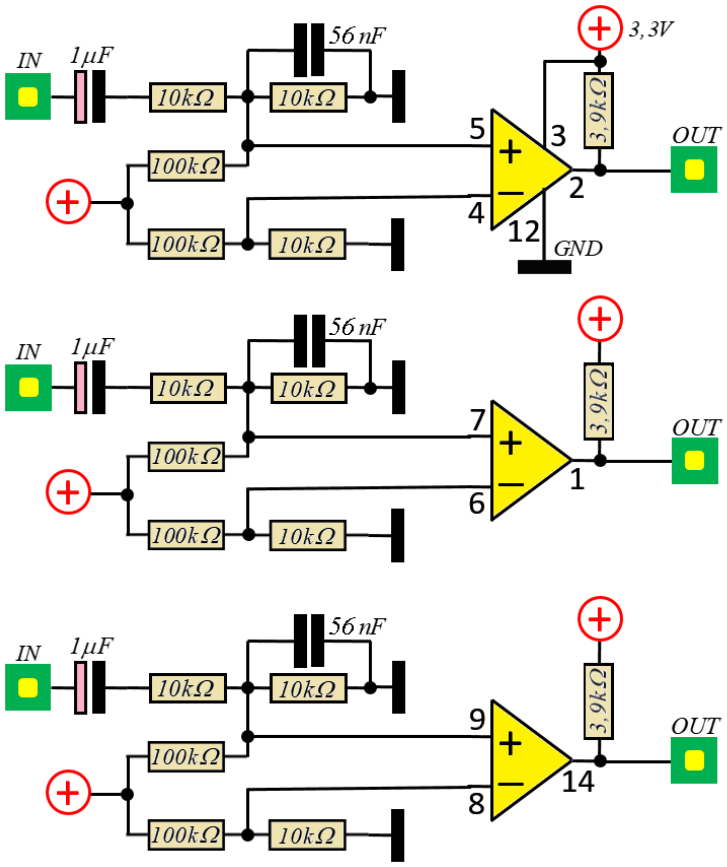


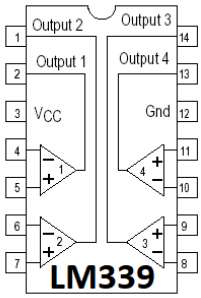
We compare these oscilloscope pictures with an oscilloscope picture of one HALL-sensor and a motor rotation of 50rps = 3000rpm:

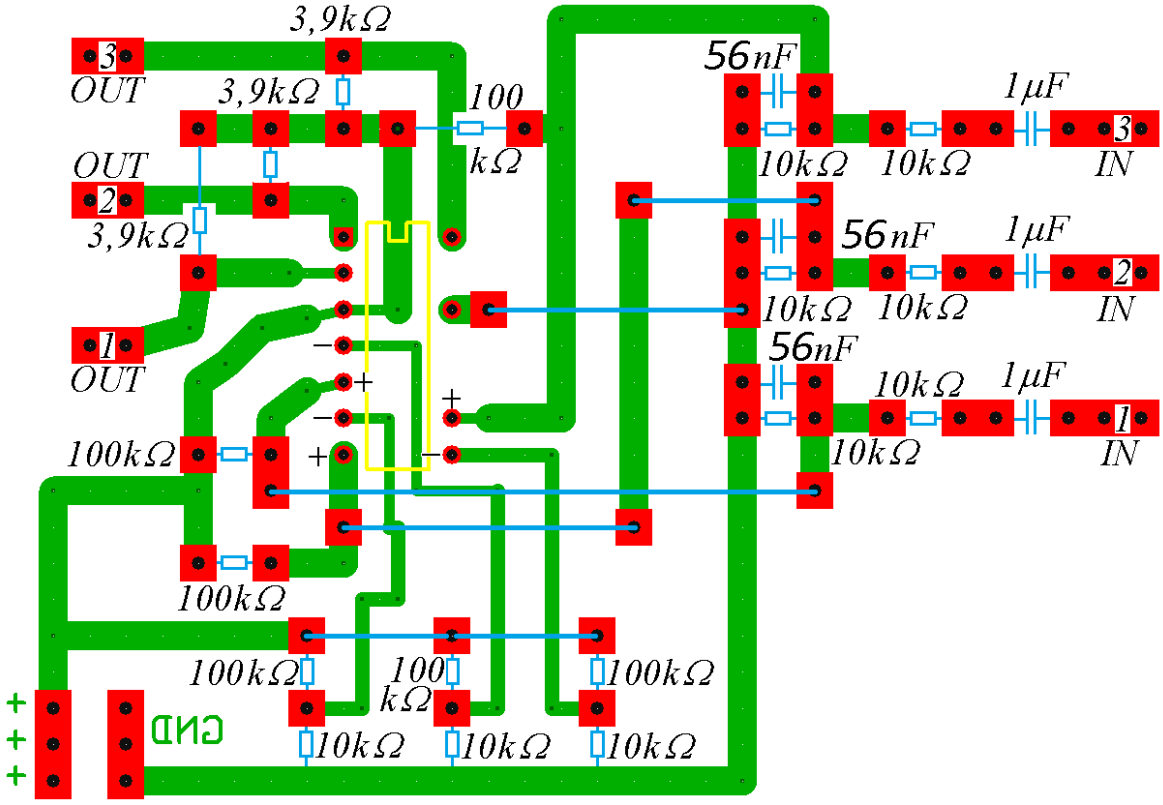


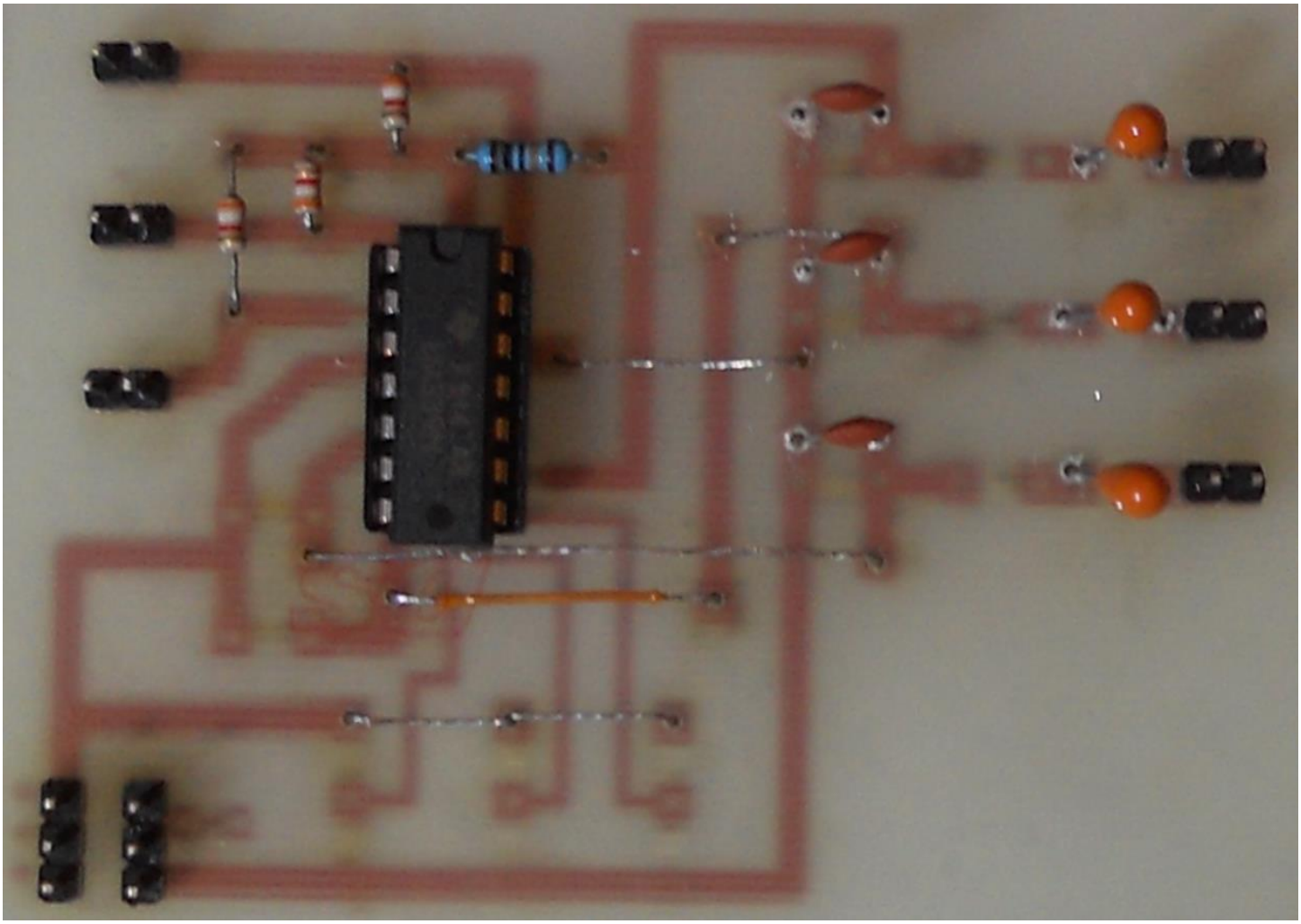
We see that we have to generate edges at zero points A0, B0 and C0 to control the BLDC-motor with the same C-program of our NUCLEO-board.

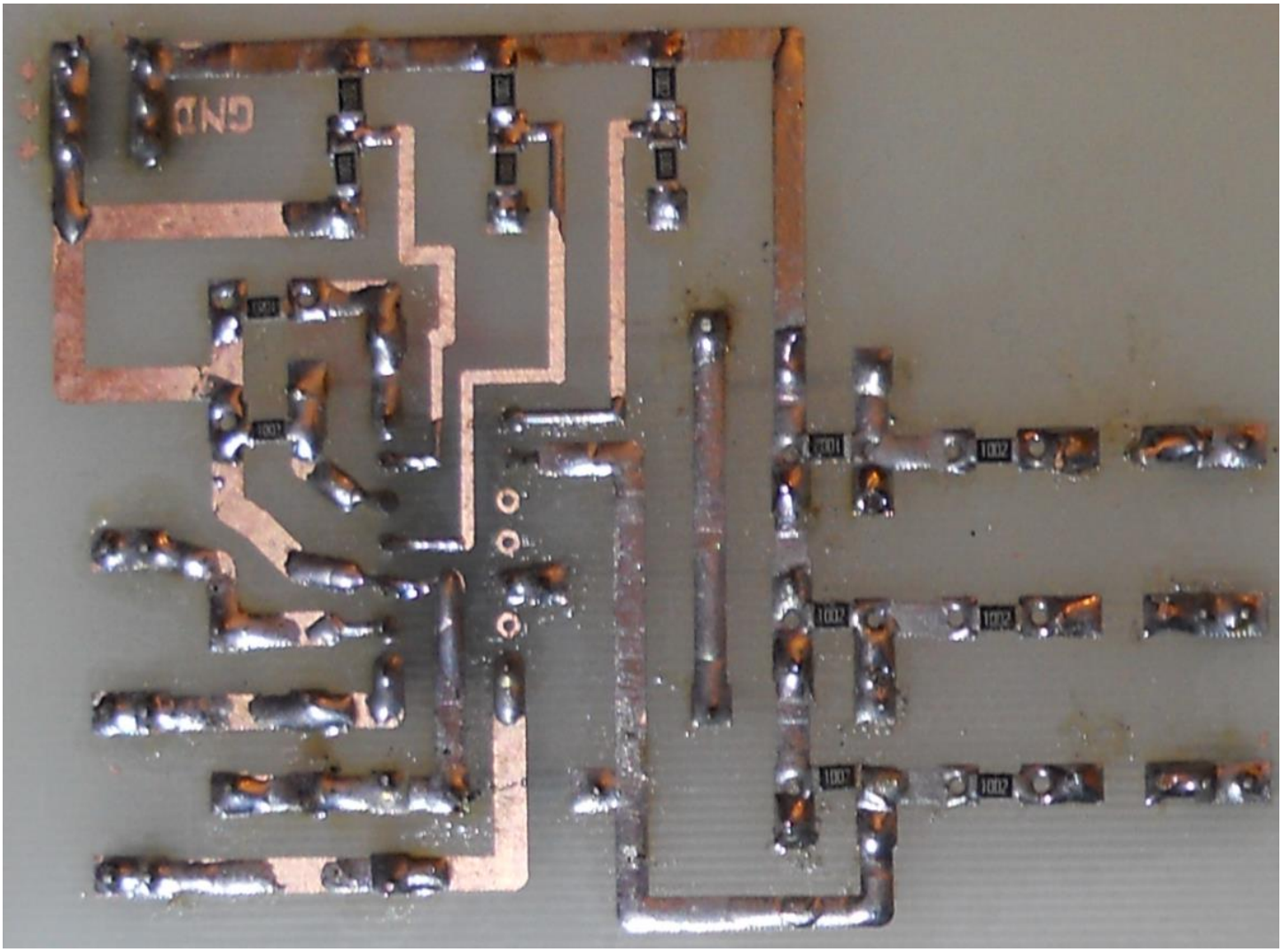
Again LM339 generates rectangular signals as external interrupts for the NUCLEO-board, but the circuit changes:

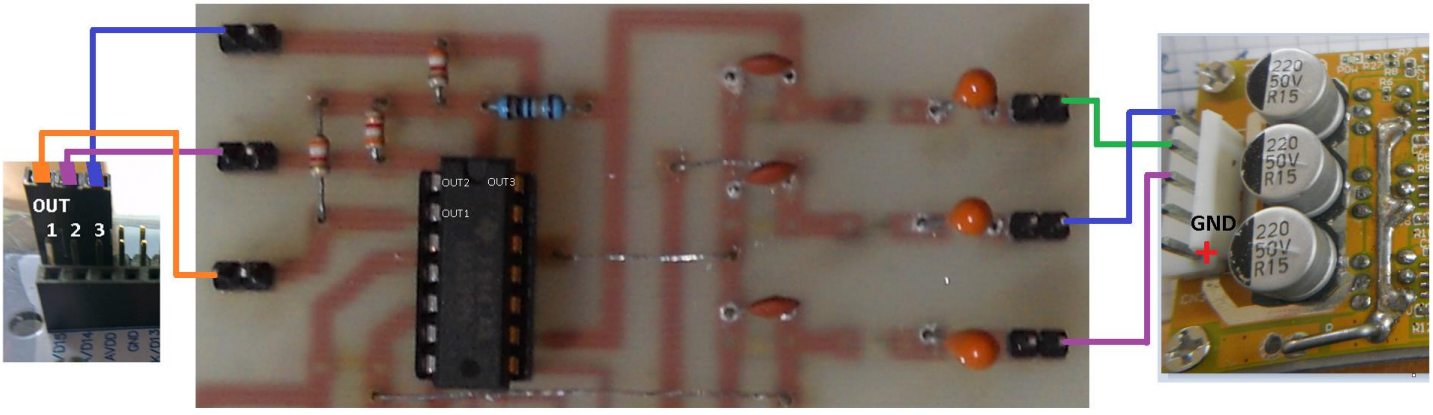


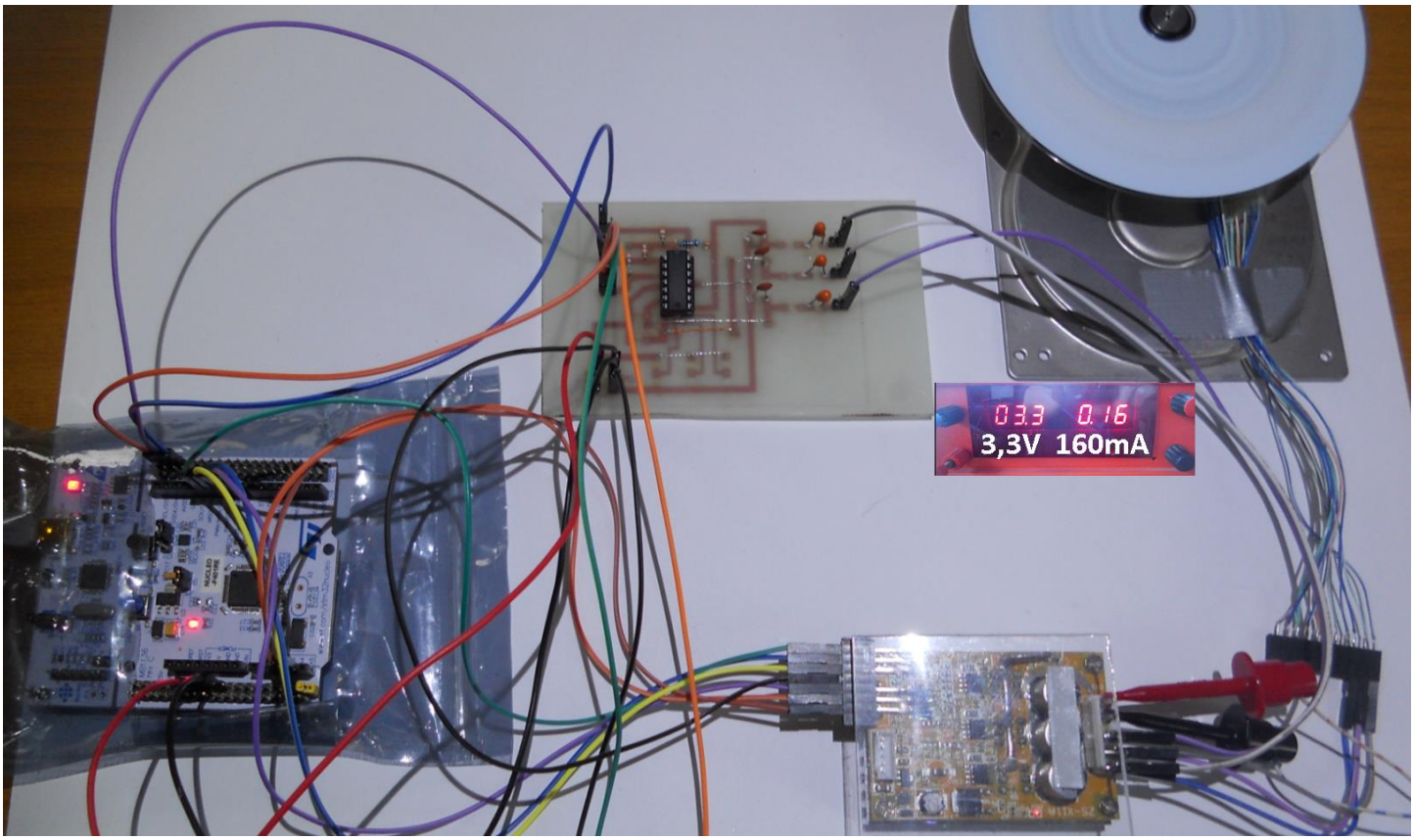



3,9kΩ
3
OUT
3,9kΩ
100
kΩ
OUT
2
3,9kΩ
1
OUT
100kΩ
100kΩ
100kΩ
100
kΩ
100kΩ
10kΩ
10kΩ
10kΩ
GND
56nF
1µF
10kΩ
10kΩ
3
IN
56nF
1µF
10kΩ
10kΩ
2
IN
56nF
10kΩ
1µF
10kΩ
1
IN




GND


OUT
1 2 3
OUT2
OUT3
OUT1
220
50V
R15
220
50V
R15
220
50V
R15
GND
+



If the CDROM-motor is running, he runs in the same silent way as controlled with HALL-sensors.

The start is a problem if a CD is present. The motor starts not as smooth as with HALL-sensors. If we switch on the motor without a CD, the motor starts immediately, but with any mass present, the motor needs a more or less hefty push.

The following change in C-code improves the start behavior a little bit:

```
uint16_t PWM_dn = 10000;
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin) {
  hall1 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_5);
  hall2 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_6);
  hall3 = HAL_GPIO_ReadPin(GPIOC, GPIO_PIN_8);
  new_step = 100*hall1 + 10*hall2 + hall3;
  if(start){
     switch (new_step){
       case 10://1,2
         TIM3->CCR2 = 0;
         TIM3->CCR3 = 0;
         GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
```

```
        GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
        TIM3->CCR1 = PWM_dn;
        GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
        break;
      case 1://3,1
        TIM3->CCR1 = 0;
        TIM3->CCR2 = 0;
        GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
        GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
        TIM3->CCR3 = PWM_dn;
        GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
        break;
      case 100://2,3
        TIM3->CCR1 = 0;
        TIM3->CCR3 = 0;
        GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
        GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
        TIM3->CCR2 = PWM_dn;
        GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
        if(PWM_dn > 8000) {PWM_dn -= 1;} else {start = !start; PWM_dn = 3000;}
        break;
      default: //because of some noise
        TIM3->CCR1 = 0;
        TIM3->CCR2 = 0;
        TIM3->CCR3 = 0;
      }
} else {
    switch (new_step){
      case 10://1,2
        TIM3->CCR2 = 0;
        TIM3->CCR3 = 0;
```

```
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR1 = PWM_dn;
  GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
  break;
 case 11://3,2
  TIM3->CCR1 = 0;
  TIM3->CCR2 = 0;
  GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR3 = PWM_dn;
  GPIOC->ODR = GPIOC->ODR | ((1<<sw2));
  break;
 case 1://3,1
  TIM3->CCR1 = 0;
  TIM3->CCR2 = 0;
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR3 = PWM_dn;
  GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
  break;
 case 101://2,1
  TIM3->CCR1 = 0;
  TIM3->CCR3 = 0;
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
  GPIOC->ODR = GPIOC->ODR & ~((1<<sw3));
  TIM3->CCR2 = PWM_dn;
  GPIOB->ODR = GPIOB->ODR | ((1<<sw1));
  break;
 case 100://2,3
  TIM3->CCR1 = 0;
```

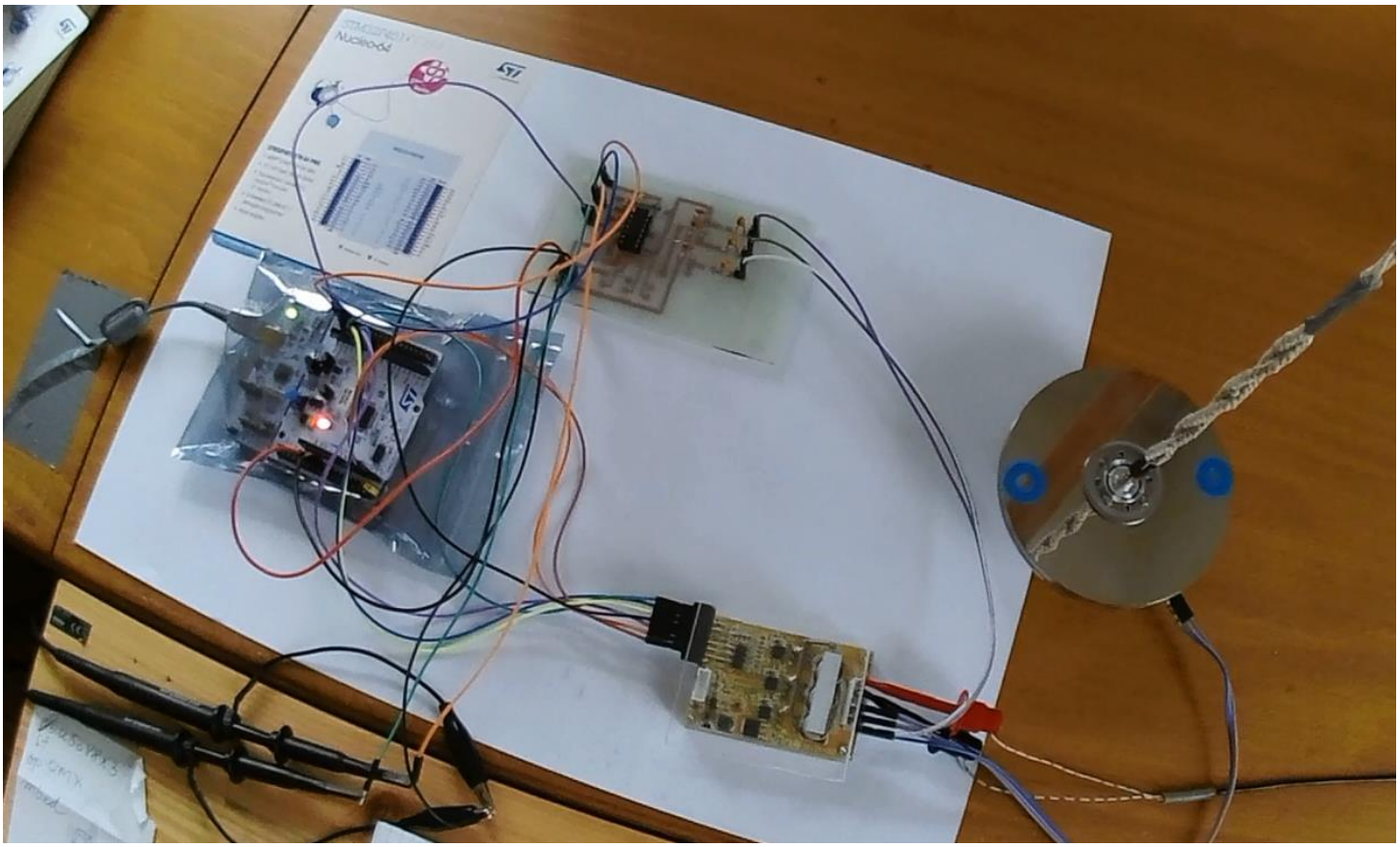
```
        TIM3->CCR3 = 0;
        GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
        GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
        TIM3->CCR2 = PWM_dn;
        GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
        break;
      case 110://1,3
        TIM3->CCR2 = 0;
        TIM3->CCR3 = 0;
        GPIOB->ODR = GPIOB->ODR & ~((1<<sw1));
        GPIOC->ODR = GPIOC->ODR & ~((1<<sw2));
        TIM3->CCR1 = PWM_dn;
        GPIOC->ODR = GPIOC->ODR | ((1<<sw3));
        if(PWM_dn < 10000) PWM_dn += 1;
      break;
      default: //because of some noise
        TIM3->CCR1 = 0;
        TIM3->CCR2 = 0;
        TIM3->CCR3 = 0;
      }
  }
}
```

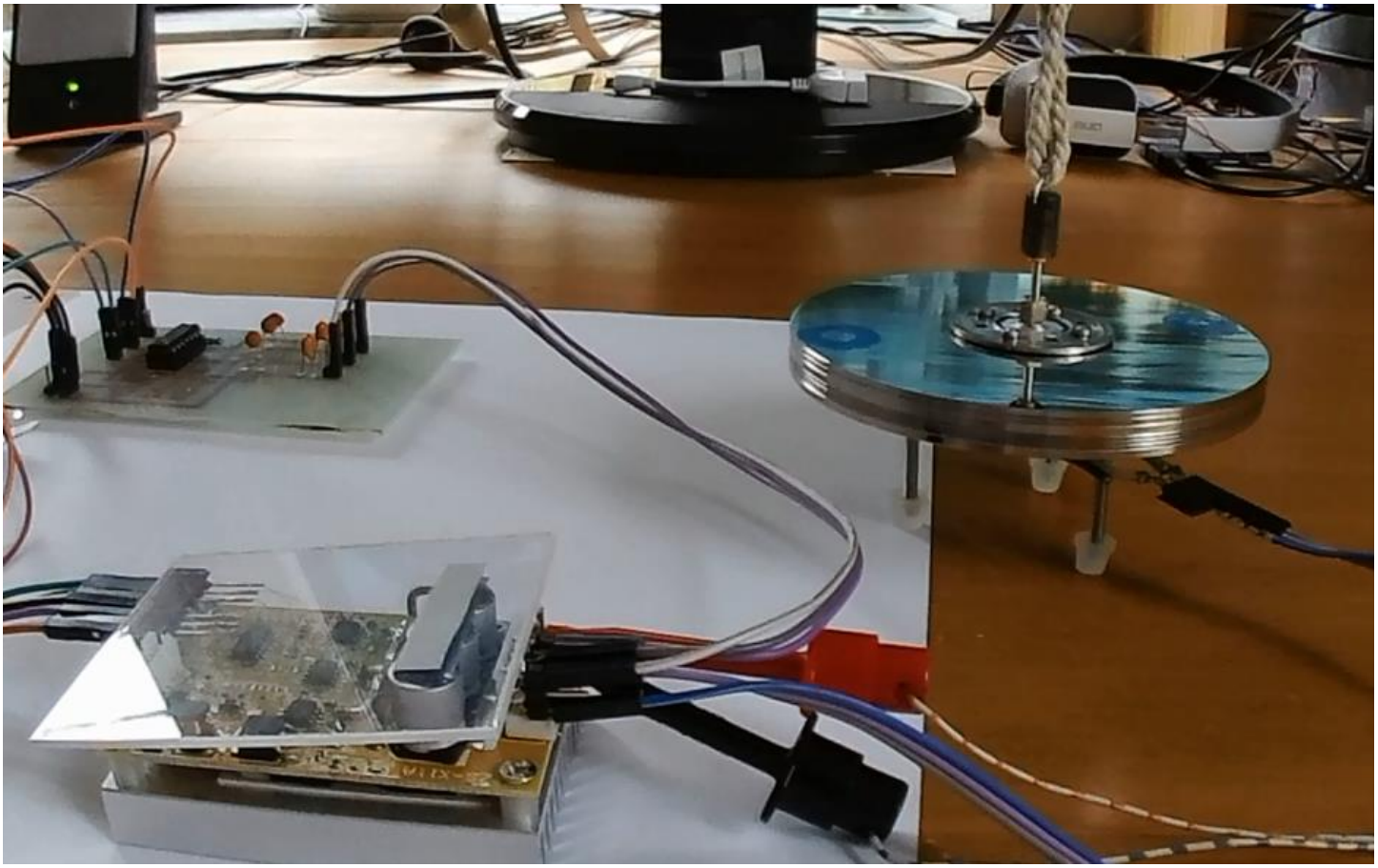
Some Applications

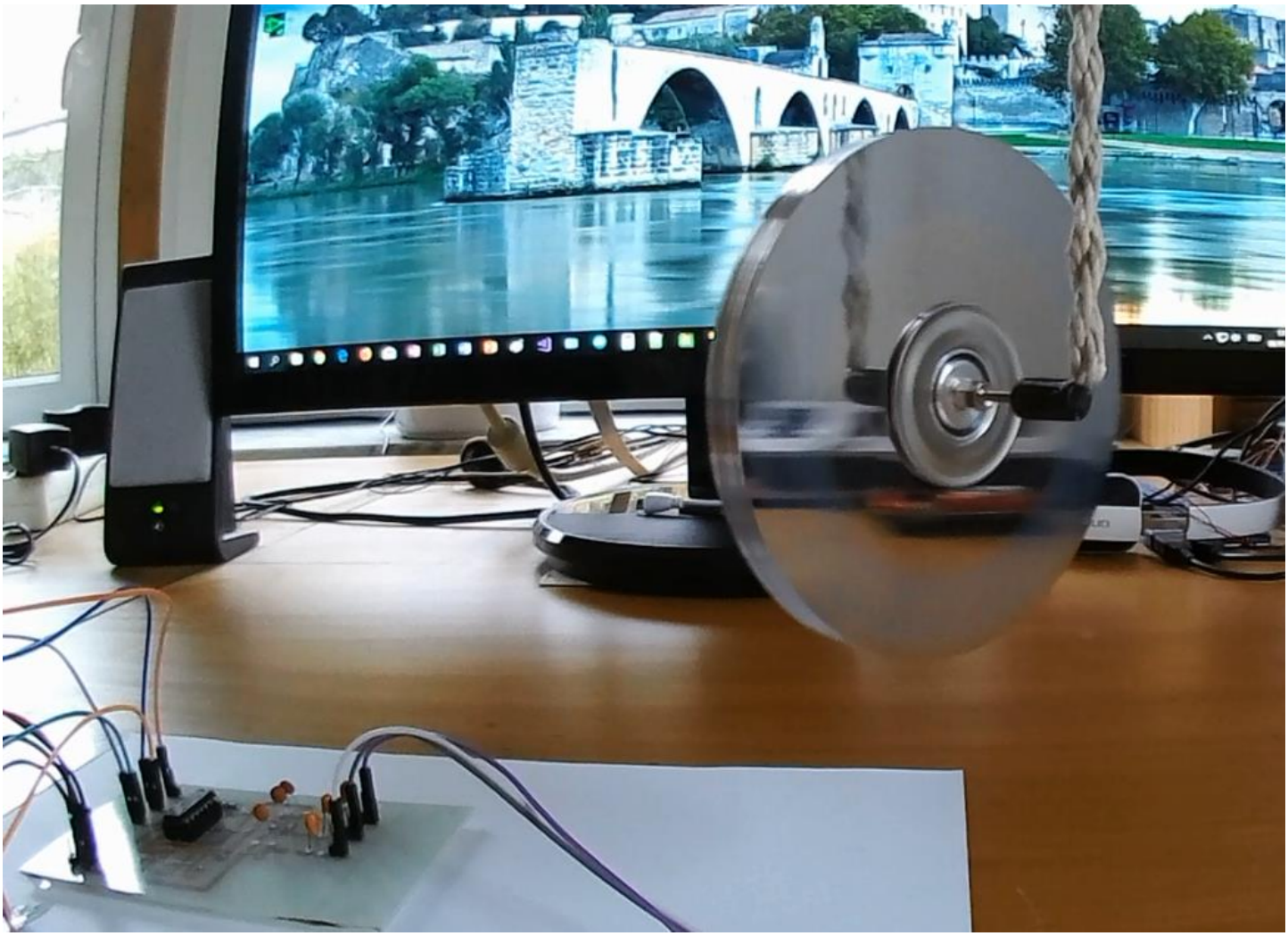
Since BEMF based on induction and induction needs motion we cannot expect, that the motor starts without any push.

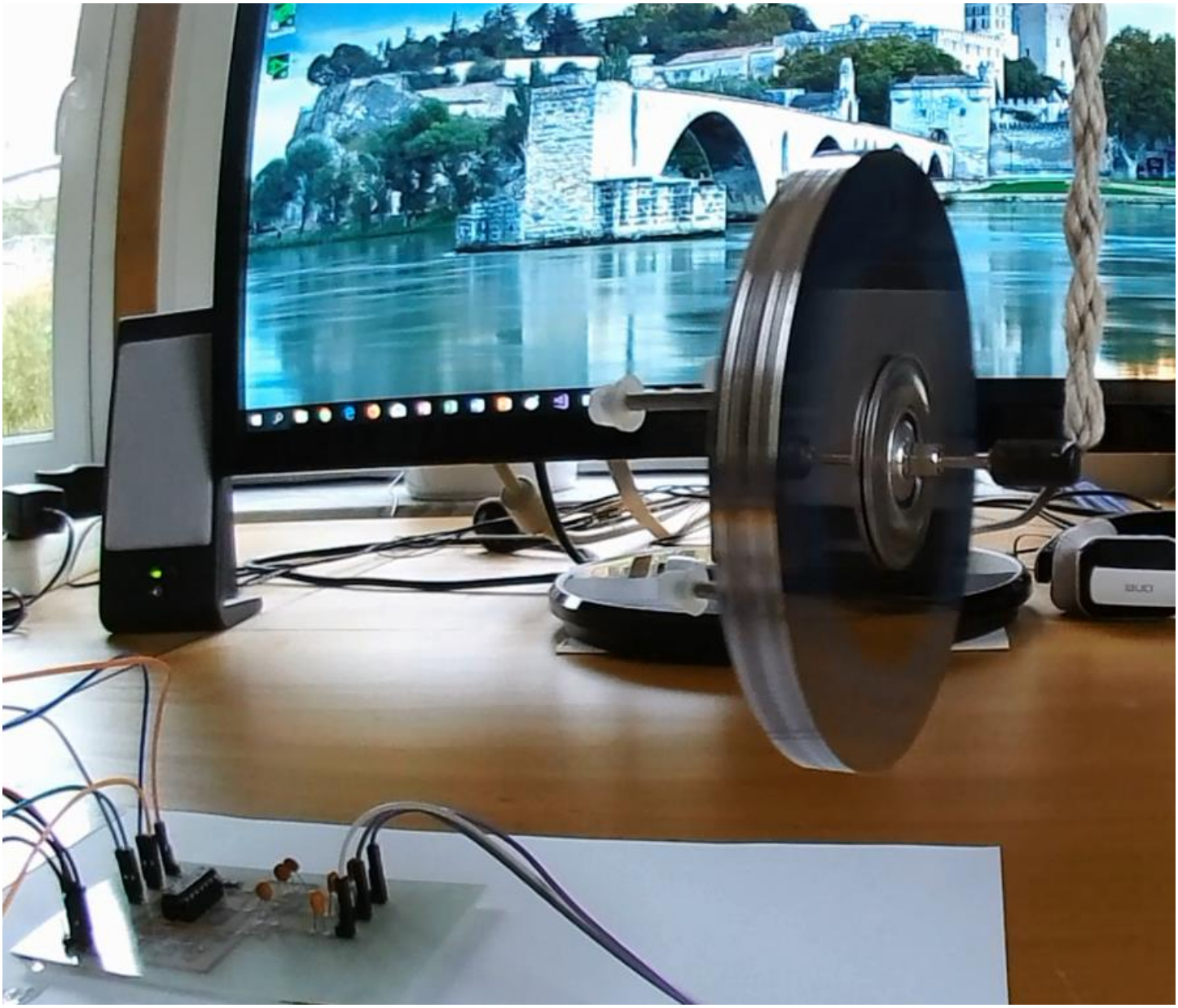
In the following we see some other motors we used together with the same program.

The first one is a hard drive motor and we used this device to conduct an experiment to show the power of an angular momentum.

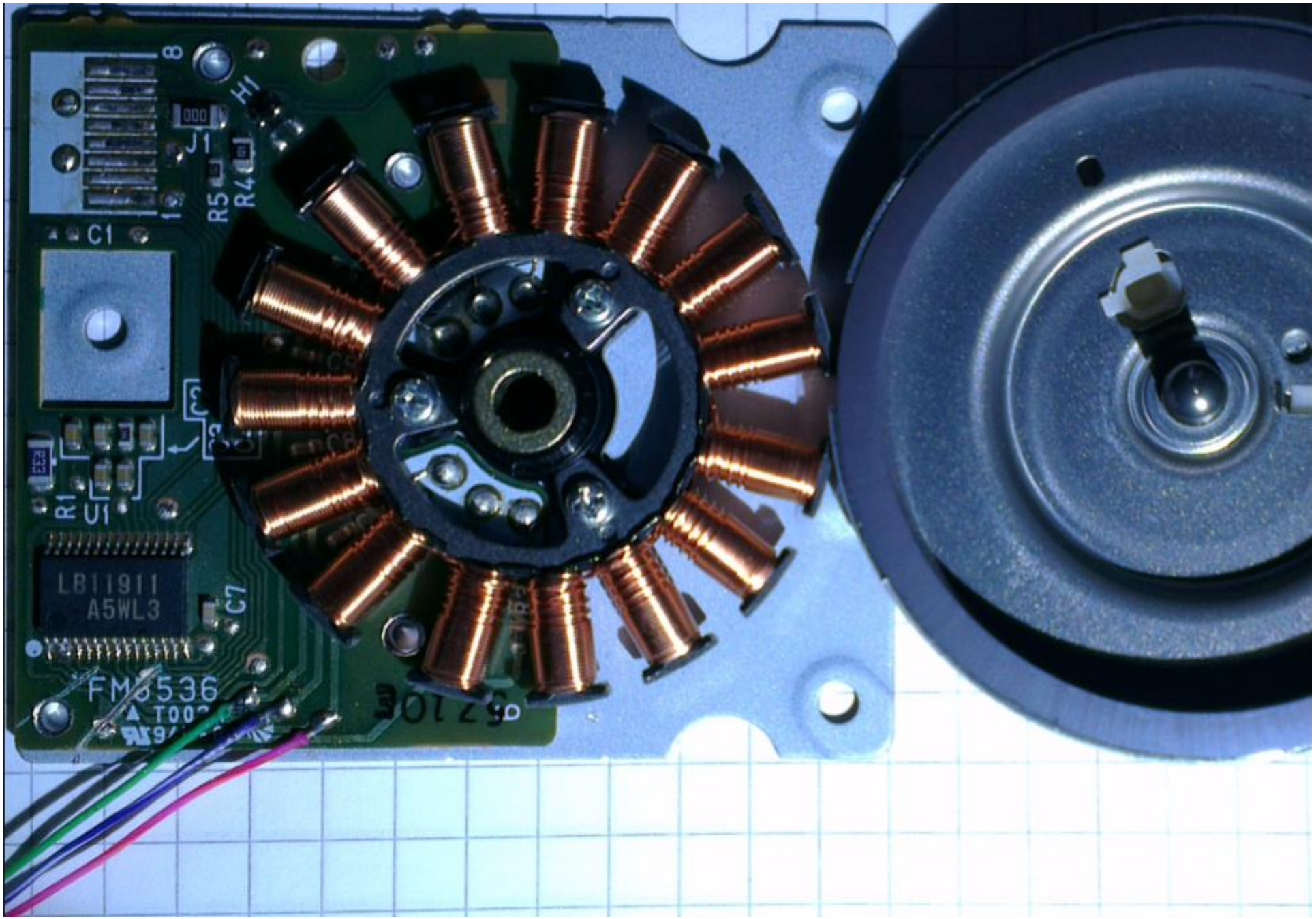


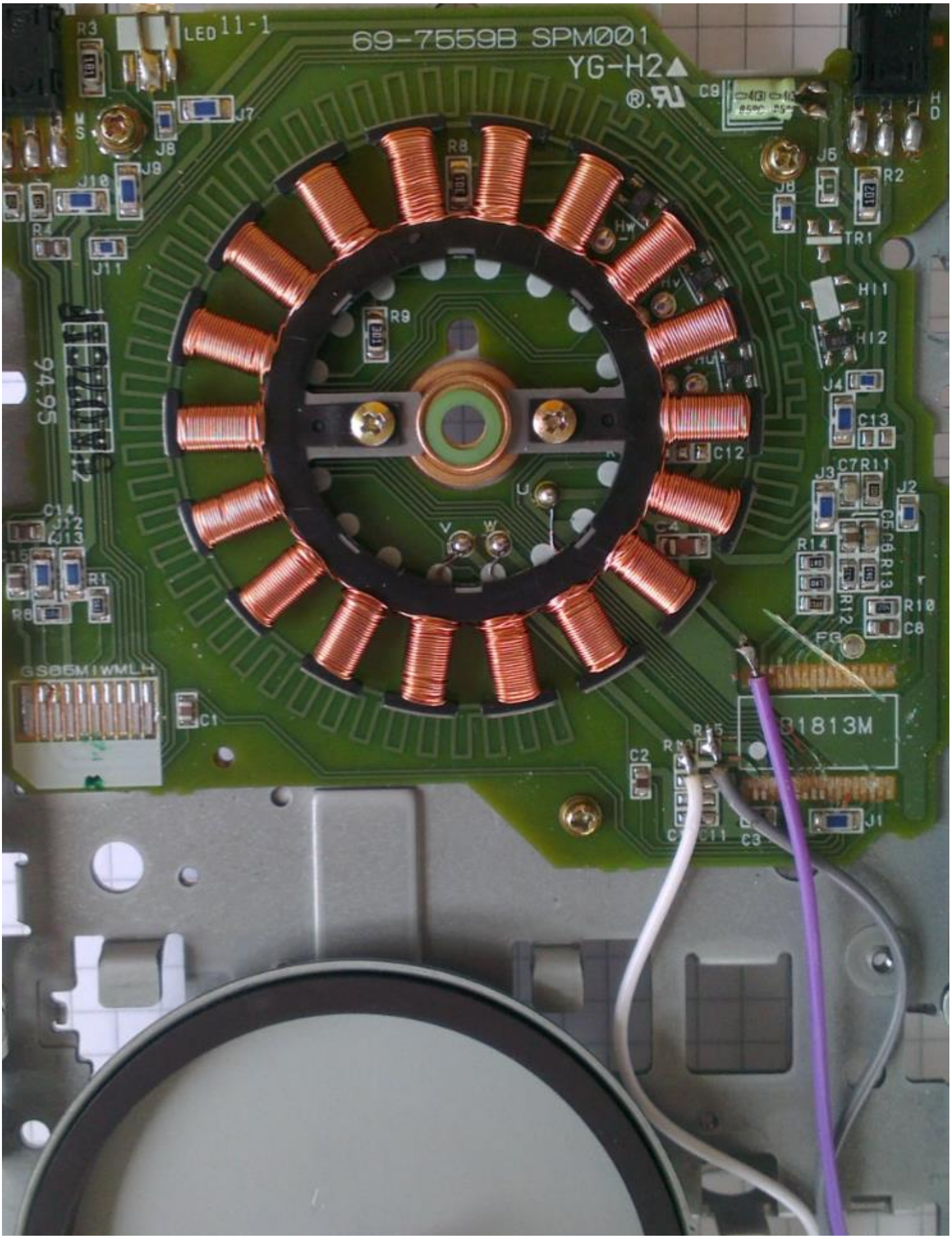






And here are two different floppy drive motors. A relatively small impulse starts both immediately.


C1
R5
R4
H1
R1
U1
LB11911
A5WL3
C7
FM6536


69-7559B SPM001
YG-H2
LED
R3
J7
J8
J9
J10
J11
R4
R8
R9
C9
J5
R2
TR1
HI1
HI2
J4
C13
C12
C7
R11
J3
J2
C5
C6
R13
R14
R12
R10
C8
FG
C14
J12
J13
R1
R6
C1
C2
C4
U
V
W
J1
C3
GS86M1WMLH
9495

… have fun with STM32!

edgarmarx@t-online.de