UNIVERSITY OF PADOVA

Embedded Real–Time Control

Laboratory report

Names - (Student ID Numbers)

Academic Year 2021-2022

Contents

1 General notes

This report was made as part of evaluation of course Embedded Real-Time Control at UNIPD. In labs we were using and all development was done on device named TurtleBot (see picture 1). As software development tool was used STM32 CUBE IDE.

We are using microcontroller from STM32F7 series, part of STMicroelectronics' STM32 family, is based on the ARM Cortex-M7 core, operating at frequencies up to 216 MHz. It is designed for high-performance embedded applications that require substantial processing power combined with real-time capabilities. The STM32F7 features up to 2 MB of dual-bank Flash memory, up to 512 KB of SRAM (with additional TCM memory), and includes an integrated Floating Point Unit (FPU) and Digital Signal Processing (DSP) instructions for efficient signal processing tasks. Key peripherals include high-speed USB OTG (with support for full-speed and high-speed modes), multiple 12-bit ADCs and DACs, timers with encoder and PWM modes, as well as extensive connectivity options such as UART, SPI, I^{2}C, CAN, SDIO, and Ethernet MAC with IEEE 1588 support. The series also includes an L1 cache (I-cache and D-cache), Chrom-ART Accelerator for graphical applications, and flexible memory interfaces for external SDRAM, SRAM, or NOR/NAND Flash. These features make the STM32F7 well-suited for applications involving advanced user interfaces, audio processing, motor control, and industrial automation.

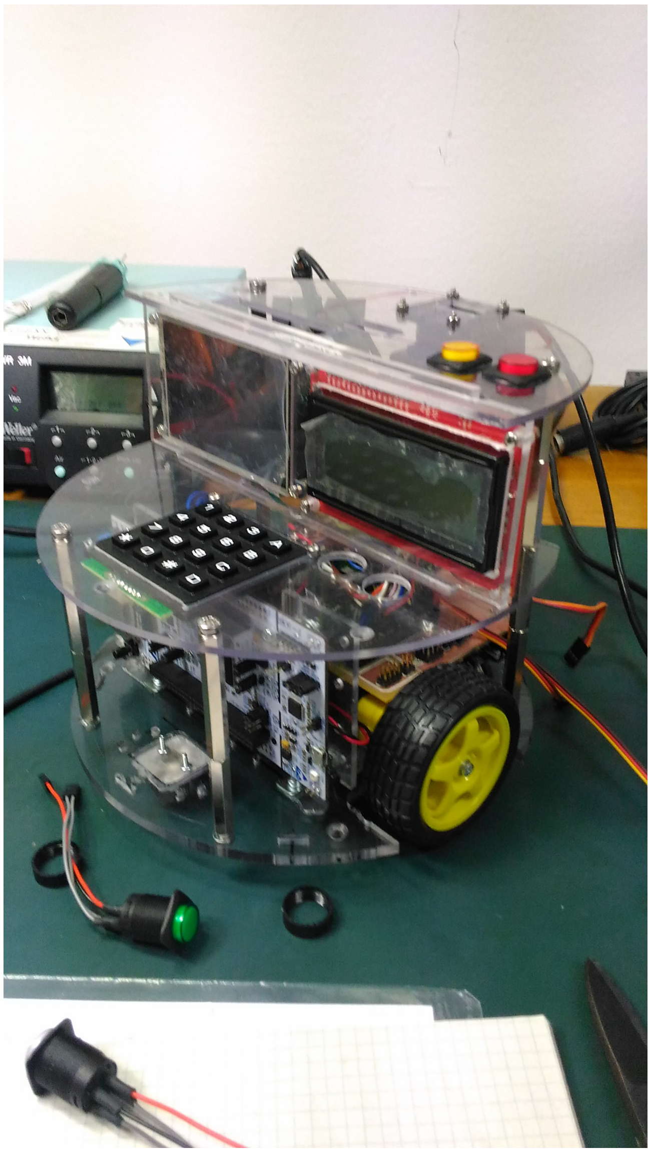
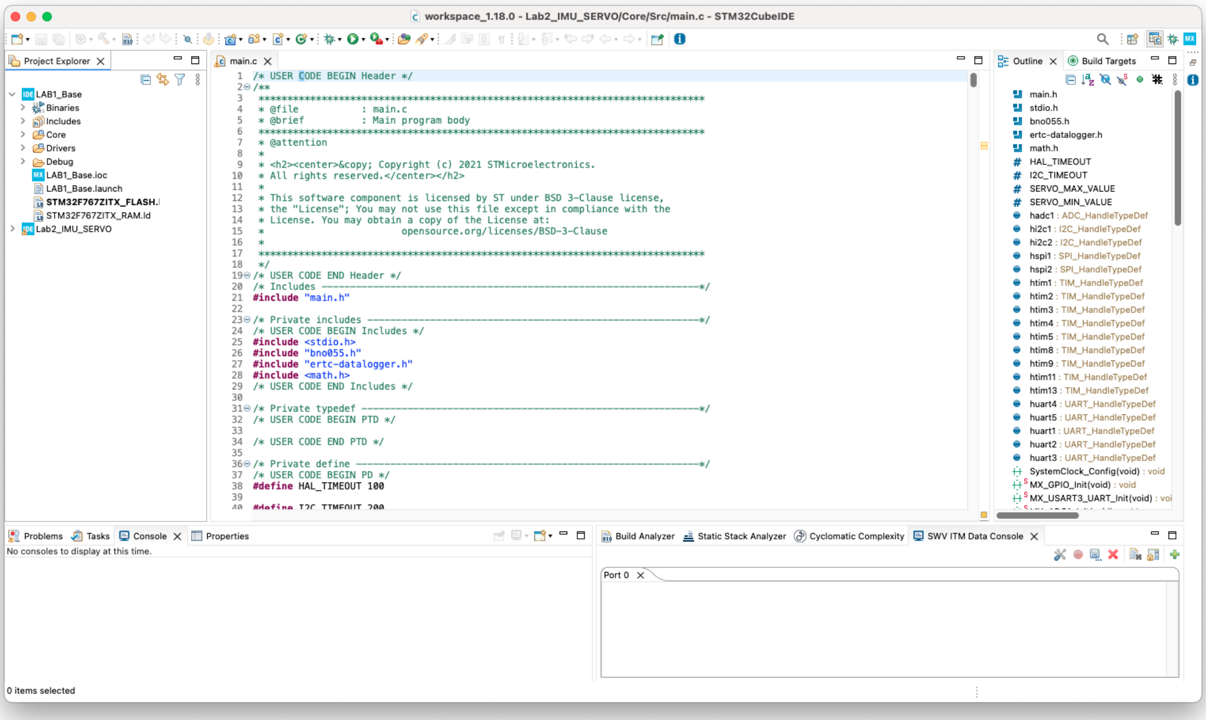


Figure 1: The TurtleBot



```
/* USER CODE BEGIN Header */
/**
 ******************************************************************************
 * @file           : main.c
 * @brief          : Main program body
 ******************************************************************************
 * @attention
 *
 * <h2><center>&copy; Copyright (c) 2021 STMicroelectronics.
 * All rights reserved.</center></h2>
 *
 * This software component is licensed by ST under BSD 3-Clause license,
 * the "License"; You may not use this file except in compliance with the
 * License. You may obtain a copy of the License at:
 *                        opensource.org/licenses/BSD-3-Clause
 *
 ******************************************************************************
 */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */
#include <stdio.h>
#include "bno055.h"
#include "ertc-datalogger.h"
#include <math.h>
/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define HAL_TIMEOUT 100

#define I2C_TIMEOUT 200
```

```
main.h
stdio.h
bno055.h
ertc-datalogger.h
math.h
HAL_TIMEOUT
I2C_TIMEOUT
SERVO_MAX_VALUE
SERVO_MIN_VALUE
hadc1 : ADC_HandleTypeDef
hi2c1 : I2C_HandleTypeDef
hi2c2 : I2C_HandleTypeDef
hspi1 : SPI_HandleTypeDef
hspi2 : SPI_HandleTypeDef
htim1 : TIM_HandleTypeDef
htim2 : TIM_HandleTypeDef
htim3 : TIM_HandleTypeDef
htim4 : TIM_HandleTypeDef
htim5 : TIM_HandleTypeDef
htim8 : TIM_HandleTypeDef
htim9 : TIM_HandleTypeDef
htim11 : TIM_HandleTypeDef
htim13 : TIM_HandleTypeDef
huart4 : UART_HandleTypeDef
huart5 : UART_HandleTypeDef
huart1 : UART_HandleTypeDef
huart2 : UART_HandleTypeDef
huart3 : UART_HandleTypeDef
SystemClock_Config(void) : void
MX_GPIO_Init(void) : void
MX_USART3_UART_Init(void) : voi
```

```
No consoles to display at this time.
0 items selected
```

Figure 2: example from STM32 cube IDE

2 Laboratory 1

Target of the first lab was to establish I2C communication between mcu STM32 and I/O expander SX1509. And to read the data from the keypad and the line sensor with help of this expander. See configuration on picture 3.

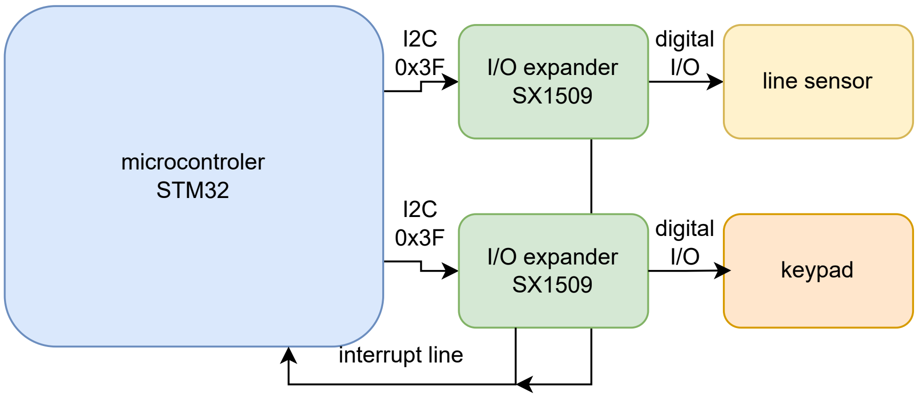


Figure 3: The TurtleBot

2.1 SX1509 I/O expander

We are using 16-ch digital I/O expander SX1509 [1]. Device has two I/O banks, represented by two registers in device memory of address 0x27 and 0x28. The I2C address of device is configurable by 2 address pins. Up to 4 devices could be at one line. We are using addresses 0x3E and 0x3F since we have 2 devices. Chip has built in debouncing, keypad engine and LED driver. See scheme on the pictre 4

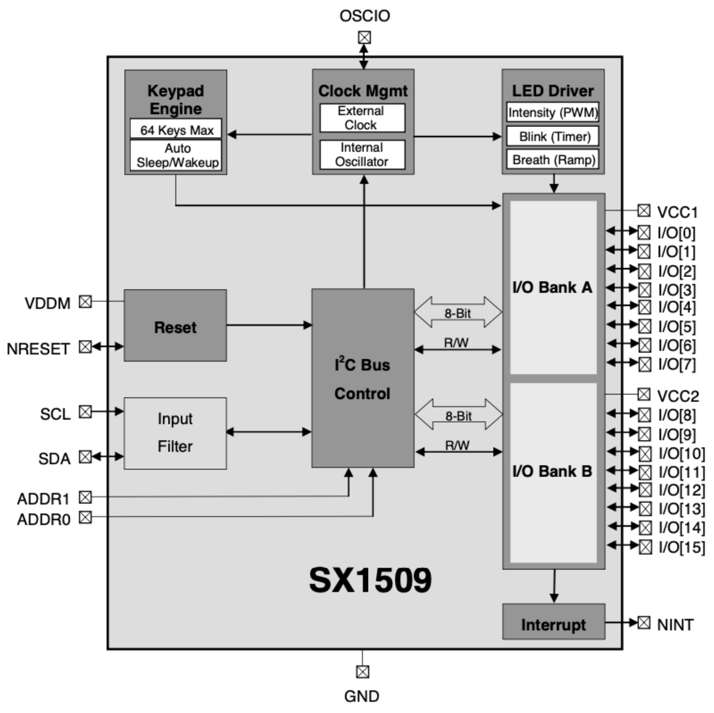


Figure 4: SX1509 block scheme

2.2 Line sensor

TurtleBot has Pololu QTR sensor[2] onboard, version with 8 channels and analog output. Each one of the outputs is connected to one of the digital inputs of SX1509. Internal logic of expander converts signal to digital one when is voltage higher/lower than treshold. Order of bits coresponds to physical state of the sensor. Principle of sensor is shown on picture 5.

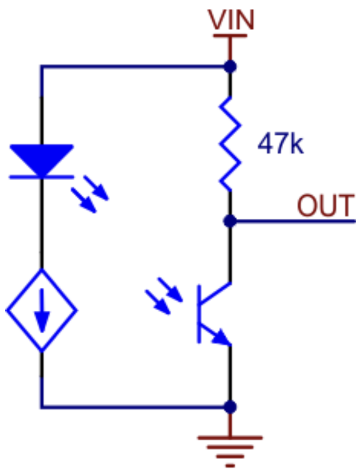


Figure 5: Scheme of single unit from line sensor

2.3 Keypad

Classical 4x4 matrix keypad is used. Digital signal is connected to I/O expander. Rows are connected to the one bank (0x28) and colums (0x27) to the other one, so four low bits of represents the keys. See keyboard configuration on picture 6.

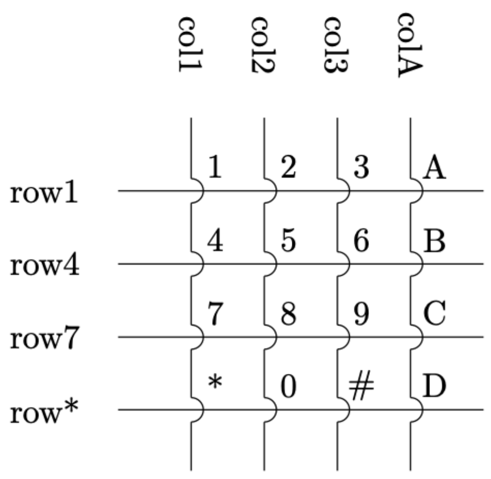


Figure 6: Scheme of keypad connection

2.4 Laboratory work

Exercise 1 & Exercise 2: Write an ISR that recognizes and correctly handles the keypad interrupts. You have to properly implement the void HAL_GPIO_EXTI_Callback(uint16_t pin) function. Then print which interrupt has been triggered,i.e., the pin. To be able to receive another interrupt from the keypad, have to read the registers REG_KEY_DATA_1 and REG_KEY_DATA_2 inside the ISR.

Extend the code of exercise 1 to handle the keypad interrupt. You have to print which keypad button has been pressed.

We have implemented one callback function, for reading keypad data only when the button is pressed. We are returning 2 global variables. One for interupt pin number and one as flag, so we can execute appropriate part of code in main. Then we calling function to read data from row and colums through I2C. See listing 1.

Listing 1: Keypad callback function

```c
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
    IRQ_GPIO_pin = GPIO_Pin;
    IRQ_GPIO_flag = 1;
    HAL_I2C_Mem_Read(&hi2c1, I2C_KEYPAD_ADDRESS, KEYPAD_ROWS_ADDRESS,
    8, key_rows_data, 1, I2C_TIMEOUT);
    HAL_I2C_Mem_Read(&hi2c1, I2C_KEYPAD_ADDRESS, KEYPAD_COLS_ADDRESS,
    8, key_cols_data, 1, I2C_TIMEOUT);
}
```

To get character from the keypad, we need to decode the data we received from the expander. For that purpose we have two functions, you can see in listing 2.

Listing 2: Keypad get cahr functions

```c
uint8_t GetKeypadPosition(uint8_t data)
{
    if(data == 254)
        return 0;
    else if(data == 253)
        return 1;
    else if(data == 251)
        return 2;
    else
        return 3;
}


```

```
char GetCharFromKeypad(uint8_t const rows, uint8_t const cols)
{
    return keypadLayout[rows][cols];
}
```

Exercise 3: Write a routine that reads the status of the line sensor and prints it. The routine must check the status with a polling period of 100ms.

We implemented function for reading data form the light sensor (see 4), and we calling this function by polling in main every 100 ms. We basically just read device registers trough I2C.

Listing 3: Keypad get cahr functions

```
void GetLightSensorStatus(uint8_t * data)
{
HAL_I2C_Mem_Read(&hi2c1, I2C_LIGHT_SENSOR_ADDRESS, LINE_SENSOR_ADDRESS,
1, data, 1, I2C_TIMEOUT);
}
```

Exercise 4: We haven't saved LAB0, since it is not part final evaluation, so we can extend it, but since our LAB0 was working properly, it is basically just changing LED delay based on keypad input. That means making a function that returns delay value based on input combination.

2.5 Conclusion

In this lab we establish I2C communication with two devices and successfully read data from these. We implemented keypad decoder, successfully read data from line sensor and haven't finished exercise 4, because we haven't saved lab0 on flash disk and were not able to login to school comuputer for 20 minutes. At least we make theoretical description of the problem, since it is quite trivial. Some of the functions are short so it would make sense to make them inline or not using function at all. We are not aiming for the most effective code and we prefer readability. Our code was not compiling with inline versions so we are using regular functions.

3 Laboratory 2

The aim of this laboratory was to implement an open-loop camera stabilization system on the TurtleBot using data from its IMU and controlling two servo motors to adjust the tilt and the pan of the camera.

3.1 IMU Data Handling

The TurtleBot is equipped with a 6-DOF IMU, comprised of a 3-axis accelerometer and 3-axis gyroscope. The accelerometer measures the linear acceleration $[m/s^2]$ along the 3 axes, while the gyroscope measures angular velocity(rad/s) around these axes. The data from the two sensors is available within `bno055_convert_double_accel_xyz_msq()` and `bno055_convert_double_gyro_xyz_rps()` respectively. The data provided by the IMU is crucial when estimating the movement and orientation of the robot, particularly for calculating the tilt as follows:

$$\theta = sin^{-1}(\frac{a_y}{g}) \tag{1}$$

where a_y is the acceleration along the y axis.

3.2 Servo Control

The two servo motors controlling the movement are operated via PWM signals generated by TIM6 of the STM32 and the angles are updated with a frequency of 100Hz using the `__HAL_TIM_PeriodElapsedCallback` function and scaled into a safe duty cycle with the `saturate()` function.

Listing 4: Angle update function

```
TIM_HandleTypeDef * timer6_ptr = &htim6;
HAL_TIM_PeriodElapsedCallback(timer6_ptr)
{
theta = asin(d_accel_xyz.y/G);
pan = (int8_t)(-(theta/3.14)*180) + PAN_BIAS;

angle = angle + ((-(d_gyro_xyz.z*0.1)/3.14)*180.)/10.;

tilt = (int8_t)angle;
}
```

As can be seen in the code, there is a strange phenomenon where dividing the angle update formula by 10 ensures a smooth pan without the need of a filter. While this makes no sense time/frequency-wise (after having done the calculations multiple times), it does in the end provide a very smooth movement.

3.3 Data Logging

Data logging was performed via the UART interface of the STM32, while using MATLAB to receive and visualize the data in real time with the `serial_datalog()` function.

3.4 Results

Lab video demonstration::Video Demonstration

The camera stabilization performance was captured and logged while the TurtleBot was tilted manually. As seen in the video, the servo effectively counteracts the tilt to maintain camera alignment, without the need of a complementary filter. In the event of a less fortunate camera setup, the complementary filter would have had the following form:

$$\theta_{fused} = \alpha \cdot \theta_{gyro} + (1-\alpha)\theta_{accel} \tag{2}$$

,where θ_{gyro} and θ_{accel} are two different sensor measurements, from the gyroscope and accelerometer, respectively, thus ensuring a smoother pan.

4 Laboratory 3

4.1 Design process

We implemented the motor controller relatively quickly and tested it. Initially, everything seemed to work fine, so we proceeded with final design adjustments. Unfortunately, we introduced a small change that caused the error to be computed with the wrong sign. This mistake cost us a significant amount of time, as we assumed the implementation was correct — and since it worked intermittently, it was even more misleading. We also implemented the jaw controller but struggled with tuning it properly. Eventually, we discovered the source of the issue. Unfortunately, we ran out of time during the lab session and were not able to tune the PID controller properly.

5 Laboratory 4

5.1 Design process

At first, we tried to implement a simple approach, but it did not work as we expected. The idea was to split the left and right side of the sensor into two separate arrays, giving us two values. Each value represented the occupancy of one half of the sensor. We then subtracted these values to get an error — ideally close to zero.

In hindsight, our implementation may not have been flawed, the real issue was likely the motor control implementation. At the time, however, we assumed the problem was in our logic. We implemented the jaw controller, without major issues. It worked as expected.

A Section in the Appendix

A.1 Subsection in the Appendix

Additional relevant information...

References

[1] SX1507/SX1508/SX1509. 2025. *Sparkfun* [online]. [accessed 2025-4-22]. Available at: https://cdn.sparkfun.com/datasheets/BreakoutBoards/sx1509.pdf

[2] Pololu line sensor. 2025. *Pololu.com* [online]. [accessed 2025-4-22]. Available at: https://www.pololu.com/docs/pdf/0J12/QTR-8x.pdf