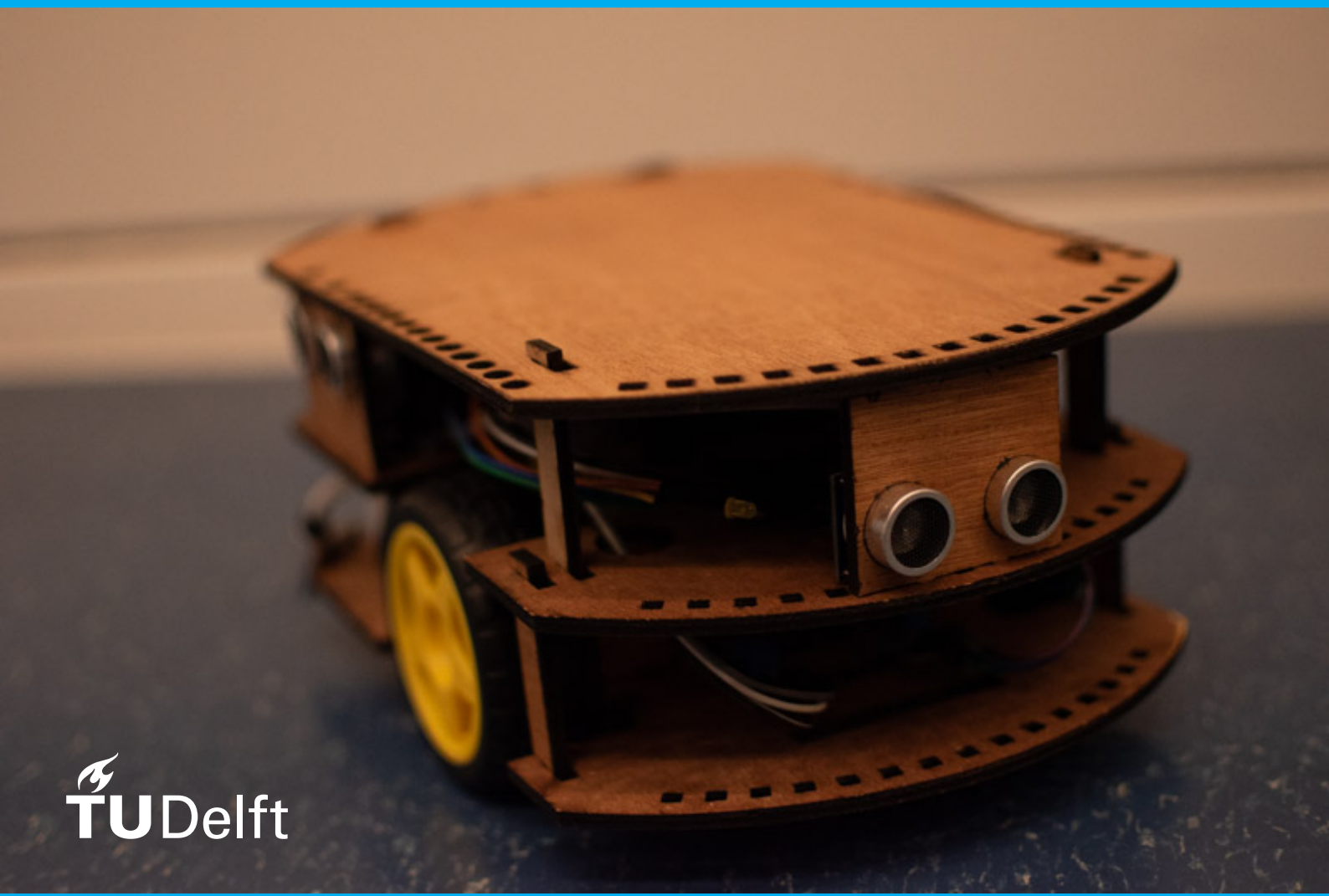
CSE2425 Embedded Software

Lab Manual 2025-2026

Q. Wang
K.G. Langendoen
V.D. van de Beek
W. Narchi
A. Kiste
M. Hadjigeorgiou
M.C. Negoiţescu
A.E. Celen



TU Delft

Preface

When redesigning the Embedded Software course practicals, we realized that the previous robot set up for the assignment was not entirely suitable. In the past years, the students spent lots of time programming on laptops/PCs to detect the lines from the pictures taken by their smartphones. The laptops/PCs then sent the information of the detected lines to the robot through Bluetooth. Based on this information, the robots performed proper actions to follow the lines. There were two motivations for revamping the assignment: 1) Programming and running the codes on laptops/PCs is not a learning objective of this course, and 2) the latency on sending the line detection information from the laptop/PC to the robot is too long. Four years ago, we moved to a setup where students could do regular embedded programming in a microcontroller. Furthermore, following the suggestion given by Prof. Martijn Wisse and Martin Klomp of 3ME, we decided to give students the raw components of the robot. The students assembled the car robot by themselves. This year, we have also designed a PCB board (by Arend-Jan van Hilten) and purchased some new hardware for this assignment that could let the students focus more on the embedded software design instead of spending too much time on debugging the hardware. Based on the validations in the past years, the students should be able to complete the assignments at the final demonstration sessions.

Qing Wang and Koen Langendoen
Delft, December 2024

Note to reader

In this manual, there will be a series of text boxes with different icons. Although the information in these boxes might not be required for the lab, they contain useful information.

Be Aware!
This type of box warns for frequently occurring mistakes and problems.

Hint
This type of box contains a hint that might be useful when solving the exercise.

Background Information
This type of box gives more information about the technologies that are used in this exercise.

Contents

1 Introduction

The purpose of this lab assignment is to become familiar with programming in an Embedded Systems environment, which comes with certain limitations. Your main tasks are to program a car robot to detect & avoid obstacles, follow lines, and sprint. You will program the Raspberry Pico H [6] board, supported by VSCode (cf. Section 2 for details). Programming Pico can be done using the RP2040 [8] implementation of the Arduino Platform, C/C++ Hardware Support Library or the C/C++ Hardware Struct Library from the SDK [5].

Be Aware!
Using Micropython is **not allowed** for any part of this lab

1.1. Hardware

The embedded platform used in the lab is RP2040 [7], which is a microcontroller that contains dual ARM 32-bit Cortex-M0+ cores. The full list of hardware that you will use to assemble the robot is as follows:

- 1x Raspberry Pi Pico H board (containing the RP2040 controller) [6, 7]
- 1x Custom PCB breakout board
- 2x HC-SR04 ultrasound ranging sensor: To detect obstacles [4]
- 1x L9110 H-Bridge: For power distribution to the motors [3]
- 2x TCRT5000 Infrared reflective sensor: To detect lines [9][1]
- 2x B83609 speed encoder: Measure the rotational speed of the robot [1]
- 1x OLED display [2]
- 1x Arduino ADKeyboard [optional]
- 1x ST-Link [optional; with 4 Female-to-Female jumper wires]
- 1x USB cable
- 1x USB power splitter
- 2x Red LEDs
- 1x 5000mAh Mi Power bank

[1] The infrared sensor's sensitivity can be adjusted by turning the screw-shaped knob with a Phillips head screwdriver; see the assembly instructions on Brightspace (page 7).

- 10x Male-to-Female Jumper wires
- 4x 4-pin Female-to-Female wires
- 1x 3-pin Female-to-Female wires
- 2x Motors with gearbox
- 2x Wheels
- 1x Caster wheel
- Wooden frames
- 1x Bag of 10 screws and 10 nuts

1.2. Timeline

This lab consists of several assignments as indicated by table 1.1. We recommend following the timeline outlined below, though this is only a recommendation and is not strictly necessary.

Table 1.1: A potential timeline for the robot lab.

Week	Assignment
4-5	Robot component pick up, assembly & testing
6	'Variable speed' & 'Obstacle detection and stopping'
7	'Line follower' & 'Obstacle avoidance'
8	'Fixed-distance traveling'
9	Demo / grading

1.3. Hardware Check

In the weeks between the hardware handout and the winter vacation, students are required to do a hardware check with a TA during one of the labs. This check consists of assembling the robot, showing a simple variant of 3.1, and visual inspection of the components. This is not graded, and no bonus will be handed out. It may be possible to do the check line. The details of the online option will be given before the start of the vacation.

1.4. Deliverables

In addition to the grading demo, there are a number of deliverables that you must provide in order to pass the lab. These should be submitted on Weblab.

1. **Report:** A short report (recommended length of 3-4 A4 pages, though it may be longer/shorter) detailing your solution for each of the completed sub-assignments. Please specify the logic of your solutions in addition to the framework (Arduino, Hardware Structs, Hardware Library, etc) that you used. More concisely, you should at least have sections for each of the following (assuming you have implemented the corresponding functionality), but you may include additional sections if needed:
   (a) Description of the obstacle detection algorithm
   (b) Description of the line detection algorithm
   (c) Description of the obstacle avoidance algorithm
   (d) Description of the algorithm on calculating the traveling distance
2. **Code:** The complete source code of your solutions. Each code file should have a header with the format specified in listing 1.1. The source files should be delivered as a ZIP file. If you have used the Arduino IDE the sketches should be called `a_assignment_b.ino`, where *a* and *b* are your student number and the assignment number, respectively.

Listing 1.1: The header for all source files

```
/**
 * Student name: [YOUR NAME]
 * Student number: [YOUR STUDENT NUMBER]
 *
 */
```

1.5. Passing Criteria

The criteria are detailed in table 1.2. For passing the lab, you will have to get at least 5 points, for example, by finishing all tasks using the Arduino library. You will receive a higher grade if you perform tasks with advanced techniques such as timer registers or finish the last task with higher accuracy.

Table 1.2: Criteria

Points / **Criteria**	**2.5 pt**	**2 pt**	**1.5 pt**	**1 pt**
Variable speed	n/a	n/a	Timer Registers	Basic (analogWrite; *can use Arduino library*)
Obstacle detection and stopping	Programmable IO	ISR Interrupt	Using Arduino Library interrupts	Basic (busy loop with digitalWrite and -Read; *can use Arduino library*)
Line follower	n/a	Registers (ADC)	n/a	Basic (analogRead; *can use Arduino library*)
Obstacle avoidance	Multicore (*use registers*)	Reorient and follow line after avoidance (*use registers*)	Reorient and follow line after avoidance (*can use Arduino library*) **OR** Obstacle successfully avoided only (*use registers*)	Obstacle successfully avoided only (*can use Arduino library*)
Fixed-distance traveling	n/a	Error ≤ 10 cm (*use registers*)	Error ≤ 10 cm (*can use Arduino library*) **OR** 30 cm ≥ Error > 10 cm (*use registers*)	30 cm ≥ Error > 10 cm (*can use Arduino library*)

What we mean with registers
With registers we mean programming using memory constants (CMSIS). Other libraries other than those mentioned can be used by special request.

2

Setup

2.1. Assembling the robot

In order to start working on the assignments, you first need to assemble the robot. **Assembly instructions can be found on the Brightspace page of this course (**https://brightspace.tudelft.nl/d2l/le/content/680733/viewContent/4055211/View **and** https://brightspace.tudelft.nl/d2l/le/content/680733/viewContent/3806716/View**). We are currently working on a new version of the instructions. This video might help you greatly with the robot assembly (**https://www.youtube.com/watch?v=AU0idiy4gbg**).**

2.2. Setting up a local environment

Using a Virtual Machine
If you choose to avoid setting up the local software or have hardware restrictions like an arm64 computer, it is possible to use a virtual machine. In that case, you can follow the instructions in section 2.3.

In order to start programming for the Raspberry Pi Pico H (RP2040) in this course, you may also set up a local environment on your own computer. We currently support three variants that you may use:

- Raspberry Pi Pico VS Code Extension (C/C++ SDK + CMSIS)
- Command-line (C/C++ SDK + CMSIS)
- Arduino IDE or Arduino Community Plugin for VS Code

It is advised to use VS Code (section 2.2.1), but instructions for the Arduino IDE are also provided (section 2.2.3).

2.2.1. Pico VS Code extension

The Raspberry Pi Foundation provides their own extension for VS Code targeted at their SDK. It provides templating, toolchain management and offline documentation for the Pico family of microcontrollers.

To use the extension, you must install VS Code, Python 3.9+, git, tar, and the GCC C/C++ compiler. It can then be downloaded from the Extensions Market Place as Raspberry Pi Pico published by the Raspberry Pi Foundation.

You can then create programs for the microcontoller using the C/C++ SDK. For Linux-based systems, you may need to install udev rules in order to upload without root, the easiest way to do so is to simply download picotool for your distribution. For the first upload you may need to put the microcontroller into flashing mode, this can be done by holding the `BOOTSEL` button while plugging the Pico into the computer. After uploading a program using the SDK, you can upload using USB without having to put the microcontroller into boot mode. You can double check this (or fix it), by making sure that your project's `CMakeLists.txt`

contains `pico_enable_stdio_usb([executable] 1)` and the main function calls `stdio_init_all()` in it's main.

More information about the SDK can be found in the Getting Started Guide

2.2.2. Pico Manual

It is also possible to use a manual command-line based workflow to program the Raspberry Pi. This involves installing picotool, the ARM toolchain and the PiSDK (or otherwise) onto your machine. These steps are documented in Appendix B and C of the Getting Started Guide. If the Pico does not show up as a USB device, it is also possible to use the `pico load [file].uf2` to flash the program and `pico reboot` to reboot into running mode.

Rootless picotool
Picotool can be used permissionless if you add two additional udev rules to your system into a new file, e.g. `/etc/udev/rules.d/99-picotool`. These are:
`SUBSYSTEM=="usb", ATTRSidVendor=="2e8a", ATTRSidProduct=="0003", MODE="660", GROUP="plugdev"`
and
`SUBSYSTEM=="usb", ATTRSidVendor=="2e8a", ATTRSidProduct=="000a", MODE="660", GROUP="plugdev"`
Your user should be a member of the `plugdev` group

2.2.3. Arduino IDE

There are two systems for programming in Arduino for the Raspberry Pi Pico that we will support in this course:

- Arduino IDE
- Arduino Community for VS Code

Instructions for setting up both environments will be roughly the same for VS code and the IDE with the only difference as to where you need to set certain settings. For this guide, we will assume that you are using the IDE since that has an easier UI to navigate.

To program the microcontroller, we use Earle F. Philhower III's [8] core rather than the mbed core that is available by default on the IDE. We do this since the mbed core does not support uploading programs without setting the microcontroller in BOOTSEL mode, which would require you to replug the Pico every time you want to upload your program. You can download the package by pressing on `File->Preferences` and adding `https://github.com/earlephilhower/arduino-pico/releases/download/global/package_rp2040_index.json` in the *Additional Boards Manager URLs* field. Press OK, go to `Tools->Boards->Board Manager` and add *Raspberry Pi Pico/RP2040*.

PlatformIO
This also why you can't use PlatformIO to program the Pico, since they don't support this Arduino core due to a licensing conflict with the Raspberry Pi foundation.

For the first upload, you must put the board in flashing mode by holding the `BOOTSEL` button while plugging in the board. You then can select *Raspberry Pi UF2*, upload a sketch, and replug the controller. From that point onward you can upload using the serial port which avoids having to putting into flashing modes.

Library OLED
Since the OLED screen is used as a debugging tool, you are allowed to use a library to program it. One library that has worked well in the past is Adafruit SSD1306. Make sure to wire the VCC pin to the 3.3v pin by the power out and not the 5V pin of the I2C connectors. You do still use the other pins of that connector

.

This allows you to print to serial, greatly easing the debugging process. You can access serial output via the Arduino IDE by going to Tools > Serial Monitor. The default `blink_182.ino` sketch provided in the virtual machine image contains an example of how to print to serial.

Permission and Arduino IDE
The UF2 support seems to depend on the system running an automated mounting mechanism like `udev2` and `gfvs`, otherwise it is not able to detect it. Your user should also be a member of `dialout` and `plugdev`. It is possible that you may need to install the udev rule: `SUBSYSTEMS=="usb", ATTRSidVendor=="2341", GROUP="plugdev", MODE="0666"` in `/etc/udev/rules.d/99-arduino`

An example of blinking the built-in LED is given in listing 2.1.

Listing 2.1: An example of blinking the LED

```
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BULTINT as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED_BUILTIN, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

More information about the Arduino core can be found on its documentation page.

2.3. Setting up a virtual machine environment

The necessary software to debug, upload and run your code on Pico consists of VS Code, the Pico Toolchain and the Arduino IDE. These programs have been set up in a **VM (Virtual Machine)** that runs Ubuntu 24.04. The VM image can be found on the Brightspace page of this course (https://drive.google.com/file/d/1c_Mx-r-wuuw2-sxwSlOlIn9DrGYI5-gA/view?usp=drive_link). To run it, you need a virtualization application (such as VirtualBox) and 20 GiB of free space.

After installing the VM, you will need to grant it access to the appropriate USB devices. If you are using VirtualBox, this can be done by right-clicking the appropriate VM in VirtualBox Manager, then navigating to Settings > USB, clicking the icon with the green plus, plugging the Pico and then selecting the device 'Raspberry Pi Pico'. You then want to replug the Pico while holding the BOOTSEL button and repeat the process, but then select 'Raspberry Pi Pi RP2 Boot'.

If you are using VirtualBox, it might be useful to set up a shared folder to transfer files between the VM and your computer[1]. You may encounter permission issues when trying to access the shared folder of your VM[2].

[1] https://carleton.ca/scs/tech-support/troubleshooting-guides/creating-a-shared-folder-in-virtualbox/
[2] https://stackoverflow.com/a/26981786/14247568

3

Assignments

You will need to use RP2040 registers for the assignments. Please refer to **Appendix A** for a short tutorial on how to blink an LED with registers in RP2040.

3.1. Assignment 1: Variable speed

In this assignment, the program you will write needs to control the motors using PWM signals and timers. The program should perform the following actions in sequence:

- Starts the robot and makes it run *(note: you will need to set the duty cycle of the PWM signal to above a certain threshold (say, above 50% in our test) to make the robot move)*.
- After three to four seconds of keeping the robot running at a constant speed, increase the duty cycle of the PWM signals to speed up the robot.
- After another three to four seconds, slow down the robot gradually and finally stop the robot.

In this assignment, you can either 1) place the robot on the floor, or 2) hold it with your hands in the air. In the latter case, the robot as a whole will not move, but the wheels will move. Both of these methods are accepted in the final demonstration of this assignment.

About PWM
Some timers of the Pico also control PWM output. Check the documentation and pin mapping to make sure that you don't introduce conflicts in your sketch.

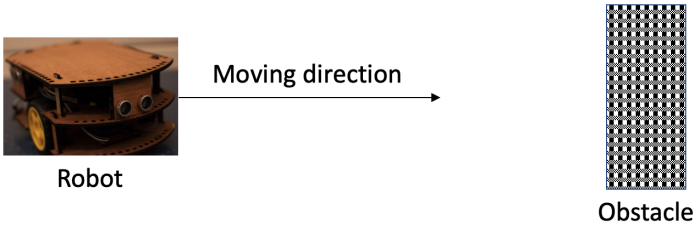
Motor Speeds
Since the course is using DC motors, it is not possible to get an exact speed. It is therefore likely that your implementation will have a slight drift. This will be accepted in the final demonstration.

3.2. Assignment 2: Obstacle detection and stopping

The robot is equipped with two HC-SR04 ultrasound ranging sensors: one placed in front, and the other placed on the right side of the robot. In this assignment, you will use the **front** ultrasound ranging sensor to detect an obstacle in front of the robot. You can use Programmable IO (2.5 pts), ISR interrupts (2 pts), Arduino library interrupts (1.5 pts), or an Arduino library busy loop (1 pt). Please refer to table 1.2 for detailed passing criteria.

The obstacle can be anything you find at home (such as a book or a box). For the final assessment, the obstacle used will be the IKEA box you received the robot hardware in. You can set up the testing environment as in fig. 3.1.

Figure 3.1: Setup for assignment 2



Your program should perform the following activities in sequence:

- Start the robot and make it run.
- Detect if there is an obstacle ahead using the front ultrasound range sensor.
- Stop the robot when it comes close to the obstacle (when the inter-robot-obstacle distance is less than 15 cm).

3.3. Assignment 3: Line follower

The robot has two TCRT5000 Infrared reflective sensors (including two infrared LEDs) attached to the bottom. The LEDs shines infrared light on the floor and the reflected light is detected by the sensors. Based on the reflected light received at the two sensors, a line follower program can be made.

For this assignment, the robot should follow a (black) line on the floor, including any curves. You should DIY (do-it-yourself) the lines, such as using printed papers with (black) lines or with black tape. For the assessment, black electrical tape will be used.

Line width
The width of the lines used during the final assessment will be approximately 20mm. You are expected to follow a 'keep-in' approach for following lines; the width of the lines is smaller than the distance between the two IR sensors on the bottom of the robot. As such, you should try to keep the line between the two IR sensors.

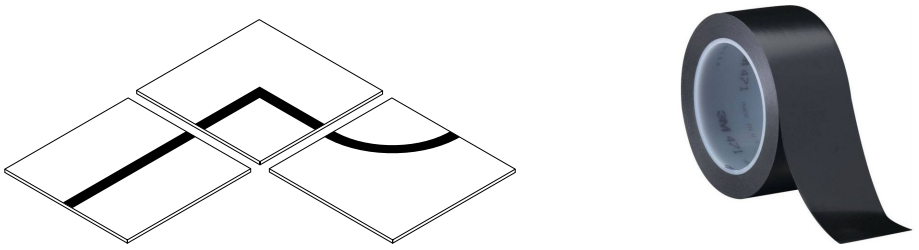


Figure 3.2: Setup for assignment 3: (left) example pieces of track that the robot should be able to navigate; (right) black tape that you might use to build the line on the floor

3.4. Assignment 4: Obstacle avoidance

In this assignment, you will program the robot to maneuver around an obstacle on the path. For the final assessment, the obstacle used will be the IKEA box you received the robot hardware in (similarly to assignment 2). This obstacle will always be in the middle of the path, i.e. there will be a black line segment before the object and a black line segment after the object.

More concretely, a program to solve this assignment should perform the following:

- Start the robot and make it run by following a straight line that you build (use the same method as assignment 3).

- Detect if there is an obstacle ahead using the front ultrasound ranging sensor
- Maneuver around the obstacle and reach the line on the other side of the obstacle
- **Optional**: Reorient the robot and continue to follow the line (bonus points, see table 1.2)

You will gain additional points for not utilising the Arduino library and doing manual register programming instead.

Side sensor
The robot components include a second HC-SR04 ultrasound sensor that is mounted on the side. Though not strictly necessary, it might be useful to utilise this sensor in order to determine when you can turn the robot to realign with the path.

3.5. Assignment 5: Fixed-distance traveling

The robot has two B83609 speed encoders that can be used to measure the robot's rotational speed. In this assignment, you should program the robot to run for a given distance, e.g. 1.5 meters. Your program should perform the following activities in sequence:

- Start the robot and make it run by following a straight line that you build (use the same method as Assignment 3).
- Stop the robot at the place where it is $X.Y$ meters away from the starting position.

You will gain additional points for not utilising the Arduino library and doing manual register programming instead. Furthermore, you will gain additional points if your fixed distance traveling has an error of < 10 cm (see table 1.2).

Distance
Your program should be flexible enough to make the robot run for any given distance. In the final demonstration on campus, you will be given a distance between 1.0 and 2.5 meters.

Noise
The B83609 speed encoders are not perfect sensors and so are prone to noise that can affect your algorithm. It might be useful to look into **debouncing** to counteract this.

3.6. Bonus Assignment: Person Detection

In this bonus assignment, your goal is to run an embedded AI model that can detect a person in an image on the robot.

The assignment consists of two main components:

1. **The Machine Learning Logic:** Hosted on Google Colab.
2. **The Embedded Implementation:** A C template provided on Brightspace.

You are expected to start by training the model and getting it ready for deployment through the Google Colab notebook, and then deploying it onto your robot using the C template. More information on how to get started is provided throughout the notebook, as well as in the README of the C template.

Completing this assignment will result in an extra **0.5 points** being added to your robot grade.

No Additional Hardware
It should be noted that this bonus assignment is meant to be done **without** any additional hardware. You are only tasked with making the model run on the robot, without an external camera.

A

A Short Tutorial on RP2040 Programming with Registers

When using the Arduino IDE, it is customary to use the provided library to program your microcontroller. The IDE implicitly puts `#include <Arduino.h>` at the top of your code to load definitions for functions such as `pinMode` or `digitalWrite`, which are used in the Blink example sketch, for example.

This library makes our job easier, but also keeps us from learning how the Pico really makes an LED blink. This appendix aims to explain how to blink an LED with the hopes of setting you on the path of figuring out how to program the Pico to do more complex tasks.

Your new best friend
For the following sections, keeping the RP2040 datasheet and SDK reference close is important! This datasheet can be found here and the reference here.

A.1. The world of registers and peripherals

The Arduino library used in the course is built on-top of the C/C++ SDK provided by the Raspberry Pi foundation. This SDK contains a couple of layers that build on each other. At its foundation is the *Hardware Registers Library/hardware_regs* which is a set of include files generated from the hardware to C defines. These define the memory locations, layouts and the offsets for each register. This library is used for the *Hardware Structs Library/hardware_structs*, which are structures that abstract memory-mapped registers blocks to reduce errors and hide complexity. Structures and then used by the `Hardware Support Library/hardware_[name]` which are light-weight C functions that interact with the registers and only adds a small overhead. Finally, these are then used in the `Higher-level libraries/pico_xxx` which provide nice abstractions across different devices and include more scaffolding to ease programming.

These functions read and write data from so-called **registers**, which are special regions in memory that control the processor and its peripherals. Peripherals are internal devices that assist the processor in its tasks, such as keeping track of time, internet connectivity and CRC calculation. In order to find out how to use these registers, we will make use of the Datasheet provided to us by the Raspberry Pi Foundation.

A.2. General-Purpose Input/Output (GPIO)

One such peripheral is the GPIO controller. On the Pico we have two banks: QSPI and the user bank. The former is used for executing code from an external flash drive and the latter is what is available for us as programmer to use. GPIOs are General Purpose Input/Output pins and you can see them on the board when look at it, they're the rows on the sides. Each of these pins are labeled with a number, if you take the board from the casing and turn it around you can see how each individual pin is mapped. For this lab, we have routed these to convenient hubs to the left. You can refer to the Brightspace for a reference what each pin on the breakout board is connected to. [1]

[1] this will be posted before 13th of December

So, how do we use this peripheral to control the pins in such a way to make an LED blink? If you look at the manual, you find on page 235 the GPIO overview. Here, you see the different functions that could be applicable. The first is Programmable IO (PIO) and the second is Software Control via SIO. After looking into it, you discover that PIO are small state machines that can take over IO tasks for the chip, which is a bit overkill. So, instead, you decide to use SIO instead.

SIO are single-cycle IO operations, where you access the entire user bank at the same time, so you can use an offset to access the corresponding pin. It is defined in the `sio_hw` struct, which is a part of the Hardware Struct Library. To set the led to high, we need to enable a bit in `gpio_set` and to disable the led, we need to use gpio_clr.

Accessing registers in Hardware Structure
The defines for registers are contained in a series of header files belonging to the hardware library. Peripherals are represented by structs, accessed with `NAME` with its members being the registers.

You decide to toggle 25, which connects to the on-board LED. This pin is turned on by setting the 25th bit of `gpio_set` to high. You try the following program:

```
#define MASK(x) (1UL << (x))
#define LEDPIN 25

sio_hw->gpio_set = _MASK(LEDPIN);
```

Only to find out the LED doesn't turn on! What is going on? After thinking about it more, you realize that each pin has different modes and that you don't know what the default is. A pin can either be PWM, SIO, PIO, Clock, etc... Therefore, you must set the pin into SIO mode before we can use it to do single-cycle IO.

Setting the mode is done by writing the correct value to the control register of the user register bank. This is located in `io_bank0_io[LEDPIN].ctrl`, which gives us the right control register for our target register. This can be set to the SIO mode value by writing to the `FUNSEL` offset. Since a control register can control many different aspects of a microcontroller, we need to make sure that we shift it an appropriate amount so that is written to the right location. The control register also sets 4 other values: `IRQOVER`, `INOVER`, `OEOVER` and `OUTOVER`. `FUNCSEL` is done at the end of the register, which means that we do not actually need to shift our number in this case.

```
// [DEVICE]->[REG] |= [DEVICE]_[REG]_[VAL][NUM]_[BIT]
io_bank0_hw->io[LEDPIN].ctrl = GPIO_FUNC_SIO << IO_BANK0_GPIO0_CTRL_FUNCSEL_LSB;
sio_hw->gpio_set = 1ul << LEDPIN;
```

It *still* doesn't work? Having lost motivation, you decide to give up. Some light reading will calm you down a bit, so you decide to read the SIO section of the reference manual in more details, when it suddenly hits you: pins can take input *or* output. It could be that your pin starts in input mode rather than output mode. Eagerly, you read ahead and see in the documentation for the `GPIO_OE` register of `SIO` that and, indeed, 0 means it is input and 1 is output. That means that it probably starts up in input rather than output mode! You set the bit to one and see what happens...

```
sio_hw->gpio_oe_set = MASK(LEDPIN);
```

You upload the code and... finally, the LED turns on! But it's not blinking yet, so let's wrap everything up and make it blink.

A.3. Putting it all together

In this short tutorial, we have looked at how to use the GPIO peripherals to control the voltage on a pin and how to configure using SIO. This provides a basic understanding of how the RP2040 chipset works.

Finally, lets make the LED blink rather than simply shining continuously. If you may recall from our hardware lecture there are few ways to achieve this. A good method is by using the `CLOCK` devices to interrupt the microcontrollers. These exceptions can then be controlled by the generic ARM *Nested Vector Interrupt Controller* peripheral. Another, less efficient, option would be using the *Real-Time Clock* that is included. However, to save on energy cost, we decide to use our Pico as a cheap space heater and running a busy loop. To do so, we simply do a loop that counts down and executes a `noop` instruction. After this loop is done, we *xor* the value in the register at position 25 to turn it off and on.

Our program won't differ much from the blink example using just the Arduino library. It still follows the traditional setup & loop architecture. However, we pick and choose exactly which registers we write to rather than using complicated functions that my write to many registers at the same time. This will save quite a bit of memory & time and will make your program more predictable. However, keep in mind that you do need to spend a lot more time writing your program and that the speed you gained comes at the cost of readability. So always keep this trade-off in mind as you pick what parts you write in Arduino, higher level C library or registers!

We finally arrive at the following:

```
#define LEDPIN 25
#define MASK (1L << (x))

void ms_delay(int ms)
{
   while (ms-- > 0) {
      volatile int x=500;
      while (x-- > 0)
         __asm("nop");
   }
}

void setup() {
  io_bank0_hw->io[LEDPIN].ctrl = GPIO_FUNC_SIO << IO_BANK0_GPIO0_CTRL_FUNCSEL_LSB;
  sio_hw->gpio_oe_set = MASK(LEDPIN)
}

void loop() {
  // put your main code here, to run repeatedly:
  sio_hw->gpio_set = MASK(LEDPIN);
  ms_delay(300);
  sio_hw->gpio_clr = MASK(LEDPIN);
}

void main() {
  setup();
  while (true) { loop(); }
}
```

Look ma, no C calls! Are you proud of me now?! I have made a small LED blink!

A.4. Further improvements

Now you know how to control registers, you can control everything the Pico has to offer.

One addition that we recommend that you add to your code is a call to `stdio_init_all()` in your `main` and adding `pico_enable_stdio_usb(blink 1)` to your CMakeLists. This is a call to the high-level API that allows you to upload to the Pico without having to press the BOOTSEL button. This library call is an exception since it does not help with your implementation directly and *always* can be used for full points.

The first step to deeper understanding the microcontroller would be to understand how to create ISRs. There are two ways to do so: adding your own handler by using a call or by defining handling directly. Although, the former is recommended since it is less fragile, the second can be done be redefining the predefined interrupt handler. You can find a list of the function names in a header that renames from their original CMSIS name to a custom RP2040 name.

```
extern "C" void isr_systick(void) {
    // Your code here
}
```

This tutorial should have given you the tools required to go out and do your own research. Further inspection of the library is useful to understand the many things the Pico is capable of. Additionally, many tutorials can be found, but keep in mind register definitions may differ between libraries and processors. A few interesting options are:

- Getting started with the RP Pico
- RP Pico C/C++ SDK (some interesting discussion of PIO and library design)
- RP2040 Datasheet

- *Embedded Systems Fundamentals with Arm Cortex-M based Microcontrollers: A Practical Approach Nucleo-F091RC Edition*
- *Efficient Embedded Systems Design Education Kit for STM32F401*
- *Fundamentals of System-on-Chip Design on Arm Cortex-M Microcontrollers*
- *Making Embedded Systems*

B

Changes over course editions

Edition 2021-2022

- Additional ways to power the STM32.

Edition 2022-2023

- Move to more modern STM32F401 from the STM32F103.
- Addition of small OLED display.
- STLink for reprogramming Z0eff.
- Replace assignment 4 which involved recognizing a robot in front with another assignment where Z0eff navigates around a box.
- Use of the STM32duino over the ArduinoSTM32 Core used previously.
- STM32LL as Arduino backend rather than LibMaple.
- Registers are clarified as being programmed in ARM CMSIS.

Edition 2023-2024

- Clarify final deliverables with dedicated report section.
- Provide specifications of line and objects used for the assignments.
- Move to PlatformIO as first-class development environment and add usage instructions.

Edition 2024-2025

- Move from the ST platform to the Raspberry Pi Pico.
- Rewrite of chapter 2 and Appendix A.
- ST-Link flashing for the Pico in Appendix C.

Edition 2025-2026

- Updated dead link.
- Updated restriction on what counts as registers.
- Added embedded AI as a bonus assignment.

Bibliography

[1] Double speed measuring module with photoelectric encoders model: Hc-020k. URL https://mrelectrobot.com/wp-content/uploads/2021/09/HC-020K.pdf.

[2] Oled display, Dec 2022. URL https://www.aliexpress.com/item/32957309383.html.

[3] Elecrow. L9110 Motor control driver chip. page 2, 2018. URL https://www.elecrow.com/download/datasheet-l9110.pdf.

[4] Indoware. Ultrasonic Ranging Module HC - SR04. *Datasheet*, pages 1–4, 2013. URL https://www.electroschematics.com/wp-content/uploads/2013/07/HCSR04-datasheet-version-1.pdf.

[5] Rasperry Pi Ltd. Raspberry pi pico-series c/c++ sdk - libraries and tools for c/c++ development on raspberry pi microcontroller, Dec 2024. URL https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf.

[6] Rasperry Pi Ltd. Raspberry pi pico datasheet, Dec 2024. URL https://datasheets.raspberrypi.com/pico/pico-datasheet.pdf.

[7] Rasperry Pi Ltd. Rp2040 pico datasheet, Dec 2024. URL https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf.

[8] Earle F. Philhower III. Arduino pico documentation, Nov 2022. URL https://arduino-pico.readthedocs.io/en/stable/.

[9] Vishay. Application of optical reflex sensors - mouser electronics. URL https://www.mouser.com/datasheet/2/427/VISH_S_A0003153719_1-2568779.pdf.