Robotics Practicals

2021

Lab Worksheet

Haptic Interfaces

Assistants: Ali Reza Manzoori and Zeynep Özge Orhan

1. Lab Description

1.1. Aim

This lab is closely linked to research projects in haptics and medical robotics carried out at the Rehabilitation and Assistive Robotics Group. It will give you a brief overview of the state of the art of haptic devices, problems which have to be addressed to generate useful sensations, the type of sensations that can be produced, as well as how they can be programmed. We will look at hardware components that make up a haptic device, discuss the importance and influence of each component and investigate typical control schemes for haptic applications.

At the end of this lab, you will have a basic understanding of the hardware and software components that makes up a haptic device, as well as of the particular features of this type of robotic device relative to more traditional robots currently used in manufacturing.

1.2. Structure

In this practical, you will get familiar with a 2-DOF haptic interface (the Pantograph device). You will not only deal with all the mechatronic components of the device but also program the device to render virtual environments.

Practical Session (4 hours):

- Introduction to haptics and force feedback
  - Basics, advantages, application areas
- Investigating mechatronics components of a haptic device
  - Mechanism, structure, actuators, sensors and control hardware
- Setting up the Pantograph device
  - Investigating data sheets, cabling
- Kinematic analysis of the Pantograph
- Drivers to read encoders
- Drivers to command motors

2. Introduction

2.1. Haptics and Haptic Interfaces

Haptics is the sense of touch and force. It enables us to dynamically interact with the physical world, to manipulate objects, feel the textures of a surface, the shape of an object etc. To better understand the interaction of a human user with a haptic device, let us first take a look at the physiology of haptic sensing. Imagine a sponge on a table in front of you. Using your arm and hand, you can first touch the table and then the sponge. The table will feel hard, and the sponge soft. The forces exerted against your finger (e.g. contact force, compliance and weight of an object, resistance) give you information about the rigidity of the touched or manipulated object. This is called **kinesthetic or force feedback**. But you can also slide your finger over the table and the sponge; feel the edges and the wooden surface of the table, and the porous surface of the sponge. This, in return, is called **tactile feedback**. In this lab, however, we will only focus on force feedback (*For a complete information, please refer to the excellent article on haptics written by* (Hayward & Maclean, 2007)).

A haptic interface is an actuated, computer controlled and instrumented device that allows a human user to touch and manipulate objects either within a virtual environment (VE) or in a real world through a slave of a teleoperated system such as for surgical robotics (Gillespie, 2005). The haptic interface ensures bilateral interactions between the user and the VE in a haptic rendering process as shown in Figure 1 (Samur, 2010). This dual way property, in other words being not only an input interface but also a feedback source for the user, gives a unique characteristic to the haptic device. However, this interaction imposes additional constraints with respect to “standard” mechatronic devices:

- A haptic device must be safe enough to interact with human motion
- Motion and force generated by the haptic interface must be sufficiently smooth without parasite effects (e.g. vibrations)

These constraints drastically influence the design of the mechanical structure, the choice of actuators, transmissions and sensors as well as the control scheme.

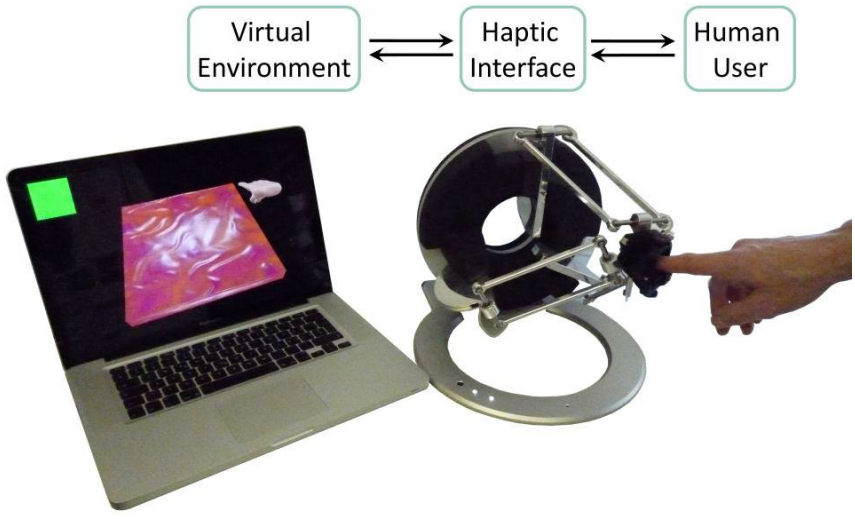


Figure 1: Haptic rendering process. Bilateral interactions between a user and a VE are realized through the haptic interface.

3. Application Areas

Although haptic interfaces are still mainly used in research, there are already several industrial applications, some of which are presented in this section.

3.1. Industrial applications:

- Teleoperation in hazardous environments (e.g. inspection of nuclear power plants)
- Surgery training (e.g., VirtaMed, Surgical Science, Mentice, Simboinix, Mimic, MOOG)
- Medical & Surgical robotics (e.g., ForceDimension, Intiutive Surgical, Hansen Medical)
- Virtual prototyping and 3D CAD (e.g., PHANTOM, Haption)
- Entertainment industry (e.g., Novint Falcon)
- Nano and micro manipulation.

Figure 2: (Left) MIRO surgical robot from DLR controlled by Force Dimension's omega.7 haptic device (Right)da Vinci surgery system.

3.2. Research applications:

- Psychophysics / Neuroscience (in particular investigation of motor control and motor learning)
- Rehabilitation of stroke patients
- Scientific visualization
- Virtual museum
- Input/output devices for the visually impaired

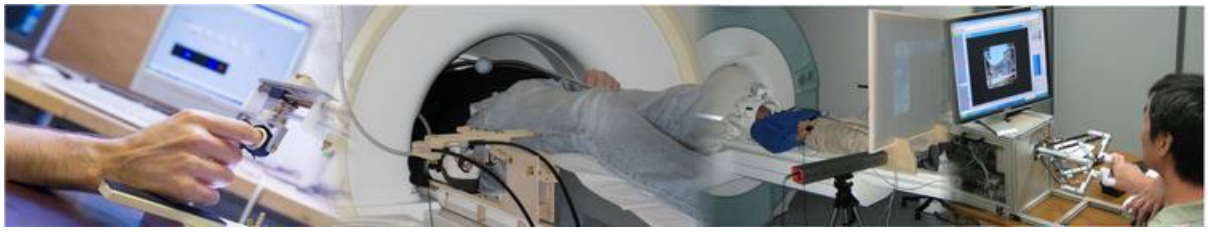


Figure 3: Haptic interfaces to investigate brain mechanisms involved in human motor control. (Courtesy of Rehabilitation Engineering Lab at ETHZ)

4. Hardware

A haptic interface is made of mainly the same components as an industrial robot for assembly or machining tasks: actuators, transmissions and sensors, attached to a task specific mechanical structure. The main difference between a haptic interface and an industrial robot is the dynamic physical interaction with a human operator.

Figure 4 sketches a haptic interface (the Pantograph device), which you will use for this practical. This 2 degree-of-freedom (DOF) interface could, for example, reflect the interaction with a virtual object in 2D space. The haptic device shown in this figure consists of actuators and sensors (for example encoders for sensing the position of the output), a kinematic linkage, a control unit including controller and motor driver, and a PC user interface that can be used for reading information from the device, tuning parameters on it, and displaying a virtual environment to the user.

Let us assume that the actuator is a backdrivable (i.e. reversible with low friction and inertia) DC motor. In this case, the controller runs in torque control mode (i.e. it controls the current running through the motor coils, which, in the static case, is proportional to the output torque by the factor K – the torque constant of the motor). A transmission can then be used to reduce or increase the speed range or output torque, and to link the output shaft to the actuator.

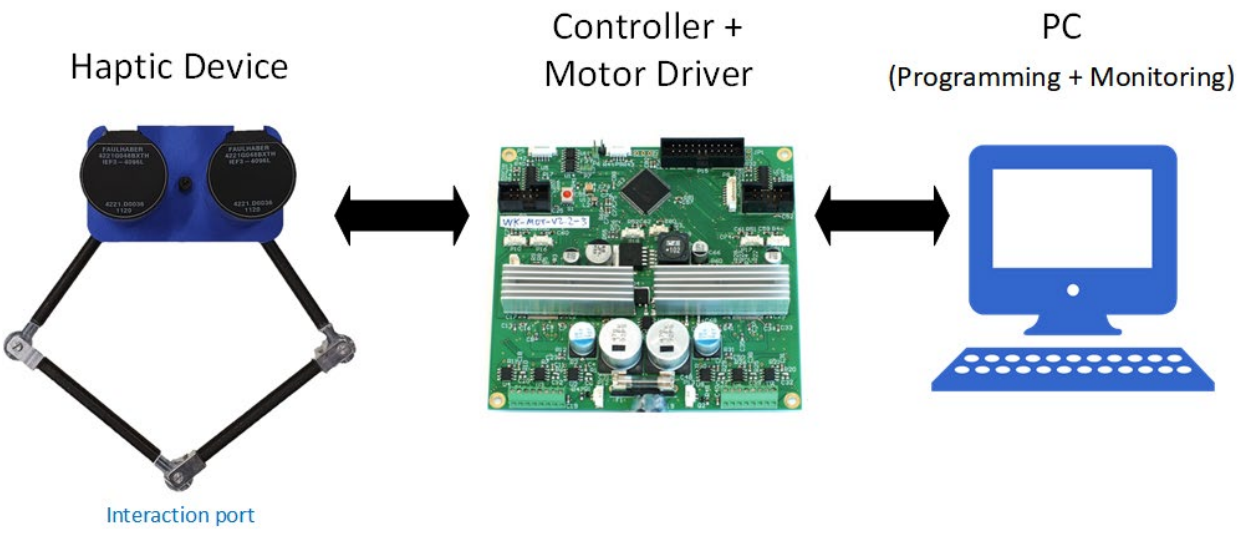


Figure 4: Components of a haptic interface

5. Control

The task of most industrial robots is to position a tool or component in space, and move it along a desired trajectory. These robots usually use a position control scheme. The loop is closed over position sensors located throughout the robot. A haptic interface, however, will interact with a human subject, who will impose the position of the output by her/his movement. Therefore, **the haptic interface must control the reaction forces at the output, rather than the device endpoint trajectory.** This is ideally achieved with control schemes based on force-feedback, but for simpler implementations it is also possible to use open-

loop force/torque control. Haptic feedback must run at a minimum of 1 kHz, to be able to compensate for vibrations up to the sensible frequency of 400 Hz.

Another point that can be considered in the case of a haptic interface is the visual feedback. As the user interacts with a virtual environment, visual information is usually presented together with the haptic feedback (various applications additionally present acoustic feedback to enrich the user experience and make it more realistic). The visual feedback must be presented at an update rate of at least 15 Hz for the user to perceive "smooth" animation.

Figure 5 schematically represents the visual and haptic feedback loops present in the Xitact IHP. The interaction forces between the haptic interface and the virtual world are calculated and updated on the haptic interface at about 1 kHz. This, together with the collision detection, requires an important amount of processing power, which surpasses that of many industrial robots. The visual feedback is generated at 20 Hz. The computer screen shows the interaction of the virtual tool with the virtual environment (Vollenweider, 2000) (Moix, 2005).

In order to produce transparent haptic rendering, a multi-purpose haptic device must be able to simulate free movement, i.e. give the user the impression that he or she is moving his hand freely, without any device attached to it. Once this is achieved, arbitrary forces can be superposed on the free movement.

The most frequently implemented force-control strategy for haptic interfaces is impedance control. In **impedance control** mode, the haptic device will generate force in reaction to a displacement generated by the human operator. This control scheme theoretically requires a force/torque sensor at the output to measure the interaction forces with the operator, but can practically be realized without such a sensor if the actuators and transmissions are backdrivable and have low friction, and the mechanical structure has low inertia (For more information on impedance control, refer to (Hogan & Buerger, 2005)).

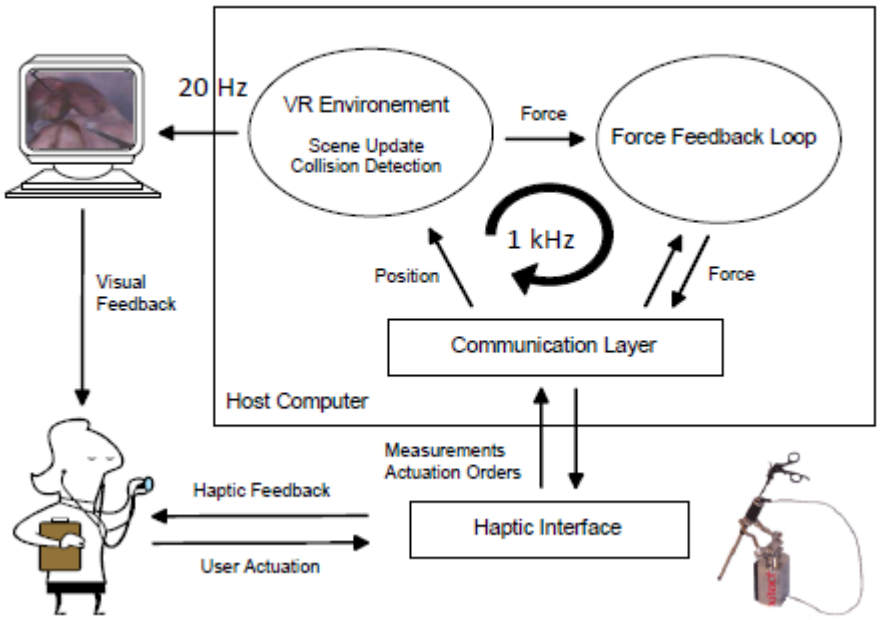


Figure 5: Schematic representation of the visual and haptic control loops of a virtual reality station

6. Bibliography

Gillespie, R. B. (2005). Haptic interface to virtual environments. In Robotics and Automation Handbook. Tom Kurfess (ed.). CRC Press.

Hayward, V., & Maclean, K. (2007). Do it yourself haptics: part i. IEEE Robotics & Automation Magazine, 14(4):88–104.

Hogan, N., & Buerger, S. (2005). Impedance and Interaction Control. In Robotics and Automation Handbook. Tom Kurfess (ed.). CRC Press.

Moix, T. (2005). Mechatronic Elements and Haptic Rendering for Computer Assisted Minimally Invasive Surgery Training. Lausanne: EPFL PhD thesis No 3306.

Samur, E. (2010). Systematic Evaluation Methodology and Performance Metrics for Haptic Interfaces. EPFL PhD Thesis No 4648.

Vollenweider, M. (2000). High Quality Virtual Reality Systems with Haptic Feedback. Lausanne: EPFL PhD thesis No 2251.

7. Practical Session

7.1. Hardware overview

The final purpose in this practical session is to implement some simple virtual environments using the haptic device shown in Figure 6. The main part of the device itself consists of 2 brushless DC (BLDC) motors, connected to a pantograph linkage that converts the rotational movement of the motors to the translational movement of the end point in the 2D plane. In robotics literature, the rotational movement of the motors is typically referred to as the "joint space", and the translational movement of the end point (also known as the end-effector) is referred to as the "task space".

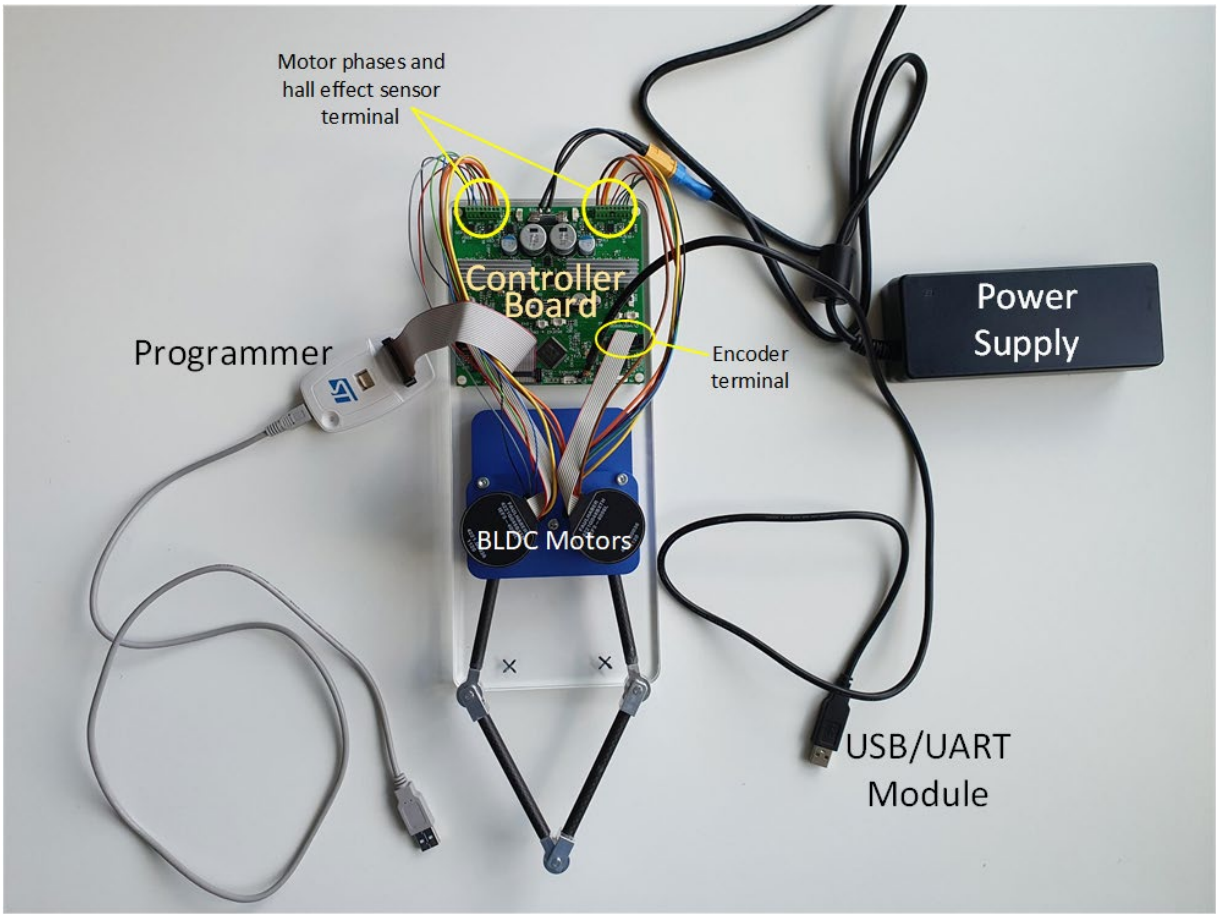


Figure 6: The pantograph haptic device and its peripherals.

The motors are driven by a controller board, which is also responsible for doing the calculations to render the virtual environment[1]. The board is powered by a DC power supply. In order to enable the board to drive the motors, the winding phases of the motors and the hall sensor outputs are connected to the green screw terminal on the board. The (incremental) shaft encoders of the motors are also connected to

[1] In many robotic devices, these 2 tasks are typically carried out by 2 separate units: an embedded controller (which only takes care of the calculations and generating commands), and an amplifier (which takes care of providing the required voltage and current to the motors to follow the commands generated by the controller).

the black box connectors on the board using flat cables, providing a more accurate information about the position of the motor shafts. You can read more about brushless DC motors and incremental encoders in Annex I. Note that in order to protect the motors, the board will limit the torque of the motors to their rated torque, which is approximately 110 mN.m. Even if you command a higher torque value, this limit on the torques will be enforced by the controller board.

All of the computations on the board are carried out by a microcontroller. You will upload your program (written in the C programming language) to render the virtual environment on this microcontroller, using a programmer (ST-link V2). This programmer is only active for a few seconds when you upload (or "flash") your code to the board, and is idle the rest of the time[2]. After uploading your code to the microcontroller, you can communicate with it from the computer to read the variables and also set controller parameters if needed. This communication is done via the UART protocol, using the USB/UART converter that allows the computer to transfer data with the device over a normal USB port.

Important precaution for working with the device:

The controller board is exposed, and this board is sensitive to electrostatic discharge. Therefore, avoid touching the board as much as possible, and if you have to touch the board (for pushing the reset button for example), touch the blue metal part of the pantograph first to discharge any possible static charge.

[2] However, it is better to keep it connected to the computer because you may need to flash your code several times. When the programmer is connected to the board, it should also be kept connected to the computer because otherwise it will block the microcontroller from running.

7.2. Theoretical Derivations

Exercise 1. Deriving the kinematic equations

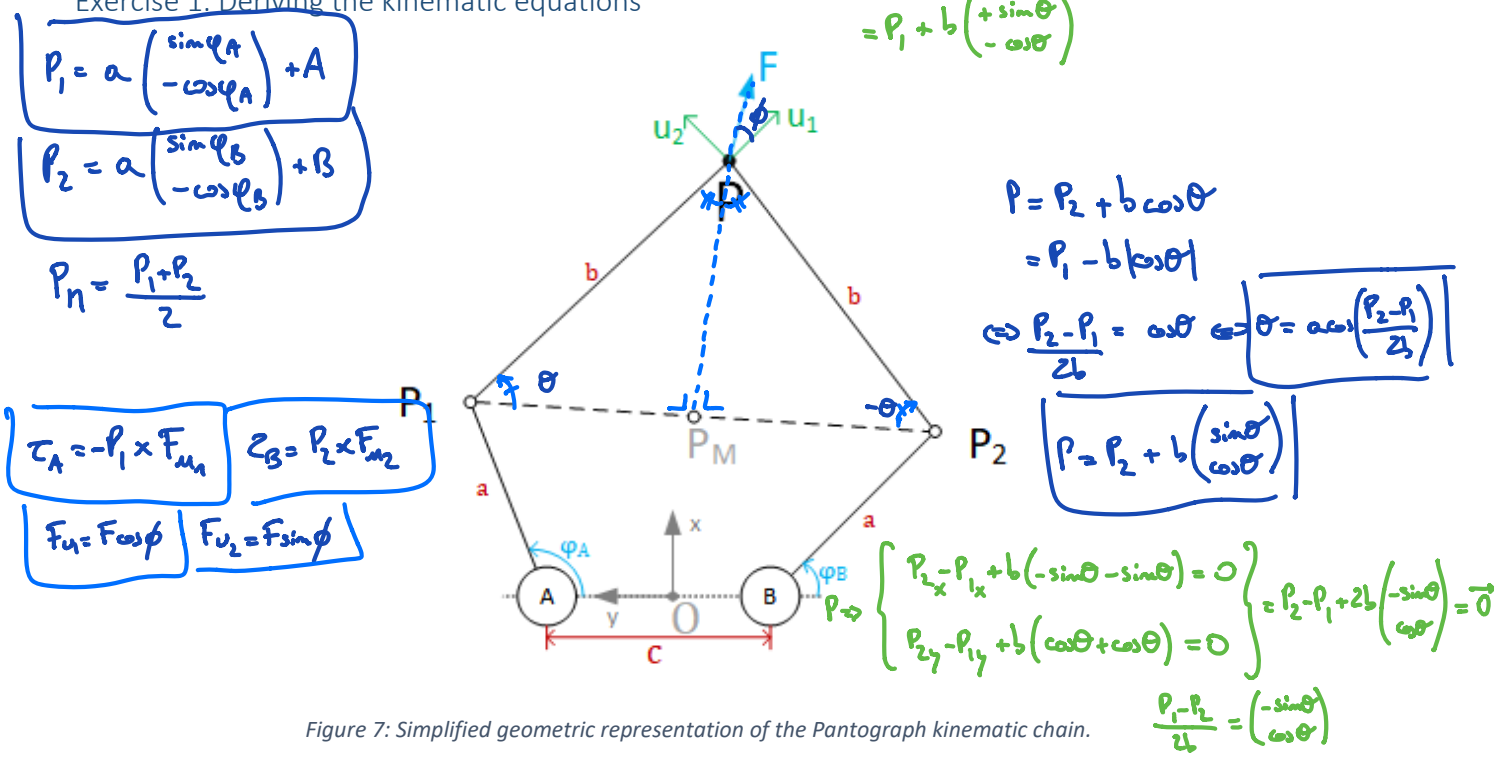


Figure 7: Simplified geometric representation of the Pantograph kinematic chain.

A) Derive, by hand, the forward kinematic equations for the endpoint of the pantograph, relating the joint space positions to task space (i.e., endpoint position as a function of motor angles). In other words, write the equation describing **P**(x,y) in the Cartesian plane in terms of angles φ_A and φ_B as shown in Figure 7 with the lengths a, b and c as fixed parameters. Pay attention to the positive direction of the X and Y axes, the positive direction of the angles, and the zero point of the angles. Keep your equations consistent with Figure 7. Also note that the origin of the coordinate frame O is located on the midpoint of the line connecting motors A and B (so we have $\boldsymbol{A}(0, +{}^{c}/_{2})$ and $\boldsymbol{B}(0, -{}^{c}/_{2})$ for example).

Hint: You can use the auxiliary point P_M (midpoint of the line connecting P_1 and P_2) and geometric relationships to simplify the derivation. This is not the only possible method, though.

B) Implement these equations in a MATLAB script and roughly verify that the results make sense using some intuitive values for the motor angles. You can use the numerical values $a = 102\ [mm], b = 111\ [m]$ and $c = 60\ [mm]$ for the lengths to do the numerical verification.

C) After checking your equations, convert them to symbolic form[3] such that you have the X and Y components of the endpoint position P, as a function of symbolic variables named `phi_A`, `phi_B`, `LENGTH_A`, `LENGTH_B` and `LENGTH_C`. You can then generate the equations to be

[3] For this step you need to have the Symbolic Math Toolbox of MATLAB installed on your computer. For a very short review of this toolbox and its most used commands, refer to this webpage.

implemented on the microcontroller (expressed in C programming syntax rather than MATLAB) using the `ccode()` function of MATLAB.

Exercise 2. Deriving the force/torque equations

A) Derive, by hand, the equations between $\boldsymbol{F}(F_x, F_y)$, the force vector applied to the end point of the device, and the motor torques τ_A and τ_B (i.e. the equations relating task-space forces to joint-space torques). Write the motor torques as a function of the end point force components with the motor angles as variables (that is, $\tau_A = f(F_x, F_y, \varphi_A, \varphi_B)$ and $\tau_B = g(F_x, F_y, \varphi_A, \varphi_B)$). Use the lengths a, b and c as fixed parameters. Pay attention to the positive direction of the force components and the torques as shown in Figure 7 (positive direction of the torques is the same as the positive direction for the angles).
Hint: You can use the auxiliary unit vectors u_1 and u_2 (which are parallel to the links from P_1 and P_2 to P) to simplify the derivation. This is not the only possible method, though.

B) Implement the equations in your MATLAB script and similar to before, roughly verify them by inputting some representative numerical values for the joint angles and force vector components.

C) After checking your equations, you can convert them to the symbolic form such that you have the torques of the motors A and B as a function of symbolic variables named `phi_A`, `phi_B`, `LENGTH_A`, `LENGTH_B`, `LENGTH_C`, `F_x` and `F_y`. You can then generate the equations to be implemented on the microcontroller directly using the `ccode()` function of MATLAB.

7.3. Flashing the base code and pantograph initialization

As a first step to begin working with the device and becoming familiar with it, you can flash the base code to the controller and check the reading of the encoder angles. Follow these steps for flashing the code:

1. Plug your power supply into the electric outlet and verify that your device is powered on by checking the LED on the board.
2. Open the microcontroller code project in the System Workbench software, and compile the code by clicking on the black downward arrow next to the "build" icon (see Figure 8) and making sure *Release* is selected (you only need to do this once, next time you can just click on the build icon).

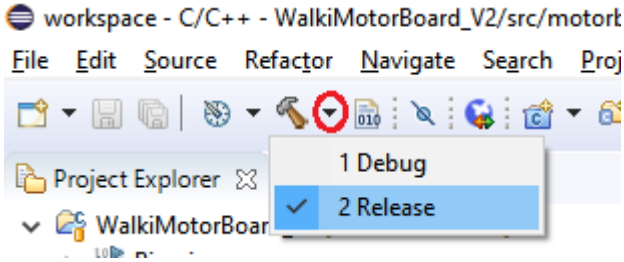


Figure 8 - Selecting "release" mode for the first-time compilation of the microcontroller code.

3. Connect the USB cable of the programmer to your computer. The red LED on the programmer will turn on if it is properly connected. If this LED is off or blinking, it means there is a problem is your connection to the computer, check with the assistants.
4. After the build process is finished successfully, you can flash the program to the microcontroller by clicking on the black downward arrow next to the "run" icon, then *Run As > 1 Ac6 STM32…* (see Figure 9). Verify that the flashing process is finished successfully by looking in the *Console* tab on

the bottom of the System Workbench window, there should be a line saying *"Verified OK"* near the end.

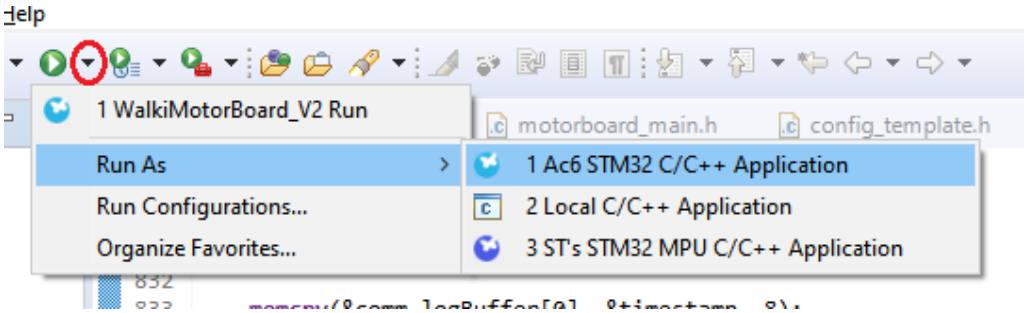


Figure 9 - Flashing the code on the microcontroller for the first time.

Now you can use the GUI application on the computer to communicate with the board and read the encoder angles. The encoders of the device are incremental, which means that they can only give the relative amount of rotation of the motor compared to the initial position at which you power on the device. But for our calculations, we need the absolute angles as defined before (refer to Figure 7). Therefore, before being able to start working with the device, you need to initialize the encoders. Follow these steps to communicate with the device via the GUI and initialize the encoders:

1. Make sure that the USB/UART module is connected to the computer's USB port. Open the PantographGUI project in Qt Creator and run it by clicking on the green "run" button on the bottom left corner of the screen (see Figure 10).

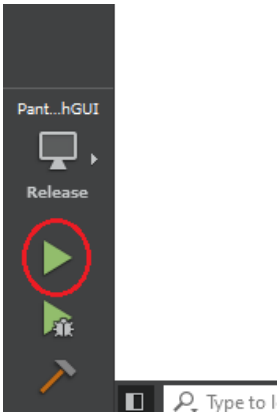


Figure 10 - Running the GUI program in Qt Creator.

2. On the first window that appears, from the "Serial port" drop down menu, choose *USB Serial Port* and then click on OK. The GUI window will appear (Figure 11).

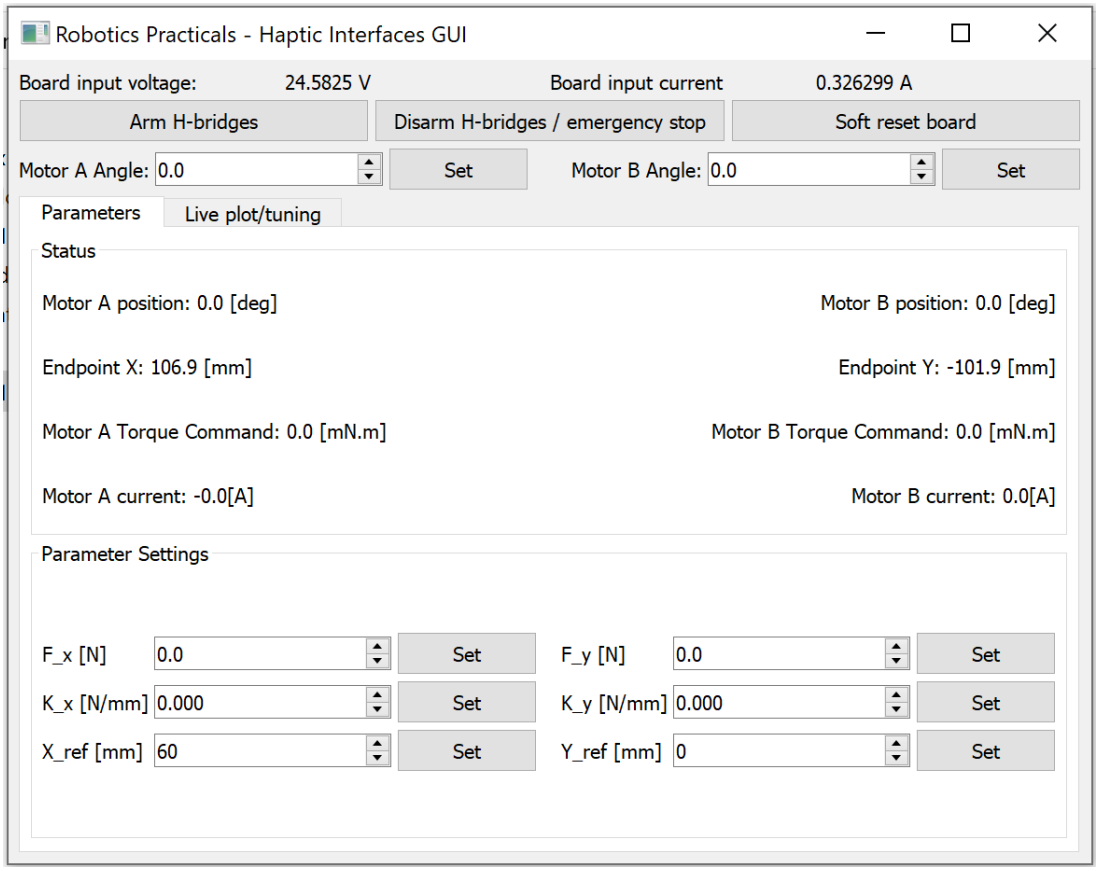


Figure 11 - The GUI window of the PC interface software.

3. If the numbers for the board input voltage and current are not being displayed, use the red reset button next to the encoder connector of motor A on the controller board (see Figure 12). Resetting the board is required each time you flash a new program to the microcontroller.

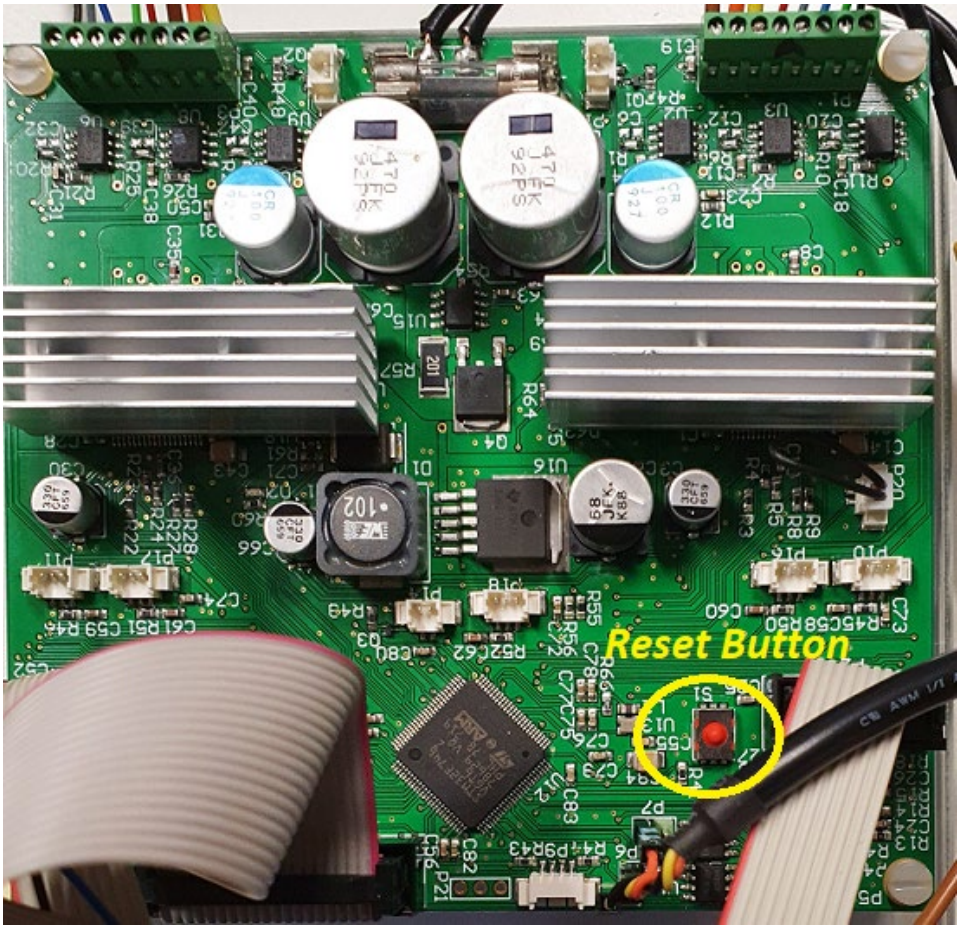


Figure 12 - Reset button on the controller board.

8. You can also monitor the messages sent from the microcontroller in the *Application Output* tab in Qt Creator. After resetting the board, you can check the messages received from the controller to verify the communication as well.
9. Try moving the links directly connected to each motor and see the angle value changing in the GUI window. In order to initialize the angles to the correct absolute values, move the link directly connected to each motor to the 90° angle (you can use the black cross marks on the base as a guide) and then enter 90 in the *"Motor A/B Angle"* field and click on the *Set* button. Verify that the angle for the corresponding motor has changed to 90. You need to do this separately for each motor. Also, each time you power off, reset, or reprogram the board, this procedure needs to be repeated.

7.4. Becoming familiar with the microcontroller code

Go back to the microcontroller code in System Workbench. On the left side of the screen in the *Project Explorer* pane, find the file `motorboard_main.c` in the `src` folder. This file is the part of the code you will be writing your program in. Around the top of this file, find the lines where the constants and variables for the pantograph are defined and initialized (indicated clearly by comments in the code). You will be using these parameters and variables in your controller. Also pay attention to the units of these parameters and variables, indicated in the comments. Then scroll down to the function `StepMainLoop()` which is where you need to insert your controller code. This function will be called at

1 KHz (that is, every 1 millisecond). You can see some sections are marked with a comment saying "`YOUR CODE HERE`".

7.5. Implementing the haptic controller

on site

Exercise 3. Implementing and testing the kinematics

Insert your code for calculating the X and Y coordinates of the endpoint position under the section marked with the comment `Forward kinematics`. Use the motor angles `phi_A` and `phi_B` defined just above, and the necessary parameters and variables defined on the top section of the file. Update the values of the `endPointXPosition` and `endPointYPosition` variables with your calculated values. You can use the C code generated by your MATLAB script from Exercise 1 here.

After entering your code and checking that it compiles without any errors, you can flash the program to the controller board. Then, connect to the board with the GUI program and initialize the encoders to the correct angles as described in the previous section. Now try moving the end point around and see the values for the *"Endpoint X/Y"* being updated in the GUI.

In order to verify your kinematics, place the end point over each of the cross marks on the base. These points are located at $(+90, \pm 30)[mm]$ and your calculated end point position must approximately agree with these values.

on site

Exercise 4. Implementing and testing the force-torque conversion

Insert your code for calculating the torques of the two motors based on the force vector applied to the end point under the section marked with the comment `Generate torque commands from end point force`. Use the force components `F_x` and `F_y` in your expressions, and update the values of the `motorATorqueCommand` and `motorBTorqueCommand` variables. You can use the C code generated by your MATLAB script from Exercise 2 here. Compile and flash your code to the board, and initialize the encoders.

A) As a first test, set `F_x` to 1 [N] and `F_y` to 0 [N] and check the *"Motor A/B Torque Command"* values in the GUI as you move the endpoint around. Do these generated commands make sense? **Check your results with the assistants before proceeding.**
B) Click on the "Arm H-Bridges" button in the GUI to activate the H-bridges on the board and start driving the motors. After a brief delay, the motors will start applying the commanded torques. Feel the direction of the force being applied at the end point, does it correspond to the force you programmed?
C) Gradually increase the value of F_x, while keeping an eye on the commanded torques to the motors. What is approximately the upper limit on the forces that the device can generate over the full workspace? If you want to increase this maximum force, what can you do?

on site

Exercise 5. Rendering virtual springs and dampers in X and Y

Write the code to simulate an elastic virtual environment at the end point of the pantograph, consisting of two separate linear springs acting in the X and Y directions respectively. Use the variables named X_ref and Y_ref as the resting position of the springs. Also, use the spring constants `K_x` and `K_y` in your equations, and save the calculated forces to `F_x` and `F_y`. Compile and flash the code to the board. Do not forget to initialize the encoders before continuing.

A) As a first test, set some sensible values for the spring stiffnesses and resting positions. Then, without activating the motors, move the end point around and check the generated torque commands. Do they intuitively make sense? **Check your results with the assistants before going to the next step.**
B) Try simulating a spring only in one direction at a time, with a proper stiffness value. Arm the H-bridges and test the behavior. Does it replicate the behavior of a real spring?
C) Set the spring stiffness in one of the directions to a very low value, e.g. 0.001 [N/mm], and the other to 0. Does this behavior correspond to that of a real spring? If not, what is causing the difference?
D) Set both of your spring constants to 0.01 [N/mm], then gradually increase the stiffness only in one direction, by increments of 0.01. Do you observe a fundamental change in the behavior as you increase the stiffness of the virtual spring (other than feeling stiffer)?
E) What is the highest stiffness that you can set and still have realistic behavior? Which issue keeps you from increasing the stiffness beyond this value? What determines the upper limit of the virtual stiffness you can render with the device?
F) Discuss your conclusions about the limitations of this device in rendering virtual springs. Which characteristics determine the lower and upper limit of the impedances (which was only a stiffness in this lab) that a haptic device can render realistically?

ANNEX I

Actuators and Sensors

Brushless DC motors

Two brushless DC motors (BLDC) are used for our haptic interface. Before looking at the working principle of a brushless motor, let's see how the "brushed motors" are working.

The "brushed motors" derive their name from having brushes within the internal structure. In a brushed motor a stator with magnets surrounds a turning rotor. When connected to a current source, opposite polarities create a magnetic field torque. Because of this, the rotor starts turning around its axis. In other words, the rotor turns under the influence of the magnetic field such that its north pole will move toward the south pole of the stator (Figure 13, right). Brushes are used to switch the polarity of the rotor magnet as the rotor rotates, so that the opposite poles on the rotor and stator are always close to each other (and therefore creating torque). This constant switching of the polarity is called "commutation". You can refer to this video for a very easy to understand explanation.

The "brushless motor" is so named since the magnetic field is provided by rotating magnets while the armature is stationary, thus eliminating the need for brush commutation. The flow of the electric current between the rotor and the stationary part of the machine is switched externally. As the rotor keeps turning, the commutator reverses the direction of the electric current (and thus the magnetic field) to create a one-way torque to keep the motor turning.

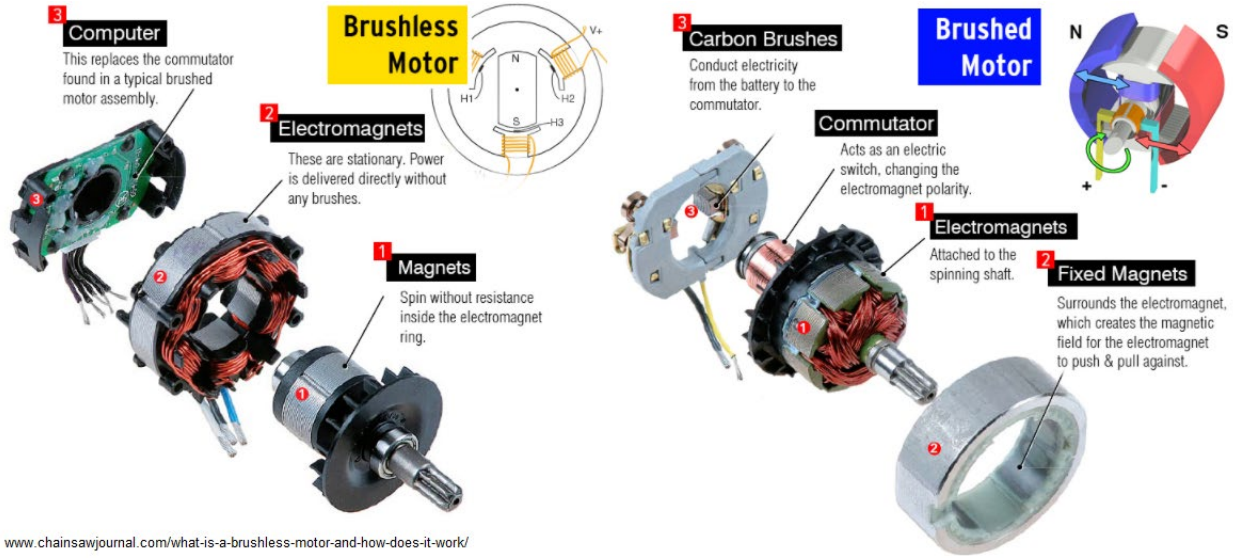


Figure 13 – Working principles of brushless and brushed motors

The difference between a brushed motor and a brushless motor is thus in how the commutation is done. In brushed motors, the commutation is done mechanically in the motor. In a brushless motor, the commutation needs to be done by an external electrical circuit, and thus we need sensing. A Hall-effect sensor which is a solid-state, magnetic field sensor is used to sense the rotation angle and thus control the current supply to the stator such that the magnetic field is in the right direction. In other words, the

Hall-effect sensor can be used to measure the rotation speed, and the angle of the rotor, and then decide when the current should be applied to the motor coils to make the magnets rotate at the right orientation.

The Hall-effect sensor employs the principle that when a conductor with current flowing through it is placed in a magnetic field, the magnetic field induces a transverse force on the charge carriers. This force pushes them to the sides of the conductor—negative to one side and positive to the other side. This creates charge on the sides of the conductor induces a voltage.

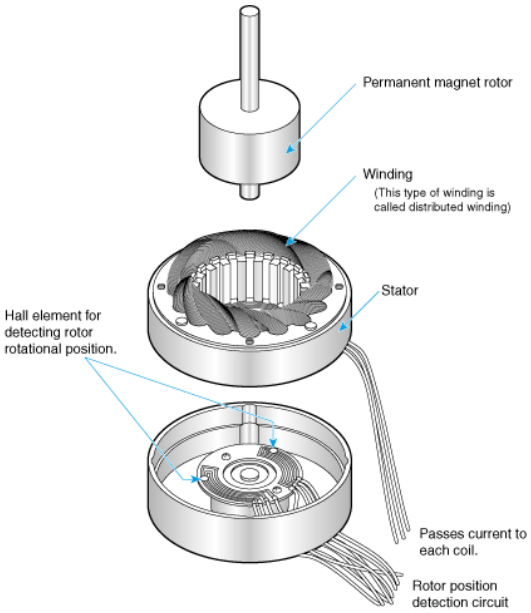


Figure 14 – Structural components of BLDC motor

Compared to other types of motors (DC brushed, stepper motors…), these ones have many advantages such as higher efficiency, high torque to weight ratio, compact size, increased reliability, low vibration and reduced noise. Thanks to these advantages, BLDC motors are often used in modern devices where low noise and low heat are required especially if devices run continuously. These primary distinctions are relevant to haptic interface design.

Encoders

To accurately determine the motor's shaft angle at any point in time, 2 position sensors are attached to one end of each motor. For such position sensors, several technologies exist such as magnetic sensing (using Hall Effect sensors) or optical encoders like the ones we are using.

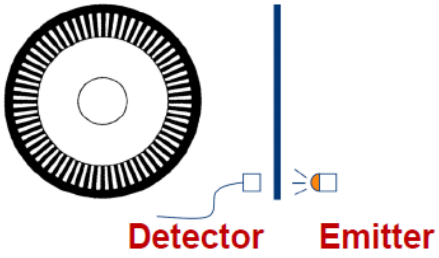


Figure 15. Optical encoder working principle

In an optical encoder, a focused beam of light is aimed at a paired photo detector and is interrupted by a coded patterned disk. Thus, the system will produce a number of pulses per revolution, linearly with the motor's shaft angle. Two types of optical encoders exist: absolute and differential encoders. For the first type, the absolute angle of the shaft is known by just reading the light pattern while differential encoders only allow incremental / decremental readings.

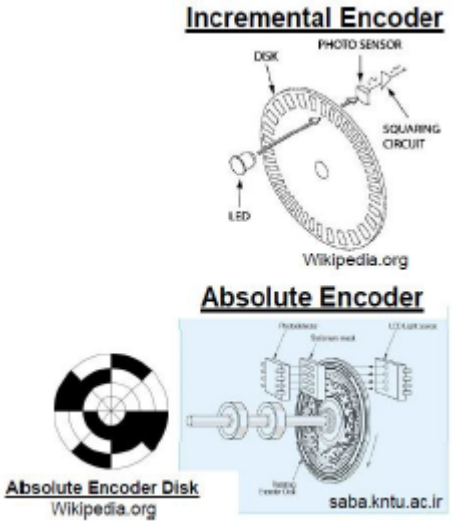
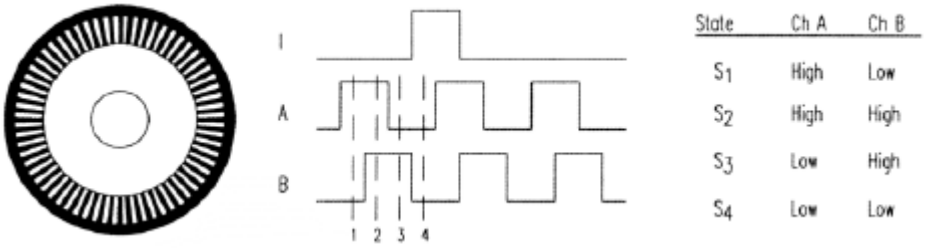


Figure 16: Different types of optical encoders

To know in which direction the shaft will turn, optical systems actually use two LEDs (2 channels) slightly misaligned in order to generate different signals depending on the rotation:



To properly convert the photo detectors output to a pure square signal such as--+ the one shown in the above diagram, differential line receivers can be used, where Schmitt triggers are incorporated.