Robotics Practical - Report

Topic 5 - Haptics

Group 29:

Simon GILGIEN 253797

Jean LESUR 284531

Filip SLEZAK 286557

Assistants:

Özge ORHAN

Ali MANZOORI

Date : March 27, 2021

Contents

1 Introduction

This practical work aims at familiarizing us with the basics of haptic devices. In the first section we derive the forward kinematics equations which are used for calculating force-toque equations. We will then implement some basic force generating commands and discuss the results.

2 Method

2.1 Forward kinematics

A key information to control the haptic device we used in the lab is the position of the endpoint. Since we have no direct way to measure it, we must derive the forward kinematic equations of the system to deduce the position of the enpoint from the angle of the motors, which we can measure.

Using the coordinate system given in the assignment and reproduced in figure 1, we can express the positions of the motors as

$$A = \begin{bmatrix} 0 \\ c/2 \end{bmatrix} \qquad\qquad B = \begin{bmatrix} 0 \\ -c/2 \end{bmatrix}$$

The positions of the articulations P_1 and P_2 can be derived using simple trigonometric properties:

$$P_1 = \begin{bmatrix} a \cdot \sin(\phi_A) \\ -a \cdot \cos(\phi_A) + c/2 \end{bmatrix} \qquad\qquad P_2 = \begin{bmatrix} b \cdot \sin(\phi_B) \\ -b \cdot \cos(\phi_B) - c/2 \end{bmatrix}$$

Then we define the angle α between the segment P_1P_2 and the y axis, taken as being positive in figure 1. Since we have the coordinate of P_1 and P_2, we can compute

$$\alpha = \operatorname{atan2}\left((P_1 - P_2)_x, (P_1 - P_2)_y\right)$$

We also define the angle $\theta = \angle P_2P_1P$.

Since the triangle $\triangle P_1P_2P$ is isosceles, the median PP_M is normal to P_1P_2, and therefore we can express θ as

$$\theta = \arccos\left(\frac{\|P_1 - P_2\|}{2b}\right)$$

Finally, we can express the position of the endpoint of the pantograph as

$$P = P_1 + \begin{bmatrix} \sin(\theta - \alpha) \\ -\cos(\theta - \alpha) \end{bmatrix}$$

We implemented this method in Matlab using its Symbolic Toolbox, and generated the corresponding C code. You will find this code in the annexes at the end of this report.

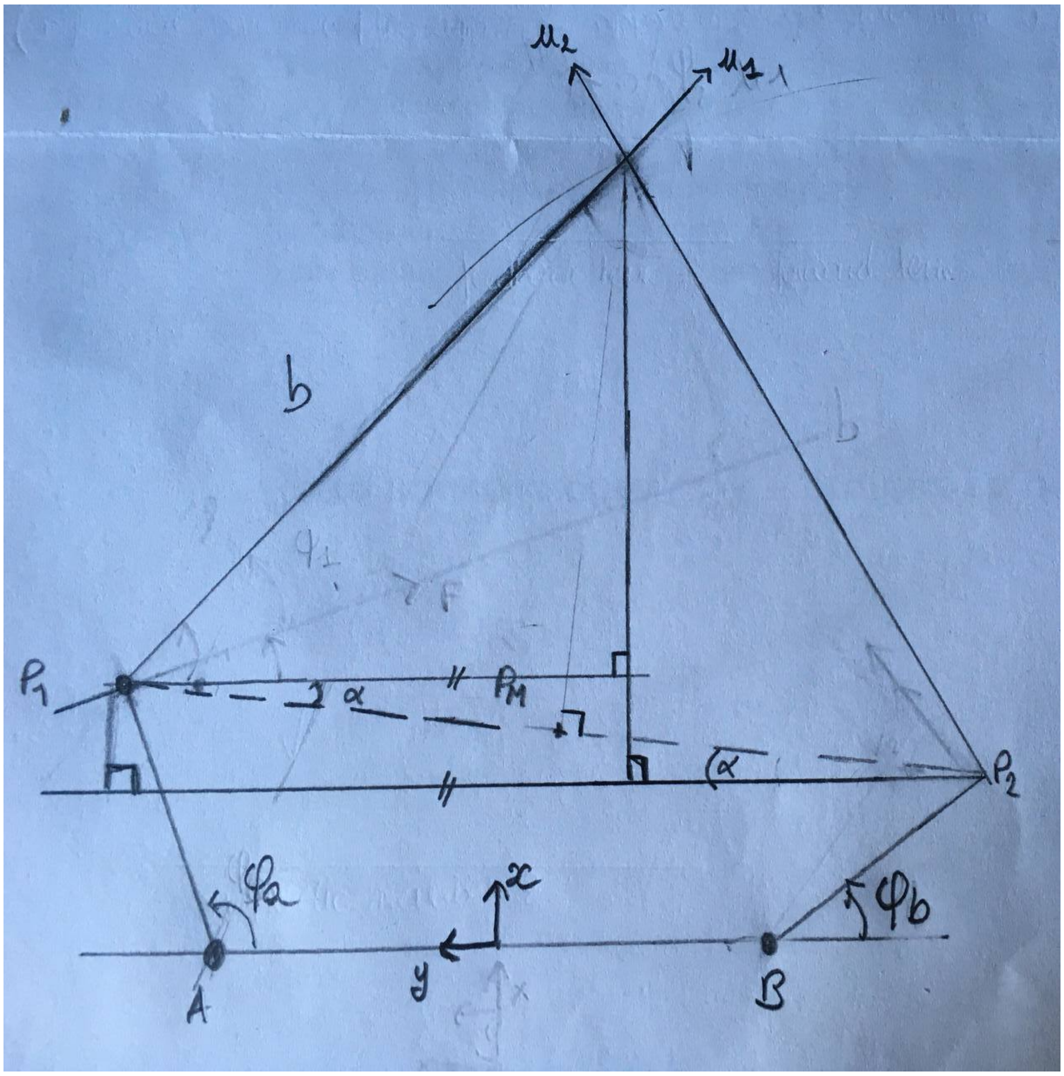

Figure 1: Simple drawing used for the exercise.

2.2 Force-torque equation

In order to control the haptic feedback of our device, we need to be able to convert the desired force on the endpoint to motor torques. The first thing to note is that since both ends of the PP_1 segment can pivot freely, no external force can be applied on this segment. Since we consider a static solution, the external normal forces on that rod at each end cancel each other, and since the resulting moment has to be null, we can conclude that no normal force is transmitted in this segment.

By symmetry, we can also conclude that the force transmitted in the PP_2 segment is also parallel to that segment. Therefore, it is interesting to express the desired force as a linear combination of u_1 and u_2, which are defined as units vectors parallel to P_1P, respectively P_2P.

Using the angles α and θ defined above, we can express u_1 and u_2 in the xy coordinate frames as

$$u_1 = \begin{bmatrix} \cos\left(\theta - \alpha - \frac{\pi}{2}\right) \\ \sin\left(\theta - \alpha - \frac{\pi}{2}\right) \end{bmatrix} \qquad u_2 = \begin{bmatrix} \cos\left(\frac{\pi}{2} - \alpha - \theta\right) \\ \sin\left(\frac{\pi}{2} - \alpha - \theta\right) \end{bmatrix}$$

Therefore the transfer matrix from base xy to the base u_1u_2 is given by

$$M = \begin{bmatrix} \cos\left(\theta - \alpha - \frac{\pi}{2}\right) & \cos\left(\frac{\pi}{2} - \alpha - \theta\right) \\ \sin\left(\theta - \alpha - \frac{\pi}{2}\right) & \sin\left(\frac{\pi}{2} - \alpha - \theta\right) \end{bmatrix}^{-1}$$

We can then express the force in each rod as

$$\begin{bmatrix} F_1 \\ F_2 \end{bmatrix} = M \begin{bmatrix} F_x \\ F_y \end{bmatrix}$$

The corresponding torque on the motors can then be expressed using the angles between the a and b segments:

$$\tau_A = -F_1 a \sin(\phi_A - (\theta - \alpha)$$
$$\tau_B = F_2 a \sin(phi_B + \theta + \alpha)$$

Again, we implemented this in Matlab, and used the Symbolic Toolbox to generate the corresponding C code.

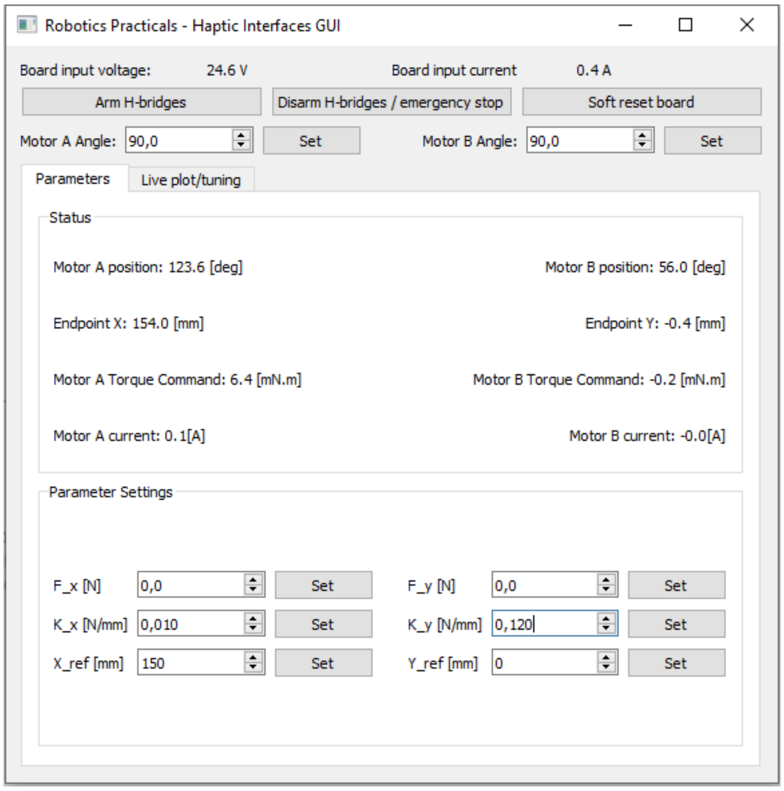

Figure 2: Interface of the control software

2.3 Implementation on the haptic device

We embedded the C code generated previously in the controller of the haptic device. The code running on the STM32 controller of the device computes the torque output to apply to achieve the desired torque, and it also computes the position of the endpoint of the device in real-time.

We then used a program on the control PC communicating with the controller via UART. This allowed us to monitor the position of the endpoint of the device, set the desired force on the endpoint, and activate the motor driver. This program also allows to calibrate the encoders, which is needed every time that the controller board is reset as the encoders only measure the relative displacement of the motors, and not their absolute position. A screenshot of the interface of this software is depicted in figure 2.

In a second time, we modified the controller code to simulate a spring on the endpoint in the x and y directions. Given a reference point and a stiffness in the x and y direction, the controller computes the required forces to apply on the endpoint and the corresponding motor torques. We can then dynamically set the reference point and the stiffness in each direction from the control software.

3 Results

3.1 Observations

Our first observation upon entering the room was how big the device actually. From the picture we had we imagined it much smaller. With such motors we were quite surprised the force was quite limited and that we could actually get the device stuck in some positions. We were then confronted with the fact that the device needed constant calibration upon each reboot and some errors might result from the imprecise way we had of doing this.

Furthermore we also explored the code structure presented to us during the practical to see what such a device involved. One thing we would have liked to try out is to lower the frequency below our sensitivity threshold to feel the difference.

3.2 Worksheet Questions

For exercises 1 and 2, please refer to sections 2.1 and 2.2.

For exercises 3 and 4, we tested our formulas on the haptic device and looked for incoherencies or incorrect edge cases before playing around with the device a little using different forces. Over the entire workspace of the device, the maximum forces were 0.3N along the x-axis and 0.9N along the y-axis, approximately. To increase this, we could modify the device geometry, use bigger motors or change the transmission to reduce friction.

For exercise 5, we simply implemented the spring formula $\vec{F} = -k(\vec{x} - \vec{x}_0)$ for the input forces in the control loop (with the particularity that k can be different in the x and y directions). Then, using the virtual environment interface, we intuitively set the virtual spring constant, to 20mN/mm for instance. Moving around the end point, the resulting torques weren't saturating the motor outputs and seemed to make sense. Following are answers to the last questions.

1. 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?

   Yes they intuitively make sense if we consider the right-hand rule orientation for the motors.

2. 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?

   The systems does act like a spring. Indeed, if we offset the point P from its resting position, this point gently returns to the resting position with an oscillating response. This oscillatory responses acts differently for different stiffness values.

3. 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?

   No it does not act like a real spring. In fact, there is some friction in the systems, the asked force is too small to overcome this friction. In addition, a BLDC motor needs to provide a minimum torque to counter its inertia.

4. 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)?

   The oscillation's frequency is increasing along one direction. The stiffness limits for the device to operate like a spring depend on the resting position. The oscillation along a given direction makes some oscillation in the other direction as well, although they are way smaller. A real spring would not behave like this, the oscillations would stay along the same direction.

5. 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?

   When setting the resting position at x=120mm and y=0, we can use a stiffness up to 60 mN/mm and still have a realistic behavior. If we set the resting position a little bit further, x=150 and y=0, the stiffness can be set to 120 mN/mm. The saturation of the motor torques keeps us from increasing the stiffness. If we want to increase the stiffness, we need to generate bigger torques.

6. 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?

   Compared to a real spring, this device is a lot bigger and needs electricity to function. Those limits are determined by the device's geometry, the inertia of the motors as well as the minimum and maximum torques it can produce. Last, having some low-friction transmission in the arms is a plus.

3.3 Additional Questions

1. What is the unique property of haptic interfaces that differs from conventional human-computer interfaces?
   They are input and output at the same time. They allow bidirectional interactions between the user and the virtual engine.

2. What are the constraints on the mechanical design of a haptic device?
   The device must be backdrivable, safe for human interaction and operate smoothly, in a realistic manner.

3. Choose one application area of haptic interfaces and explain how the haptic device is used and which advantages it has.
   Let's consider latest generation of iPhones to have a home button. By integrating haptic feedback into their button, Apple got rid of the play between the button and frame. This prevents water and dust from getting inside the device, thus rendering it waterproof.

4. Typically, what 2 electronics units are needed to control a robotic device?
   An actuator and a sensor to get the position of the motor's rotor.

5. Why do we have to initialize the Pantograph device angles after each resetting?
   This Pantograph device uses a relative encoder and upon reset the program doesn't remember the last value of the angles. Adding to that the likely fact that the user might move the device between two runs of the program. The easiest way to get a correct estimate of the angles and prevent unexpected behaviour when running the program is to manually calibrate the device.

6. What is impedance control? Why should the control loop frequency be high?
   Impedance control adapts the output of the haptic device depending on the impedance it receives from the environment/user. The interactive input is then taken into account in the output of the haptic device. This usually requires the actuator and sensor combination mentioned above but sometimes a backdrivable actuator can be used if the friction/inertia are sufficiently low for the interaction to be fast enough. The control loop frequency must be high to prevent parasite vibrations uncomfortable to the user. In this PW we run the loop at 1kHZ, well above the 400Hz sensible limit.

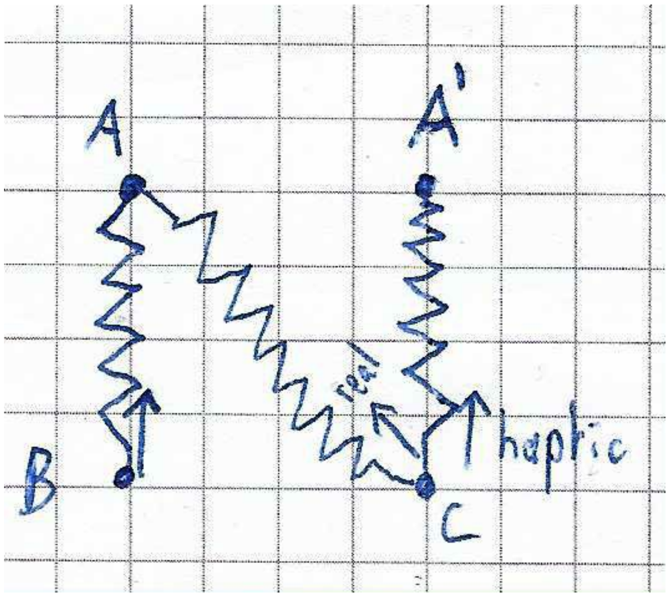

Figure 3: Spring scheme used for the discussion.

4 Discussion

Haptic devices present pros and cons related to their use. On one side, they allow the user to get some feedback out of the device, faking a touch sensation. In our practical this was not so relevant, but we could imagine the use of this technology in teleoperating a robot for example. On the other side, their mechanical constraints can shrink the workspace and the need for high bandwidth and low power consumption implies the use of small motors that can't always provide sufficient torque for an application.

Those devices can be compared with a regular mechanical spring as they have an oscillatory regime that tends to push them toward the resting position. Using the figure 3, if a regular spring is attached to point A and we pull it to B, the force will be toward A. If we pull it to C, the force will still be toward A. Thus a regular spring always pulls toward its resting position. However, the 'fake' spring generated by the haptic device has another behavior. If we virtually attach it to A and pull it to C, the force will be upwards but towards a new resting point : A'. The behaviors are thus different due to the construct of the haptic device but there is an area in the workspace in which for range of forces the devices behaves like a spring.

5 Conclusion

In conclusion, we were glad to attend this practical work. We discovered the basics of haptic devices robots : their concept that was unknown to some of us, especially their mechanical constraints and how to generate simple force commands. We never used such an haptic device and the sensations generated were very interesting !

We would have appreciated a little bit more context for this practical. It seems like we made some measurements and played with the device a little but maybe actually putting the device to use for something might have been more interesting to us. For example, feeling a virtual surface on the computer through the force feedback of the haptic device would have been great.

6 Annexes

In this section you will find the matlab script used to generate the C code to compute the position of point P as well as for the motor torques. You will also find the main C code for the motorboard. Note that we left the C code as generated by matlab although it is hard to read.

Matlab Script :

```
% HAPTICS Practical Work
% function P = haptics(phi_A, phi_B, LENGTH_A, LENGTH_B, LENGTH_C)
    syms phi_A phi_B LENGTH_A LENGTH_B LENGTH_C

    P1 = [LENGTH_A * sin(phi_A); 0.5*LENGTH_C - LENGTH_A*cos(phi_A)];
    P2 = [LENGTH_A * sin(phi_B); -0.5*LENGTH_C - LENGTH_A*cos(phi_B)];

    %PM = (P1 + P2)/2;

    alpha = atan2(P1(1) - P2(1), P1(2) - P2(2));
    theta = acos(0.5*norm(P1-P2)/LENGTH_B);

    P = P2 + LENGTH_B*[sin(theta+alpha); cos(theta+alpha)];
    ccode(P, 'File', 'P.c')

% end

% function [tau_A, tau_B] = haptics2(phi_A, phi_B, LENGTH_A, LENGTH_B,
LENGTH_C, F_x, F_y)
    syms phi_A phi_B LENGTH_A LENGTH_B LENGTH_C F_x F_y

    P1 = [LENGTH_A * sin(phi_A); 0.5*LENGTH_C - LENGTH_A*cos(phi_A)];
    P2 = [LENGTH_A * sin(phi_B); -0.5*LENGTH_C - LENGTH_A*cos(phi_B)];

    alpha = atan2(P1(1) - P2(1), P1(2) - P2(2));
    theta = acos(0.5*norm(P1-P2)/LENGTH_B);

    Fxy = [F_x;F_y];

    psi1 = theta - alpha - pi/2;
    psi2 = pi/2 - theta - alpha;
    M = inv([cos(psi1), cos(psi2);
             sin(psi1), sin(psi2)]);

   Fu = M*Fxy;
```

```
    tau_A = -Fu(1) * LENGTH_A * sin(phi_A - (theta - alpha));
    tau_B = Fu(2) * LENGTH_A * sin(phi_B + theta + alpha);

    ccode(tau_A, 'File', 'TauA.c')
    ccode(tau_B, 'File', 'TauB.c')
% end
```

Motorboard main :

```
#include "motorboard_main.h"
#include "communication.h"
#include "motor_regulator.h"
#include "safety_monitor.h"
#include "trajectory_interpolator.h"
#include "drivers/adc.h"
#include "drivers/callback_timers.h"
#include "drivers/fan.h"
#include "drivers/h_bridge.h"
#include "drivers/hall_resolver.h"
#include "drivers/incr_encoder.h"
#include "drivers/input_mosfet.h"
#include "drivers/led.h"
#include "drivers/spi.h"
#include "drivers/adc124s101.h"
#include "drivers/i2c.h"
#include "drivers/relays.h"
#include "lib/potentiometer.h"
#include "lib/thermistor.h"
#include "lib/utils.h"

#define CPU_COUNTER_IDLE_VALUE 35997.0f // Value measured with a special firmware with everything deactivated.

volatile uint64_t timestamp; // [us]

//********* Haptic pantograph constants *********
#define LENGTH_A 102.f          // Link a of the pantograph (directly connected to the motor) [mm]
#define LENGTH_B 111.f          // Link b of the pantograph (connected to the end point) [mm]
#define LENGTH_C 60.f           // Distance c (between the two motors) [mm]

//********* Haptic pantograph controller variables initialization *********
float32_t endPointXPosition = 0.0f;                           // X coordinate of the pantograph end point position [mm]
```

```
float32_t endPointYPosition = 0.0f;                          // Y coordinate of the pantograph end point position [mm]
volatile int32_t X_ref = 0;                                   // X coordinate of the reference (equilibrium) position of the spring [mm]
volatile int32_t Y_ref = 0;                                   // Y coordinate of the reference (equilibrium) position of the spring [mm]
volatile float32_t motorATorqueCommand = 0.0f;  // Calculated torque command for motor A [mN.m]
volatile float32_t motorBTorqueCommand = 0.0f;  // Calculated torque command for motor B [mN.m]
volatile float32_t F_x = 0.0f;                                // X component of the end point force [N]
volatile float32_t F_y = 0.0f;                                // Y component of the end point force [N]
volatile float32_t K_x = 0.0f;                                // X spring stiffness [N/mm]
volatile float32_t K_y = 0.0f;                                // Y spring stiffness [N/mm]

/**
 * @brief Update the current regulators loops.
 * @remark This function is called by an interrupt routine at a high rate (10
 * kHz), so all operations performed in this function should be as quick as
 * possible.
 */
void StepCurrentLoop(void)
{
    float32_t dt;

    dbgio_Set(&dbgio_1, GPIO_PIN_SET);

    // Increment the timestamp.
    timestamp += (uint64_t)cbt_GetTimerPeriod(&cbt_currentTimer);

    // Acquire all the ADC channels (except the phase current sensors).
    adc_AcquireAllChannels();

    // Step the current regulation.
    dt = MICROSECOND_TO_SECOND_F((float32_t)cbt_GetTimerPeriod(&cbt_currentTimer));
    mr_StepCurrentRegulation(&mr_motorA, dt);
    mr_StepCurrentRegulation(&mr_motorB, dt);
```

```c

    dbgio_Set(&dbgio_1, GPIO_PIN_RESET);
}

/**
 * @brief Update the main loop of the haptic pantograph controller.
 * @remark This is function is called by an interrupt routine at 1 KHz.
 */
void StepMainLoop(void)
{
    dbgio_Set(&dbgio_2, GPIO_PIN_SET);


    // You can use this function to send messages to be displayed in the debug terminal of the PC
    //comm_SendDebugMessageDecimated(1000, "Test message\n");

    // Read motor angles and speeds
    float phi_A = mr_motorA.currentMotorAngle * PI / 180.f;         // [rad]
    float phi_B = mr_motorB.currentMotorAngle * PI / 180.f;         // [rad]

    // Forward kinematics -- **** YOUR CODE HERE ****
    endPointXPosition = LENGTH_B*sin(atan2(LENGTH_A*sin(phi_A)-LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))+LENGTH_A*sin(phi_B);
    endPointYPosition = LENGTH_C*(-1.0/2.0)+LENGTH_B*cos(atan2(LENGTH_A*sin(phi_A)-LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))-LENGTH_A*cos(phi_B);

    // Calculate the end point force to render virtual environment -- **** YOUR CODE HERE ****
    F_x = K_x * (X_ref - endPointXPosition);
    F_y = K_y * (Y_ref - endPointYPosition);

    // Generate torque commands from end point force -- **** YOUR CODE HERE ****
    motorATorqueCommand = LENGTH_A*sin(phi_A+atan2(LENGTH_A*sin(phi_A)-LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-
```

```
LENGTH_A*sin(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(
phi_A)+LENGTH_A*cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*((F_y
*cos(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B)))/(cos
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*sin
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))-sin
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*cos
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B)))+(F_x*sin
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B)))/(cos
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*sin
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))-sin
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*cos
```

```
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))));
    motorBTorqueCommand = LENGTH_A*sin(phi_B+atan2(LENGTH_A*sin(
phi_A)-LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+
LENGTH_A*cos(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(
phi_A)+LENGTH_A*cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*((F_y
*cos(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B)))/(cos
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*sin
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))-sin
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*cos
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B)))+(F_x*sin
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B)))/(cos
(3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
(phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
(phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*sin
(3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
```

```
        (phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
        (phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
        cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))-sin
        (3.141592653589793*(-1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
        LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
        (phi_B))+acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
        (phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
        cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))*cos
        (3.141592653589793*(1.0/2.0)+atan2(LENGTH_A*sin(phi_A)-
        LENGTH_A*sin(phi_B),LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*cos
        (phi_B))-acos((sqrt(pow(fabs(LENGTH_A*sin(phi_A)-LENGTH_A*sin
        (phi_B)),2.0)+pow(fabs(LENGTH_C-LENGTH_A*cos(phi_A)+LENGTH_A*
        cos(phi_B)),2.0))*(1.0/2.0))/LENGTH_B))));

    mr_SetTorque(&mr_motorA, motorATorqueCommand);
    mr_SetTorque(&mr_motorB, motorBTorqueCommand);


    // Run a step of the position regulation loop (only to update
       motor position reading)
    mr_StepPositionRegulation(&mr_motorA, POS_DT, timestamp);
    mr_StepPositionRegulation(&mr_motorB, POS_DT, timestamp);

    // General logging to the PC.
    comm_SendPantographLog(timestamp, &mr_motorA, &mr_motorB);

    // Monitor for safety.
    saf_Step(timestamp, POS_DT);

    dbgio_Set(&dbgio_2, GPIO_PIN_RESET);
}

/**
  * @brief Main function.
  * Setups all the modules, and run a low-priority loop to update
     the
  * communication, the input mosfet manager and the trajectory
     generator.
  * @remark This function never returns.
  * @remark The low-level HAL (CubeMX) should be initialized before
     the call to
  * this function.
  */
void motorboardMain(void)
{
    uint32_t flashBorRegValue; // Brownout setting in the flash
       register.
```

```

    /*dt_Timer* cpuUsageTimer;
    uint32_t cpuUsageCounter;*/
#if HAS_POTENTIOMETERS
    float32_t potJointPosA, potJointPosB;
#endif

    //
    timestamp = 0;

    // Wait a small time to let the electronics stabilize, in case they have
    // just been powered up.
    utils_DelayMs(100);

    // Setup the peripherals.
    comm_Init();
    led_Init();
    adc_Init();
    im_Init();
    dbgio_Init();
    i2c_Init();

    // Setup the motors controllers and the input mosfet.
    mr_Init(&mr_motorA);
    mr_Init(&mr_motorB);

    // Start into a safe state.
    im_DisableVPower();

    // Start the regulation loops.
    cbt_Init(&cbt_currentTimer, StepCurrentLoop, (uint32_t)SECOND_TO_MICROSECOND_F(CURRENT_DT));
    cbt_Init(&cbt_mainLoopTimer, StepMainLoop, (uint32_t)SECOND_TO_MICROSECOND_F(POS_DT));


    // Initialize the encoders with the potentiometer, if available.
#if HAS_POTENTIOMETERS
    utils_DelayMs(100);
    potJointPosA = pot_Get(&pot_a);
    potJointPosB = pot_Get(&pot_b);
    enc_SetMotorShaftPosition(&enc_a, JOINTS_ANGLES_TO_MOTOR_ANGLE_A(potJointPosA, potJointPosB));
    enc_SetMotorShaftPosition(&enc_b, JOINTS_ANGLES_TO_MOTOR_ANGLE_B(potJointPosA, potJointPosB));
#elif HAS_ENCODERS
```

```
    enc_SetMotorShaftPosition(&enc_a, JOINTS_ANGLES_TO_MOTOR_ANGLE_A(0.0f, 0.0f));
    enc_SetMotorShaftPosition(&enc_b, JOINTS_ANGLES_TO_MOTOR_ANGLE_B(0.0f, 0.0f));
#endif


    // Check the flash option bytes.
    flashBorRegValue = ((FLASH->OPTCR & FLASH_OPTCR_BOR_LEV_Msk) >> FLASH_OPTCR_BOR_LEV_Pos);
    if(flashBorRegValue != 0x0)
    {
        comm_SendDebugMessage("Wrong flash option byte: BOR_LEV is 0x%x instead"
                              " of 0x%x.", flashBorRegValue, 0);
    }

    // Start the safety supervisor.
    saf_Init();

    //
    comm_SendDebugMessage("Motorboard ready to control %s.\r\n", HW_NAME);

        // Main loop.
    // All that is executing here has the lowest priority of execution, and is
    // not real-time (communication, various tests...).
        while(1)
        {
        int di;

        for(di=0; di<10; di++)
        {
            // Update the communication.
            comm_Step();

            // Update the input mosfet manager.
            im_StepInputMosfet(timestamp);


            // Measure CPU time.
            /*while(dt_GetDuration(cpuUsageTimer) < 5000)
                cpuUsageCounter++;

            bfilt_Step(&cpuUsageFilter, 1.0f - ((float32_t)cpuUsageCounter) / CPU_COUNTER_IDLE_VALUE);
```

```
                dt_Start(cpuUsageTimer);
                cpuUsageCounter = 0;*/
            }
        }
}
```