Master Project, 2016, Microengineering
Idiap Research Institute

Improving the Drawing Skills of a Humanoid Robot with Visual Feedback

Noémie Jaquier

Supervisor	Dr. Sylvain Calinon
Professor	Prof. Auke Ijspeert

June 17, 2016

Project Description

Baxter is a bimanual robot driven by series elastic actuators. The intrinsic compliance of the arms has the advantage that it is safe for the end-users sharing the robot's workspace. However, it comes at the expense of being less precise and less repeatable than a standard stiff industrial robot.

In this project, it is proposed to exploit the camera and sensors embedded within the robot to adjust the robot's arm movement and compensate for this imprecision. The vision and feedback capabilities will be tested in a drawing experiment.

The project aims at developing a controller reproducing a set of strokes by exploiting both force and position aspects, and using vision to evaluate the dissimilarity between planned and resulting strokes.

Idiap Research Institute Master Project Summary 17.06.2016

Improving the Drawing Skills of a Humanoid Robot with Vision Feedback

Noémie Jaquier, Microengineering Section

Assistant: *Dr. Sylvain Calinon*

Professor: *Prof. Auke Ijspeert*

The goal of this project is to develop a controller for the humanoid robot Baxter to reproduce a simple image with pen strokes while exploiting the compliance of the arms of the robot and the visual feedback provided by its cameras.

First, the drawing plane is estimated using the ViSP library and the image acquired by the robot's left hand camera. The black dots placed at each corner of the drawing zone are detected and tracked. The pose of the drawing plane is then estimated using Dementhon's approach for its initialisation and updated using a virtual visual servoing approach.

To evaluate this estimation, the drawing plane is estimated precisely with the intervention of the user. Moreover, the visual estimation is readjusted using the forces applied at the end-effector. Typical estimations are shown by Figure I. Compared to the reference, the visual estimation is shifted of a few centimetres along z-axis. However, it is parallel to the reference, conversely to the haptic estimation, that is finally not used.

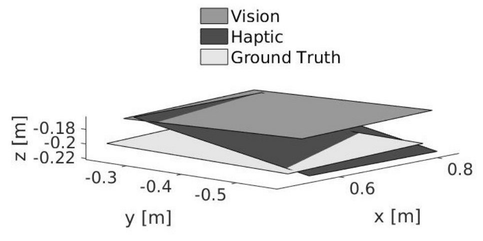


Figure I : Estimated drawing planes.

Two controllers are implemented to make Baxter draw with its right arm. The commanded velocity for the joints of the velocity controller is given by

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}_{pos}^{\dagger}(\dot{\boldsymbol{x}}_{\text{ff}}^{pos} + K_{ssp}\,\Delta\boldsymbol{x}_t^{pos}) + (\boldsymbol{I} - \boldsymbol{J}_{pos}^{\dagger}\boldsymbol{J}_{pos})((\boldsymbol{J}_{or,2}, \boldsymbol{J}_{or,3})^{\top})^{\dagger}\,K_{nsp}\,(\Delta\boldsymbol{x}_{t,2}^{or}, \Delta\boldsymbol{x}_{t,3}^{or})^{\top}.$$

The height of the end-effector relatively to the drawing plane is adapted in function of the force applied at the end-effector.

The torque controller is implemented with

$$\boldsymbol{\tau} = \boldsymbol{K}_p(\hat{\boldsymbol{q}}_t - \boldsymbol{q}_t) - \boldsymbol{K}_v\dot{\boldsymbol{q}}_t + \boldsymbol{J}^{\top}(\boldsymbol{q}_t)\boldsymbol{F} + \boldsymbol{G}(\boldsymbol{q})$$

The gains emulate the inertia matrix of the robot and are set as diagonal matrices. The gains of the second joint of the arm are designed so that it remains soft and helps to adapt the height of the end-effector relatively to the drawing plane.

Figure II shows an example of images drawn with each controller. The drawn shapes are evaluated visually and using match values computed with Hu moments. The images drawn with the velocity controller are often more accurate than those drawn with the torque controller. However, the obtained images are most of the time recognisable.

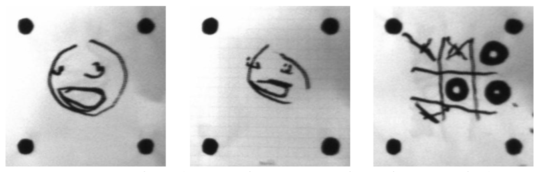


Figure II : smiley faces drawn with velocity (left) and torque (centre) controller and example of tic-tac-toe game (right) where Baxter draws the crosses.

A tic-tac-toe game is implemented using the minimax recursive algorithm. OpenCV functions are used to recognise the user's move. An example is show by Figure II.

The quality of the drawn images may be improved by computing precisely the inertia matrix and by using a controller in operational space. The camera may also be used to readjust the drawn strokes online.

Contents

1 Introduction **1**

2 Baxter Research SDK Robot **1**

3 Visual Detection and Estimation of the Drawing Plane **2**

3.1 Setup 2

3.2 Dots Detection and Tracking 3

3.3 Pose Estimation from Four Coplanar Points 4

3.3.1 Dementhon Approach 4

3.3.2 Virtual Visual Servoing Approach 5

3.3.3 Drawing Plane Pose Estimation 6

3.4 Drawing Plane in Robot Coordinates 6

4 Trajectory Generation **7**

4.1 Trajectory Generation from an Image 7

4.1.1 Preprocessing of the Image 7

4.1.2 Thinning Operation 7

4.1.3 Contours Extraction 8

4.1.4 Generation of Trajectories from Contours 8

4.2 From Drawing Plane to Robot Coordinates 9

5 Control of the Robot **10**

5.1 Representing the Pose of the End-Effector 10

5.1.1 Representing Orientation 10

5.1.2 Rotation Matrix 10

5.1.3 Euler Angles 11

5.1.4 Quaternions 12

5.2 Forward Kinematics 12

5.3 Inverse Kinematics 12

5.4 Inverse Kinematics Computation to Draw with Baxter 13

5.5 Joints Control 14

5.5.1 Simple Velocity Controller 14

5.5.2 Velocity Controller with Adaptive Height 14

5.5.3 Torque Controller 15

6 Visual Feedback **16**

6.1 Recovering the Performed Trajectory 16

6.2 Projection and Processing of the Drawing 16

6.3 Evaluation of the Dissimilarity between Expected, Planned and Drawn Strokes 17

7 Drawing's steps **18**

8 Results **19**
8.1 Drawing Realisation in Pictures . . . 19
8.2 Evaluation of the Visual Estimation of the Drawing Plane . . . 20
8.2.1 Estimation of the Reference . . . 20
8.2.2 Haptic Estimation of the Drawing Plane . . . 20
8.2.3 Evaluation of the Different Estimations of the Drawing Plane . . . 21
8.3 Evaluation of the Robot Controllers . . . 23
8.3.1 Velocity Controller . . . 23
8.3.2 Torque Controller . . . 27
8.3.3 Discussion and Comparison Between Controllers . . . 33

9 Play Tic-Tac-Toe with Baxter **34**
9.1 Game Description . . . 34
9.2 Process of a Game with Baxter . . . 35
9.3 Choice of Baxter's Next Move . . . 35
9.3.1 Minimax Algorithm . . . 35
9.3.2 Implementation of the Minimax Algorithm . . . 35
9.4 Detection of The Opponent's Move . . . 36
9.4.1 Preprocessing of the Images . . . 36
9.4.2 Canny Edges Detection . . . 36
9.4.3 Contours Extraction and Sorting . . . 38
9.4.4 Move Recognition . . . 39
9.5 Results . . . 39
9.5.1 A Game Realisation in Pictures . . . 39
9.5.2 Drawings of the Board . . . 40
9.5.3 Drawing of the Symbols . . . 40
9.5.4 Recognition of the Opponent's Move . . . 41
9.5.5 Discussion . . . 43

10 Conclusion and Future Work **43**

References **44**

1 Introduction

Drawing is an activity that we perform during our whole life. From the youngest age, toddlers already draw. This act of scribbling helps for example to develop their perception, exercise their cognitive abilities and improve their coordination and muscle control. Later, drawing capability is part of everyday life to explain notions with sketches or for artistic purposes. As humanoid robots are developed to be human-like, to behave similarly to humans and to collaborate and share social activities with them, it is relevant to study if we can provide them the capacity to draw.

Several robots already have the capacity to draw human portraits. For example, Paul is a three joints planar arm which produces observational drawings of people [1, 2]. Its sketches are considered as being of artistic value by professionals. A HOAP2 robot was also programmed to draw aesthetic portraits after having analysed the face of the model [3]. An other example of a drawing robot is E-David, an industrial robot that can use brushes or pencils to paint using several colours [4]. ISAC is a dual armed humanoid robot that can mimic an artist drawing simple shapes, taking advantage of its force sensors at the end-effector [5]. Moreover, programming robots to draw may have an educational purpose. For example, Standford students designed an artist robot turning JPEG images into sketches and a robot that wirelessly translates a sketch made by a tele-operator into strokes a white board [6].

The industrial manipulators used in these applications are often stiff and controlled by velocity commands, so that the drawing can be really precise if the robot is accurate enough. However, since there is no feedback about the drawing environment, a small inaccuracy in the modelling of the environment, in the model of the robot kinematic chain or in the control of the robot can result in the pen not being in contact with the paper or pushing strongly against the drawing plane. The system needs to be recalibrated if, for example, the drawing plane changed.

The goal of this project is to develop a controller for the humanoid robot Baxter to reproduce a simple image with pen strokes. The project aims to exploit the compliance of the arms of the robot and the visual feedback provided by the cameras to reproduce a set of strokes and evaluate the dissimilarity between the planned and resulting drawings.

ViSP, an open source Visual Servoing Platform library, was used to determine the drawing plane.[1] The extraction of desired strokes from the source image and the evaluation of their reproduction in the drawing were done using OpenCV functions. Two different controllers were implemented and tested on Baxter. Several experiments were realised on Baxter to evaluate the accuracy and efficiency of the visual estimations and the different controllers. Finally, a tic-tac-toe game was implemented to illustrate the drawing capabilities of the robot in an interactive way.

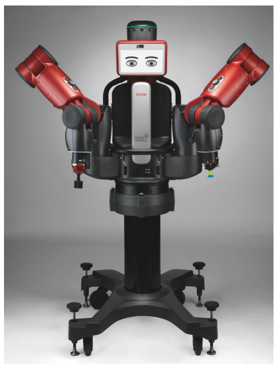
Section 2 of this report presents the Baxter robot. Section 3 describes the detection of the drawing plane and the computation of the corresponding referential. The generation of trajectories necessary to draw an image is presented in Section 4. Section 5 presents the control of the robot using inverse kinematics and two different control modes. The visual feedback and the evaluation of drawn strokes is shown in Section 6. Section 7 describes the computation steps required to draw with Baxter. The developed tools are then tested and evaluated in Section 8 and the tic-tac-toe game implementation is described by Section 9. Section 10 finally concludes the report and makes suggestions about future work.

2 Baxter Research SDK Robot

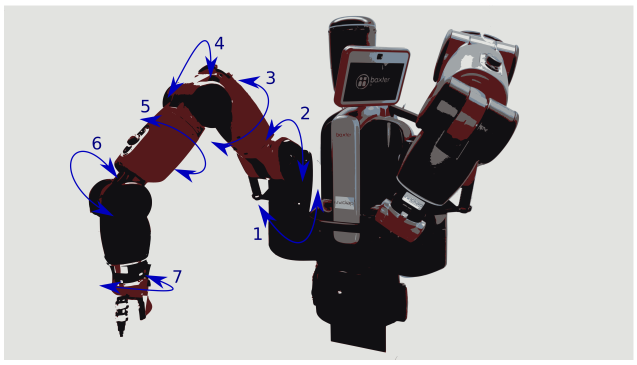
Baxter is a humanoid robot developed by Rethink Robotics.[2] Two pictures of the robot are shown in Figure 1. It possesses two 7 degrees of freedom arms driven by series elastic actuators. Thus, its arms are

[1]https://visp.inria.fr/
[2]http://www.rethinkrobotics.com/

intrinsically compliant, so that it is safer for the users to share the robot workspace. Baxter also has feedback capabilities thanks to the torque sensors that are embedded at each joint. The drawback of the intrinsic compliance is that the movements of the robot are less precise and less repeatable than those of a standard industrial robot. Four modes are currently available to control the arm joints: position, velocity, torque and raw position modes [7]. Grippers are placed as end-effector of each arm and can be used to hold objects.



(a) Baxter Research SDK Robot. Figure adapted from [8].



(b) Arm joints of Baxter.

Figure 1: Baxter pictures.

Baxter has three cameras, one at each hand and one on the head. Is also possesses four user interface panels composed of three push buttons and one indexing scroll wheel, two shoulders buttons and two buttons and one touch sensor at the cuff of each hand. A 3-axis accelerometer is located inside each cuff and each hand has two IR Range sensors.

Baxter's head can be rotated to left or right direction and down to make the robot nod. Twelve sonar distance sensors surrounded by yellow LEDs are mounted around the head of the robot.

Baxter is controlled via ROS topics and services.[3] Baxter's arms can be moved manually by pressing the cuff of the hand. In this case, a controller compensating the effect of gravity is activated and the arms can easily be placed by the user at a desired position.

3 Visual Detection and Estimation of the Drawing Plane

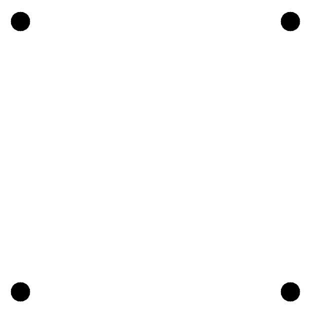
3.1 Setup

The drawing plane is detected and estimated using the ViSP library [9] with the image acquired by one of Baxter's cameras. ViSP, standing for Visual Servoing Platform, is an open source library providing tools to develop visual tracking and visual servoing applications with control laws applicable on robotic systems.[4]

As the positions of Baxter's arm joints are more precisely measured than the position of its head, one of the hand camera is used to acquire the image to detect the drawing plane. The right hand of the robot was chosen to hold the pen and draw in order to mimic human behaviour and to facilitate interactions with the user as most people are right-handed. The right hand camera may be hidden behind the pen, so that the left hand camera is used to acquire images of the drawings.

The drawing plane is placed in front of the robot and consists of a white sheet with one black dot at each corner of the drawing zone. An example of placement of the dots and of the general setup with Baxter are shown in Figure 2. For the experiments presented in this report, the drawing plane is placed horizontally. However, it could be rotated as it is detected and estimated before each drawing.

[3]http://www.ros.org/
[4]https://visp.inria.fr/


(a) Dots disposition on the white sheet.

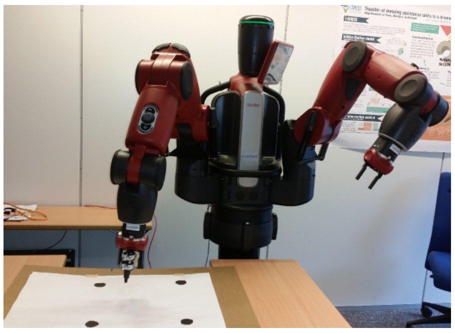

(b) General setup to make Baxter draw.

Figure 2: Disposition and placement of the drawing plane in front of the robot.

3.2 Dots Detection and Tracking

The parameters needed to fully describe a dot are the followings :

- Width;
- Height;
- Area;
- Size precision;
- Ellipsoid shape precision;
- Minimum grey level;
- Maximum grey level;
- Grey level precision.

Algorithm 1: Dots tracking

```
$N$ = required number of dots;
if first call of the function then
    initialisation = true;
end
if !initialisation then
    for dot $\leftarrow$ 1 to $N$ do
        Search dot in restricted area around the centre of gravity of the previous dot;
        if no dot found then
            initialisation = true;
        end
    end
    if less than $N$ dots successfully tracked then
        initialisation = true;
    end
end
if initialisation then
    Search dots in the whole image;
    Sort the detected dots;
end
```

To detect the dots with a set of desired features, each pixel of the image is tested to determine if it belongs to a valid dot. On the one hand, if the current pixel has a grey level outside the allowed interval or if it already belongs to a detected dot, there is no further check and the next pixel is analysed. On the other hand, if the pixel may belong to a undetected dot, a blob is formed with the neighbouring pixels of the correct colour. If its features correspond to the defined ones, the blob is added to the list of detected dots.

To track a dot, the detection algorithm is performed in an area around the last known position of the dot in the image. If one blob with correct features is found in this area, it is assumed that it is the displaced dot.

Algorithm 1 describes the tracking of the four dots on the drawing plane. The dots need to be ordered so that the pose of the drawing plane remains the same if the tracking is reinitialised. To respect the image processing convention, the dots are listed clockwise with the upper left one at the beginning of the list.

3.3 Pose Estimation from Four Coplanar Points

3.3.1 Dementhon Approach

Dementhon's approach estimates the object position and orientation in camera coordinates ${}^c\boldsymbol{M}_o \in \mathbb{R}^{4\times4}$ without previous initialisation [10, 11]. The matrix ${}^c\boldsymbol{M}_o$ is composed of a rotation matrix and a translation matrix, as shown by

$$ {}^c\boldsymbol{M}_o = \begin{pmatrix} \boldsymbol{R}^{3\times3} & \boldsymbol{t}^{3\times1} \\ \boldsymbol{0}^{1\times3} & 1 \end{pmatrix}. \tag{1}$$

The rows of the rotation matrix R are the coordinates of the unit vectors $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, $\boldsymbol{e}_3$ of the camera coordinate system expressed in the object coordinate system with $\boldsymbol{e}_3 = \boldsymbol{e}_1 \wedge \boldsymbol{e}_2$. The translation vector $\boldsymbol{t}$ is the position of the reference point $(X_o; Y_o; Z_o)$ for the object in camera coordinates.

Generally, the object is defined by four or more feature points M_i. Each point is defined with its 3D coordinates expressed in object frame of reference $({}^oX_i; {}^oY_i; {}^oZ_i)$ and its 2D coordinates expressed in the image frame of reference $(x_i; y_i)$.

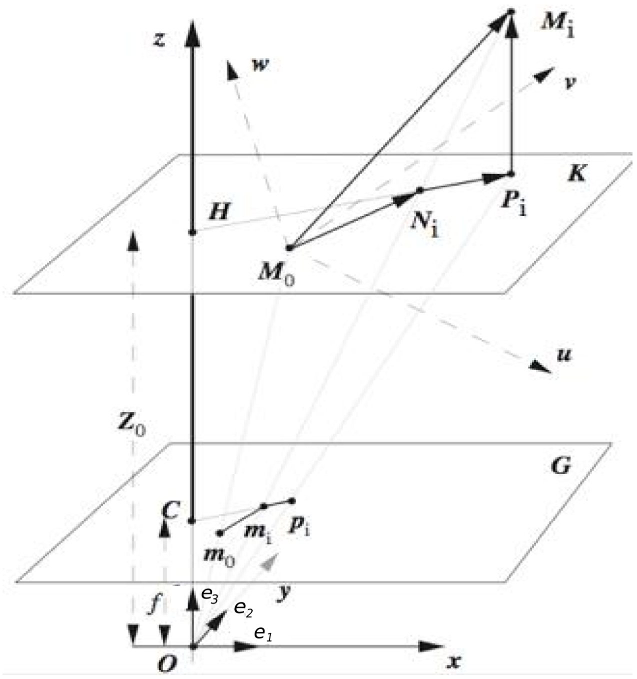


Figure 3: Perspective projection (m_i) and scaled orthographic projection (p_i) for an object point M_i and a reference point M_o. Figure adapted from [10, 11].

Figure 3 shows the classic pinhole camera model with the following elements:

O, the centre of projection of the camera;

$\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3$, the unit vectors of the camera coordinate system;

f, the focal length of the camera;

G, the image plane;

M_o, the reference point for the object;

M_i, the feature points of the object with coordinates $(X_i; Y_i; Z_i)$;

m_i, the image of points M_i by perspective projection with coordinates $(x_i; y_i)$;

p_i, the image of points M_i by a scaled orthographic projection with coordinates $(x'_i; y'_i)$.

The intrinsic parameters of the camera are known.

The object pose is fully described by $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, x_o, y_o and Z_o. An exact object pose is defined as an object pose for which the equalities

$$x_i = f\frac{X_i}{Z_i},\ y_i = f\frac{Y_i}{Z_i}, \tag{2}$$

are verified, so that the object points M_i fall on the lines of sight of the image point m_i. A necessary and sufficient condition for a pose to be an exact pose is that

$$\begin{array}{c} \boldsymbol{m}^\top \frac{f}{Z_o}\boldsymbol{e}_1 = x_i(1 + \frac{1}{Z_o}\boldsymbol{m}^\top \boldsymbol{e}_3) - x_o \\ \boldsymbol{m}^\top \frac{f}{Z_o}\boldsymbol{e}_2 = y_i(1 + \frac{1}{Z_o}\boldsymbol{m}^\top \boldsymbol{e}_3) - x_o \end{array} \tag{3}$$

are satisfied for all points M_i, where $\boldsymbol{m}^\top$ is the vector going from M_o to M_i.

The POSIT algorithm (Pose from Orthography and Scaling with Iterations), described by Algorithm 2 is then used to find the pose of the object. If the feature points are coplanar, two approximate poses are found. If one of those poses is behind the camera, the other one is chosen. Otherwise, the pose minimising the average distance between the actual image points (x_i, y_i) and the projected points from the computed pose $(x'_i, y'_i)^\top$ is kept. Thus, the pose is evaluated by computing the average distance between actual image points $(x_i; y_i)$ and the projected points from the computed pose $(x'_i, y'_i)^\top$.

Algorithm 2: POSIT

```
$\epsilon_i = 0$, $n = 1$;
while $\|\epsilon_{i(n)} - \epsilon_{i(n-1)}\| >$ threshold do
    Solve Equation 3 for $\boldsymbol{e}_1$, $\boldsymbol{e}_2$ and $Z_o$;
    Compute $\epsilon_i(n)$;
    $n++$;
end
```

Dementhon's approach is fully described for non-coplanar feature points in [10] and for coplanar feature points in [11]. This approach is implemented in ViSP.

3.3.2 Virtual Visual Servoing Approach

The virtual visual servoing approach estimates the object position and orientation in camera coordinates ${}^c\boldsymbol{M}o \in \mathbb{R}^{4\times4}$ considering the problem as a visual servoing problem.

The goal of visual servoing is to move a camera to observe an object at a given position and orientation in the image. The final position of the camera is reached by minimising the error between a desired state of image features and the current state of these features. The goal of the pose computation is to minimise the error between the observed image features $\boldsymbol{p}$ and the back-projection of these features $\boldsymbol{p}'$ according to the matrix ${}^c\boldsymbol{M}_o$. A virtual camera is thus moved from the initial pose to the final pose using a visual servoing control law in order to ensure this minimisation.

The error to be minimised is given by

$$\boldsymbol{e} = \boldsymbol{C}(\boldsymbol{p}' - \boldsymbol{p}), \tag{4}$$

where $\boldsymbol{C}$ is the combination matrix. $\boldsymbol{C}$ has to be chosen so that $\boldsymbol{C}\boldsymbol{L}_p$ is full rank, where $\boldsymbol{L}_p$ is the interaction matrix that links the image features $\boldsymbol{p}$ to the virtual camera velocity $\boldsymbol{v}_c$ as shown by

$$\dot{\boldsymbol{p}}' = \boldsymbol{L}_p \boldsymbol{v}_c \tag{5}$$

The following control law can be obtained by deriving Equation 4 and defining an exponentially decoupled decrease of the error $\boldsymbol{e}$.

$$\boldsymbol{v}_c = -\lambda \boldsymbol{L}_p^{\dagger} \boldsymbol{e} \tag{6}$$

In this case, $\boldsymbol{C}$ is equal to the pseudo-inverse of the interaction matrix $\boldsymbol{L}_p^{\dagger}$. λ is a proportional coefficient that determine the rate of the decay of the error.

The interaction matrix for one feature point is given by

$$\boldsymbol{L}_{pi} = \begin{pmatrix} -\frac{1}{Z_i} & 0 & \frac{x_i}{Z_i} & x_i y_i & -(1-x_i^2) & y_i \\ 0 & -\frac{1}{Z_i} & \frac{y_i}{Z_i} & (1-y_i^2) & -x_i y_i & -x_i \end{pmatrix}. \tag{7}$$

$(x, y)^\top$ are the 2D coordinates of the point expressed in the image frame of reference and $(X, Y, Z)^\top$ are its 3D coordinates expressed in camera frame of reference computed from the position of the point in object frame of reference and the estimation of ${}^c\boldsymbol{M}_o$ for the previous image. The interaction matrix has a row size equal to two times the number of feature points.

The virtual camera is then moved at velocity $\boldsymbol{\nu}_c$ resulting in the change of camera pose between two images given by an homogeneous matrix $\boldsymbol{M}$. Finally, the estimation of the pose is given by

$${}^c\boldsymbol{M}_{o(n+1)} = \boldsymbol{M}\,{}^c\boldsymbol{M}_{o(n)}, \tag{8}$$

where n is the image number.

The advantages of this algorithm is that it is simple and can consider various geometrical features to compute the pose by deriving the interaction matrix for these features. The drawback is that the convergence of the control law is not guaranteed for large errors $\|\boldsymbol{p}' - \boldsymbol{p}\|$. However, the motion between two successive images acquired at video rate is small enough to ensure the convergence, so that only the initialisation of the pose can be problematic.

The virtual visual servoing approach is fully described in [12]. It is also implemented in ViSP.

3.3.3 Drawing Plane Pose Estimation

The drawing plane pose is estimated only if the four dots have been correctly tracked by the dot tracking function. If the pose is computed for the first time, it is estimated with Dementhon approach to avoid convergence problems.

The estimation of the pose for the next images is computed using the virtual visual servoing approach and the last estimation is used to compute the current pose.

If one or several points are lost for a given number of frames during the tracking, the pose is initialised again with Dementhon approach.

3.4 Drawing Plane in Robot Coordinates

The matrix ${}^c\boldsymbol{M}_o$ allows to compute the drawing plane in camera coordinates. As Baxter is controlled with commands in the robot coordinates, it is necessary to compute the drawing plane in this coordinates with Equation 9. The matrices ${}^r\boldsymbol{M}_c \in \mathbb{R}^{4\times4}$ and ${}^r\boldsymbol{M}_o \in \mathbb{R}^{4\times4}$ represents the camera position and orientation in robot coordinates and the object position and orientation in robot coordinates, which can be computed with

$${}^r\boldsymbol{M}_o = {}^r\boldsymbol{M}_c\,{}^c\boldsymbol{M}_o \tag{9}$$

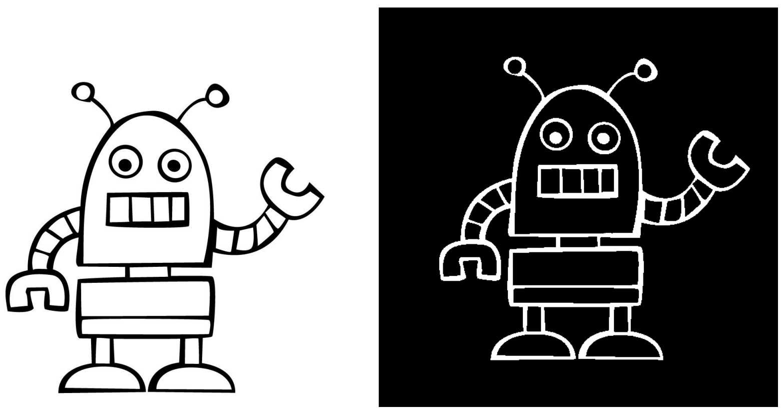
4 Trajectory Generation

4.1 Trajectory Generation from an Image

4.1.1 Preprocessing of the Image

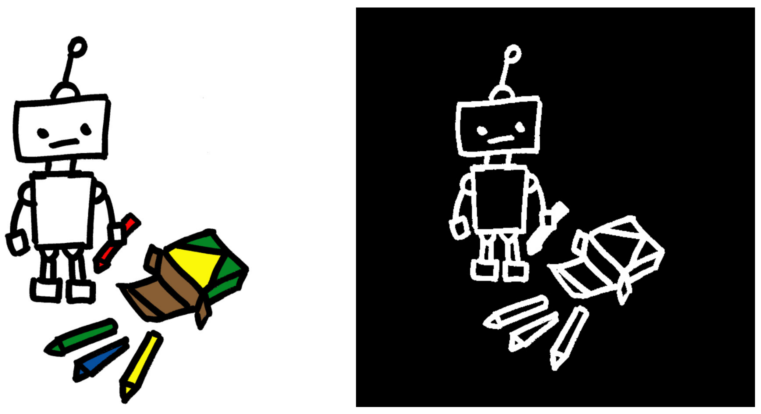
The images used to generate trajectories to be drawn are pictures with simple shapes as shown by Figures 4a and 5a.

The image is first converted from RGB to grayscale, and a threshold is then used to convert it to a binary image. Figures 4b and 5b show examples of thresholding. The threshold value was fixed at 70 for the two images.



(a) Original image. (b) Image after threshold operation.

Figure 4: Example of thresholding.



(a) Original image. (b) Image after threshold operation.

Figure 5: Example of thresholding.

4.1.2 Thinning Operation

After the thresholding operation, the image is thinned used Guo-Hall algorithm in order to extract the contours. The neighbourhood of a pixel p in the image is defined as shown by Table 1. Guo-Hall algorithm is a two-subiteration algorithm which preserves connectivity in the images and produces thin results while alternatively deleting north and east, then south and west boundary pixels. It is described by Algorithm 3, where $\neg$, $\land$ and $\lor$ refer respectively to logical complement, AND and OR.

Figures 6a and 7a show examples of thinned images. Guo-Hall algorithm is fully described in [13] and has been implemented for OpenCV in [14].

p_1	p_2	p_3
p_8	p	p_4
p_7	p_6	p_5

Table 1: Neighbourhood definition for pixel p.

```
Algorithm 3: Guo-Hall
while points are deleted do
    for all pixels p do
        $C(p) = \neg p_2 \wedge (p_3 \vee p_4) + \neg p_4 \wedge (p_5 \vee p_6) + \neg p_6 \wedge (p_7 \vee p_8) + \neg p_8 \wedge (p_1 \vee p_2)$;
        $N_1(p) = (p_1 \vee p_2) + (p_3 \vee p_4) + (p_5 \vee p_6) + (p_7 \vee p_8)$;
        $N_2(p) = (p_2 \vee p_3) + (p_4 \vee p_5) + (p_6 \vee p_7) + (p_8 \vee p_1)$;
        $N(p) = min(N_1(p), N_2(p))$;
        if {
            a. $C(p) = 1$
            b. $2 \leq N(p) \leq 3$
            c. Apply one of the following:
              1. $(p_2 \vee p_3 \vee \neg p_5) \wedge p_4 = 0$ in odd iterations
              2. $(p_6 \vee p_7 \vee \neg p_1) \wedge p_8 = 0$ in even iterations
        } then
            Delete pixel $p$;
        end
    end
end
```

4.1.3 Contours Extraction

After the thinning, the contours of the image are extracted using the function *findContours* of OpenCV [15]. To find the contour, all pixels of the image are scanned. If a pixel is white, it belongs to a new border and it is considered at the start point. The border is then followed by turning clockwise around the start pixel considering its Moore neighbourhood, until an other white pixel is found. This pixel is marked as a border and the border following continues by turning clockwise around it. When the start pixel is visited a second time, the detection of the current contour is done and the scan of the image continue. This algorithm is explained in [16].

The function is first used to extract the external contours in shown by Figures 6b and 7b. Those contours are then removed from the original image and the internal contours are extracted as shown in Figures 6c and 7c. This two-steps extraction avoid having the same part of the image included in two or more different contours.

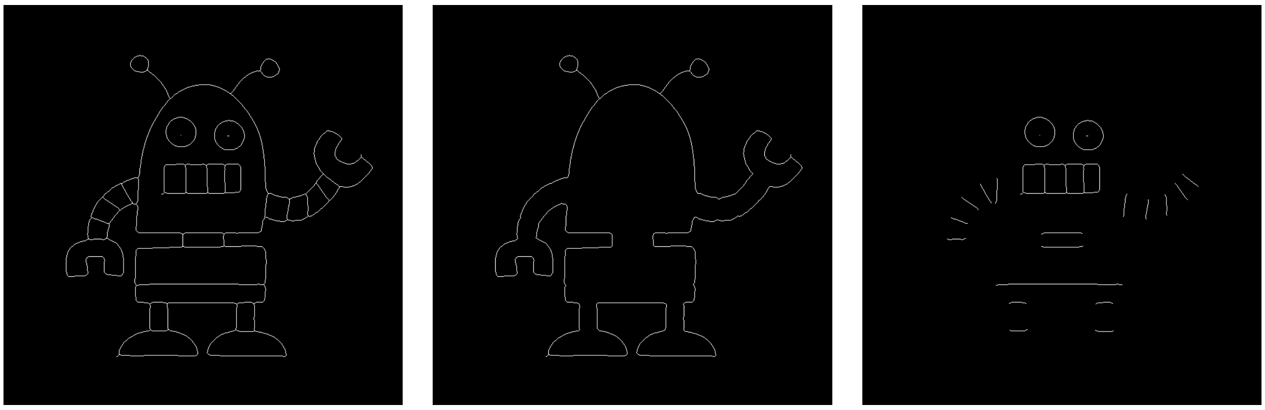
4.1.4 Generation of Trajectories from Contours

The function *findContours* of OpenCV stores the contours as vectors of points. Each point is transformed into a trajectory point using

$$\begin{aligned} x &= (u - w/2)r, \\ y &= (v - h/2)r, \end{aligned} \tag{10}$$

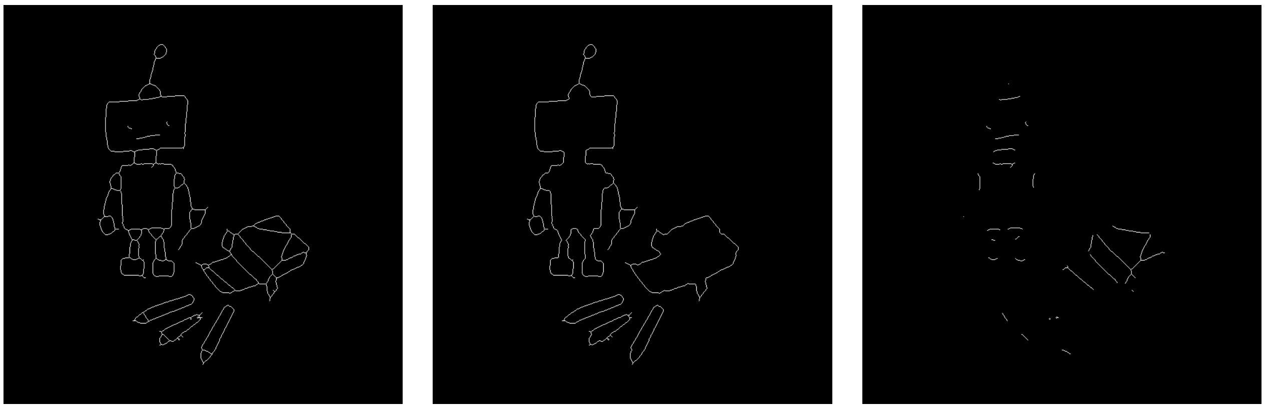
where $(x, y)^\top$ are the coordinates of the point in the drawing plane, $(u, v)^\top$ are the pixels coordinates of the point in the image, w and h are respectively the width and the height of the image in pixels and r is the length of one pixel in meter defined in function of the desired size of the drawn image.

As one trajectory point is given to the robot per control cycle, the distance between two points determines the drawing speed. The maximum speed is given by $v = d_{\max} f$, where $d_{\max} = \sqrt{dx^2_{\max} + dy^2_{\max}}$ is the maximum



(a) Thinned image. (b) External contours. (c) Internal contours.

Figure 6: Example of thinning and contours extraction.



(a) Thinned image. (b) External contours. (c) Internal contours.

Figure 7: Example of thinning and contours extraction.

distance between two points and $f = 500Hz$ is the frequency of the control loop of the robot. The maximum drawing speed is chosen to be reasonable for the robot and to have visually a smooth movement. Thus, if the distance along x or y between two trajectory points is above $dx_{\max}$ or $dy_{\max}$, computed with the maximum drawing speed, new trajectory points are added in-between to keep the drawing speed below the maximum speed.

4.2 From Drawing Plane to Robot Coordinates

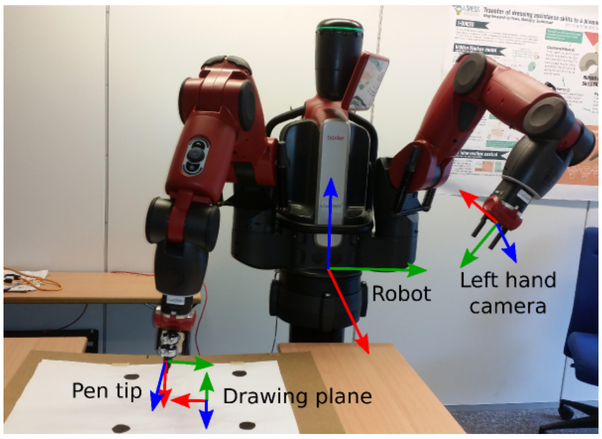
The coordinates of the computed trajectory points then need to be transferred into robot coordinates. Figure 8a shows the coordinate systems of the drawing plane, the left hand camera, the pen and the robot. The pen frame of reference is a shifted version of the right gripper frame of reference, as the pen is aligned with the gripper.

The desired orientation of the robot's arm during the drawing is shown in Figure 8b. This arm configuration was chosen because Baxter is designed to manipulate objects in this configuration, and the gravity is better compensated around this pose (i.e., the modeling and calibration procedure have been optimized for this configuration).

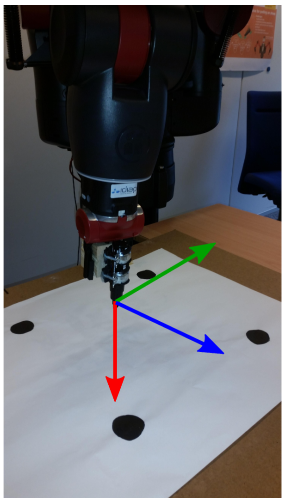
The coordinates of a trajectory point in the drawing plane coordinates can be expressed by ${}^{o}\boldsymbol{X} = \begin{pmatrix} \boldsymbol{I}^{3\times3} & \boldsymbol{t}^{3\times1} \\ \boldsymbol{0}^{1\times3} & 1 \end{pmatrix}$ with $\boldsymbol{t} = (x, y, z)^{\top}$ and $z = 0$ as the trajectory lies on the drawing plane. The homogeneous matrix corresponding to the coordinates of this trajectory point in robot frame of reference is given by

$$^{r}\boldsymbol{X} = \begin{pmatrix} \boldsymbol{R}_z(\pi/2) & \boldsymbol{0} \\ \boldsymbol{0} & 1 \end{pmatrix} {}^{r}\boldsymbol{M}_o \, {}^{o}\boldsymbol{X}. \tag{11}$$

The rotation of $\pi/2$ around z-axis of the drawing plane is necessary due to the conventions of the different frames of reference.



(a) Frames of reference of Baxter: robot, left hand camera, pen tip and drawing plane.



(b) Orientation of the arm relative to the drawing plane.

Figure 8: Position and orientation of different frames of reference : x, y and z axis are red, green and blue.

5 Control of the Robot

5.1 Representing the Pose of the End-Effector

5.1.1 Representing Orientation

The position and orientation of a rigid body define its pose. In this case, the goal is to define the pose of the end-effector of the robot, namely the pen tip, expressed in robot coordinates (see Figure 8a). The position of the end-effector is commonly represented with Cartesian coordinates $(x, y, z)^\top$. However, its orientation is represented in various forms, including rotation matrix, Euler angles or quaternions [17].

5.1.2 Rotation Matrix

A rotation matrix is an orthogonal matrix with a determinant equal to 1. The columns of the rotation matrix are the basis vectors of the end-effector coordinates expressed in robot coordinates. Moreover, its rows are the basis vectors of the robot coordinates expressed in end-effector coordinates.

To represent a pose, the rotation matrix can be integrated into homogeneous matrices ${}^{e}\boldsymbol{M}_r$ and ${}^{r}\boldsymbol{M}_e$ defining respectively the pose of the robot in end-effector coordinates and the pose of the end-effector in robot coordinates. These two matrices are

$$\begin{aligned} {}^{e}\boldsymbol{M}_r &= \begin{pmatrix} R^{3\times3} & {}^{e}O_r^{3\times1} \\ 0^{1\times3} & 1 \end{pmatrix}, \\ {}^{r}\boldsymbol{M}_e &= \begin{pmatrix} R^{\top\,3\times3} & {}^{r}O_e^{3\times1} \\ 0^{1\times3} & 1 \end{pmatrix}, \end{aligned} \tag{12}$$

where ${}^{e}O_r$ and ${}^{r}O_e$ are respectively the origin of the robot frame of reference in end-effector coordinates and the origin of the end-effector frame of reference in robot coordinates. The multiplication of these two matrices is equal to the identity matrix.

The advantage of homogeneous matrices is that the multiplication of two homogeneous matrices yields another homogeneous matrix whose application to a point is the same than the successive applications of the two original matrices.

5.1.3 Euler Angles

A common way to represent the orientation is to use a vector of three Euler angles $(\phi, \theta, \psi)^\top$. A vector of three Euler angle describes a sequence of three coordinates rotations $\boldsymbol{R}_{ijk} = \boldsymbol{R}_i(\phi)\boldsymbol{R}_j(\theta)\boldsymbol{R}_k(\psi)$. A valid rotation sequence is able to span the space of all three dimensional rotations. A valid rotation sequence often used is the sequence $(1, 2, 3)$ for which ϕ, θ and ψ are also called *roll*, *pitch* and *yaw*. This rotation sequence is shown by Figure 9.

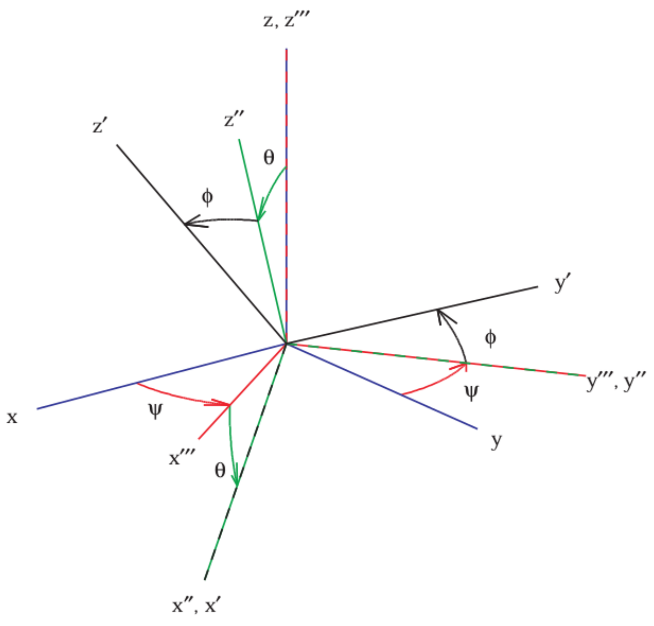


Figure 9: Representation of Euler angles $(1, 2, 3)$ sequence. Figure adapted from [17].

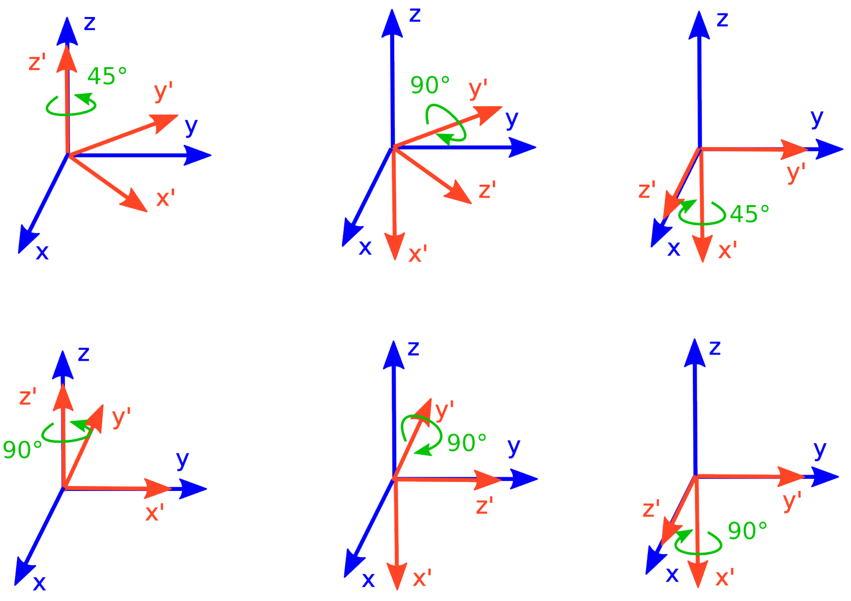


Figure 10: Example of two Euler sequences describing the same orientation.

Thus, the pose of a rigid body can be described by $\boldsymbol{x} = (x, y, z, \phi, \theta, \psi)^\top$.

The main advantage of Euler angles is that they are easy to interpret. Thus, they are very popular. However, some important functions of these angles have singularities. These singularities arise when the second Euler is at a critical value and make the fist and third Euler angles indistinguishable. For example, if the pitch angle of the sequence (1,2,3) is equal to 90 deg, a roll and yaw angles of 45 deg describe the same orientation than a roll and yaw angles of 90 deg.

5.1.4 Quaternions

Quaternions are a generalisation of complex numbers. A quaternion is written as $q = q_w + q_x i + q_y j + q_z k$ or $\boldsymbol{q} = (q_x, q_y, q_z, q_w)^\top$. The inverse of a quaternion is $q^{-1} = (-q_x, -q_y, -q_z, q_w)^\top$. Moreover, quaternion multiplication is not commutative. The result of the multiplication between $\boldsymbol{q}$ and $\boldsymbol{p}$ is given by

$$\boldsymbol{qp} = \begin{pmatrix} q_w p_w - (q_x, q_y, q_z)(p_x, p_y, p_z)^\top \\ q_w (p_x, p_y, p_z)^\top + p_w (q_x, q_y, q_z)^\top - (q_x, q_y, q_z)^\top \wedge (p_x, p_y, p_z)^\top \end{pmatrix} \tag{13}$$

and the difference between two quaternions is $\boldsymbol{qp}^{-1}$.

A unit quaternion has a norm $\|\boldsymbol{q}\| = \sqrt{q_w^2 + q_x^2 + q_y^2 + q_z^2}$ equal to 1. It can be used to represent the orientation of a rigid body as it is equivalent to a pure rotation.

The rotation of an object by the result of the multiplication of two unit quaternions is equivalent to the rotation of the object by the first quaternion followed by the rotation by the second one. Thus, sequences of rotation can be represented by unit quaternions. Similarly, the difference between two unit quaternions is equivalent to the rotation performed to go from the first quaternion to the second. Thus, the pose of a rigid body can be described by $\boldsymbol{x} = (x, y, z, q_x, q_y, q_z, q_w)^\top$.

A rotation matrix can be computed from a unit quaternion as shown by

$$\boldsymbol{R}_q(\boldsymbol{q}) = \begin{pmatrix} q_w q_w + q_x q_x - q_y q_y - q_z q_z & 2q_x q_y - 2q_w q_z & 2q_w q_y + 2q_x q_z \\ 2q_w q_z + 2q_x q_y & q_w q_w - q_x q_x + q_y q_y - q_z z & 2q_y q_z - 2q_w q_x \\ 2q_x q_z - 2q_w q_y & 2q_w q_x + 2q_y q_z & q_w q_w - q_x q_x - q_y q_y + q_z q_z \end{pmatrix}. \tag{14}$$

A rotation vector can also be computed from a unit quaternion, namely

$$\boldsymbol{v}(\boldsymbol{q}) = 2 \arccos \left(\frac{(q_x, q_y, q_z)^\top}{\|(q_x, q_y, q_z)^\top\|} \right). \tag{15}$$

The rotation vector representation is similar to the axis-angle representation as it gives a rotation axis and an angle. The axis of rotation is given by the direction of the vector and the angle of rotation is defined by its amplitude.

The advantages of using unit quaternions to represent the orientation of a rigid body are that no singularity occurs in their important functions and that they are well-suited to integrate the velocity of a rigid body over the time. The main disadvantage is that they have no intuitive physical meaning.

5.2 Forward Kinematics

The kinematic chain is a mathematical model in which rigid bodies, or links, are connected by joints. Kinematic chains of serial robots, like Baxter's arm, are formed by links connected in series. The joint variables $(q_1, q_2, ..., q_n)^\top \in \mathbb{R}^n$ of the robot are the linear or angular inputs sent to the actuators. In the case of Baxter's arms, $n = 7$ for each arm. The operational variables define the pose of the end-effector. The forward kinematics express the operational variables $(x_1, x_2, ..., x_m)^\top \in \mathbb{R}^m$, often equivalent to $(x, y, z, \phi, \psi, \theta)^\top$ with $m = 6$, in function of the joint variables, so that $\boldsymbol{x} = \mathrm{FK}(\boldsymbol{q})$.

5.3 Inverse Kinematics

The inverse kinematics conversely expresses the joint variables in function of the operational variables, so that $\boldsymbol{q} = \mathrm{IK}(\boldsymbol{x})$. For serial robots, the inverse kinematics can have more than one solution. There are thus more than one configuration for some position of the end-effector and inverse kinematics are complex to solve.

The robot Jacobian $\boldsymbol{J}(\boldsymbol{q}) \in \mathbb{R}^{m \times n}$ relates the joint velocities $\dot{\boldsymbol{q}}$ to the linear and angular velocities $\dot{\boldsymbol{x}}$ of the end-effector as shown by

$$\dot{\boldsymbol{x}} = \boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} \text{ with } \boldsymbol{J}(\boldsymbol{q}) = \frac{\delta}{\delta \boldsymbol{q}} \mathrm{FK}|_{\boldsymbol{q}}. \tag{16}$$

$J(\boldsymbol{q})$ will further be written as J in order to make the equations lighter. The number of solutions of Equation 16 is determined by the rank of the Jacobian. If $\mathrm{rank}(\boldsymbol{J}(\boldsymbol{q})) < n$, there is no solution, if $\mathrm{rank}(\boldsymbol{J}(\boldsymbol{q})) = n$, the solution is unique and finally, if $\mathrm{rank}(\boldsymbol{J}(\boldsymbol{q})) > n$, there is a infinite number of solution forming the solution space. In this last case, the robot is redundant and a null space is defined by the solution to $\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} = 0$.

The Jacobian can be decomposed in two parts, one for position and one for orientation, as shown by

$$\boldsymbol{J} = \begin{pmatrix} \boldsymbol{J}_{\text{pos}} \\ \boldsymbol{J}_{\text{or}} \end{pmatrix}. \tag{17}$$

More generally, each row i of the Jacobian stands for the component x_i of $\boldsymbol{x}$.

The desired joint angles is computed at each time step to follow a trajectory $\hat{\boldsymbol{x}}$ in Cartesian space using

$$\boldsymbol{x}_t = \mathrm{FK}(\boldsymbol{q}_t), \tag{18}$$

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}^{\dagger}(\boldsymbol{q}_t)\, K_{\text{ssp}}\, (\hat{\boldsymbol{x}}_t - \boldsymbol{x}_t) + \boldsymbol{N}(\boldsymbol{q}_t)\, K_{\text{nsp}}\, g(\boldsymbol{q}_t), \tag{19}$$

$$\hat{\boldsymbol{q}}_{t+1} = \boldsymbol{q}_t + \hat{\dot{\boldsymbol{q}}}_t dt. \tag{20}$$

Equation 18 computes the pose of the end-effector at time t in function of the joint angles using the forward kinematics. Then, Equation 19 computes the desired angular velocities of the joints and Equation 20 updates the desired joint angles.

$\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{\top}(\boldsymbol{J}\boldsymbol{J}^{\top})^{-1}$ is the Moore-Penrose pseudo-inverse of the Jacobian. The Moore-Penrose pseudo-inverse provides a least-square solution to a system of linear equations $\boldsymbol{J}\dot{\boldsymbol{q}} = \dot{\boldsymbol{x}}$. It aims to minimise the sum of squared residuals $\|\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}\|^2$ with respect to the vector $\dot{\boldsymbol{q}}$. The damped pseudo-inverse provides a regularised least-square solution to a system of linear equations and aims to minimise $\|\boldsymbol{J}\dot{\boldsymbol{q}} - \dot{\boldsymbol{x}}\|^2 + \|\lambda \boldsymbol{I}\|$, where $\boldsymbol{I} \in \mathbb{R}^{mxm}$ is the identity matrix and λ is a scalar. In robotics, the term $\boldsymbol{J}\boldsymbol{J}^{\top}$ is small when the robot is close to a singularity and thus, it is difficult to invert. Using the damped Moore-Penrose pseudo-inverse allows to avoid the singularities. Thus, in Equation 19, the damped pseudo-inverse is used, so that $\boldsymbol{J}^{\dagger} = \boldsymbol{J}^{\top}(\boldsymbol{J}\boldsymbol{J}^{\top} + \lambda \boldsymbol{I})^{-1}$, with λ a small factor.

In Equation 19, $\boldsymbol{N}(\hat{\boldsymbol{q}}_t)$ is the null space $(\boldsymbol{I} - \boldsymbol{J}^{\dagger}\boldsymbol{J})$ of the Jacobian and $g(\hat{\boldsymbol{q}}_t)$ is a function to optimise. Any vector $\boldsymbol{N}(\hat{\boldsymbol{q}}_t)g(\hat{\boldsymbol{q}}_t)$ added to a vector of the solution space yields a new vector belonging to the solution space.

K_{ssp} and K_{nsp} are gains determining respectively the importance of the solution space term and of the null space term.

5.4 Inverse Kinematics Computation to Draw with Baxter

To draw with Baxter, the Equations 18, 19 and 20 are used to compute the inverse kinematics, but they are slightly adapted to serve this specific purpose. In the following equations, $\boldsymbol{x}'$ is defined as the pose described with Cartesian positions and quaternions $(x, y, z, q_x, q_y, q_z, q_w)^{\top}$ and $\boldsymbol{x}$ is defined as the pose described with Cartesian positions and Euler angles $(x, y, z, \phi, \psi, \theta)^{\top}$.

The trajectory points are obtained as described in Section 4. The trajectory point ${}^{r}X$ of Equation 11 can also be described by $\hat{\boldsymbol{x}}'_t = (\hat{x}, \hat{y}, \hat{z}, \hat{q}_x, \hat{q}_y, \hat{q}_z, \hat{q}_w)^{\top}$ by transforming the rotation matrix into a quaternion vector.

In order to follow the drawing trajectory $\hat{\boldsymbol{x}}'^{d}$, the forwards kinematics are first used at each time step to compute the pose at time t with

$$\boldsymbol{x}'_t = \mathrm{FK}(\boldsymbol{q}_t). \tag{21}$$

The difference $\Delta \boldsymbol{x}_t$ between the pose given by the trajectory and the current pose is then computed using

$$\Delta \boldsymbol{x}_t = \begin{pmatrix} \Delta \boldsymbol{x}_t^{\text{pos}} \\ \Delta \boldsymbol{x}_t^{\text{or}} \end{pmatrix} = \begin{pmatrix} (\hat{x}, \hat{y}, \hat{z})_t^{\top} - (x, y, z)_t^{\top} \\ \boldsymbol{v}\left((\hat{q}_x, \hat{q}_y, \hat{q}_z, \hat{q}_w)_t^{\top}((q_x, q_y, q_z, q_w)_t^{\top})^{-1}\right) \end{pmatrix}, \tag{22}$$

where $\boldsymbol{v}$ is the rotation vector computed from the quaternion $\boldsymbol{q}$ and $\boldsymbol{q}^{-1}$ is the inverse of a quaternion as described at the beginning of the section.

The first part of the matrix is the difference between the Cartesian positions of the two vectors and the second part is the rotation vector obtained from the difference of the quaternions of the two vectors.

When one is drawing, the most important component of the pose is the position of the pen tip. The orientation should be maintained, so that the pen tip can be in contact with the sheet. Moreover, small variations around this vertical position can be considered without altering the drawing, and the pen can be rotated around the vertical axis without any consequence on the drawing. Thus, the two last rows of the Jacobian for the orientation and the two last components of the difference in orientation $\boldsymbol{\Delta x}_t^{\text{or}}$ are used to compute the desired velocities.

The desired joint velocities are computed at time t while taking advantages of these information with

$$\hat{\dot{\boldsymbol{q}}}_t = \boldsymbol{J}_{pos}^{\dagger}(\dot{\boldsymbol{x}}_{\text{ff}}^{\text{pos}} + K_{\text{ssp}}\,\Delta\boldsymbol{x}_t^{\text{pos}}) + (\boldsymbol{I} - \boldsymbol{J}_{\text{pos}}^{\dagger}\boldsymbol{J}_{\text{pos}})((\boldsymbol{J}_{\text{or},2}, \boldsymbol{J}_{\text{or},3})^{\top})^{\dagger}\, K_{\text{nsp}}\,(\Delta\boldsymbol{x}_{t,2}^{\text{or}}, \Delta\boldsymbol{x}_{t,3}^{\text{or}})^{\top}. \tag{23}$$

Thus, only the position of the end-effector is considered in the velocity component in solution space. The feed-forward velocity $\dot{\boldsymbol{x}}_{\text{ff}}^{\text{pos}}$ from

$$\dot{\boldsymbol{x}}_{\text{ff}}^{\text{pos}} = \frac{(\hat{x}, \hat{y}, \hat{z})_t^{\top} - (x, y, z)_t^{\top}}{dt} \tag{24}$$

is also added to this term. This feed-forward term allows to reduce the error faster and to keep it smaller than a system relying only on feedback. Moreover, the row of the orientation Jacobian corresponding to the z-axis of the end-effector is deleted in order that it remains free. Finally, the other orientation constraints are treated in the velocity component of the null space, so that the orientation is maintained around the perfect orientation perpendicular to the sheet.

The obtained desired joint velocities are then clamped, so that $\|\hat{\dot{q}}_{t,i}\| < \dot{q}^{\max}$. Finally, the desired joint angles are updated using Equation 20.

5.5 Joints Control

5.5.1 Simple Velocity Controller

A velocity controller can be implemented for Baxter by giving a velocity command equal to $\hat{\dot{\boldsymbol{q}}}_t$ obtained with the inverse kinematics computation to the robot.

5.5.2 Velocity Controller with Adaptive Height

As the drawing plane may not be a perfect plane or may not be detected precisely enough, a controller that adapts the height in function of the measured force between the end-effector and the sheet is designed.

At each time step, the forces ${}^{r}\boldsymbol{F} = ({}^{r}F_x, {}^{r}F_y, {}^{r}F_z)^{\top}$ applied at the end-effector are measured by the robot through the torque sensors at the joints by assuming a single contact point at the pen tip. They are given in robot coordinates. As the sheet may not be aligned with the robot frame of reference, these forces are projected into the drawing plane frame of reference with

$${}^{o}\boldsymbol{F} = {}^{c}\boldsymbol{M}_o^{-1}\,{}^{r}\boldsymbol{M}_c^{-1}\,{}^{r}\boldsymbol{F}. \tag{25}$$

A maximum and minimum forces ${}^{o}F_z^{\max}$ and ${}^{o}F_z^{\min}$ of contact are defined to draw, so that the pen tip lies on the sheet by pressing against it with a reasonable force. Moreover, a maximum distance $\Delta z^{\max}$ between the current and initial positions along the z-axis of the drawing plane coordinates is selected. ${}^{o}F_z^{\max}$, ${}^{o}F_z^{\min}$ and $\Delta z^{\max}$ are selected empirically.

Algorithm 4 describes the update of the z-component of $\hat{\boldsymbol{x}}_t$ at time t in function of the projected force along z-axis of the drawing plane frame of reference. It is important to note that the pen tip goes toward the drawing plane if $\hat{z}'$ decreases (due to the orientation of the drawing plane frame of reference).

After the update of the trajectory point coordinates, the inverse kinematics are computed and the velocity command $\hat{\dot{\boldsymbol{q}}}_t$ is transmitted to the robot.

Algorithm 4: Adjustment of the height of the pen tip in function of its contact force with the sheet

```
$k$ a chosen gain;
$\hat{z}_t$ the z-component of $\hat{\boldsymbol{x}}_t$;
if ${}^oF_z > {}^oF_z^{\max}$ then
    if $\hat{z}_0 - \hat{z}_t < \Delta z^{\max}$ then
        $\hat{z}_t -= k({}^oF_z - {}^oF_z^{\max})$
    end
else
    if ${}^oF_z < {}^oF_z^{\min}$ then
        if $\hat{z}_t - \hat{z}_0 < \Delta z^{\max}$ then
            $\hat{z}_t += k({}^oF_z^{\min} - {}^oF_z)$
        end
end
```

5.5.3 Torque Controller

The dynamical model of the robot is described by

$$\boldsymbol{\tau} = \boldsymbol{H}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}}) + \boldsymbol{G}(\boldsymbol{q}), \tag{26}$$

where $\boldsymbol{H}(\boldsymbol{q})$ is the inertia matrix of the robot, $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ is the matrix due to Coriolis effect and centrifugal force, $\boldsymbol{G}(\boldsymbol{q})$ is the gravity component, $\boldsymbol{\tau} \in \mathbb{R}^{nx1}$ is the vector of torque applied to each joint and $\boldsymbol{q}$, $\dot{\boldsymbol{q}}$, $\ddot{\boldsymbol{q}}$ are respectively the current positions, velocities and accelerations of the robot's joints.

A torque controller was implemented with

$$\boldsymbol{\tau} = \boldsymbol{K}_p(\hat{\boldsymbol{q}}_t - \boldsymbol{q}_t) - \boldsymbol{K}_v\dot{\boldsymbol{q}}_t + \boldsymbol{J}^{\top}(\boldsymbol{q}_t)\boldsymbol{F} + \boldsymbol{G}(\boldsymbol{q}) \text{ with } \dot{\boldsymbol{q}}_t = \frac{\boldsymbol{q}_t - \boldsymbol{q}_{t-1}}{dt}, \tag{27}$$

where $\hat{\boldsymbol{q}}_t$ is the desired joint angles obtained by computing the inverse kinematics, $\boldsymbol{q}_t$ are the joint angles measured at time t, $\hat{\boldsymbol{q}}_t$ is the desired joint positions and $\dot{\boldsymbol{q}}_t$ are the computed current joint velocities. K_p and K_v are gains.

With the above controller, the influence of Coriolis effect and centrifugal force $\boldsymbol{C}(\boldsymbol{q}, \dot{\boldsymbol{q}})$ in (26) is assumed to be negligible. The inertia matrix $\boldsymbol{H}(\boldsymbol{q})$ computed with the model provided by Rethink Robotics is inaccurate and cannot be used in practice. A related effect is achieved by adjusting the gains K_p and K_v for each joint separately. The term $\boldsymbol{J}^{\top}(\boldsymbol{q}_t)\boldsymbol{F}$ is used to add a small torque corresponding to a force at the end-effector toward the drawing place, with $\boldsymbol{F} \in \mathbb{R}^{6\times1}$ a vector of forces and twists applied to the end-effector. A force pushing the end-effector in the direction of the drawing plane is defined so that ${}^r\boldsymbol{F} = {}^r\boldsymbol{M}_o{}^o\boldsymbol{F}$ with ${}^o\boldsymbol{F} = (0, 0, F_z, 0, 0, 0)^{\top}$. Before being transmitted to the robot, the computed torques are clamped, so that $\|\tau_{t,i}\| < \tau^{\max}$.

$\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are set to diagonal matrices, with different gains for each joint. Namely, by reducing the gains of the joints close to the end-effector compared to those of the other joints. This is necessary because the joints that are far from the end-effector need to move their mass and the masses of all following joints, so that they need higher torques for the same displacement.

The first part of Equation 27 acts similarly to a spring-damper system, where K_p would be the spring constant and K_v the damping coefficient. Thus, the ratio $\zeta = \frac{K_v}{2\sqrt{K_p}}$ determines the behaviour of the system. If $0 < \zeta < 1$, the system is underdamped, if $\zeta > 1$, the system is overdamped and if $\zeta = 1$, the system is critically damped. A critically damped system converges to the desired value as fast as possible without oscillations. A system ideally damped with $\zeta = \frac{1}{\sqrt{2}}$ converges to the desired value with the minimum integral of error over time, that is the minimum total surface between the curve and the desired value. The different behaviours of the system are shown by Figure 11.

For the drawing application, we would like to have the gains $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ in a proportion corresponding to the behavior of an ideal damping ratio. In practice, setting the gains with an ideal damping ratio did not result in the behavior expected theoretically, due to external perturbations such as various forms of frictions that are not modeled, as well as filters on the velocity signals whose characteristics are unknown. Thus, the gains for each joint were determined separately by perturbing each joint of the Baxter's arm by hand around a desired pose, and adjusting the gains until each joint has the desired tracking behaviour for the application.

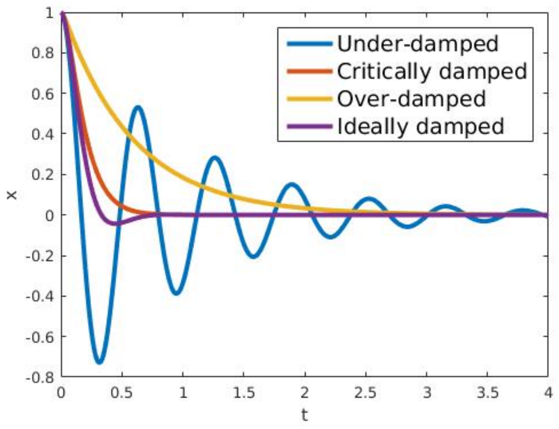


Figure 11: Behaviours of a damped mass-spring system.

The designed controller separates the computation of the inverse kinematics and the computation of the torques and control the robot in joint space. An other possibility is to control the robot directly in operational space. To make Baxter draw, the first option was chosen because the model of the inertia matrix is inaccurate and cannot be used. Moreover, this allowed use to adjust the gains individually for each joint. Finally, the computation of the inverse kinematics is the same for the velocity and the torque controller, so that both the velocity and torque controllers provided by the robot could be used with the same implementation.

More information about the control of robots are given in [18, 19, 20].

6 Visual Feedback

6.1 Recovering the Performed Trajectory

During the reproduction of the trajectories by the robot, the current pose of the end-effector can be determined by measuring the joint angles and convert them to a pose using the forward kinematics, so that $\boldsymbol{x}_t = \mathrm{FK}(\boldsymbol{q_t})$. The coordinates of these poses in the drawing plane frame of reference are then computed using

$$ {}^{o}\boldsymbol{X} = {}^{r}\boldsymbol{M}_o^{-1} \begin{pmatrix} R_z(-\pi/2) & 0 \\ 0 & 1 \end{pmatrix} {}^{r}\boldsymbol{X}, \tag{28} $$

corresponding to the inverse operation of Equation 11.

An image with this trajectory is then generated with the transformation

$$ \begin{aligned} u &= x/r + w/2 \\ v &= y/r + h/2 \end{aligned}, \tag{29} $$

corresponding to the inverse operation of Equation 10. A filled circle is drawn at each location $(u, v)^\top$ on the image. The image obtained is a reproduction of the drawing that should appear on the sheet in front of the robot if the pen tip was correctly placed on the sheet during the whole drawing.

6.2 Projection and Processing of the Drawing

Once Baxter has finished to draw, a picture of the result is taken with the left hand camera. As the camera is not perfectly in front of the drawing, the image is first projected to obtain an image of the drawing from a front view. An example of this transformation is shown by Figure 12.

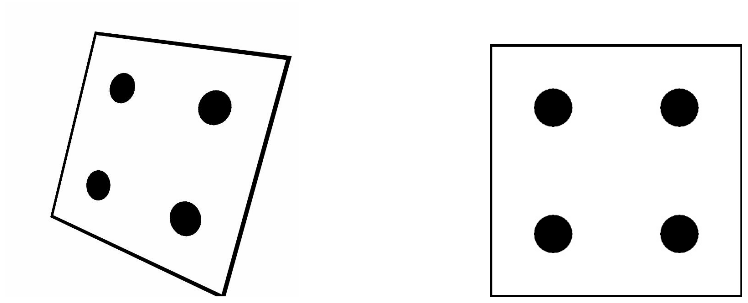

(a) Image from the camera. (b) Projected image.

Figure 12: Transformation of the image taken by the camera into a front view image.

The projection is done with the functions *getPerspectiveTransform* and *warpPerspective* of OpenCV [15]. The first function computes a perspective transform from two sets of four points. The first set of points is the 2D coordinates in the image $(x, y)^\top$ of the four points used to compute the pose of the drawing plane in Section 3. The second set of points is the desired positions of those four points in the transformed image. The second function applies the perspective transform to the image.

After the projection, a threshold operation is applied. The pixels in a defined square at each corner of the image are then set as equal to zero in order to remove the four dots used to determine the pose of the drawing plane.

Figure 13 shows an example of the projection and processing of a drawing.

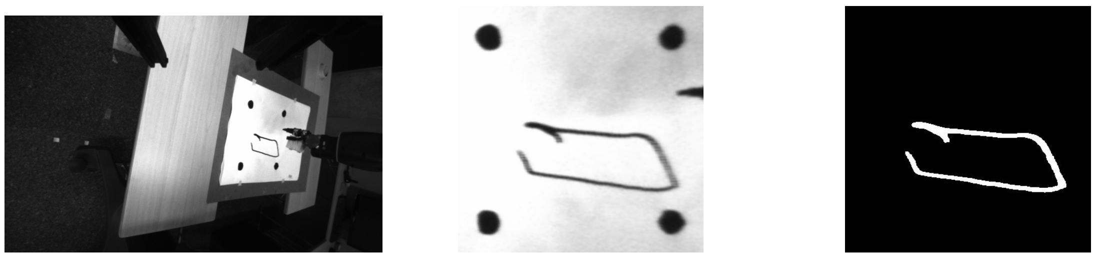

(a) Image from the camera. (b) Projected image. (c) Processed image.

Figure 13: Example of projection and processing of the final drawing.

6.3 Evaluation of the Dissimilarity between Expected, Planned and Drawn Strokes

The expected, planned and drawn images are then compared. The expected strokes correspond to the picture that should be drawn by the robot. In order to have a constant contour width, a new image is generated from the trajectory files using Equations 28 and 29. The planned strokes correspond to the image drawn from the recovered trajectory and the drawn strokes are the final drawing that appears on the sheet.

The expected and planned images are compared to evaluate the reproduction of the desired trajectories by Baxter. The expected and drawn images are compared to evaluate the contact between the pen tip and the sheet.

First, a visual comparison of the image is done by observing the differences between them. Then, they are compared with the function *matchShapes* of OpenCV. The similarity between two shapes is given by

$$e = \sum_{i=1}^{7} \left| \frac{1}{m_i^A} - \frac{1}{m_i^B} \right|. \tag{30}$$

The lower the result, the better match between shapes A and B it is. Matching an image with itself gives a result equal to 0. m_i^s for shape s is computed by

$$m_i^s = \operatorname{sign}(h_i^s) \log(h_i^s), \tag{31}$$

where h_i^s is the ith Hu moment invariants. Hu moments are computed from the normalised central moments of the image and are invariant with respect to translation, scale and orientation [21]. This comparison method has been chosen because the goal of the robot is to reproduce the shape of the picture. It does not matter if the drawing is a little bit shifted or rotated with respect to the original image.

7 Drawing's steps

The tools described in the previous section are gathered to perform the drawing of an image with Baxter. The steps of the drawing are described in Algorithm 5. Images are acquired with the left-hand camera and Baxter's right arm is used to draw. The initial pose of the robot is shown in Figure 14.

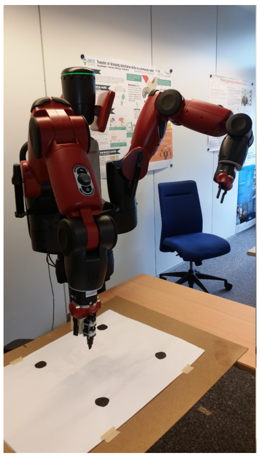


Figure 14: Initial pose of Baxter before the execution of the drawing.

Algorithm 5: Steps involved in drawing

```
-Baxter's arm in initial pose;
do
    -Acquire image with the left hand camera;
    -Detect and track dots with desired features in the image;
while four dots not correctly tracked;
-Estimation of the pose of the drawing plane with the four dots;
-Compute the frame of reference of the drawing plane in robot coordinates;
-Generate the trajectories to be drawn from a given image;
for all drawing trajectories do
    -Go to the first point of the drawing trajectory with Baxter's right arm;
    -Complete the drawing trajectory;
    -Go to the final point, 5 cm above the last point of the current trajectory;
end
-Go back to the initial pose;
-Take a picture of the drawing;
-Generate a picture with the executed trajectory;
-Generate a picture with the desired trajectory;
-Compare the three images;
```

A trajectory between the current pose and the final desired pose is generated when the robot goes to the initial or final point of the current trajectory. The trajectories are generated in order to avoid large pose errors that could lead to large torques or instabilities of the controller. Trajectory points are placed between the two poses so that the difference between $x_{i,t+1}$ and $x_{i,t}$ is smaller or equal to a predetermined maximum distance.

When the robot is tracking a trajectory, a reference point is given as desired pose at each time step. The distance between two consecutive points is chosen to achieve a reasonable and natural drawing speed. The ROS loop runs at 500 Hz. At each time step, the different lines of code are executed and if it takes less than the loop cycle time, the system waits until it reaches this duration. This allows us to know precisely the time between two executions of the loop, as one loop takes less than the allowed time to be executed.

8 Results

8.1 Drawing Realisation in Pictures

Figure 15 shows the estimation of the drawing plane made at the beginning of the drawing. The picture was taken by Baxter's left hand camera.

Figure 16 shows different steps of the drawing of a smiley face by Baxter.

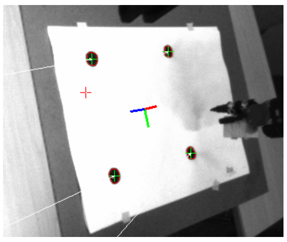
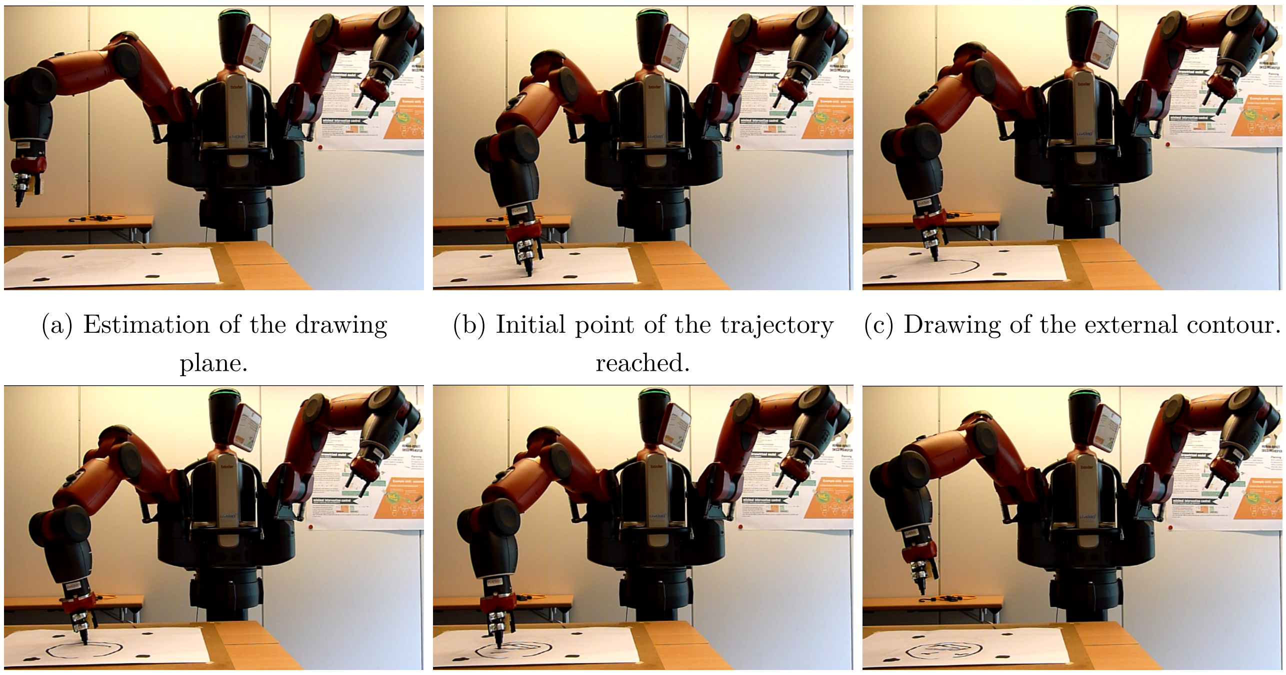


Figure 15: Estimation of the drawing plane: x, y and z axis are red, green and blue.



(a) Estimation of the drawing plane.

(b) Initial point of the trajectory reached.

(c) Drawing of the external contour.

(d) Drawing of the mouth.

(e) Drawing of the first eye.

(f) Drawing completed.

Figure 16: Baxter drawing a smiley face.

8.2 Evaluation of the Visual Estimation of the Drawing Plane

8.2.1 Estimation of the Reference

The goal is to obtain a precise estimation of the drawing plane, with the intervention of the user, that will be used for further evaluation.

The pose of the drawing plane can be estimated by registering the position of the robot end-effector at three different points of the plane and by computing the vectors corresponding to the axis of its frame of reference. To do this, a torque controller compensating for the effect of gravity is used to move Baxter's arms manually to place the end-effector successively on the upper-left, upper-right, down-left black dots of the drawing plane. The Cartesian position in robot's coordinates is registered for each point. Figure 17 shows the disposition of the three points and the drawing plane coordinates. $\boldsymbol{p}_{12}$ and $\boldsymbol{p}_{13}$ are defined as the vectors going from point 1 to 2, respectively 3.

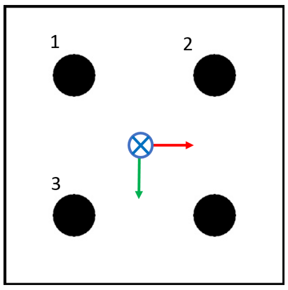


Figure 17: Disposition of three points used to compute the drawing plane coordinates (x, y and z axis are red, green and blue).

The homogeneous matrix ${}^{r}\boldsymbol{M}_{o}$ is then computed such that the columns of the rotation matrix R are the basis vectors $\boldsymbol{e}_1$, $\boldsymbol{e}_2$, $\boldsymbol{e}_3$ given by

$$\begin{aligned} \boldsymbol{e}_1 &= \boldsymbol{p}_{12}/\|\boldsymbol{p}_{12}\| \\ \boldsymbol{e}_2 &= \boldsymbol{p}_{13}/\|\boldsymbol{p}_{13}\| \\ \boldsymbol{e}_3 &= \boldsymbol{e}_1 \wedge \boldsymbol{e}_2 \end{aligned} \tag{32}$$

and t is the origin of the drawing plane coordinates in the robot frame of reference given by

$${}^{r}O = \frac{P_2 + P_3}{2}. \tag{33}$$

As the forward kinematics of the robot allows us to know precisely the Cartesian position of the end-effector, one can consider that the precision of the measured points depends only on the precision of the user who placed the end-effector on the plane. This estimation of the drawing plane will further be considered as the reference (ground truth).

8.2.2 Haptic Estimation of the Drawing Plane

The goal of this estimation is to use three positions of the end-effector acquired without the intervention of the user to estimate the pose of drawing plane and to compare it with the estimation of Section 3. As this estimation uses the visual estimation of the drawing plane, it could be used to improve the computed pose.

After the estimation of the pose of the drawing plane with the four points, the following procedure is successively done for the upper-left, upper-right, down-left black dots of the drawing plane. A trajectory is computed from the current pose of the end-effector to the pose of the dot in the robot frame of reference. The trajectory is completed using a torque controller. When the final point of the trajectory is reached, the desired position along z-axis of the drawing plane coordinates is adapted using Algorithm 4 as in the velocity controller with adaptive height. The goal of this adaptation is to use the measured forces to place the pen tip in contact with the sheet with a reasonable pressure between the two objects. Thus, when the position is fixed, the pose of the end-effector is registered.

The maximum and minimum forces ${}^{o}F_z$ between the pen tip and the sheet are respectively 5 N and 3.5 N. The maximum distance $\Delta z^{\max}$ between the current and initial positions along the z-axis of the drawing plane coordinates is equal to 10 cm.

The three obtained positions are then used to compute the homogeneous matrix ${}^{r}\boldsymbol{M}_o$ with the same method as for the estimation of the drawing plane with end-effector positions.

8.2.3 Evaluation of the Different Estimations of the Drawing Plane

The estimation of the reference is first evaluated to verify that the obtained drawing plane can indeed be considered as ground truth. As the sheet is placed horizontally at a fixed height in front of the robot, the following criteria are used to determine the reliability of the reference. The acute angle $\angle(\boldsymbol{n}_{\mathrm{gt}}, -\hat{\boldsymbol{z}})$ between the normal vector of the plane $\boldsymbol{n}_{\mathrm{gt}}$ and the vector $-\hat{\boldsymbol{z}}$ opposed to z-axis of the robot coordinates is used to determine if the plane is correctly estimated as horizontal. The variation of this angle and of the measured height ${}^{r}z$ between several estimations is then measured.

Table 2 gives the mean and the standard deviation of the height ${}^{r}z$ and the angle $\angle(\boldsymbol{n}_{\mathrm{gt}}, -\hat{\boldsymbol{z}})$ over 5 estimations of the pose reference for the drawing plane. The standard deviation of the height is equal to 2.8 mm and the standard deviation of the angle is 0.49 deg. Both values are small, meaning that the estimation is repeatable, and that our estimate for the reference is reliable.

Height ${}^{r}z$ [m]	**Angle $\angle(\overrightarrow{\boldsymbol{n}_{\mathrm{gt}}}, -\hat{\boldsymbol{z}})$ [deg]**
-0.20 ± 0.003	1.85 ± 0.49

Table 2: Mean and standard deviation of the height of the drawing plane and the angle between its normal axis and the vertical axis.

Figure 18 shows a typical example of measured forces in robot coordinates and the same forces in drawing plane coordinates. Due to the respective frames of reference of the robot and drawing plane, the z-forces are inverted from one frame of reference to the second one.

The position of the pen tip along z-axis of the drawing plane is adapted for the three dots at times 5s, 12s and 20s. The increasing, respectively decreasing, z-forces in robot and drawing plane coordinates show that the pen tip goes down in direction of the sheet to reach the desired range of contact force. Thus, the pose estimated visually seems to be higher than the ground truth.

Figure 19 shows a typical example of the three different estimations of the pose with the frame of reference of each estimation. Table 3 compares the ground truth with the other estimations. ${}^{r}\Delta x$, ${}^{r}\Delta y$ and ${}^{r}\Delta z$ are the distance between the origin of the coordinates of the ground truth and the origin of the other estimation in robot frame of reference. $\angle(\overrightarrow{\boldsymbol{n}_{\mathrm{gt}}}, \overrightarrow{\boldsymbol{n}})$ is the angle between the normal vector of the ground truth and the normal vector other estimation. It is also the acute angle between the two estimated planes.

The visual estimation of the pose of the drawing plane is, as supposed, above the ground truth with a mean distance of 33.9 mm. Thus, the robot needs to be controlled by a controller which allows to adapt the height of the end-effector, like a velocity controller with adaptive height or a torque controller with an additional force, as presented in Section 5. The estimated pose is also shifted along x and y-axis. However, those shifts do not prevent the drawing as the shift along z-axis that can stop the contact between the pen tip and the sheet. Thus, these shifts of 36.6 and 23.8 mm are acceptable. Moreover, the angle between the visual estimation and the ground truth has a mean of 2.89 deg, so that the drawing plane remains estimated as a horizontal plane. The standard deviation of the distances and angles between the visual estimation and the ground truth are small, so that the visual estimation is considered as repeatable.

The distance between z-coordinates of the origin of the haptic estimation and the origin of the ground truth is smaller than the same distance between the visual estimation and the ground truth. However, as one can

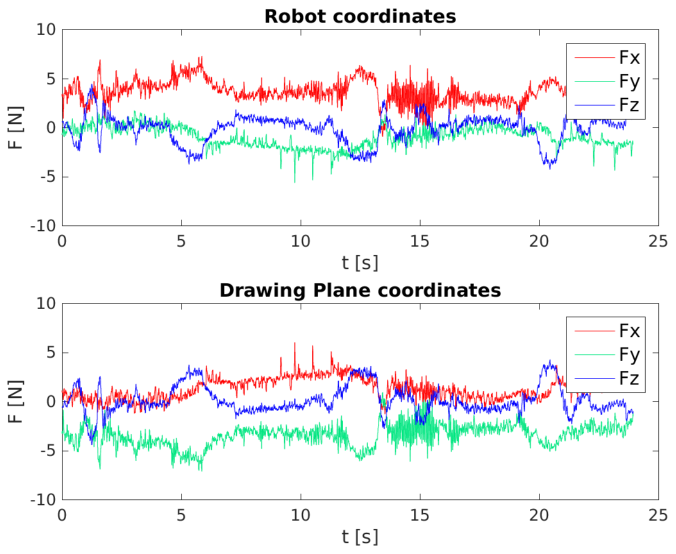


Figure 18: Measured and projected forces applied on the end-effector during the haptic estimation of the pose of the drawing plane.

see in Figure 19, the haptic estimation of the drawing plane is not horizontal and has an angle of 8.22 deg with the reference. This difference can be explained by the fact that the measured forces may be imprecise as they are influenced by the torques applied on the joints of the arm. For example, the forces can increase if the pose of the arm is close to a singularity or close to the robot's torso. The distances along x-axis are comparable to the distances obtained with the visual estimation, but have a higher standard deviation. The distances along y-axis are smaller than the distances obtained with the visual estimation.

Thus, the haptic estimation in this setup does not seem to provide much improvement over the visual estimation, so that only the visual estimation of the drawing plane is used in the final setup, with a controller that allows to adapt the height of the pen tip.

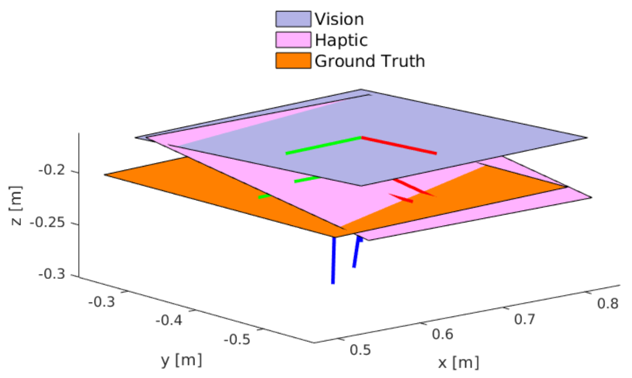


Figure 19: Estimated drawing planes with their frame of reference (x, y and z axis are red, green and blue).

Estimation	$^r\Delta x$ [mm]	$^r\Delta y$ [mm]	$^r\Delta z$ [mm]	$\angle(\overrightarrow{n_{gt}}, \overrightarrow{n})$ [deg]
Haptic	28.0 ± 15.7	7.5 ± 4.7	9.7 ± 3.6	8.2166 ± 0.7784
Visual	36.6 ± 4.6	23.8 ± 6.3	33.9 ± 4.5	2.8913 ± 0.7841

Table 3: Comparison between the ground truth and haptic and visual estimations of the drawing plane. The mean and standard deviation of the distances between the origin of their frames of reference and the acute angle between their normal vectors are given.

8.3 Evaluation of the Robot Controllers

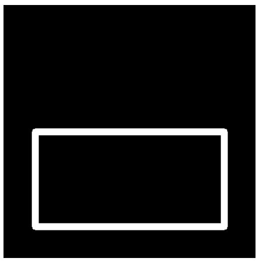
8.3.1 Velocity Controller

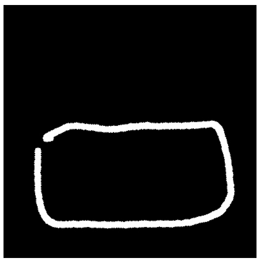
The velocity controller with an adaptive height is tested with parameters ${}^{o}F_z^{\max} = 5$ N, ${}^{o}F_z^{\min} = 3.5$ N, $\Delta z^{\max} = 10$ cm and $k = 3 \cdot 10^{-5}$, where k is the gain for the adjustment of the height. The gains of the inverse kinematics are set such that $K_{\text{spp}} = K_{\text{nsp}} = 100$. The parameters have been adjusted empirically to obtain natural drawing movements. The desired joint velocities are clamped with $\dot{q}^{\max} = 0.001$.

Two different shapes, a rectangle (30×15 cm) and a circle ($\varnothing 10$ cm), are drawn at two different maximum drawing speeds of respectively 0.1 m/s and 0.05 m/s for the rectangle, and 0.05 m/s and 0.025 m/s for the circle. Theses speeds are higher for rectangles than for circles as the rectangles are bigger and composed only of straight lines. The first point of the rectangles is defined as the upper-left corner. For the circles, it is the leftmost point. Both shapes are then drawn clockwise.

Figures 20 and 21 show the different images obtained after the fast and slow drawings of a rectangle. The mains differences between the expected and planned images are the accuracy of the formed angles at the corners and the straightness of the strokes. The angles are rounded for both drawings. However, the strokes appear straighter if the drawing is done faster. Both drawn images have a truncated left side, meaning that the pen tip did not have a good contact with the sheet. This could be due to the error of estimation of the drawing plane and to the latency of the controller to adapt the height of the end-effector. This latency is determined by the gain k. If the gain is higher, the time for the end-effector to go upper or lower is reduced. However, it can induce an oscillatory behaviour, where the end-effector goes down, hits the sheet with a high force and then goes up again to keep the force under the minimum value to draw, and so on.

Figure 22 shows the position of each joint during the drawing of the rectangle. The desired positions of the joints $\hat{\boldsymbol{q}}_t$ at each time step are the red curves and the positions $\boldsymbol{q}_t$ are the blue curves. The shapes of the two curves are similar during the whole drawing process, even if the position curve has often a small shift in time compared to the desired position curve. For the slow drawing, the positions of joints 2 and 6 do not follow precisely the desired position between $t = 15$ s and $t = 20$ s. This corresponds to the drawing of the lower-right corner of the rectangle.


(a) Expected image.


(b) Planned image.

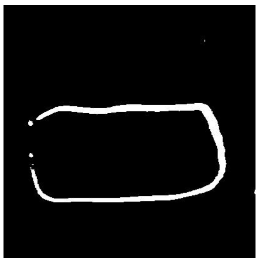
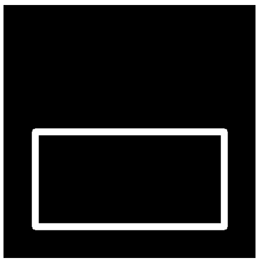
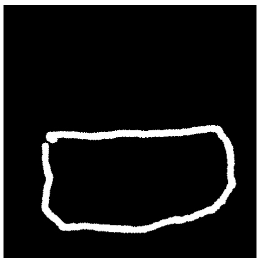

(c) Drawn image.

Figure 20: Images obtained for the fast drawing of the rectangle with the velocity controller.


(a) Expected image.


(b) Planned image.

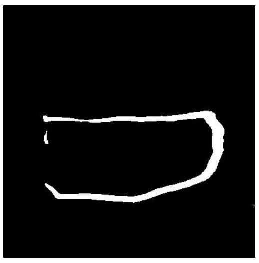
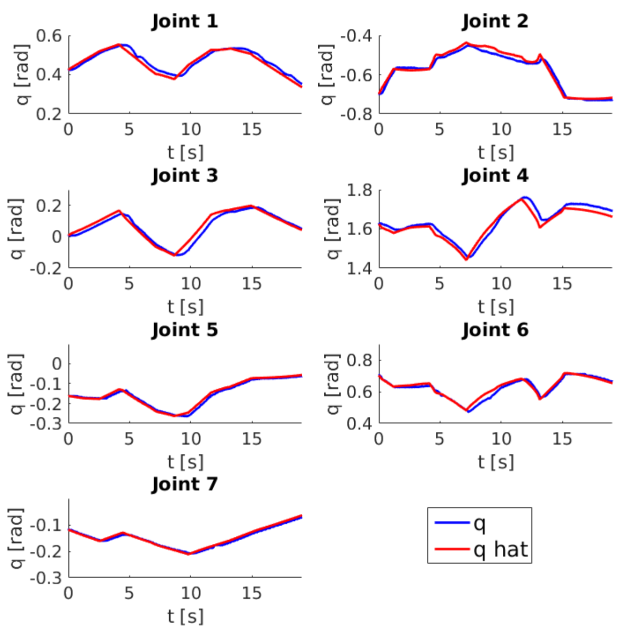

(c) Drawn image.

Figure 21: Images obtained for the slow drawing of the rectangle with the velocity controller.


(a) Fast drawing.

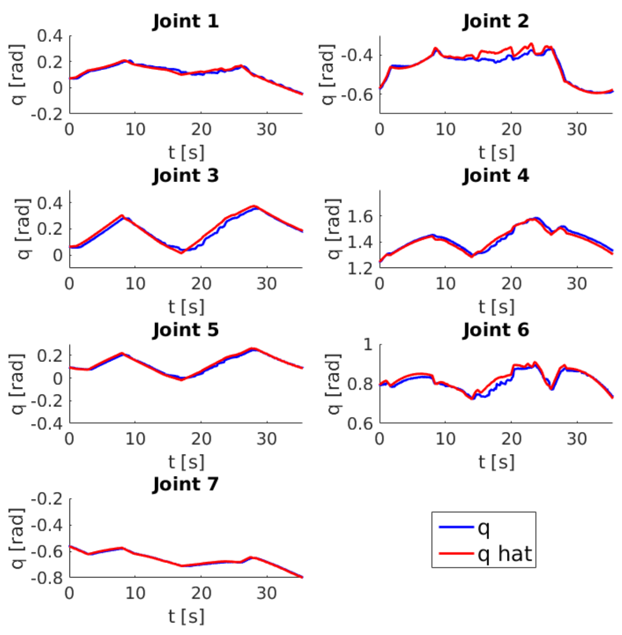
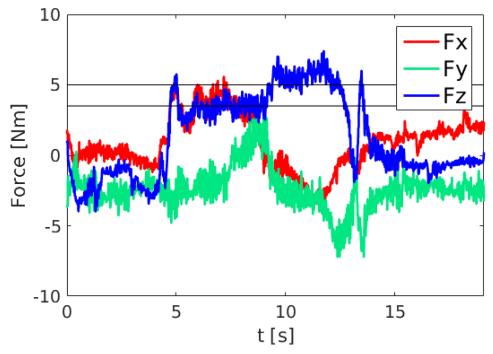

(b) Slow drawing.

Figure 22: Joints positions for the drawing of the rectangle with the velocity controller.

Figure 23 shows the forces applied at the end-effector during the drawing (in drawing plane coordinates). The force along z-axis shows the contact between the pen tip and the sheet, indicating when the robot is drawing. The two horizontal black lines are the maximum and minimum forces of contact ${}^{o}F_z^{\max}$ and ${}^{o}F_z^{\min}$. For the left and right plots, the robot draws respectively between $t = 5$ and $t = 14$ s, and between $t = 8$ and $t = 28$ s. Before and after those time windows, the robot goes respectively to the first point of the trajectory of the drawing and back to its initial position. The error in position of joints 2 and 6 between $t = 15$ s and $t = 20$ s for the slow drawing of Figure 22 corresponds to a high force applied at the end-effector along z-axis during this time window. This force is related to high absolute values of the forces along x and y-axis, meaning that the pen tip is pressing too hard on the sheet, which may disturb the movement of the robot. A similar behaviour is observed for the fast drawing for the time window during which the force along z-axis is above $max^{o}F_z$. In this case, as the force remains close to the limit, the drawing is less disturbed.


(a) Fast drawing.

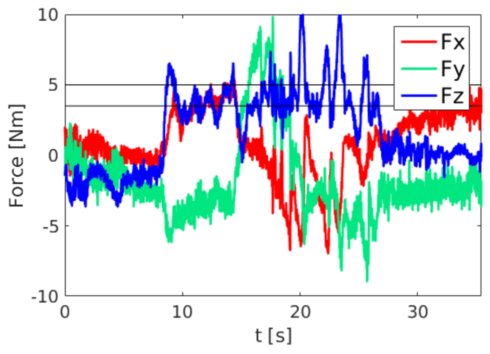
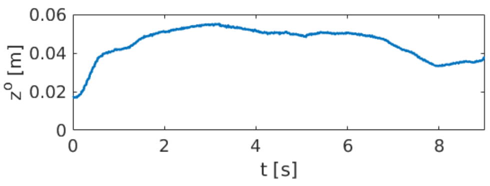

(b) Slow drawing.

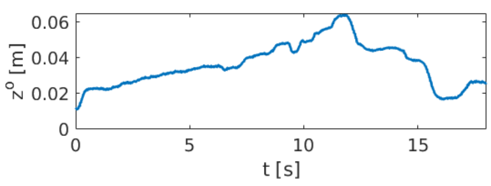
Figure 23: Forces applied at the end-effector in drawing plane coordinates for the drawings of the rectangle with the velocity controller.

Figure 24 shows the adaptation of the height of the end-effector in drawing plane coordinates in function of the applied vertical force. Only the period for which the robot is drawing is represented in these graphs.

$z = 0$ is the desired height at the beginning of the trajectory. Thus, the end-effector of the robot has to be 1 to 6 cm lower than the estimated drawing plane in order to have the pen tip in contact with the sheet. This corresponds to the estimation obtained in the previous section. For both drawings, the height is correctly adapted in function of the vertical force. When the force along z-axis of the plots of Figure 23 is above or under the limits, the height decreases so that the pen tip go further, and respectively increases when closer to the sheet.



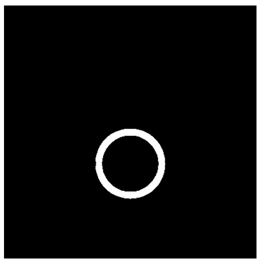
(a) Fast drawing.

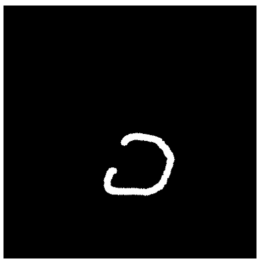


(b) Slow drawing.

Figure 24: Height of the pen tip in drawing plane coordinates for the drawings of the rectangle with the velocity controller.

Figures 25 and 26 show the different images obtained after the fast and slow drawings of a circle. The main difference between the expected and planned images are the roundness of the circle and the closing of the contour. Indeed, the circles of the planned images are not closed. The circle drawn fast seems to be more flattened than the circle drawn slowly. This can be explained by the latency of the controller that have not enough time to go close to the current desired pose before it is updated during the execution of the trajectory of the fast drawing. However, the effect of the latency is reduced for the slow drawing as two consecutive poses are closer to each other. Both circles are composed of two strokes, meaning that the pen tip did not touch the sheet in some parts of the drawing.


(a) Expected image.


(b) Planned image.

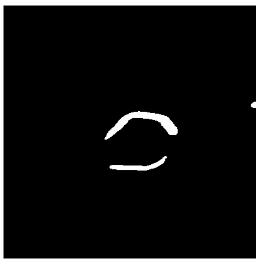
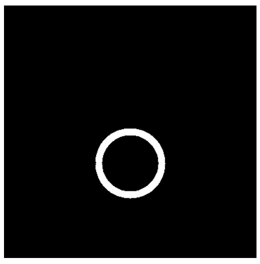
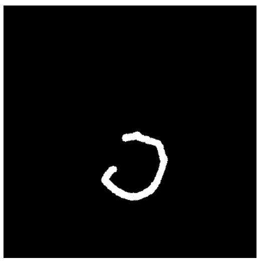

(c) Drawn image.

Figure 25: Images obtained for the fast drawing of the circle with the velocity controller.


(a) Expected image.


(b) Planned image.

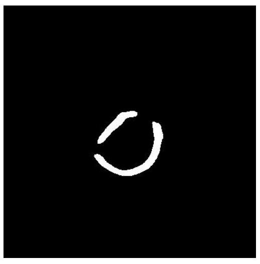
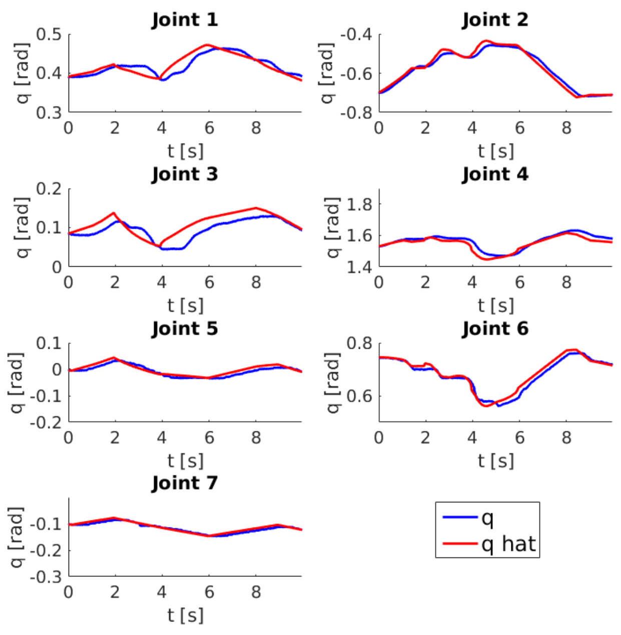
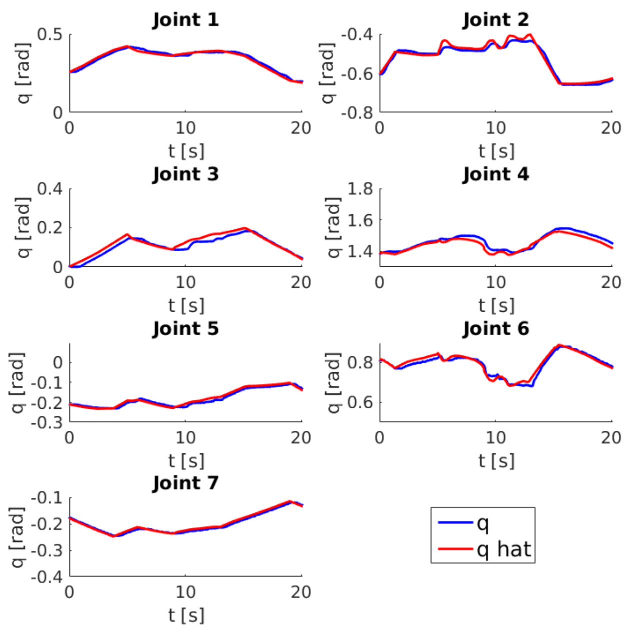

(c) Drawn image.

Figure 26: Images obtained for the slow drawing of the circle with the velocity controller.

Figure 27 shows the position of each joint during the drawing of the circle. The desired positions of the joints $\hat{\boldsymbol{q}}_t$ at each time step are the red curves and the positions $\boldsymbol{q}_t$ are the blue curves. For the fast drawing, the positions of joints 1 and 3 have a consequent error with respect to their desired position, in particular between time $t = 4$ and $t = 6$ s. It corresponds to the drawing of the low part of the circle, explaining its flattened shape. The curves of positions and desired positions are similar to each other for the others joints. For the slow drawing, there is no consequent error between desired and current position for each joint.



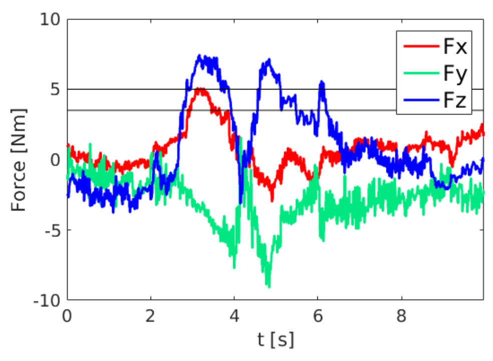
(a) Fast drawing.



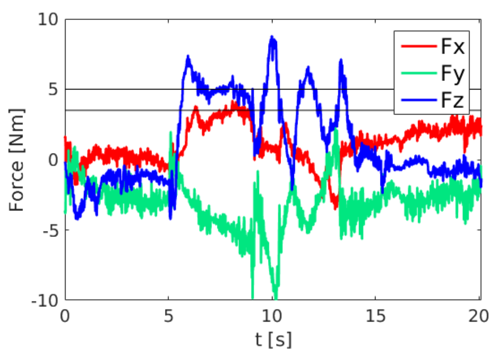
(b) Slow drawing.

Figure 27: Joints positions for the drawing of the circle with the velocity controller.

Figure 28 shows the forces applied at the end-effector during the drawing in drawing plane coordinates. The two horizontal black lines are the maximum and minimum forces of contact ${}^{o}F_z^{\max}$ and ${}^{o}F_z^{\min}$. For the left and right plots, the robot draws respectively between $t = 3$ and $t = 7$ s, and between $t = 5$ and $t = 13$ s. As for the drawing of rectangles, high forces along z-axis are associated with high absolute value of horizontal forces, showing that the pen tip presses more on the sheet.



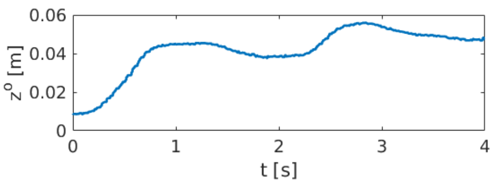
(a) Fast drawing.



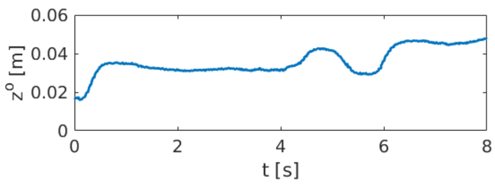
(b) Slow drawing.

Figure 28: Forces applied at the end-effector in drawing plane coordinates for the drawings of the circle with the velocity controller.

Figure 29 shows the adaptation of the height of the end-effector in drawing plane coordinates in function of the applied vertical force. Only the period for which the robot is drawing is represented in these graphs. As for the drawing of rectangles, the pen tip has to be 1 to 6 cm lower than the estimated drawing plane in order to be in contact with the sheet. For both drawings, the height is correctly adapted in function of the vertical force. The height is increased of a few centimetres at the beginning of the drawing and does not vary much after this first rectification. This confirms that the orientation of the drawing plane is well estimated.



(a) Fast drawing.



(b) Slow drawing.

Figure 29: Height of the pen tip in drawing plane coordinates for the drawings of the circle with the velocity controller.

Table 4 shows the match values of Equation 30 from Section 6 between the expected and planned or drawn strokes for the fast and slow drawings of the rectangle and the circle. Each shape was drawn three times. The lower the match value is, the better match there is between the two shapes.

According to the match values for the drawing of the rectangle, the planned and drawn images are more similar to the expected image if it is drawn fast. This may be due to the fact that the strokes appears straighter if the rectangle is drawn fast. The variance of both match values are quite high for the rectangle drawn slowly. This is due to the high match values of one of the three rectangle that was drawn in two strokes instead of one.

The match value between expected and planned circle is smaller if the circle is drawn slowly. This may be due to the shape of the circle drawn fast that is more flattened than the circle drawn slowly. Moreover, the contour of the circles drawn slowly are more completed. According to the match values between expected and drawn images, the circle drawn slowly is also closer to the circle of reference. Visually, it also seems more flattened than the same shape drawn slowly. The variance is high for the match values of the circles drawn fast, meaning that the resulting image may be quite different between drawings.

Drawing	$\boldsymbol{match(expected, planned)}$	$\boldsymbol{match(expected, drawn)}$
Fast rectangle	8.98 ± 9.63	4.46 ± 0.97
Slow rectangle	19.53 ± 19.37	52.72 ± 68.69
Fast circle	24.03 ± 21.27	47.71 ± 42.24
Slow circle	0.98 ± 0.56	16.19 ± 7.30

Table 4: Matching of the planned and drawn shapes with the expected shapes for fast and slow drawings of a rectangle and a circle with the velocity controller.

8.3.2 Torque Controller

The torque controller is tested with $F_z = -4$ N to push the end-effector in the direction of the drawing plane. The gains $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are given by

$$\boldsymbol{K}_p = \begin{pmatrix} 98 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 42 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 42 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 35 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 16.8 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 11.2 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 5.6 \end{pmatrix}, \boldsymbol{K}_v = \begin{pmatrix} 84 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 14 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 28 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 28 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 10.5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 9.1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 2.8 \end{pmatrix}. \quad (34)$$

The gains of the inverse kinematics are set such that $K_{\text{spp}} = K_{\text{nsp}} = 100$. The desired joint velocities are clamped with $\dot{q}^{\max} = 0.001$ rad/s and the commanded torques are clamped with $\tau^{\max} = 5$ Nm. The parameters have been adjusted by hand to have a good drawing. The gains $\boldsymbol{K}_p$ and $\boldsymbol{K}_v$ are set in order to have joints that are stiff enough to draw. The gains of the joints close to the end-effector have lower gains. Moreover, joint 2 has low gains. Since this joint mostly influences the movement of the end-effector in the vertical direction, keeping it soft allows us to have a soft behaviour of the end-effector with respect to the sheet. If the desired position of the end-effector is under the drawing plane, the pressure of the pen tip on the sheet is moderated by joint 2 that will compensate and shift up with the pressure.

The same shapes as those drawn with the velocity controller are drawn with the torque controller with the same maximum velocities.

Figures 30 and 31 show the different images obtained after the fast and slow drawings of a rectangle. The sides of the planned and drawn rectangles of the fast drawing are inclined with respect to the expected image. Thus, the corners are not right angles. This may be due to the latency of the controller. The position of the end-effector is late with respect to its desired position, so that it has not the time to end drawing a side of the rectangle before it is asked to follow the next one, resulting in uncompleted sides and wrong angles in the rectangle. This effect is reduced if the maximum drawing velocity is decreased. Indeed, the angles of the rectangle drawn slowly are closer to right angles. Moreover, the last side of the rectangle is truncated for both maximum drawing velocities. It may be due to a high force between the pen tip and the sheet that disturb the movement of the robot.

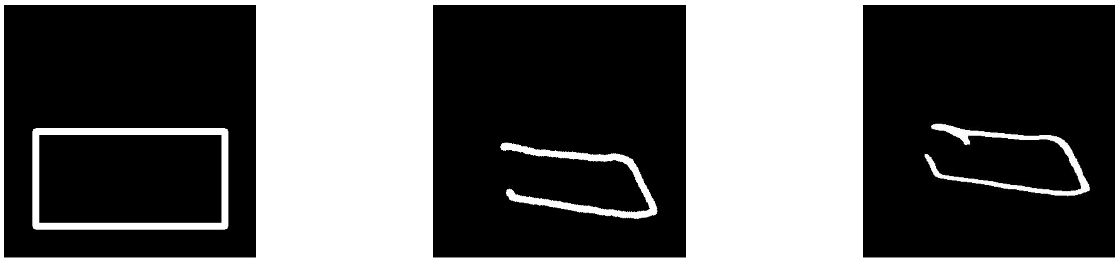

(a) Expected image. (b) Planned image. (c) Drawn image.

Figure 30: Images obtained for the fast drawing of the rectangle with the torque controller.

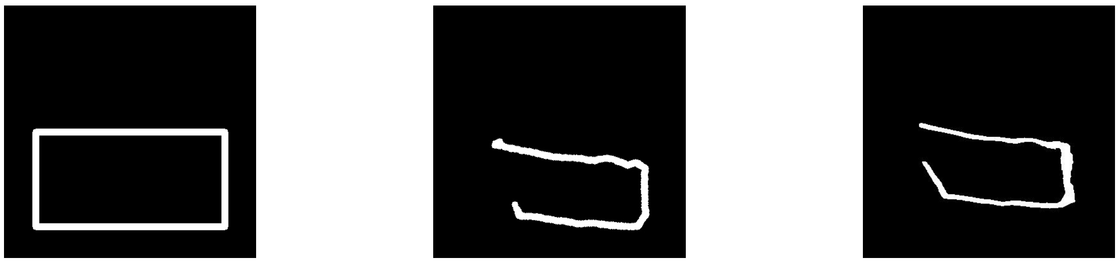
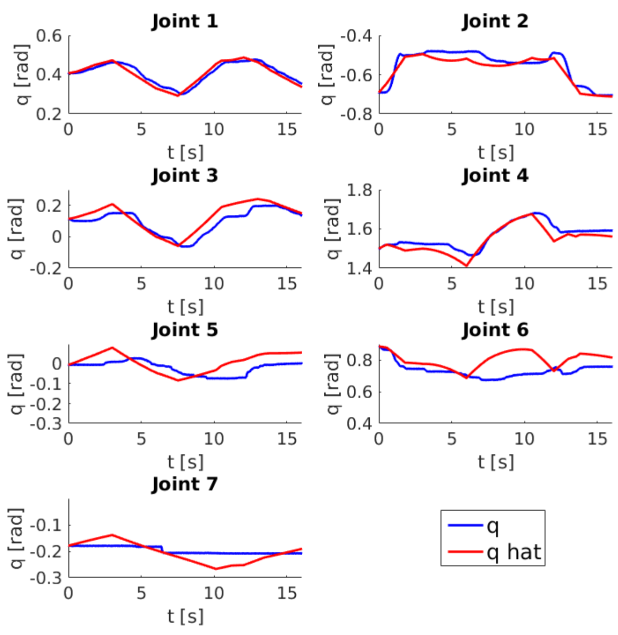
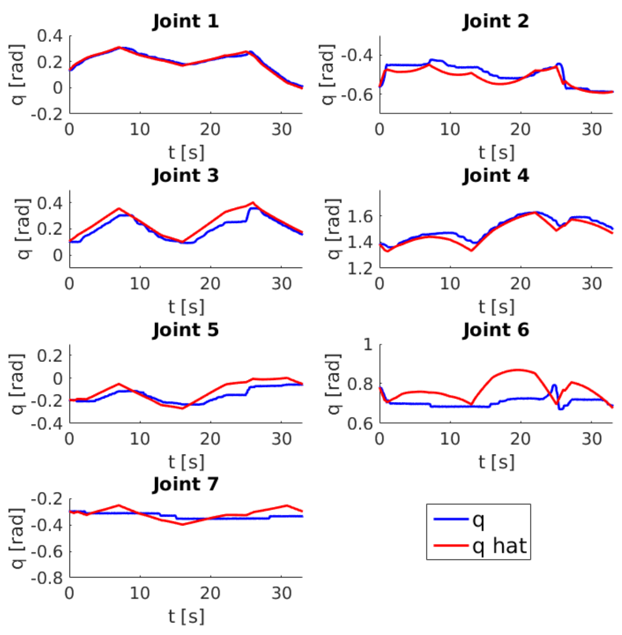

(a) Expected image. (b) Planned image. (c) Drawn image.

Figure 31: Images obtained for the slow drawing of the rectangle with the torque controller.

Figure 32 shows the position of each joint during the drawing of the rectangle. The desired positions of the joints at each time step $\hat{\boldsymbol{q}}_t$ are the red curves and the positions $\boldsymbol{q}_t$ are the blue curves. The shapes of the two curves are similar for joints 1 to 5, even if the position curve is often slightly shifted in time compared to the desired position curve. The position of two last joints have a consequent error with respect to the desired position. It may be due to the fact that their gains were set in order to have stable joints while the robot was not in contact with anything. Their behaviour may change and they may damped by the contact between the pen tip and the sheet. However, their K_p were not augmented in order to keep a stable behaviour when the pen tip does not touch the sheet. Despite its low stiffness, joint 2 is able to follow its desired position. The shift in angle observed between the two curves show the adaptation of the angle in function of the force applied on the end-effector.



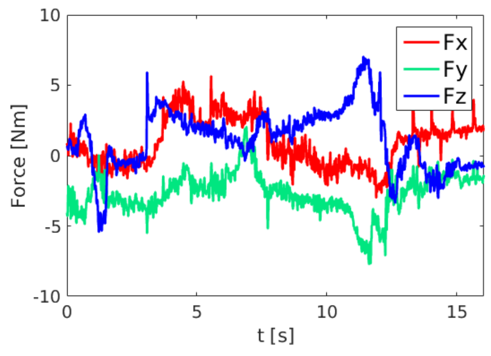
(a) Fast drawing.



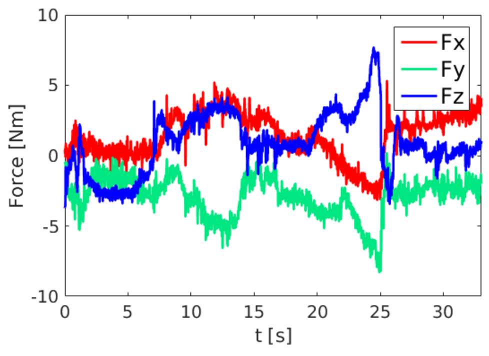
(b) Slow drawing.

Figure 32: Joints positions for the drawing of the rectangle with the torque controller.

Figure 33 shows the forces applied at the end-effector of the robot during the drawing (in drawing plane coordinates). The absolute values of the forces along y and z-axis are higher than for the rest of the drawing between $t = 10$ and $t = 12$ s for the fast drawing and between $t = 24$ to 25 s for the slow drawing. These time windows correspond to the drawing of the left-down corner of the rectangle and explain the inaccuracy of the drawing of the last side of the shape. The increase of forces may be due to the error of estimation of the drawing plane or to the configuration of the robot's arm when this corner of the rectangle is drawn that may be close to a singularity. Moreover, the increase of the force along z-axis at the beginning of the drawing, $t = 3$ and $t = 6$ s for the fast and slow drawing, correspond to the first important shift in angle between the desired and current joint positions of joint 2. This shows that the flexibility of this joint compared to the other and the force applied down during the drawing allow to have a better contact between the pen tip and the sheet.



(a) Fast drawing.

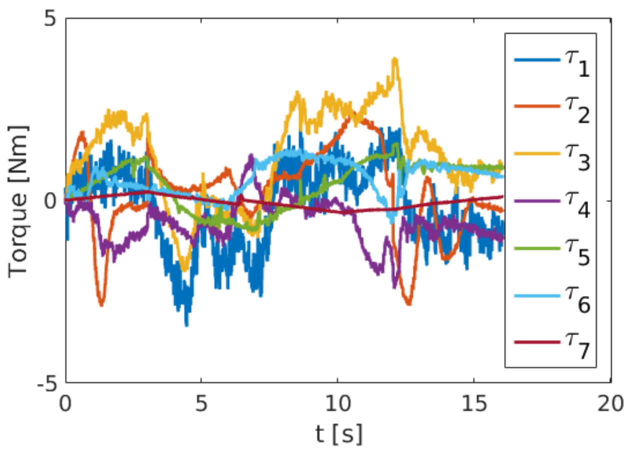


(b) Slow drawing.

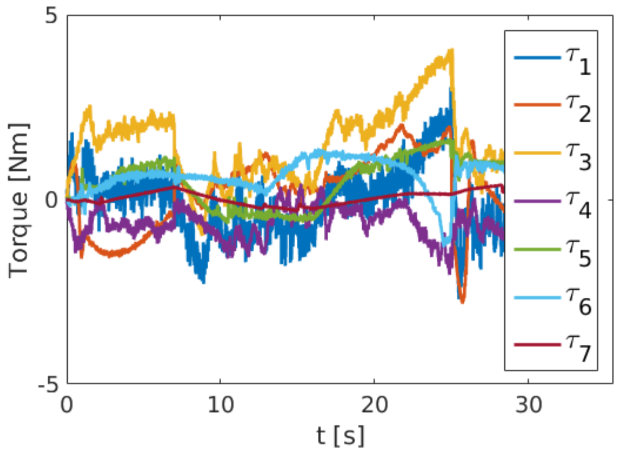
Figure 33: Forces applied at the end-effector in drawing plane coordinates for the drawings of the rectangle with the torque controller.

Figure 34 shows the torques applied to each joint during the drawing of the rectangle. The amplitude of the torques of joints 1 to 6 are in the same range. Only the torque applied on joint 7 is smaller than the other for both drawings. Moreover, the torques do not need to be clamped as their absolute value does not exceed

$\tau^{\max}$. One may observe that the torques have a similar shape for the fast and slow drawings, meaning that the same joints are used to move the end-effector during the same parts of the rectangle. The slow drawing is only stretched in time, allowing more time to follow the trajectory and improving the precision of the drawing.



(a) Fast drawing.



(b) Slow drawing.

Figure 34: Torques applied to each joint for the drawing of the rectangle with the torque controller.

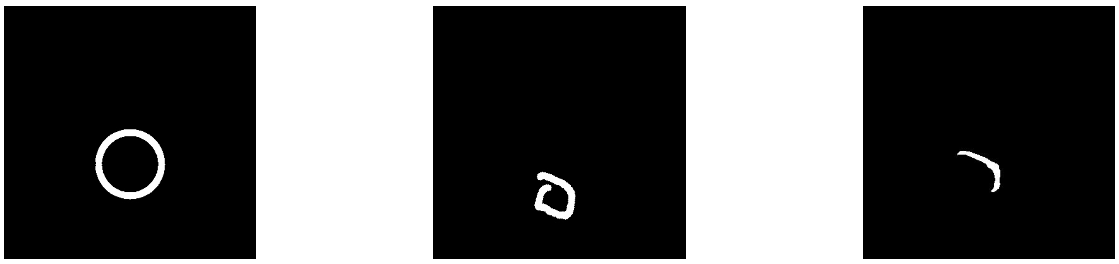

(a) Expected image. (b) Planned image. (c) Drawn image.

Figure 35: Images obtained for the fast drawing of the circle with the torque controller.

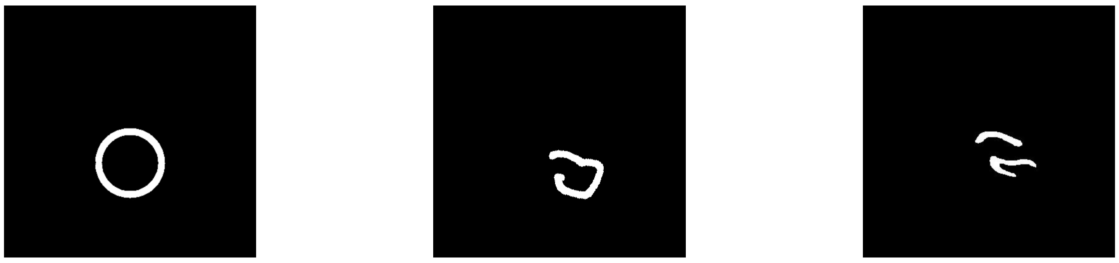
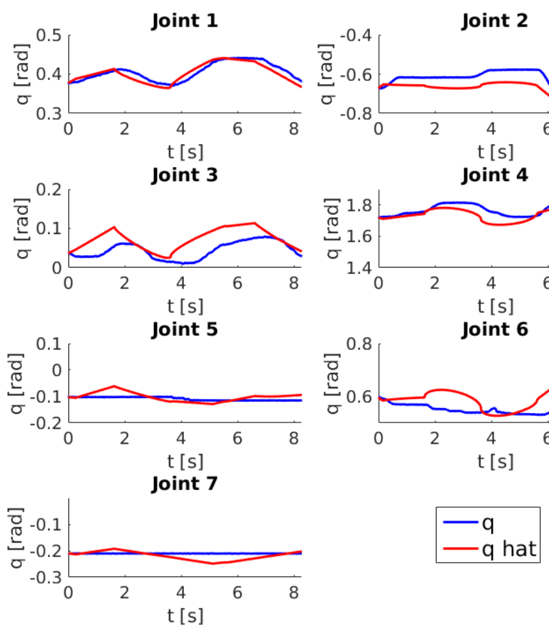

(a) Expected image. (b) Planned image. (c) Drawn image.

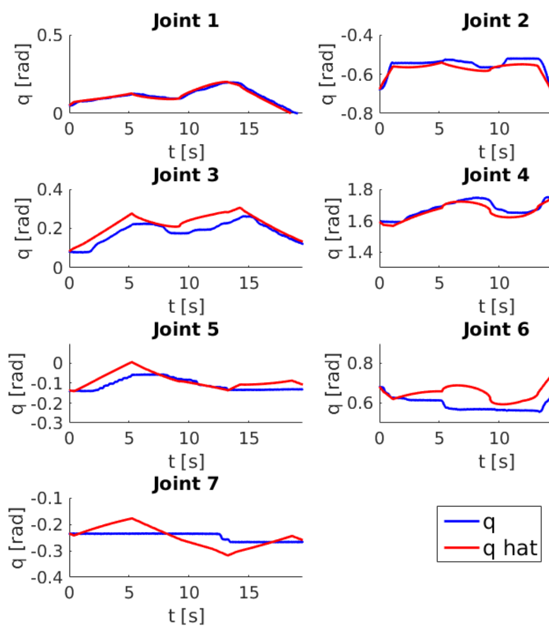
Figure 36: Images obtained for the slow drawing of the circle with the torque controller.

Figures 35 and 36 show the different images obtained after the fast and slow drawings of a circle. The main differences between the expected and planned images are the diameter and the roundness of the circle. This may be explained by the latency and some inaccuracies of the controller. The position of the end-effector is late with respect to its desired position so that the diameter of the circle is reduced. Moreover, the diameter of the circle is more decreased for the fast drawing of the circle than for the slow drawing. The circles of the drawn images are incomplete. For the fast drawing, only the first part of the circle is drawn and the pen tip does not have any contact with the sheet afterwards. For the slow drawing, the circle is drawn in two stokes and the pen tip stays in contact with the sheet at the end of the circle while to robot goes back to its initial pose. The fact that the pen tip does not stay in contact with the drawing plane may be due to the error of estimation of the plane and to the force down that is not high enough to compensate this error for a short drawing.

Figure 37 shows the position of each joint during the drawing of the circle. The desired positions of the joints at each time step $\hat{\boldsymbol{q}}_t$ are the red curves and the positions $\boldsymbol{q}_t$ are the blue curves. The error between the position of the joints and their desired position is quite important for nearly all joints during the fast drawing. This explains the inaccuracy of the drawn circle. For the slow drawing, the two curves are similar for joints 1 to 5. The errors for joints 6 and 7 are similar to the errors obtained during the drawing of the rectangle and have the same cause. Similarly to the drawing of the rectangle, joint 2 is able to follow its desired position.

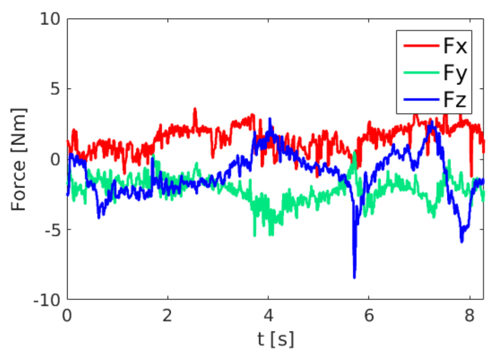


(a) Fast drawing.

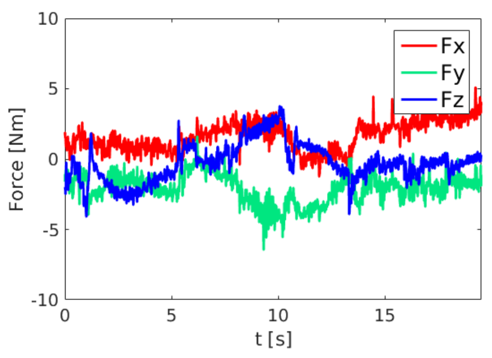


(b) Slow drawing.

Figure 37: Joints positions for the drawing of the circle with the torque controller.



(a) Fast drawing.

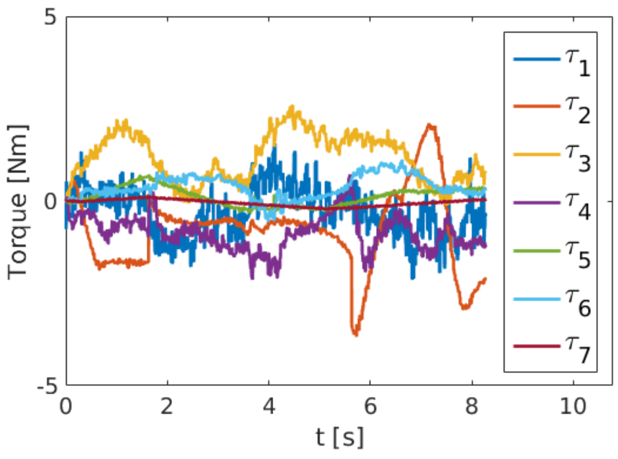


(b) Slow drawing.

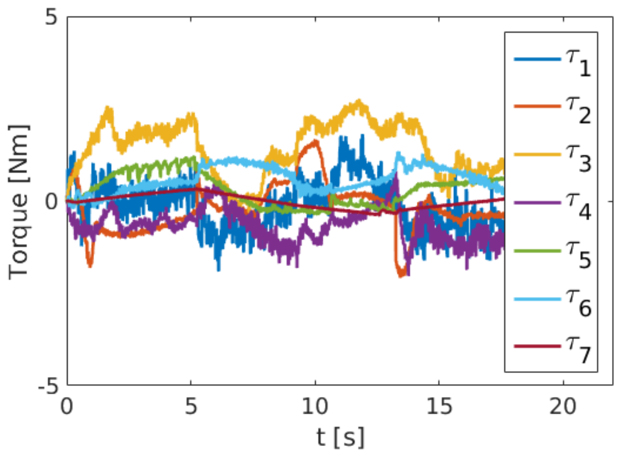
Figure 38: Forces applied at the end-effector in drawing plane coordinates for the drawings of the circle with the torque controller.

Figure 38 shows the forces applied at the end-effector during the drawing in drawing plane coordinates. Those plots confirm that the pen tip is not in contact with the sheet during parts of the drawing as the force along z-axis is quite lower than the force obtained during the drawing of the rectangle. Thus, it is difficult to determine the exact timing of the drawing. Moreover, the absolute values of the forces along x and y-axis are also low, meaning that there is low horizontal forces so that the pen tip is not always moving on the sheet.

Figure 39 shows the torques applied to each joint during the drawing of the circle. As for the drawing of rectangles, the torques do not need to be clamped as their absolute value does not exceed $\tau^{\max}$. Moreover, they have a similar shape for the fast and slow drawings. Only joint 2 has quite different shapes of torques between the two drawings. It may be due to the fact that the pen tip is more often in contact with the sheet for the slow drawing than for the fast one.



(a) Fast drawing.



(b) Slow drawing.

Figure 39: Torques applied to each joint for the drawing of the circle with the torque controller.

Table 5 shows the match values of Equation 30 from Section 6 between the expected and planned or drawn strokes for the fast and slow drawings of the rectangle and the circle. Each shape was drawn three times. The lower the match value is, the better match between two shapes it is.

Conversely to the drawing of the rectangle with the velocity controller, the match values indicate that the planned and drawn images are more similar to the expected image if it is drawn slowly. This may be due to the orientation of the small sides of the rectangle that are drawn in a diagonal direction if the shape is drawn fast. The variances for both match values are also high for the rectangle drawn fast.

According to the match value, the planned and drawn images of the circle drawn fast is more similar to the expected image than the same image drawn slowly. The fast drawing results visually in a smaller but rounder circle than the slow drawing in the planned image. As the match value is computed with Hu moments invariant to scaling, it is expected that the roundest circle is closer to the expected one. The variance of both match values of the circle drawn slowly are high. This is due to the high match values of the circle of Figure 36 for which horizontal line appears in the image. The line comes from the fact that the pen was still in contact with the sheet when the robot was going back to its initial pose.

Drawing	$\boldsymbol{match(expected, planned)}$	$\boldsymbol{match(expected, drawn)}$
Fast rectangle	56.88 ± 72.62	15.13 ± 25.61
Slow rectangle	21.50 ± 12.33	2.64 ± 3.84
Fast circle	1.98 ± 1.50	2.33 ± 2.07
Slow circle	21.70 ± 21.58	15.61 ± 22.6

Table 5: Matching of the planned and drawn shapes with the expected shapes for fast and slow drawings of a rectangle and a circle with the torque controller.

8.3.3 Discussion and Comparison Between Controllers

The velocity controller obtain better match values if the rectangle is drawn fast. Conversely, the match values of the torque controller are lower if the rectangle is drawn slowly. The match values between expected and drawn images computed with the Hu moments are respectively 4.46 ± 0.97 and 15.13 ± 26.61 for the velocity and torque controller drawing fast and 52.72 ± 68.69 and 2.64 ± 3.84 if the rectangle is drawn slowly. Moreover, the rectangles drawn fast for the velocity controller and slowly for the torque controller are also visually the most similar to the expected shape.

According to the match values, the inverse observation hold for the circle, as they are respectively equal to 47.71 ± 42.24 and 2.33 ± 2.07 for the velocity and torque controller drawing fast and 16.19 ± 7.30 and 15.61 ± 22.6 for the circle drawn slowly. Indeed, visually, the circle drawn slowly by the velocity controller is rounder than the one drawn fast. However, the circle drawn slowly by the torque controller is less truncated and less shrunken than the one drawn fast, as the second one appears quite small. Thus, the circle drawn slowly is preferred for both controllers.

The difference of performance in the drawing of the two shapes in function of the maximum velocity of drawing may be explained by the characteristics of each shape. The rectangle has bigger dimensions than the circle, that is 30×15 cm against $\varnothing 10$ cm. Moreover, it is composed of straight lines, as the circle is composed of one round stroke. Thus, the latency of the controller results in angles that are not perfectly right, with slightly truncated sides. As the sides are quite long, the modifications in the shape remain discrete. However, the circle is shrunken and it is more visible as it has a small diameter. As the latency of the controller is more and more visible while the shape is drawn faster, the circle needs to be drawn with a smaller maximum speed than the rectangle. Thus, a maximum velocity equal to 0.1 m/s is good to draw the rectangle as a maximum velocity of 0.025 m/s is preferred to draw the circle for both controllers. One can deduce that the smaller the shape is, the smaller maximum velocity is needed to draw it.

According to the match values, the rectangle drawn fast with the velocity controller is more similar to the expected rectangle than the one drawn with the torque controller. Moreover, visually, the size of the rectangle of reference is better reproduced with the velocity controller. The circle drawn slowly with the velocity controller are similar to the one drawn with the torque controller according to the match values. However, the circle drawn with the velocity controller is visually closer to the expected shape, as the pen tip is not correctly in contact with the sheet during an important part of the drawing with the torque controller.

In order to compare the performances of the controllers for more complex shapes and for a drawing composed of several contours, a smiley face was drawn with the robot. The parameters of the controller were not changed. The maximal drawing velocity was defined as 0.05 m/s for both drawings. The external and internal contours are shown in Figure 40. The external contour consists in a circle of diameter of approximately 20 cm. There are three internal contours, one for the mouth and two for the eyes. The three shapes are close to half-circles. The resulting drawing of each controller are shown in Figure 41. In this case, the match values between the expected and drawn images are respectively equal to 15.50 and 5.49 for the velocity and torque controller.

The smiley face drawn with the velocity controller has a quite round external contour drawn in one stroke and the mouth is close to a half-circle. The eyes are drawn less accurately, but they have a size close to the size of the model. The external contour drawn with the torque controller is less round than the other one. The circle is smaller than expected, more flattened and drawn with two strokes. The mouth and the eyes are also smaller than expected, but the half-circle shapes are recognisable. More generally, one can recognise the smiley in both drawings, even if the one from the velocity is drawn globally with a better accuracy.

The drawing also shows that both controllers are efficient to adapt their height relatively to the drawing plane to have a more or less constant contact between the pen tip and the sheet. Even if the torque controller was not able to keep this contact during the whole drawing of the circle, it is here able to draw the smiley so that it is similar to the reference and that one can recognise it. For both controllers, some parts of the contours

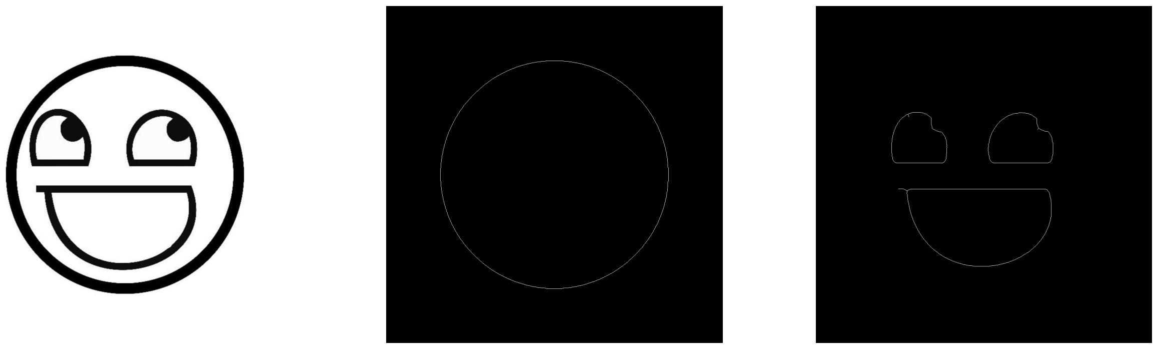

(a) Smiley. (b) External contours. (c) Internal contours.

Figure 40: Image and contours to draw a smiley with Baxter.

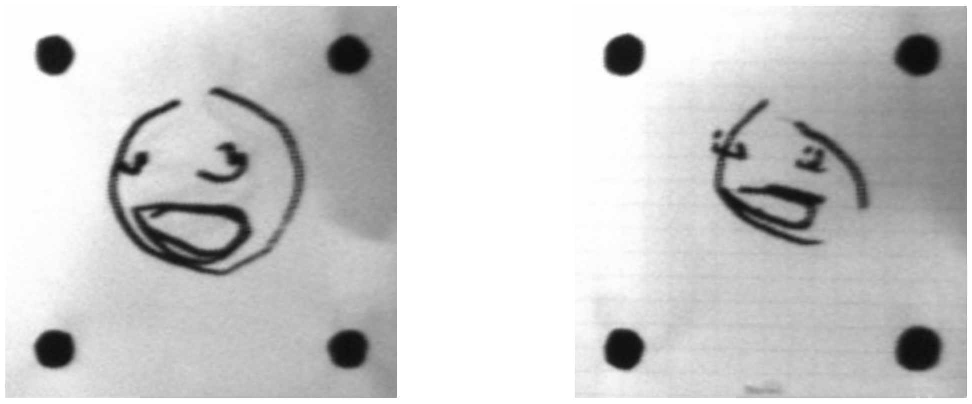

(a) Velocity controller. (b) Torque controller.

Figure 41: Projected images drawn by the robot.

are not drawn because the pen tip does not push enough against the sheet and some part are drawn thicker because of the inverse behaviour, but the majority of them are drawn correctly.

The main advantage of the velocity controller is that is it more accurate than the torque controller. However, the torque controller may be as accurate if the inertia matrix was precisely known. The main advantage of using a torque controller in joint space instead of a controller in operational space is that the gains of the joints are easier to tune individually as the inertia matrix is not known precisely. The drawings done with a controller in operational space with a precisely known inertia matrix may be better than those obtained with the actual controller. Such controller would allow a spring behaviour to be simulated on the robot, so that it would be stiff on the drawing plane, but soft in the perpendicular direction to this plane to obtain a constant contact between the pen tip and the sheet.

9 Play Tic-Tac-Toe with Baxter

9.1 Game Description

Tic-tac-toe is a two-players game played on a 3×3 grid. Each player plays his turn by marking one space with his mark. The two different marks are a cross X and a circle O. The player who succeeds to place three of his marks in the same row, column or diagonal wins the game.

Figure 42 shows an example of a complete game, where O won by filling a diagonal with his marks.

Figure 42: Example of completed tic-tac-toe game [22].

9.2 Process of a Game with Baxter

The setup to play tic-tac-toe with Baxter is similar to the drawing setup described previously. A sheet with four black dots is located in front of the robot which acquires images with the left hand camera and draw with the right hand. The torque controller is used to control the robot with the same parameters as those used to draw rectangles and circles.

A game with Baxter begins with the robot drawing the four lines composing the board. The robot performs the drawing using the method described in Section 7. Then, the user is asked if he/she wants to be the first or the second player and which of the two marks, a cross or a circle, he/she wants to play with.

When it is Baxter's turn, its next move is first determined by the minimax algorithm that will be described below. Then, the trajectory for a cross or a circle for Baxter's move is generated. The coordinate system of the sheet is computed using the four dots and Baxter draws its mark in the right space. When it is the opponent's turn, he/she can take the sheet, draw his/her mark where he/she wants to play and put the sheet back in front of Baxter.

9.3 Choice of Baxter's Next Move

9.3.1 Minimax Algorithm

The minimax recursive algorithm is used in decision theory and games theory. It aims to choose the next move in a n-players game while minimising the possible loss for a worst case scenario. For two-players games, each available move of the player who is playing is evaluated with the goal to pick the move with the maximum score. The score for each move is determined by looking at each possibility for the opponent to play after this move and by taking the move that minimises the score of the first player. The algorithm continues by flipping from one player to the other until a final value is found.

The minimax algorithm is used to play a perfect game of tic-tac-toe, so that the robot wins or the result is a draw. Figure 43 shows an example of the algorithm's execution for a tic-tac-toe game. The score is defined as equal to 10 if the robot wins, -10 if it loses and 0 if it draws. In state 1 of Figure 43, it is X's turn to play. It has three available moves, represented at states 2, 3 and 4 for which minimax algorithm is called. At state 2, X wins, so that the final score for this state is 10. States 3 and 4 generates respectively states 5 and 6 and states 7 and 8 for which minimax is called. At states 5 and 7, O wins, so that the scores are -10. States 6 and 8 generate respectively states 9 an 10 for which X wins, so that the final score is 10. The final scores for states 3 and 4 are the worse of their respective substates, that is -10 for both. Thus, state 2 is finally chosen by the minimax algorithm as the next move for player X.

9.3.2 Implementation of the Minimax Algorithm

Algorithm 6 describes the implementation of the minimax algorithm for the tic-tac-toe game. The score is equal to 10 if a row, a column or a diagonal of the board is filled by Baxter's symbol. It is equal to -10 if a row, a column or a diagonal of the board is filled by the opponent's symbol. Otherwise, it is a draw and the score is equal to 0.

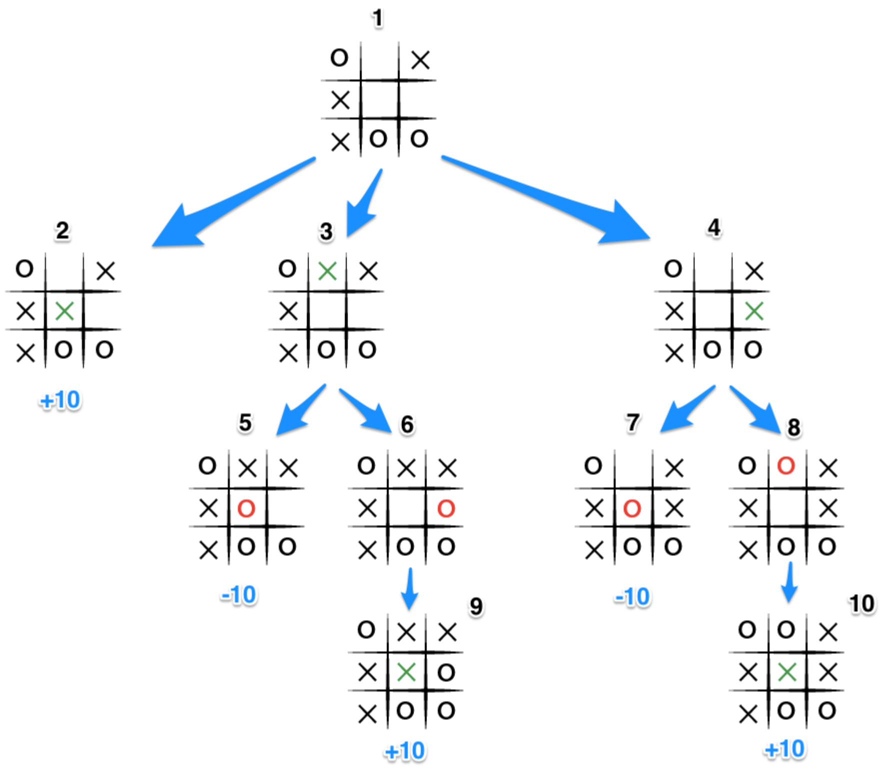


Figure 43: Example of minimax execution in a tic-tac-toe game [23].

9.4 Detection of The Opponent's Move

9.4.1 Preprocessing of the Images

To detect the opponent's move, two images are acquired respectively before and after the opponent plays. Both images are first projected using the projection method of Section 6. Figure 44 shows an example of original and projected images before and after the move of the opponent.

The absolute difference between the two projected images is then performed using the function *absdiff* of OpenCV. A threshold operation is then applied to the resultant image with a threshold value equal to 130. This value was chosen in order to keep only the evident differences between the projected images. Figure 45 shows an example of the absolute difference between the two projected images and the same image after the thresholding operation.

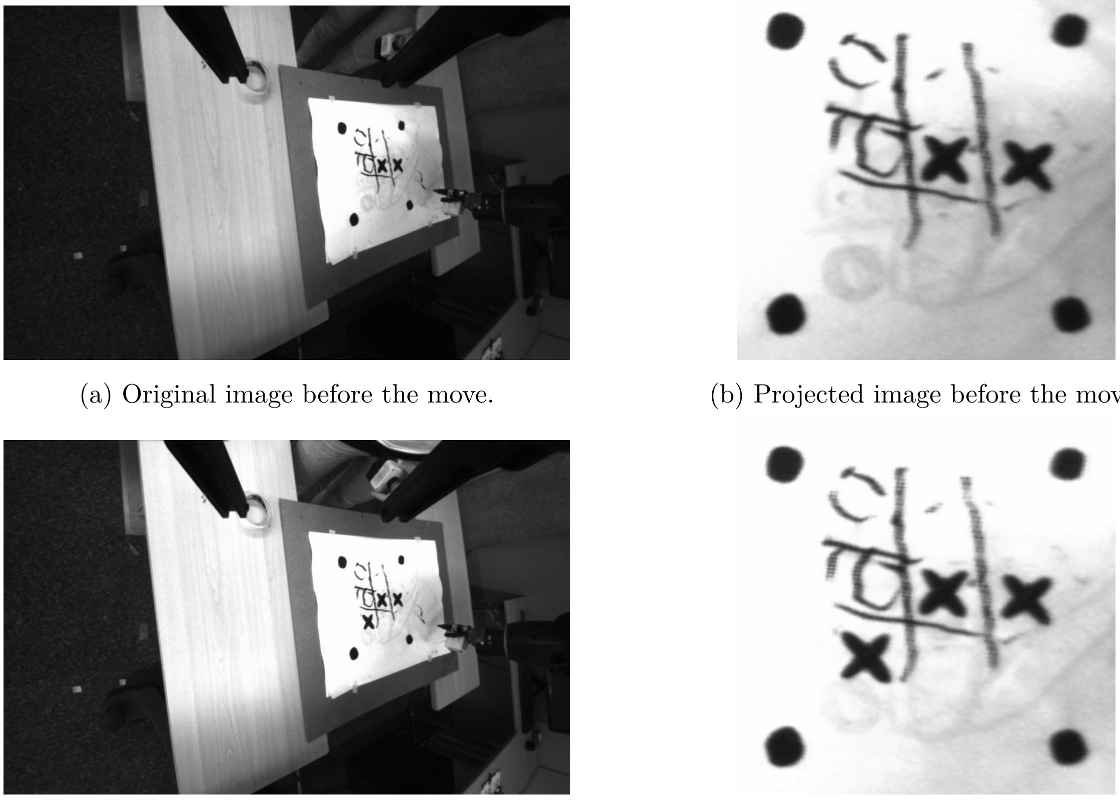
9.4.2 Canny Edges Detection

The edges of the images are then detected using the Canny edge detector implemented in OpenCV. The steps of this edge detection algorithm are presented by Figure 46. The image f is first filtered with a Gaussian filter to remove the noise. Then, a pair of convolution masks h_1 and h_2 are applied to find the horizontal and vertical derivatives g_1 and g_2. The gradient strength $||g||$ and direction θ are computed and non-maxima suppression is applied. Two threshold are then used to determine if a pixel is accepted as an edge or not. If the pixel is above the upper threshold, it is accepted. If it is below the lower threshold, it is rejected. Finally, if it is between the two thresholds, it is accepted only if it is connected to a pixel that is itself above the upper threshold.

Figure 47 shows an example of the application of Canny detector on the thresholded image. The lower threshold is equal to 100, the upper threshold is 300 and the size of the convolution masks is 3.

Algorithm 6: Minimax

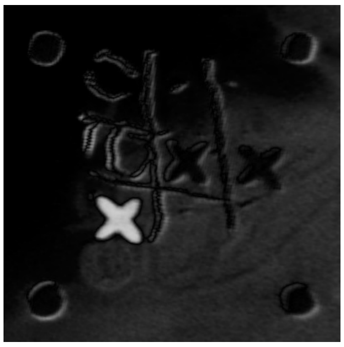
```
Empty list of scores;
Empty list of moves;
if game is over then
    return score(game)
end
for each possible move do
    Generate a new possible game;
    (list of scores).pushback(minimax(possible game));
    (list of moves).pushback(move);
end
if robot's turn then
    max score index = max(list of scores);
    Chosen move = (list of moves)[max score index];
    return (list of scores)[max score index]
end
else
    min score index = min(list of scores);
    Chosen move = (list of moves)[min score index];
    return (list of scores)[min score index]
end
```



(a) Original image before the move. (b) Projected image before the move.

(c) Original image after the move. (d) Projected image after the move.

Figure 44: Original and projected images before and after the move of the opponent.


(a) Absolute difference between projected images.

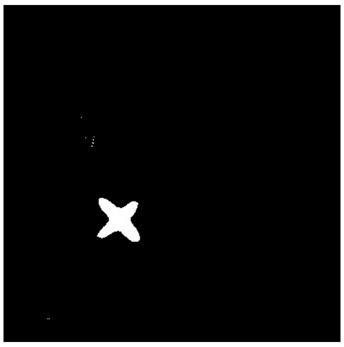

(b) Image after threshold operation.

Figure 45: Images obtained after the absolute difference of the projected images before and after the opponent move and after a thresholding operation.

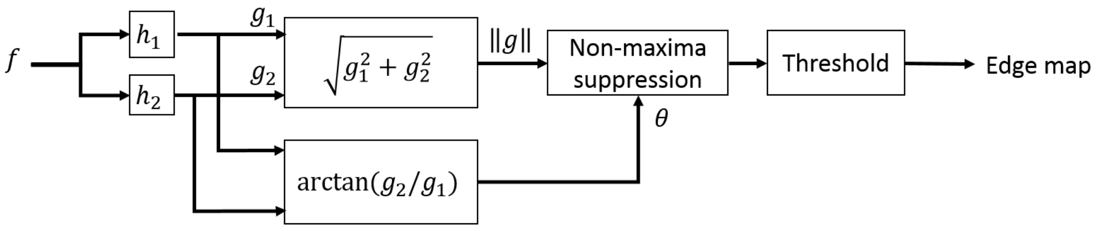


Figure 46: Canny edge detection algorithm [24].

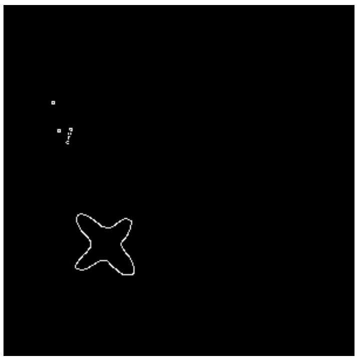


Figure 47: Image resulting from the application of the Canny edge detector on the thresholded image.

9.4.3 Contours Extraction and Sorting

The contours of the image are extracted using the function *findContours* of OpenCV. As the goal is to find the location of the new cross X or circle O added by the opponent, only the contours having an area between 900 and 3000 are kept. In this kind of images, only one contour corresponding to the mark added by the opponent remains. If more than one contour remain, they are all contours of the symbol and are very similar so that the first detected contour is kept. As most contours result from the difference between the rest of the two projected images and thus, have a small area, it is guaranteed that a contour with an area between 900 and 2200 is the contour of a symbol. The area corresponding to the contour of a cross or a circle is large enough to recognise a large range of them as the drawing of these symbols can change from one user to an other. Moreover, the detection method is the same for both symbols as the user is expected to draw his/her symbol. Figure 48 shows an example of the remaining contour.

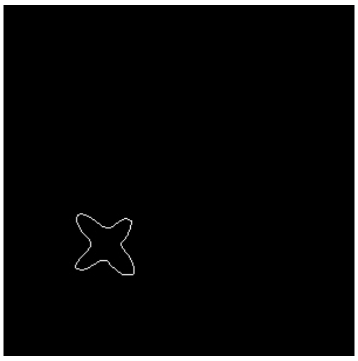


Figure 48: Contour sorted from the edges image.

9.4.4 Move Recognition

The spatial moments of the contour are computed with

$$m_{ij} = \sum_x \sum_y x^i y^j I(x,y), \tag{35}$$

where $I(x,y)$ is the value of the pixel $(x,y)^\top$ in the image. The centre of mass of the contours is given by

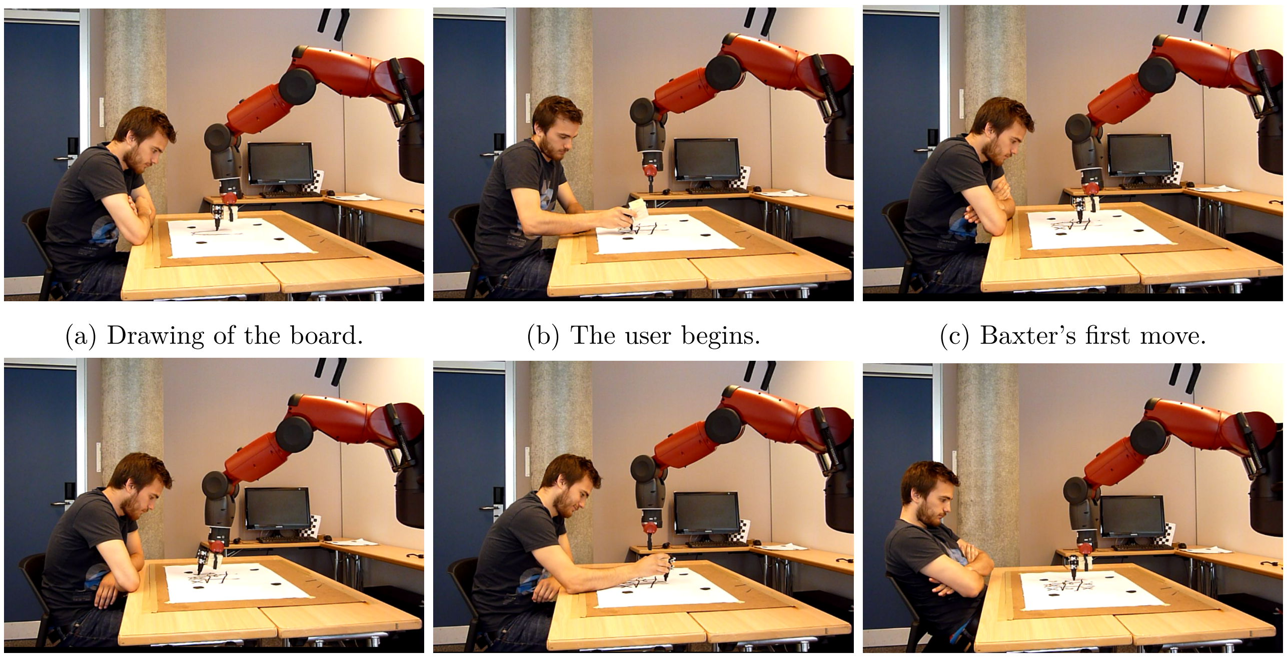
$$\overline{x} = \frac{m_{10}}{m_{00}}, \ \overline{y} = \frac{m_{01}}{m_{00}}. \tag{36}$$

The location of the opponent's move is determined as the space that contains the centre of mass of the detected contour. For example, the contour of Figure 48 is located in the bottom-left space of the board.

9.5 Results

9.5.1 A Game Realisation in Pictures

Figure 49 shows an example of a game of tic-tac-toe between Baxter and the user. Figure 50 shows the projected images of the steps of the game.



(a) Drawing of the board. (b) The user begins. (c) Baxter's first move.

(d) Baxter's second move. (e) User's third move. (f) Baxter's last move.

Figure 49: Steps of the tic-tac-toe game between Baxter and the user.

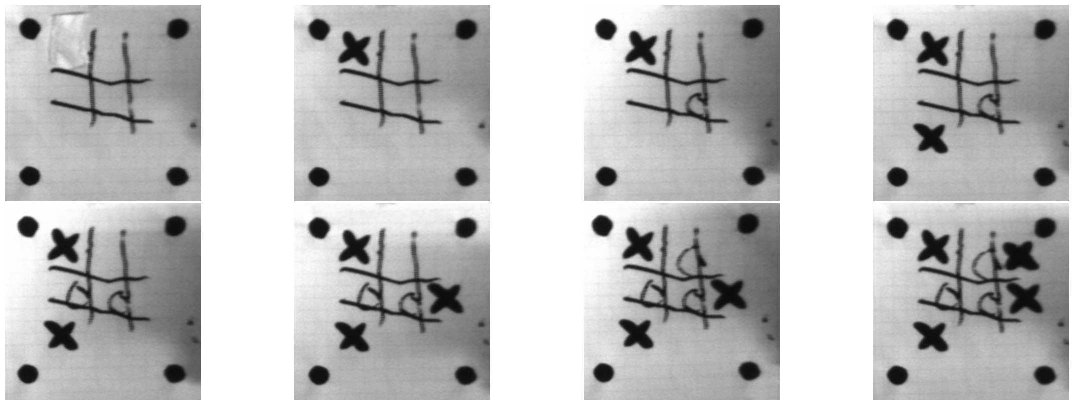
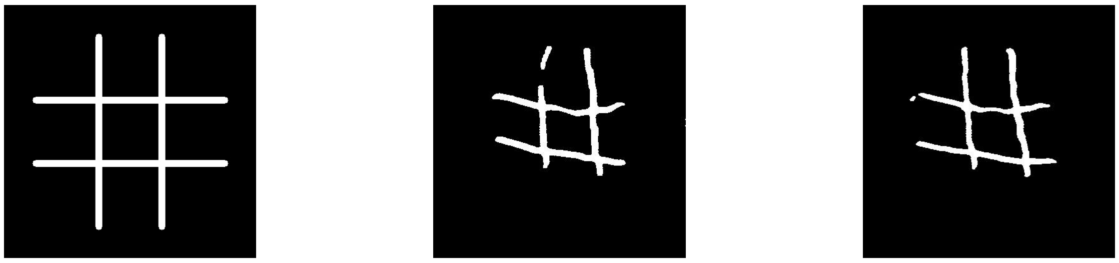


Figure 50: Projected images of steps of a tic-tac-toe game.

9.5.2 Drawings of the Board

The board is drawn using the torque controller with the same maximum drawing velocity of 0.1 m/s as the velocity defined to draw the rectangle. Baxter draws the board line by line. The two vertical lines are drawn followed by the two horizontal ones. The drawings is done from top to bottom and from left to right.

Figure 51 shows the expected board and two examples of the board drawn by Baxter. The drawn images were projected and a thresholding operation was applied. The match values between the expected and drawn images for 6 drawn boards are 8.41, 3.02, 0.57, 2.42, 2.63 and 0.63, so that the mean and standard deviation are 2.94 ± 2.87.



(a) Expected board. (b) Example of drawn board. (c) Example of drawn board.

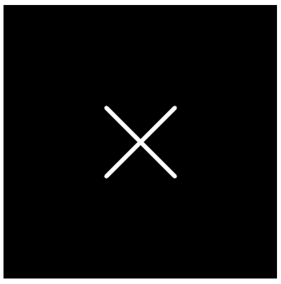
Figure 51: Expected and examples of drawn boards of tic-tac-toe game.

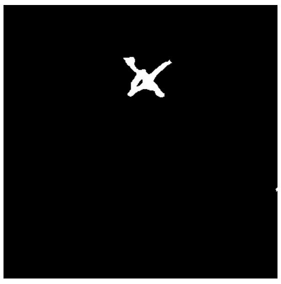
Visually, the last part of the lines are not perfectly drawn by the robot. This effect is more important for the vertical lines than for the horizontal ones. As for the drawing of the fast rectangle, this may be due to the latency of the controller. Moreover, for the vertical lines, the vertical force between the pen tip and the sheet is higher when the end effector is closer to the origin of the robot's frame of reference. This may be related to the pose of the robot and to close singularities which may disturb the movement of the robot. The match values obtained are slightly higher than the value obtained for the rectangle drawn fast.

9.5.3 Drawing of the Symbols

The symbols are drawn using the torque controller with the same maximum drawing velocity of 0.025 m/s as the velocity defined to draw circles slowly. Figures 52 and 53 show the expected symbols and two examples of drawings for crosses and circles. The match values with the expected images were computed for 8 crosses and 8 circles. The match values of the crosses are 4.52 ± 5.08 and those of the circles are 2.89 ± 1.18.

The drawn shapes tend to be smaller than the expected ones, as it was observed for the drawing of the circle with the torque controller. The match values of the circles are also similar to those obtained for the fast drawing of the circle. The match values of the crosses are slightly higher and the standard deviation is high so that some crosses are really similar to the model and others are not.


(a) Expected cross.


(b) Example of drawn cross.

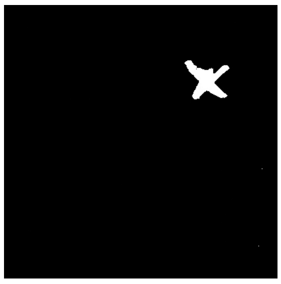
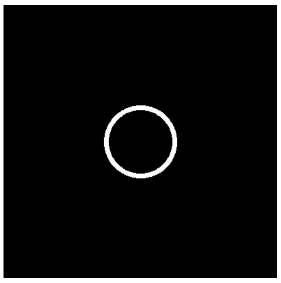
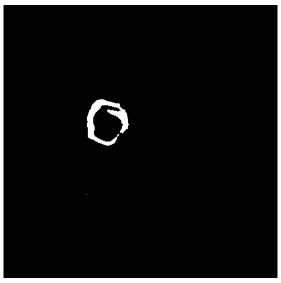

(c) Example of drawn cross.

Figure 52: Expected and examples of drawn cross of tic-tac-toe game.


(a) Expected circle.


(b) Example of drawn circle.

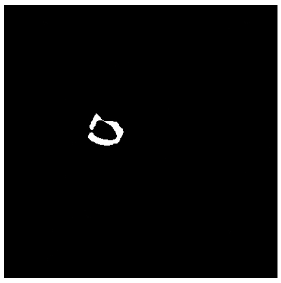
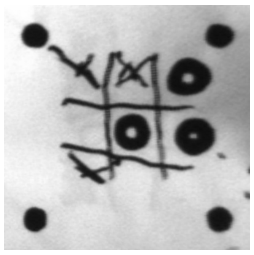

(c) Example of drawn circle.

Figure 53: Expected and examples of drawn circles of tic-tac-toe game.

Figure 54 shows two images of the game where Baxter draws circles or crosses. One can observe that the size of the symbols fits the size of the locations of the board. In this case, the crosses are located in the centre of the squares, but the circles are slightly shifted in the direction of the right side of the locations. However, all symbols are inside their respective squares of the board.


(a) Baxter draws crosses.

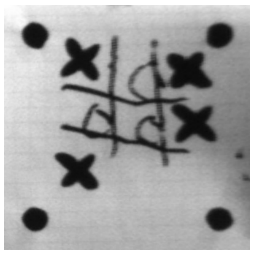

(b) Baxter draws circles.

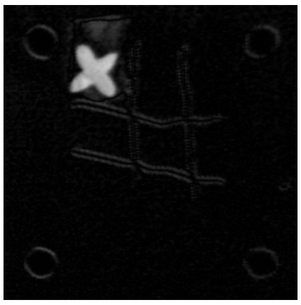
Figure 54: Examples of game states.

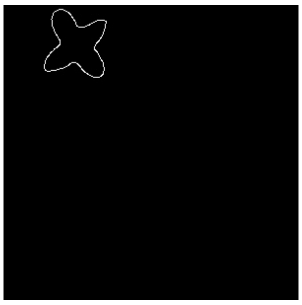
9.5.4 Recognition of the Opponent's Move

A necessary condition to detect a symbol is that it is drawn big enough with thick strokes. Examples are shown in Figure 54.

Figures 55 and 56 show examples of the detection of a cross and a circle drawn by the user. For these two examples, the contours remaining after the edges detection are only those belonging to the symbol. For the cross, the sorted contour corresponds to the top-left location (only the external contour of the circle is sorted).

Figures 57 and 58 show examples of the detection of a cross and a circle drawn by the user for which the absolute difference between the projected image is noisier than for the previous examples. In this case, lots of contours are detected by the Canny edge detector. However, only the contour of the symbol is sorted. Thus, the detection of the symbol is efficient even if the two projected images are slightly shifted compared to each other.


(a) Absolute difference between projected images.


(b) Edges detected.

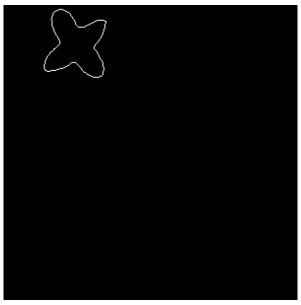
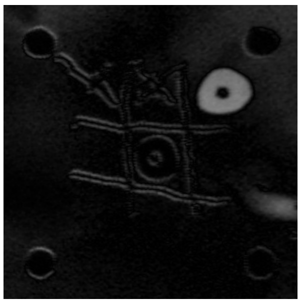
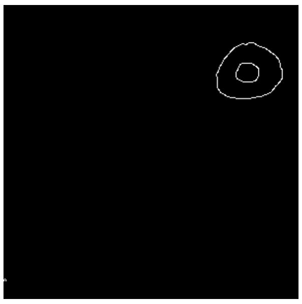

(c) Sorted contour.

Figure 55: Example of the detection of a cross drawn by the user.


(a) Absolute difference between projected images.


(b) Edges detected.

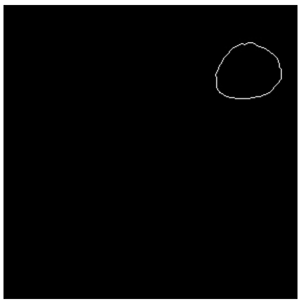
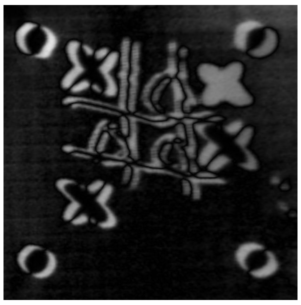
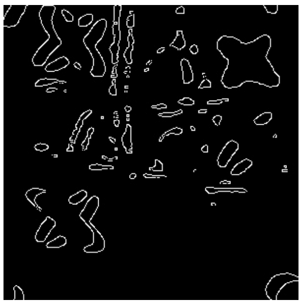

(c) Sorted contour.

Figure 56: Example of the detection of a circle drawn by the user.


(a) Absolute difference between projected images.


(b) Edges detected.

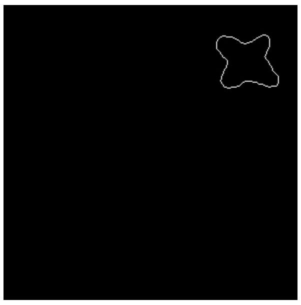
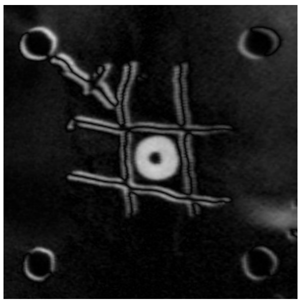
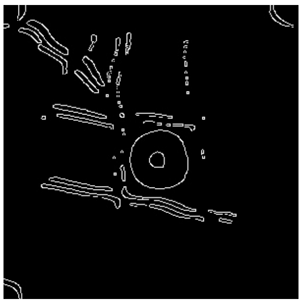

(c) Sorted contour.

Figure 57: Example of the detection of a cross drawn by the user with a noisy difference between projected images.


(a) Absolute difference between projected images.


(b) Edges detected.

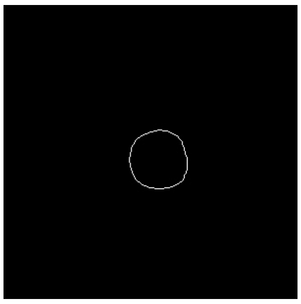

(c) Sorted contour.

Figure 58: Example of the detection of a circle drawn by the user with a noisy difference between projected images.

9.5.5 Discussion

The designed torque controller allows Baxter to play tic-tac-toe against the user. Even if the lines composing the board are truncated, it is possible to see all the locations of the grid to play and draw at the right places. Moreover, the symbols are easily recognisable as crosses or circles even if they are not totally identical to the expected image. The method used to detect the user's moves is efficient to locate the drawn symbol.

The passive and active compliances of the robot are an advantage for this kind of application, as it allows the user to share safely the robot space. Thus, the opponent can stay close to the robot while it is drawing its symbol. Moreover, the method used to project the drawing and identify the user's moves allows the player to move the sheet, in order to draw the symbol and put the sheet back in front of the robot. As the drawing plane is detected before each movement of the robot, it is adapting to a new location of the drawing plane.

10 Conclusion and Future Work

Drawing capabilities have been implemented on Baxter robot. A method to detect the drawing plane was implemented using one of Baxter's camera and the open source library ViSP and compared with a precise estimation obtained with the intervention of the user. Thus, the automatic estimation of the drawing plane made the robot autonomous to draw. Trajectories were generated from pictures with simple shapes using OpenCV functions. Moreover, two different controllers were designed and implemented so that the robot followed the desired trajectories and reproduce the picture on a sheet. The controllers were tested for different maximum drawing speed and for different simple shapes. The drawn images were then evaluated visually and with the moment invariants of images. Finally, a tic-tac-toe game was implemented for which a human is playing against Baxter.

The first observation being made is that the drawing plane is generally estimated a few centimetres higher than it is accorded to the ground truth. Thus, the controllers were designed so that the height of the pen tip relatively to the drawing plane could be adapted during the drawing. Second, both controllers were efficient to adapt their height so that the pen tip remains most of the time in contact with the sheet. Moreover, both are performing better if the shapes drawn are big enough. The images obtained with the velocity controller were more accurate than those obtained with the torque controller. However, the images drawn by both controllers were most of the time recognisable.

The torque controller was used to play tic-tac-toe with Baxter. The drawing of the board was also done by the robot. Even if the end of the lines were truncated, it was usable for the user to play against the robot. The crosses and circles drawn were corresponding to the expected symbols and the symbols drawn by the user were identified and located on the board as long as they were drawn big enough with thick strokes. The compliance of the robot and the torque controller were advantageous for this application as they allows the user to share safely the workspace of the robot and to interact with it.

To improve the quality of the drawn images, the torque controller could be improved by computing precisely the inertia matrix of the robot. In this case, a torque controller in operational space could be used instead of the implemented controller in task space. Thus, a spring behaviour could be simulated by the robot, so that the end-effector could be stiff on the drawing plane and soft in its perpendicular direction. Moreover, the robot could correct the mistakes made during the drawing and complete the missing strokes. Thinking further, the evaluation of the drawn image could be done online, using the robot's camera to evaluate each strokes and accordingly adjust Baxter's movements. The accuracy of the shapes drawn may be improved and the contact between the pen tip and the sheet may be more constant.

Noémie Jaquier,

N. Jaquier

References

[1] P. A. Tresset and F. Fol Leymarie. Sketches by Paul the robot. In *Proc. of the Annual Symp. on Computational Aesthetics in Graphics, Visualization, and Imaging*, pages 17–24, Annecy, France, 2012. Eurographics Association.

[2] P. A. Tresset and F. Fol Leymarie. Portrait drawing by Paul the robot. *Computers & Graphics*, 37:348–363, 2013.

[3] S. Calinon, J. Epiney, and A. Billard. A humanoid robot drawing human portraits. In *5th IEEE-RAS International Conference on Humanoid Robots, 2005.*, pages 161–166, Dec 2005.

[4] P. Tresset and O. Deussen. Artistically skilled embodied agents. *Proc. of AISB2014*, 2014.

[5] A. Srikaew, M. E. Cambron, S. Northrup, R. A. Peters, D. M. Wilkes, and K. Kawamura. Humanoid drawing robot. In *In Proceedings of the IASTED International Conference on Robotics and Manufacturing*, 1998.

[6] T. Perry and P. Alto. Inside an experimental robotics class: A robot sketch artist, a robot that plays dominos, and more, 2016 (accessed 06.2016). `http://spectrum.ieee.org/view-from-the-valley/robotics/robotics-software/a-robot-sketch-artist-a-robot-that-plays-dominos-and-more`.

[7] Rethink Robotics. Baxter API Reference, 2015 (accessed 02-06.2016). `http://sdk.rethinkrobotics.com/wiki/API_Reference#Sonar`.

[8] Rethink Robotics. *Baxter Datasheet*, October 2015.

[9] E. Marchand, F. Spindler, and F. Chaumette. ViSP for visual servoing: a generic software platform with a wide class of robot control skills. *IEEE Robotics and Automation Magazine*, 12:40–52, December 2005.

[10] D. F. Dementhon and L. S. Davis. Model-based object pose in 25 lines of code. *International Journal of Computer Vision*, 15(1):123–141, 1995.

[11] D. Oberkampf, D. Dementhon, and L. S. Davis. Iterative pose estimation using coplanar feature points. *Computer Vision and Image Understanding*, 63:495–511, 1996.

[12] E. Marchand and F. Chaumette. Virtual visual servoing: A framework for real-time augmented reality. In G. Drettakis and H.-P. Seidel, editors, *EUROGRAPHICS 2002 Conference Proceeding*, volume 21(3) of *Computer Graphics Forum*, pages 289–298, Saarebrücken, Germany, September 2002.

[13] Z. Guo and R. W. Hall. Parallel thinning with two-subiteration algorithms. *Commun. ACM*, 32:359–373, March 1989.

[14] Nash. Implementation of Guo-Hall algorithm, 2013 (accessed 04.2016). `https://web.archive.org/web/20160314104646/http://opencv-code.com/quick-tips/implementation-of-guo-hall-thinning-algorithm/`.

[15] Itseez. OpenCV 3.0.1 online documentation, 2016 (accessed 04-06.2016). `http://docs.opencv.org/3.1.0/#gsc.tab=0`.

[16] S. Suzuki and K. Abe. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30:32–46, 1985.

[17] J. Diebel. Representing attitude: Euler angles, unit quaternions, and rotation vectors, 2006.

[18] B. Siciliano and O. Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.

[19] M. Bouri and H. Bleuler. Bases de la robotique. *Course for EPFL Master students*, 2012.

[20] M. W. Spong. *Robot Dynamics and Control*. John Wiley & Sons, Inc., New York, NY, USA, 1st edition, 1989.

[21] M. Hu. Visual pattern recognition by moment invariants. *IRE Transactions on Information Theory*, 8:179–187, February 1962.

[22] Wikipedia. Tic-tac-toe, 2013 (accessed 05.2016). `https://en.wikipedia.org/wiki/Tic-tac-toe`.

[23] Never Stop Building LLC. Tic tac toe : understanding the minimax algorithm, 2013 (accessed 05.2016). `http://neverstopbuilding.com/minimax`.

[24] M. Unser. Image processing, volume 1. *Course for EPFL Master students*, 2012.

[25] S. Calinon. *Robot Programming by Demonstration*. CRC Press, Inc., Boca Raton, FL, USA, 1st edition, 2009.