THE UNIVERSITY *of* EDINBURGH

Robotics: Science and Systems

Course Assignment 1

Henrique Ferrolho - s1683857
henrique.ferrolho@gmail.com

November 4, 2016

1 Forward and inverse kinematics

1.1 (10 marks)

$$q = \begin{pmatrix} 0.66 \\ -0.44 \\ 1.06 \\ 1.33 \end{pmatrix}$$

$$T_{base \to L_1} = \begin{bmatrix} cos(q_1) & -sin(q_1) & 0 & 0 \\ sin(q_1) & cos(q_1) & 0 & 0 \\ 0 & 0 & 1 & 0.16 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{L_1 \to L_2} = \begin{bmatrix} cos(q_2) & 0 & sin(q_2) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(q_2) & 0 & cos(q_2) & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{L_2 \to L_3} = \begin{bmatrix} cos(q_3) & 0 & sin(q_3) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(q_3) & 0 & cos(q_3) & 0.2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{L_3 \to L_4} = \begin{bmatrix} cos(q_4) & 0 & sin(q_4) & 0 \\ 0 & 1 & 0 & 0 \\ -sin(q_4) & 0 & cos(q_4) & 0.15 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_{L_4 \to gripper} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0.35 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\begin{aligned} T_{base \to gripper} &= T_{base \to L_1} \cdot T_{L_1 \to L_2} \cdot T_{L_2 \to L_3} \cdot T_{L_3 \to L_4} \cdot T_{L_4 \to gripper} \\ &= \begin{bmatrix} -0.29243998 & -0.61311685 & 0.73387096 & 0.25840905 \\ -0.22696411 & 0.78999223 & 0.56956086 & 0.20055253 \\ -0.92895972 & 0 & -0.37018083 & 0.48346881 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned}$$

And therefore:

$$\boldsymbol{y_t} = \begin{bmatrix} 0.25840905 \\ 0.20055253 \\ 0.48346881 \end{bmatrix}$$

1.2 (10 marks)

Kinematic map $y = \Phi(q)$:

$$y = \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \end{bmatrix}$$

$$J(q) = \begin{bmatrix} \frac{\partial x}{\partial q_1} & \frac{\partial x}{\partial q_2} & \frac{\partial x}{\partial q_3} \\ \frac{\partial y}{\partial q_1} & \frac{\partial y}{\partial q_2} & \frac{\partial y}{\partial q_3} \\ \frac{\partial z}{\partial q_1} & \frac{\partial z}{\partial q_2} & \frac{\partial z}{\partial q_3} \end{bmatrix}$$

1.3 (10 marks)

2 Signal filtering & State Estimation

2.1 (10 marks)

Real world applications require signal filtering because they are not controlled environments - conveyor belts in factories are ideal controlled environments, but most of real world applications are not like that.
Since the environment is not controlled, sensors are subjected to events which introduce unwanted noise. Signal filtering is crucial to remove this undesired noise, and preserve the important signal information as clearly as possible.

The types of filters that are commonly used are:

- **Low-pass filter** - low frequencies lower than cut-off frequency are passed
- **High-pass filter** - high frequencies higher than cut-off frequency are passed
- **Band-pass filter** - only frequencies within a frequency band are passed
- **Band-stop filter** - only frequencies within a frequency band are attenuated
- **Notch filter** - rejects a particular frequency

The problem described is a noise reduction/removal problem, and as such a low-pass filter should be used. If the acceleration frequency is known, then the cut-off frequency can be set to a value close to it.

2.2 (10 marks)

State estimation is the process by which the state of a system is calculated. In real world applications, systems often have multiple measurements from different sensors built into them. These measurements may vary from one another, and in such cases the system still needs to be able to predict the real system state as close to reality as possible.

State estimation is important for scenarios like that: by weighting all the variance of each measurement and estimating the most likely state for the system.

The *least squares* (LS) estimate is:

$$\begin{aligned}\hat{x}^{LS} &= \arg\ \min_x \sum_{k=1}^{N} (y_k - H_k x)^T (y_k - H_k x) \\ &= \arg\ \min_x (\mathbf{y} - \mathbf{H}x)^T (\mathbf{y} - \mathbf{H}x)\end{aligned}$$

Direct differentiation and setting the result to zero gives the estimate:

$$\hat{x}^{LS} = (\mathbf{H}^T\mathbf{H})^{-1}\mathbf{H}^T\mathbf{y}$$

2.3 (20 marks)

The dynamics of the system can be formulated as follows:

$$m\ddot{z}(t) = -\gamma\dot{z}(t) - kz(t)$$

Given measurements z_1, α_1 and z_2, α_2:

$$\begin{aligned}\hat{x} &= [\frac{\sigma_{z_2}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2}]z_1 + [\frac{\sigma_{z_1}^2}{\sigma_{z_1}^2 + \sigma_{z_2}^2}]z_2 \\ &= [\frac{\alpha_{z_2}^4}{\alpha_{z_1}^4 + \alpha_{z_2}^4}]z_1 + [\frac{\alpha_{z_1}^4}{\alpha_{z_1}^4 + \alpha_{z_2}^4}]z_2\end{aligned}$$

The majority of sensors are subject to high frequency noise, so a low-pass filter should be used to clear the noise from both measurements.

Since the on-board transducer has high accuracy it might not need to pass through the low-pass filter. Nonetheless, if it does need to go through the filter, its threshold would be much lower than the one used on the external vision sensor.

The identification problem can be formulated as follows:

$$w_0 = \sqrt{\frac{k}{m}}$$

$$\xi = \frac{\gamma}{2\sqrt{km}}$$

For a series of k measurements, 1, 2, $\ldots k$, construct the least square problem:

$$\begin{bmatrix} \ddot{z_1} \\ \ddot{z_2} \\ \vdots \\ \ddot{z_k} \end{bmatrix} = \begin{bmatrix} -z_1, -2\dot{z_1} \\ -z_2, -2\dot{z_2} \\ \vdots \\ -z_k, -2\dot{z_k} \end{bmatrix} \begin{bmatrix} w_0^2 \\ \xi w_0 \end{bmatrix}$$

3 Computer Vision

3.1 Camera Geometry (10 marks)

$f = 10\,\text{mm}$

$D = 10\,\text{m} = 10\,000\,\text{mm}$

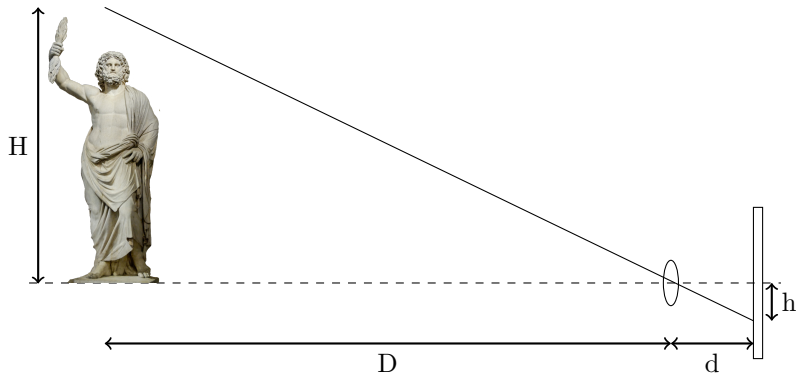
$sensor : 10\,\text{mm} \times 10\,\text{mm} = 1000 \times 1000 \text{ pixels}$

$h = 100 \text{ pixels} = 1\,\text{mm}$

$$\frac{1}{D} + \frac{1}{d} = \frac{1}{f}$$
$$\frac{1}{d} = \frac{1}{f} - \frac{1}{D}$$
$$\frac{1}{d} = \frac{1}{10} - \frac{1}{10000}$$
$$\frac{1}{d} = \frac{1000}{10000} - \frac{1}{10000}$$
$$\frac{1}{d} = \frac{999}{10000}$$
$$d = \frac{10000}{999}$$

$$\frac{d}{D} = \frac{h}{H}$$
$$H = \frac{hD}{d}$$
$$H = \frac{1 \times 10000}{\frac{10000}{999}}$$
$$H = 999\,\text{mm} = 0.999\,\text{m}$$

The observed statue is 0.999 m tall.



The distance D to the statue needs to be known in order to calculate the distance d between the lens and the film/sensor, using the focal length f. Furthermore, given D, d, and h, it is easy to calculate the height of the statue H.

3.2 Image Processing (10 marks)

The following row of images/samples is a visual representation of the *intermediate* keypoints of the car counter program.

Firstly the program fetches the frames from the live camera feed - the first image from the row of images above is an example of a frame fetched from the camera.
Afterwards it does a *image segmentation* by applying a *background removal* technique - second image, counting from the left, in the images row above.
At this point it is useful to turn the image into a binary image - middle image sample.
Then two *morphological transformations* are applied to remove the noise left in the frame - also known as *morphological closing* and *opening* - penultimate image sample.
Finally the last image from the samples row above shows the *region of interest* - region in blue, activated by the passing car (red segment of the detection region).

The general idea is to count every time a white spot is detected in the blue line - actually, to introduce some tolerance, it should not be a line with a single pixel thickness, rather a 5 pixels thick line, for example.

Furthermore, to save computation time, and since we only care for what is going on inside the blue region, we can *crop* everything else besides that region right after the frame capture. Nonetheless, the entire frame of each of the samples above is used (no cropping) for better understanding of the main idea.

Algorithm 1 Car counter program

```
1: procedure CarCounter
2:     carPassing ← False
3:     carCounter ← 0
4:     loop
5:         frame ← cam.fetchFrame()
6:         CropRegionOfInterest(frame)   ▷ Crop to save computation time
7:         SubtractBg(frame)
8:         BinaryThreshold(frame)
9:         MorphOpen(frame)
10:        MorphClose(frame)
11:        if carFound(frame) then
12:            if not carPassing then
13:                carPassing ← True
14:                counter ← counter + 1
15:            end if
16:        else if carPassing then
17:            carPassing ← False
18:        end if
19:    end loop
20: end procedure

21: function SubtractBg(frame)
22:    hourOfDay ← getTime().hour
23:    bg ← DB.image("backgrounds/" + hourOfDay + ".png")
24:    frame.subtractImg(bg)
25: end function

26: function MorphOpen(frame)
27:    erode(frame)
28:    dillate(frame)
29: end function

30: function MorphClose(frame)
31:    dillate(frame)
32:    erode(frame)
33: end function

34: function CarFound(frame)
35:    for i ← 0, frame.size do
36:        for j ← 0, frame[i].size do
37:            if frame[i][j] = RGB(255, 255, 255) then
38:                return True
39:            end if
40:        end for
41:    end for
42:    return False
43: end function
```

3.3 Shape Recognition (10 marks)

The first thing to do in order to narrow the search results by 50% is to check if the card has *red* or *black* symbols - a simple *RGB normalization* can be used. After this step the search results are narrowed either to *spades and clubs*, or *hearts and diamonds*.

A Bayes classifier needs to be trained using *4 classes*, one for each suit.