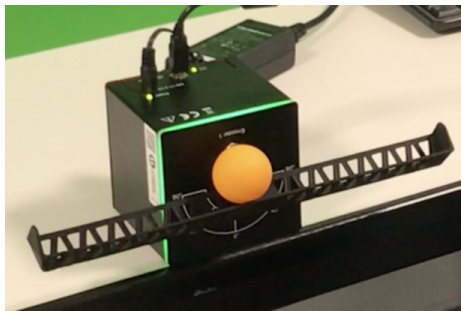




# Vision-based cascade control of a Ball & Beam system



Hugues Garnier  
Floriane Collin  
3A IA2R

January 2024

# Contents

<b>1</b>	<b>Vision-based cascade control of a Ball &amp; Beam system</b>	<b>2</b>
1.1	Objectives and layout of the Lab	2
1.2	Download of the files required for the lab	3
1.3	Hardware required	3
1.3.1	The QUBE servo-motor	3
1.3.2	Logitech C270 HD camera	4
1.3.3	Ball & Beam	4
1.4	Data-driven modelling & PD control of the beam angular position	5
1.4.1	Control performance requirements	5
1.4.2	Recording of the step response experiment	6
1.4.3	Data-driven model identification and validation	7
1.4.4	Design of a PD control	8
1.5	Vision-based ball position measurement	10
1.5.1	Image acquisition	10
1.5.2	Image processing	11
1.5.3	Ball position measurement	13
1.5.4	Use of image compare block to detect ball object	15
1.5.5	Convert and filter ball measurement	16
1.6	Ball & Beam cascade control design	18
1.6.1	Control performance requirements	18
1.6.2	Transfer function model	18
1.6.3	Cascade control design	19
1.6.4	Control performance in simulation	21
1.7	Ball & Beam cascade control implementation	21
1.7.1	Validation of the control on the real system	21
1.7.2	Control tuning	22
1.8	Troubleshooting	23

## Acknowledgements

The contents of this lab has been inspired and adapted from an initial version provided by Quanser Inc<sup>1</sup>. This is fully acknowledged.

Special thanks go also to Florian Fritz, former IA2R student, for having provided some contributions to the content of this lab.

---

<sup>1</sup>[www.quanser.com](http://www.quanser.com)

# Lab 1

---

## Vision-based cascade control of a Ball & Beam system

### 1.1 Objectives and layout of the Lab

Vision-based control techniques are at the intersection of the fields of robotics, automatic control and computer vision as shown in 1.1. They use the information provided by a vision sensor to control the movements of a dynamic system.

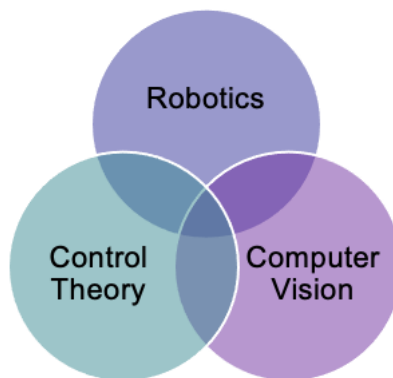


Figure 1.1: Vision-based control techniques at the intersection of robotics, control and computer vision


These techniques have become very popular in the recent years thanks to deep learning and the computing power of modern computers.

Moreover, Industry 4.0 is interested by these techniques with the idea of making mobile robots more aware of their environment and therefore able to adapt to it in the presence of humans around.

Contrary to the classical control that uses measure from electronic sensor, image-based control uses a camera and extracts useful information from it. Image processing becomes then prominent.

In this lab, we will try to understand how to estimate the position information of an object from an image. Then, we will use the information in order to design a ball & beam controller. It should be able to stabilize a ball on a beam at various positions using the QUBE Servo 2 from Quanser. Watch for example, the short presentation video of what you should be able to achieve at the end of this lab: [www.youtube.com/watch?v=BvLVn8tZBLs&t=2s](http://www.youtube.com/watch?v=BvLVn8tZBLs&t=2s).

## 1.2 Download of the files required for the lab

1. Download the zipped file *Lab\_QUBE\_Ball\_Beam.zip* from the course website. Save and unzip it in the local disc folder `C:/temp/`.
2. Start Matlab.
3. Important ! By clicking on the browse for folder icon , **change the current folder of Matlab so that it becomes your `C:/temp/Lab_QUBE_Ball_Beam` folder** that contains the files needed for this lab.

## 1.3 Hardware required

The required hardware for this lab includes:

- a QUBE-Servo 2;
- a USB Webcam Logitech C270 HD;
- a 3D printed beam;
- a ping-pong ball;
- a LED lighting panel.

### 1.3.1 The QUBE servo-motor

The Quanser QUBE-Servo 2, pictured in Figure 1.2, is a compact rotary servo system that can be used to perform a variety of classic servo control and inverted pendulum based experiments. The QUBE can be controlled by a computer via USB connection.

The system is driven using a direct-drive 18V brushed DC motor. The motor is powered by a built-in Pulse Width Modulation (PWM) amplifier with integrated current sense. The beam can be easily attached or interchanged using magnets mounted on the QUBE-Servo 2 module connector.

Single-ended rotary encoders are used to measure the angular position of the DC motor and pendulum, while the angular velocity of the motor can be estimated from the encoder or measured using an integrated software-based tachometer.



(a) QUBE-Servo 2 with Inertia Disc Module      (b) QUBE-Servo 2 with Pendulum Module

Figure 1.2: QUBE-Servo 2 with the 2 different modules

### 1.3.2 Logitech C270 HD camera



Figure 1.3: Logitech C270 HD

The Logitech C270 HD Webcam is an easy to use USB Webcam to interface with Simulink. It can be mounted on a screen and be used to provide the visual information about the ball position. The video capture is up to 1289 \* 720 pixels but we will use only 640\*480 for simplicity. As for the image acquisition, the webcam provide 30 frames per second. You can find the full technical documentation if needed on this [link](#).

### 1.3.3 Ball & Beam

A 3D printed beam is mounted on the QUBE-Servo 2 instead of the inertial disk or the rotary pendulum module. It allows the ball to roll from a side to the other of the beam. The aim is to stabilize the ball on the beam at various distance from the center. We will use an hollow orange ball because it makes the tracking easier but you can test the plain white ball if you want to go further at the end of the lab.

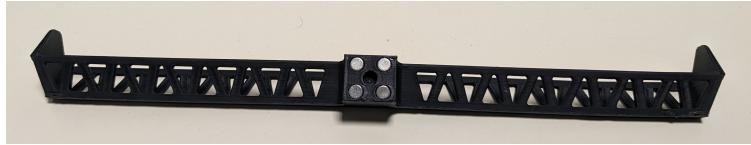


Figure 1.4: 3D printed Beam



Figure 1.5: Balls

## 1.4 Data-driven modelling & PD control of the beam angular position

In this part we will design the servo control of the beam. It is similar to the inertial disk as the beam can rotate at 360 degrees. Make sure however that the beam cannot hit the table when rotating by placing the QUBE-Servo 2 at the edge of your desk.

### 1.4.1 Control performance requirements

The performance requirements for the **beam angular position** control are described in Table 1.1.

Requirement	Assessment criteria	Level
Control the angular servo position	Position setpoint tracking	No steady-state error
	Motor input voltage	limited to [-10V ; +10 V]
	Settling time at 5 %	As short as possible
	Overshoot	Less than or equal to 5%
	Peak-time	0.1s
	Disturbance rejection	Rejection of load effects

Table 1.1: Performance requirements for the beam angle control

The determination of a model is the first crucial step for the design of a feedback control system.

The input and output of the DC motor of the QUBE with the inertia disk are:

- input: voltage of the motor  $u(t)$  in V;
- output: beam angular position  $\theta(t)$  in rad.

The motor voltage-to-angular position transfer function takes the form of a first-order plus pure integrator model

$$\frac{\Theta(s)}{U(s)} = \frac{K}{s(1 + Ts)} \quad (1.1)$$

where  $\Theta(s) = \mathcal{L}[\theta(t)]$  and  $U(s) = \mathcal{L}[u(t)]$ .  $K$  in rad/(V-s) is the model steady-state gain,  $T$  in s is the model time-constant.

As the angular velocity is the time-derivative of the angular position, both variables are linked in the Laplace domain by an integrator (or derivator) so that (1.1) can be expressed as:

$$\frac{\Theta(s)}{U(s)} = \frac{\Theta(s)}{\Omega(s)} \times \frac{\Omega(s)}{U(s)} = \frac{1}{s} \times \frac{K}{1 + Ts} \quad (1.2)$$

where  $\Omega(s) = \mathcal{L}[\omega(t)]$  is the motor angular velocity (or speed) of the inertia disk.

Identifying a system having a pure integrator is tricky and it is better when the measure is available to identify the response between the motor angular velocity and the input voltage since the model takes the form of a simple first-order system.

The QUBE-Servo 2 motor voltage-to-angular velocity transfer function has therefore the well-known first-order form which parameters can be easily estimated from a step response experiment:

$$\frac{\Omega(s)}{U(s)} = \frac{K}{1 + Ts}, \quad (1.3)$$

### 1.4.2 Recording of the step response experiment

1. Open the file *Step\_resp\_Qube.slx* in Simulink.
2. Click on **Monitor & Tune** in the Hardware panel to run the step test that lasts 10 seconds only. The red color of the Qube should turn green if the test succeeds.
3. Observe the angular velocity response for a positive step from 0 to 2 V sent after 1s followed by a negative step from 2 V to 0 on the motor voltage after 5 seconds. The amplitude of the steps can be modified according to your own reasoning and choice. To do so, click on the two step blocks and modify the value.

The angular position (in rad) and velocity (in rad/s) responses to the positive and negative step input sent to the motor voltage have been recorded and saved in the *data\_step\_Qube.mat* file. They are plotted in Figure 1.6, which can be reproduced with Matlab by executing the *step\_response\_plot.m* file.



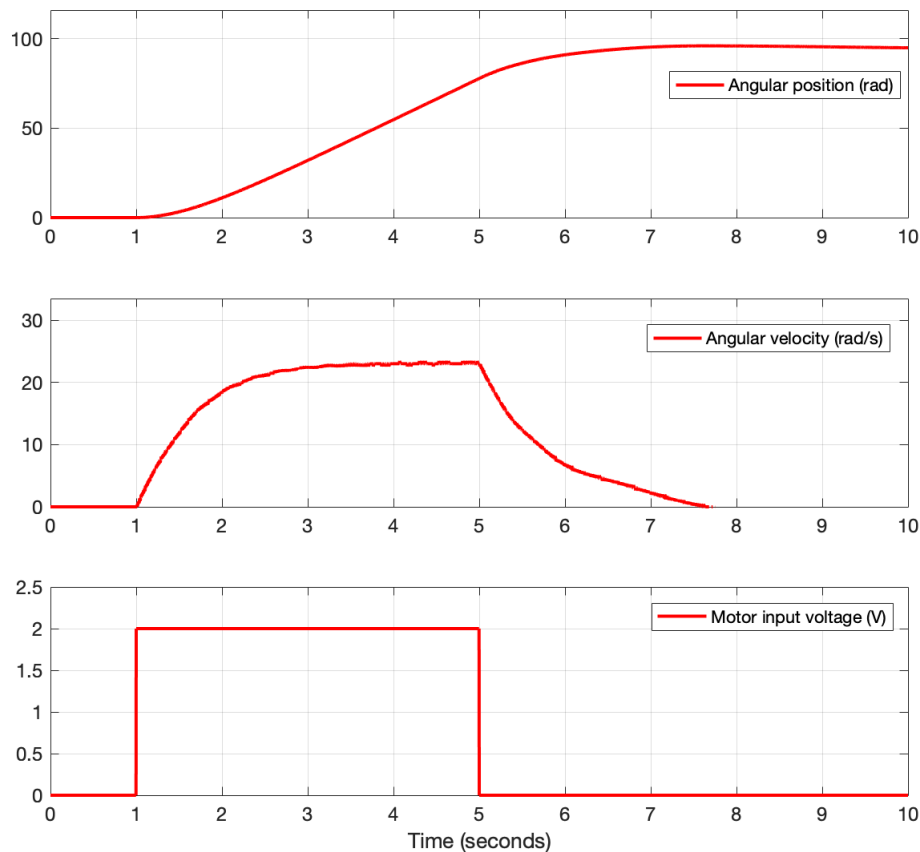


Figure 1.6: Beam angular position and velocity responses to a positive and negative step input sent to the motor voltage

### 1.4.3 Data-driven model identification and validation

It is possible to use basic methods to determine the parameters  $K$  and  $T$  of a first-order transfer function model from the positive step response for example.

1. Open the file `step_response_plot.m` in Matlab and run it.
2. From the angular velocity response plot, determine the parameters of a first-order model.
3. Open the file `test_your_model.m` in Matlab.
4. At the beginning of the file, modify the default values of the steady-state gain  $K$  and time-constant  $T$  parameters, then run the program.
5. Compare the measured and model angular velocity responses.
6. Adjust by trial and error the two model parameters to improve the fit between the measured and simulated angular velocity responses.

Note that the response to the negative step is quite different to the response to the positive step due to some nonlinear effects. This is not crucial for the model-based control design that will follow.

### 1.4.4 Design of a PD control

To balance the beam angular position we use a variation of a PD control where the derivative action is applied to the output instead of the control error. This form of PD control is also known as proportional-velocity (PV) control.

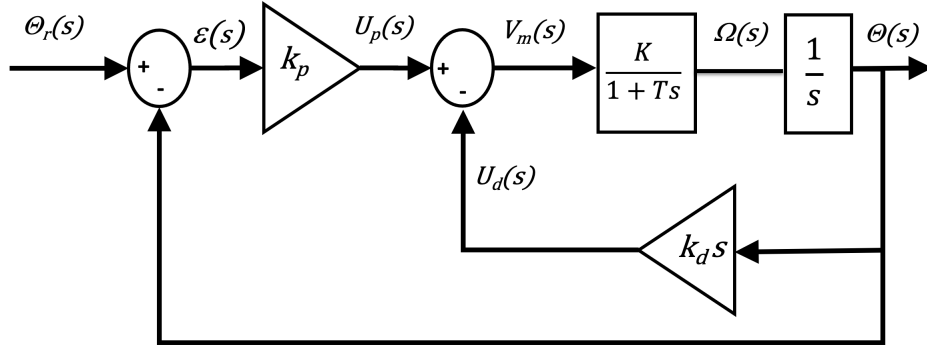


Figure 1.7: Proportional-velocity (PV) control

One of the advantages of having the derivative action on the output is to avoid the spike effect for a step change on the setpoint. The other advantage here is to obtain a closed-loop transfer function of the form of a standard second-order transfer function (with just a constant coefficient for the numerator and no zero).

In Figure 1.7 we have the relationship for the inner loop:

$$F_i(s) = \frac{\Theta(s)}{U_p(s)} = \frac{C(s)G(s)}{1 + C(s)G(s)H(s)}$$

with

$$C(s) = \frac{K}{1 + Ts}, G(s) = \frac{1}{s}, H(s) = k_d s$$

which leads to

$$F_i(s) = \frac{K}{s(Ts + 1 + Kk_d)}$$

The closed-loop transfer function then becomes

$$F_{cl}(s) = \frac{\Theta(s)}{\Theta_r(s)} = \frac{Kk_p}{Ts^2 + (1 + Kk_d)s + Kk_p}$$

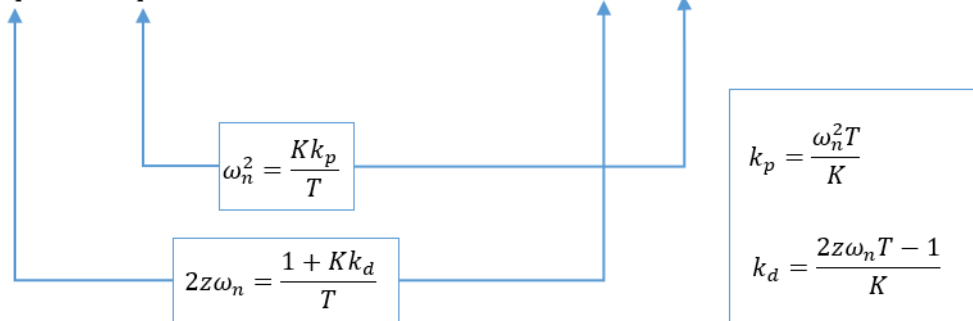
Finally, from the closed-loop transfer function, we can determine the values of  $k_p$  and  $k_d$  of the PV controller by identification with the standard second-order transfer function as shown below.

Closed-loop transfer function

$$\frac{Y(s)}{R(s)} = \frac{Kk_p/T}{s^2 + \frac{(1 + Kk_d)s}{T} + \frac{Kk_p}{T}}$$

Prototype second-order transfer function

$$\frac{Y(s)}{R(s)} = \frac{\omega_n^2}{s^2 + 2z\omega_n s + \omega_n^2}$$



1. Open the file `cd_qube_servo_pd.m`;
2. Enter the first-order model parameters  $K$  and  $T$  previously identified.
3. Run the `.m` file to calculate the gains.
4. Open the file `q_qube2_pd_control_tune.slx` in Simulink.
5. Click on **Run** to simulate and observe the response.
6. Are the requirements satisfied ?
7. Close the Simulink file.

Remark: If the closed-loop control becomes unstable, set the controller gains  $k_p$  to 10 and  $k_d$  to 1 before increasing them progressively to get good control performances.

## 1.5 Vision-based ball position measurement

In the first part, we designed a PD controller for the angular position of the beam. In classical control theory the measurement of the ball position would be directly provided by a sensor. The specificity of this lab is to design a vision-based control. We will use image processing to get the ball center position estimation/measure from the camera images.

To get a short overview on how to use cameras in vision-based control applications, watch the first 8 mn of the Quanser webinar:

[youtu.be/M2aEgXVvfXc?si=iSNF6iXjtb5J25I](https://youtu.be/M2aEgXVvfXc?si=iSNF6iXjtb5J25I)

The different tasks can be designed as follows:

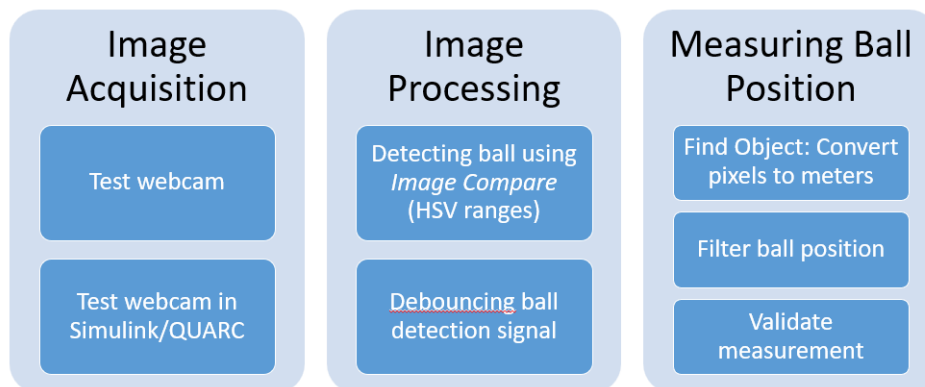


Figure 1.8: Ball position measurements tasks

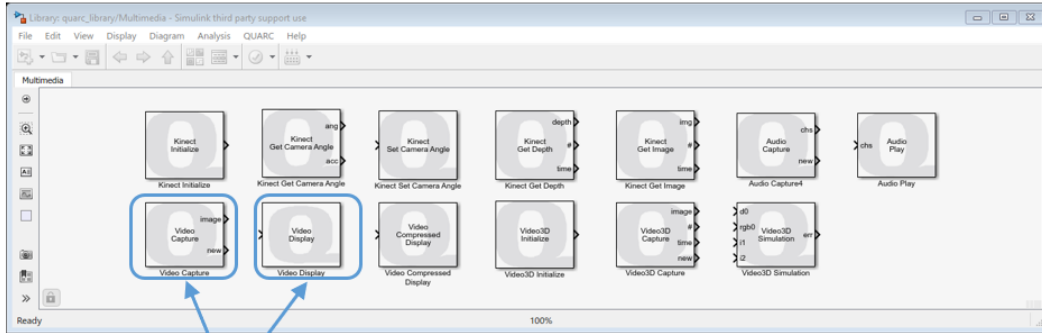
### 1.5.1 Image acquisition



Figure 1.9: Image acquisition

The QUARC library provides video acquisition blocks as shown in Figure 1.10.

1. Connect the camera to the USB port of the computer.
2. Open the file `q_camera_test_tune.slx` in Simulink.
3. Click on **Monitor & Tune** in the Hardware panel to run the test.
4. Observe the output video by double click on the *Video Display* block. If the Video display remains black, see the Troubleshooting section at the end of the document.
5. By now you should see the video capture of the camera. Adjust the position of the camera on the seat to make sure the beam length fits within the width of the camera view.

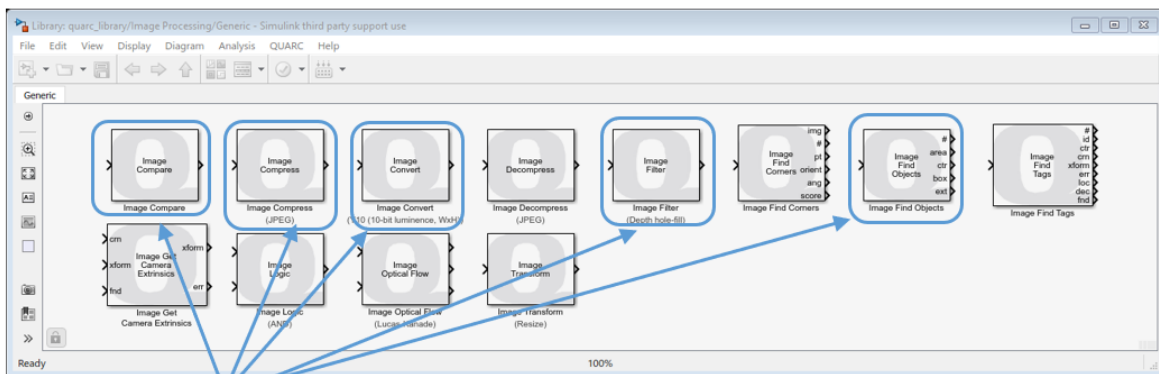


- **Video Capture** block to acquire video from webcam.
- **Video Display** block to view video feed.

Figure 1.10: Image and video acquisition QUARC blocks

### 1.5.2 Image processing

The QUARC Library provides Image processing blocks to track the ball and filter the images.



Use:

- **Image Compare:** filter out image to view only ball based on HUE colour ranges
- **Image Compress** to view video and save on bandwidth.
- **Image Convert:** convert raw image to HUE HSV files
- **Image Filter:** use to reduce noise in Image Compare output
- **Image Find Objects:** detect ball using the output of the Image Compare/Filter

Figure 1.11: Image processing QUARC Blocks

1. Open the file *image\_processing.slx*. The file contains all the blocks and subsystems used in the image processing.
2. Double click on the *Image Processing* subsystem.

The input is an RGB imaged compressed by 20% from the *Image Acquisition* block. It filters images according to the HSV (hue, saturation, brightness) ranges entered in the image.



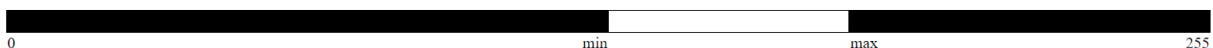
Figure 1.12: Hue - Saturation - Value

**Hue** is just another word for the color. **Saturation** is the intensity of color. Highly saturated colors are for instance those that appear bright and desaturated colors the ones that look grayed. **Value** is the brightness, dark or light, compared to a scale of grays from black to white.

We can identify an object in an image by looking at the changes of these parameters within the image.

**Image Transform** Firstly, the *Image Transform* block converts the raw image using a specific transformation algorithm. The input is an RGB image specified as a three-dimensional  $M \times N \times 3$  matrix where  $M$  is the image height and  $N$  is the image width. It contains the red, green and blue components for each pixel in the image. The algorithm used is the RGB to HSV from Quanser. It converts the RGB matrix at the input to an HSV matrix. The HSV matrix has the same size and data type as the input matrix. Hues, saturations and values are all computed to lie within the range of the data type. Hence, for an uint8 RGB matrix input the hue, saturation and value output will range from 0 to 255.

**Image Compare** Then, we use the *Image Compare* Block to compare the pixels from the HSV input image to given ranges. The output is one or more bitonal images whose pixels are 255 if the condition for that pixel is true and 0 otherwise. There are two fundamental types of comparisons: saturating and wrapping. The normal, or saturating, comparisons simply compare each pixel to the minimum and maximum values and output the result. In this case, minimum and maximum values are saturated to the range of the input image data type prior to performing the comparison. In the wrapping comparisons, the minimum and maximum values are truncated to the range of the input image data type so that they wrap around like a circular buffer or modulus arithmetic. However, the wrapping comparison takes this roll-over into account.



**Image filter** The *Image Filter* block filters an image or a proportion of an image in order to reduce noise. To filter only a portion of the image, a rectangular region of interest is defined. The image may be filtered within the region of interest or outside the region of interest. The location of the region of interest may be specified via block dialog box parameters or by an external input, allowing the region of interest to be changed dynamically.

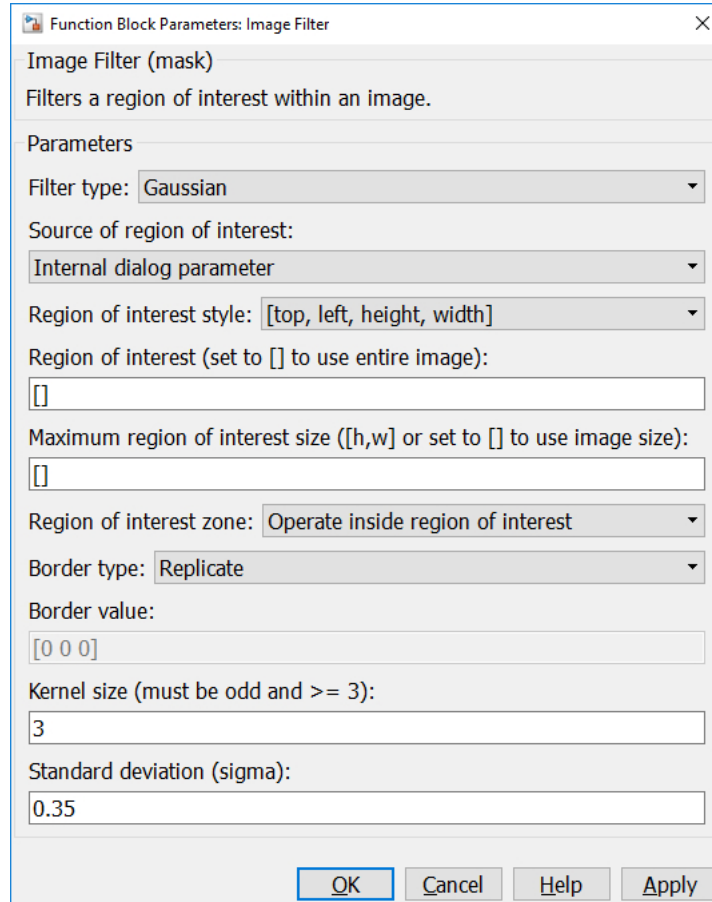


Figure 1.13: Image Filter parameters

There are several filter types that can be applied. We will use *Depth hole-fill* which is specifically designed for depth images from an RGBD camera that fills in holes where the RGBD camera could not determine the depth of a pixel. It treats a zero pixel as an unknown depth. This filter is typically applied after the Depth temporal low-pass filter so that it is operating on as much valid depth data as possible.

We could have use *Box* which blurs the image by averaging the pixels in the neighborhood or a *Low-pass* filter.

### 1.5.3 Ball position measurement

Now that we have a filtered image, we need to find the position of the ball on the beam. The first step is to use the *Image Find Objects* block to output the centroid of the ball.

The *Image Find Objects* block finds objects within an image. It assumes that the image has been preprocessed to isolate the colors of interest. It detects objects consisting of non-zero pixels in the image.

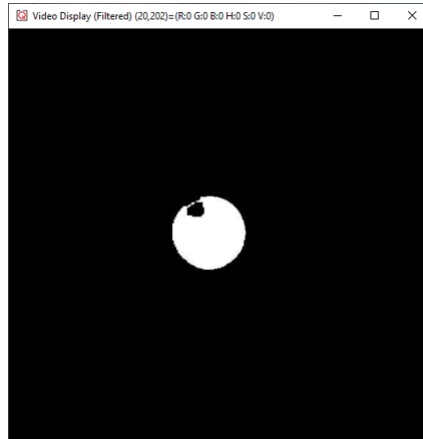


Figure 1.14: Filtered image after image processing

For each object found, the block outputs the bounding box, number of pixels in the object, and centroid of the object. The bounding boxes are output as a  $4 \times N$  matrix, where  $N$  is the number of objects found. The number of pixels are output as an  $N$ -vector and the centroids are output as a  $2 \times N$  matrix. Each bounding box is a 4-tuple of the form: minimum row, minimum column, maximum row, maximum column. Each centroid is a 2-tuple of the form: row, column.

The *Image Find Objects* block is optimized for finding large numbers of objects. It returns the objects according to the selected sort order. If the *Find largest objects* option is checked then it will find the  $N$  largest objects in the image, where  $N$  is the number of objects requested, and what constitutes the "largest" being determined by the sort criteria.

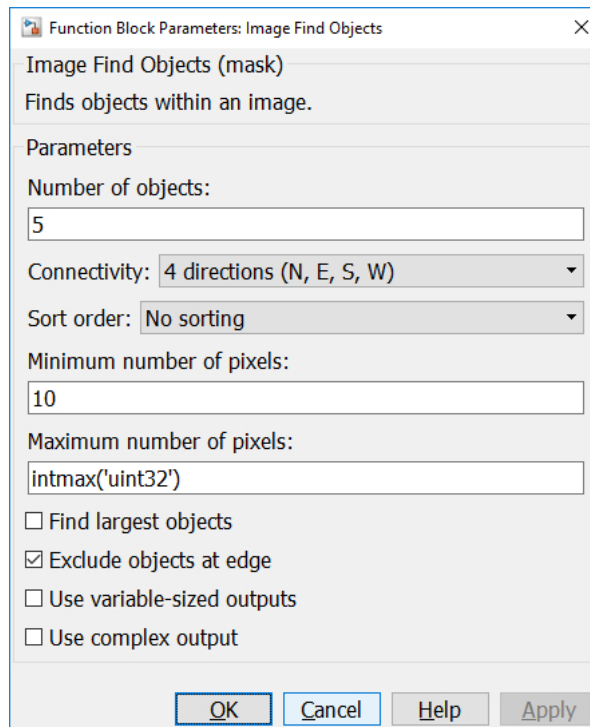


Figure 1.15: Image Find Objects dialog box



### 1.5.4 Use of image compare block to detect ball object

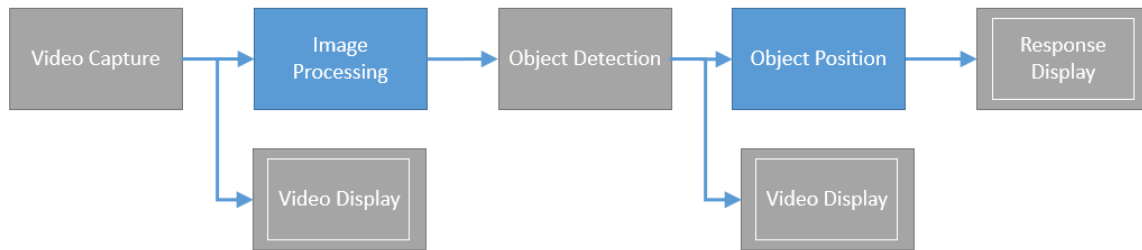


Figure 1.16: Ball detection and position measure from the camera

1. Open the file `q_camera_detect_object_tune.slx` in Simulink.
2. Run and look at the HSV (Hue, Saturation, Value) ranges in the raw video for the ball as shown un Figure 1.17.
3. Place your cursor at different position to estimate the ranges HSV of the ball.
4. Enter those values on the *Image Compare* block window in the Image Processing (HUE) subsystem.
5. Adjust the values if necessary, so the filtering image is not noisy and detect the ball properly.
6. We also use a Detect Object subsystem with an *Image Find Objects* block and a De-bouncer. It gives us the number of object detected and the center of the object. Does the result make sense ? Move the ball and see what happen.
7. Close the file.

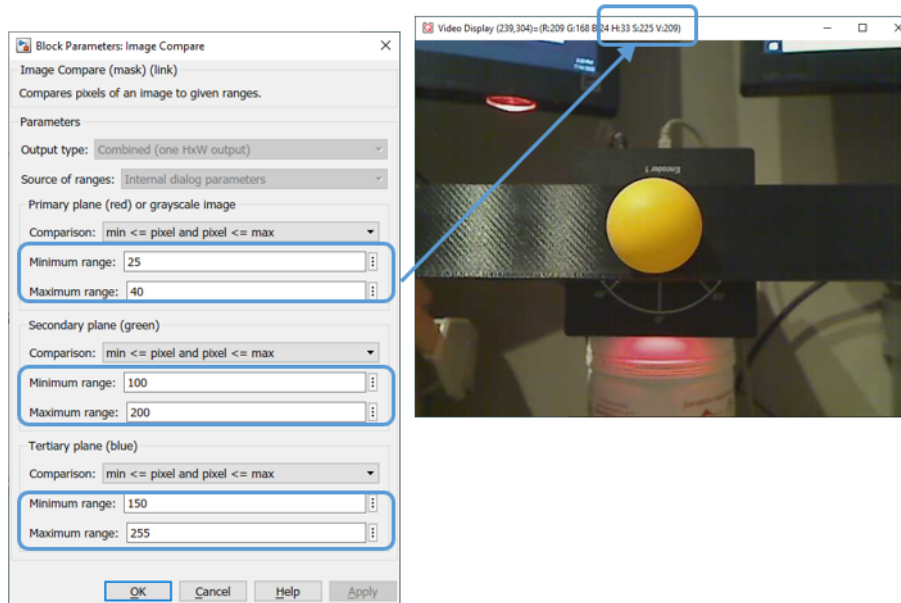


Figure 1.17: Ball detection with Image Compare

### 1.5.5 Convert and filter ball measurement

We can detect if there is a ball and where it is located in the pixel image. In this part, we will convert the value into meters using camera resolution and beam length. Then, we will filter the image and adjust its frequency cut-off frequency according to the camera rate.

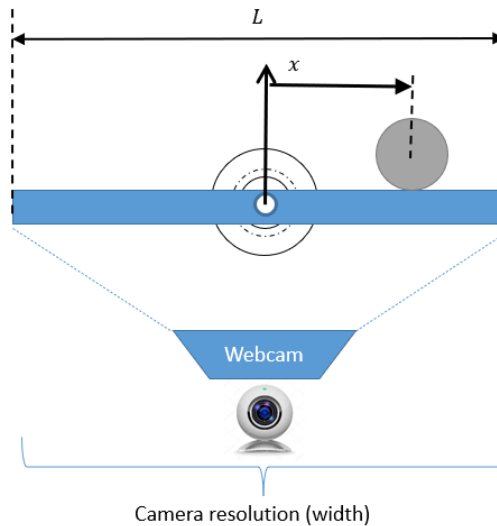


Figure 1.18: Convert object detection into position

The conversion of the pixels in  $col$  into position measurement is given by the formula:

$$x = L \left( \frac{col}{cam_{res}} - 0,5 \right) \quad (1.4)$$

where  $L$  is the length of the beam (0.3m=30cm here) and the camera resolution  $cam_{res}$  is 640.  $col$  is determined by the image processing step explained above.

1. Open the file `q_camera_measure_ball_tune.slx` in Simulink.

2. If necessary, place something behind the Qube to hide some electric wires. This is not mandatory but can help sometimes.
3. Make sure your Image Compare values from the previous part are set in.
4. Go in the Measure Ball Position subsystem and enter the right value for the length of the beam so it matches the formula given in (1.4).
5. Click on Monitor & Tune in the hardware panel to run the test in Simulink.
6. Verify that the correct position is measured.
7. Move the ball and make sure the signal is not too noisy.
8. Based on the same principle, you can try to add the measurement in the  $y$  axe (row).

## 1.6 Ball & Beam cascade control design

### 1.6.1 Control performance requirements

The performance requirements for the **ball position control** are described in Table 1.2.

Requirement	Assessment criteria	Level
Control the ball position	Position setpoint tracking	Steady-state error as short as possible
	Motor input voltage	limited to [-10V ; +10 V]
	Settling time at 4 %	As short as possible
	Overshoot	less than or equal to 2.5%
	Disturbance rejection	Rejection of load effects

Table 1.2: Performance requirements for beam position control

### 1.6.2 Transfer function model

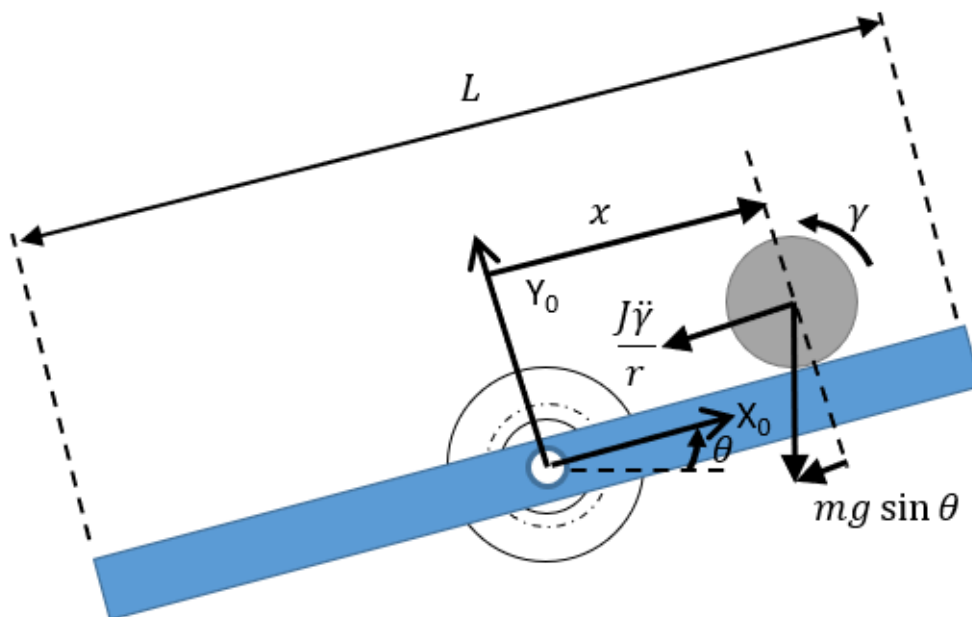


Figure 1.19: Ball & beam schematic

From Figure 1.19, applying Newton's law of motion, we have

$$m\ddot{x} + \frac{J\ddot{\gamma}}{r} - mg \sin \theta = 0 \quad (1.5)$$

$$m\ddot{x} + \frac{J\ddot{x}}{r^2} = mg \sin \theta \quad (1.6)$$

$$m\ddot{x} + \frac{2m}{3}\ddot{x} = mg \sin \theta \quad (1.7)$$

$$\ddot{x} = \frac{3}{5}g \sin \theta \quad (1.8)$$

$J$  is the moment of inertia which for a spherical shell is given by

$$J = \frac{2}{3}mr^2$$

while for a solid sphere, we have

$$J = \frac{2}{5}mr^2$$

Then, linearizing the model about  $\theta = 0$  yields

$$\ddot{x} \approx \frac{3}{5}g\theta$$

In the Laplace domain, assuming zero initial conditions, we have

$$s^2X(s) = \frac{3}{5}g\Theta(s) \quad (1.9)$$

$$\Rightarrow X(s) = \frac{K_{bb}}{s^2}\Theta(s) \quad (1.10)$$

where  $K_{bb} = \frac{3}{5}g$ .

The linearized model of the ball & beam takes the form of a double integrator transfer function.

### 1.6.3 Cascade control design

Cascade control is a multiloop control structure which is used when there are more than one measurements, but only one control variable is available.

This is the case here since we have two measurements: the beam angle and ball position but only one control variable available which is the motor voltage.

The inner and the outer control loops are formed, each with an individual feedback controller as shown in Figure 1.20.

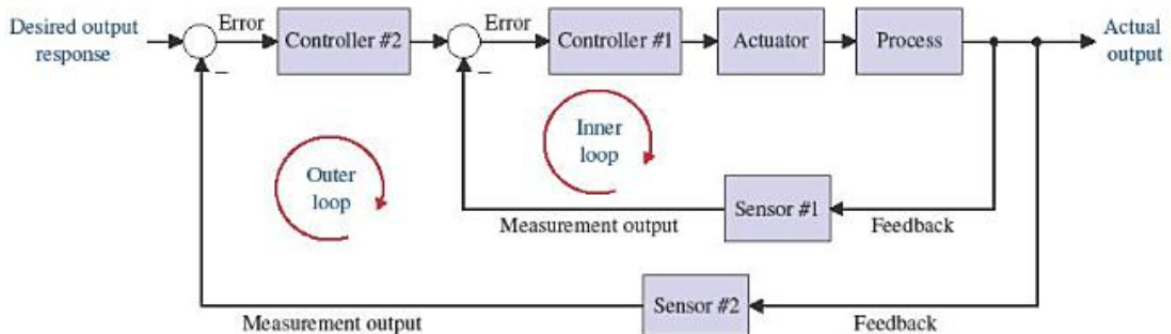


Figure 1.20: Multiloop control structure of cascade control

The full Ball & Beam cascade control is shown in Figure 1.21. Note that it is assumed that the inner loop for the beam position control is very fast in comparison with the outer loop and so it is assumed that  $\Theta(s) \approx \Theta_r(s)$ .

With this ideal assumption, the ideal ball & beam control is (again) a simple PD controller for the ball position control over the beam.

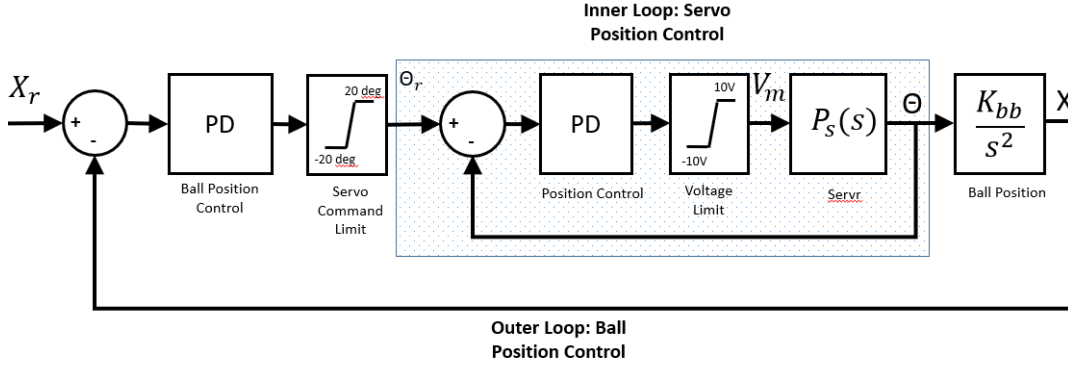


Figure 1.21: Ball & beam full cascade control

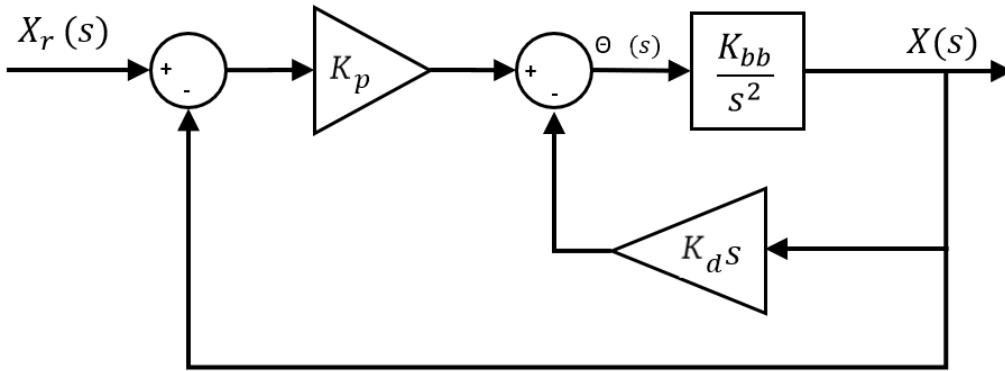


Figure 1.22: Ideal ball & beam control

$$X(s) = \frac{K_{bb}}{s^2} \Theta_r(s)$$

From Figure 1.22, we have the relationship for the inner loop:

$$F_i(s) = \frac{X(s)}{\Theta_r(s)} = \frac{C(s)G(s)}{1 + C(s)G(s)H(s)}$$

with

$$C(s) = 1, G(s) = \frac{K_{bb}}{s^2}, H(s) = K_d s$$

$$F_i(s) = \frac{K_{bb}}{s(1 + K_{bb}K_d)}$$

Finally the full closed-loop transfer function:

$$\Rightarrow \frac{X(s)}{X_r(s)} = \frac{K_p K_{bb}}{s^2 + K_d K_{bb} s + K_p K_{bb}} \quad (1.11)$$

We then can determine  $K_p$  and  $K_d$  by identifying the closed-loop transfer function to a prototype second-order transfer function.

$$\frac{X(s)}{X_r(s)} = \frac{\omega_n^2}{s^2 + 2z\omega_n s + \omega_n^2} \quad (1.12)$$

which leads to

$$K_p = \frac{\omega_n^2}{K_{bb}} \quad \text{and} \quad K_d = \frac{2z\omega_n}{K_{bb}} \quad (1.13)$$

### 1.6.4 Control performance in simulation

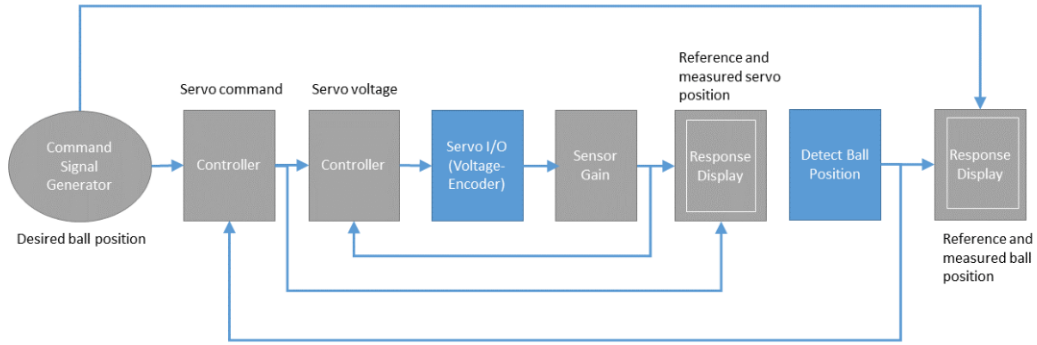


Figure 1.23: Ball & Beam closed-loop scheme

1. Open the file `cd_ball_and_beam.m` and enter the parameters from the control performance requirements.
2. Run the file.
3. Open the file `s_ball_beam_control_tune.slx` in Simulink.
4. Simulate the response of the system.

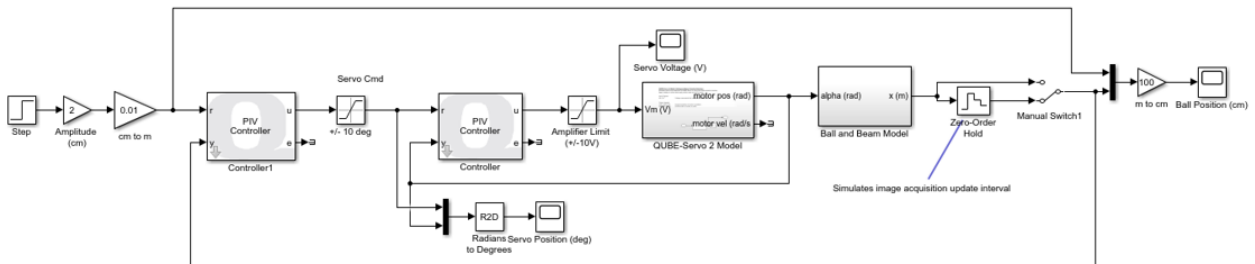


Figure 1.24: Ball & beam control simulation

## 1.7 Ball & Beam cascade control implementation

### 1.7.1 Validation of the control on the real system

1. Open the file `q_ball_beam_control_tune.slx` in Simulink.
2. Click on *Detect Ball Position* block, then on *Image Processing* block, then on *Image Compare* and enter your HSE values to detect the orange ball.
3. Click on **Monitor & Tune** in the Hardware panel to run the test.

4. Observe the ball response to a -3/3cm square wave reference.
5. Observe the ball response to a -3/3cm triangular wave reference.

### 1.7.2 Control tuning

Remember that the open-loop transfer function model of the ball position is a double integrator. It has a double pole at the origin and is therefore an unstable system. As such the tuning of the PD control can be very sensitive to the choice of the numerical values of the PD controller.

The major issue is to find a best trade-off between stability and the steady-state error. Here are some considerations that you can experiment to improve the control performance.

#### If the ball is difficult to stabilize

1. Verify that the ball position measurement is correct.
2. Ensure the ball position measurement is not too noisy.
3. Increase the settling time ( $t_s$ ), allow for response to take longer to settle, to generate lower control gains. This will make the control more robust but tracking performance will decrease.

#### If there is a large steady-state error

1. Verify that the beam is initially at 0 angle. Verify also that the beam ends at the edges of the camera view.
2. Lower settling time ( $t_s$ ).
3. Add an integral gain: starting from 0.02 and increasing it by 0.01. Setting the integral gain too high makes the response more oscillatory.

#### Filtering effects and considerations

The default filter cut-off frequencies set in the PD controller are 7.5Hz for the ball position and 35Hz for the beam position control.

Beam filter servo cut-off frequency:

1. Lowering it will remove noise, but going too low will degrade the tracking performance of the ball.
2. Increasing it can improve performance, but setting it too high will cause a visible high-frequency noise.

Ball filter cut-off frequency:

1. Lowering it will remove noise, but may lead to higher overshoot.
2. Increasing it can improve the ball tracking, but it will increase noise and makes the system less stable (usually noticeable at 40 Hz).



## 1.8 Troubleshooting

- If the camera does not display the image when you run the file `q_camera_test_tune.slx`, click on Image Acquisition block, then Video capture. Finally click on the 3 dots ... on Image identifier, and select the logi 270 camera.
- If you have issues for compiling and an error saying "*Error occured while executing External MEX-file*". Then, disconnect and reconnect the camera into the USB port of the computer and then restart.
- How to fix the following QUARC license issue that may appear:
  - ▷ If the following error message appears when using Simulink: "**Error occurred while executing External Mode MEX-file 'quarc\_comm': One of the arguments is invalid.**"

You will need to re-activate the QUARC License on your computer. Follow the steps below:

- Log out.
- Log in with the following login: `.\admin_TPauto`
- Ask the teacher to enter the password.
- Go to the `C:\temp\` directory.
- Double-click on the licence file: `Quarc essentials Polytech Nancy`.
- Tick both options and save the file.
- Unplug and re-plug the QUBE-Servo 2 to reset the micro-controller.
- Log out the admin account.
- Log in with your student account.
- Start Matlab and Simulink again.