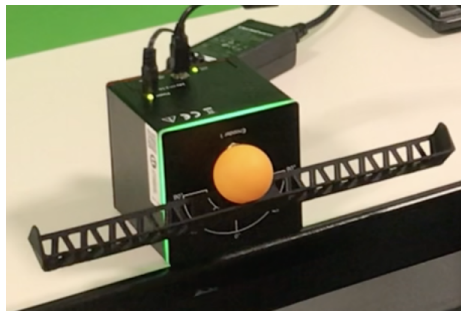




Vision-based cascade control of a Ball & Beam system



Hugues Garnier

Florian Fritz

(5A IA2R SIA student in 2022-2023)

September 2023

Contents

1	Image-based cascade control of a Ball & Beam system	2
1.1	Objectives and layout of the Lab	2
1.2	Download of the files required for the lab	2
1.3	Hardware required & other requirements	3
1.3.1	The QUBE servo-motor	3
1.3.2	Logitech C270 HD camera	3
1.3.3	Ball & Beam	4
1.4	Reminder and sampling and anti-aliasing filter	4
1.5	Data-driven modelling & PD control of the beam angular position	5
1.5.1	Control performance requirements	5
1.5.2	Recording of the step response experiment	6
1.5.3	Data-driven model learning and validation	7
1.5.4	Design of a PD control	8
1.6	Image-based ball position measurement	10
1.6.1	Image acquisition	10
1.6.2	Image processing	11
1.6.3	Ball position measurement	13
1.6.4	Use of image compare block to detect ball object	15
1.6.5	Convert and filter ball measurement	16
1.7	Ball & Beam cascade control design	17
1.7.1	Control performance requirements	17
1.7.2	Transfer function model	17
1.7.3	Control design	18
1.7.4	Control performance in simulation	19
1.8	Ball & Beam cascade control implementation	20
1.8.1	Validation of the control on the real system	20
1.8.2	Control tuning	20
1.9	Troubleshooting	21
1.10	Summary and reflections	21

Acknowledgements

The contents of this lab has been inspired and adapted from an initial version provided by Quanser Inc¹. This is fully acknowledged.

¹www.quanser.com

Lab 1

Image-based cascade control of a Ball & Beam system

1.1 Objectives and layout of the Lab

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


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 the necessary information in it. Image processing becomes then prominent.

In this lab, we will try to understand how to extract 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 from Quanser : www.youtube.com/watch?v=pm2LDu-1b30.

1.2 Download of the files required for the lab

1. Download the zipped file *Lab_Ball_Beam.zip* from the course website. Save and unzip it in the local disk 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_Ball_Beam` folder** that contains the files needed for this lab.

1.3 Hardware required & other requirements

The required hardware for this lab includes:

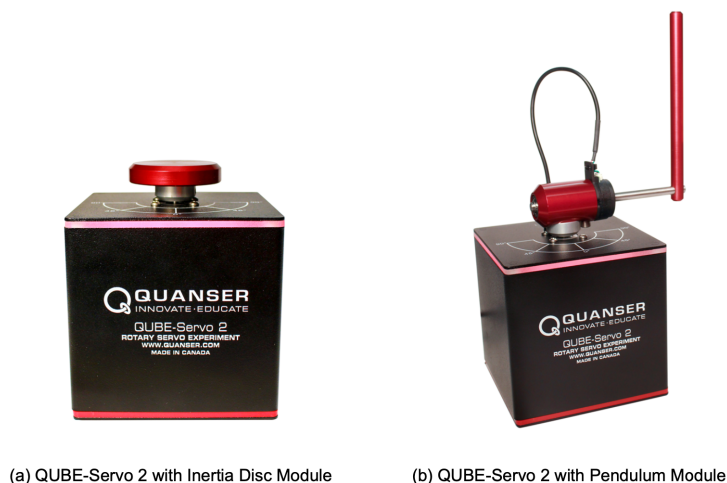
- a QUBE-Servo 2;
- a USB Webcam Logitech C270 HD;
- a 3D printed beam & ball;

1.3.1 The QUBE servo-motor

The Quanser QUBE-Servo 2, pictured in Figure 1.1, 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 1.1: QUBE-Servo 2 with the 2 different modules

1.3.2 Logitech C270 HD camera

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](#).



Figure 1.2: Logitech C270 HD

1.3.3 Ball & Beam

A 3D printed beam is mounted on the QUBE-Servo 2 instead of the inertial disc or the rotary pendulum module. It allows the ball to roll from a side to the other. 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.

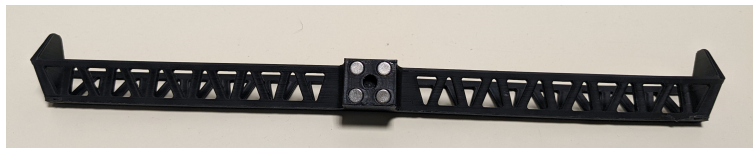


Figure 1.3: 3D printed Beam



Figure 1.4: Balls

1.4 Reminder and sampling and anti-aliasing filter

The camera framerate is 30 ips. The period is: $1/30 = 0.033s$

The camera update interval is then $img_rate = Ts * \text{ceil}(0.033 / Ts) = 0.034s$

Based on the Shannon sampling theorem, to prevent aliasing, the filter cut-off frequency should be chosen a bit smaller than half the sampling frequency.

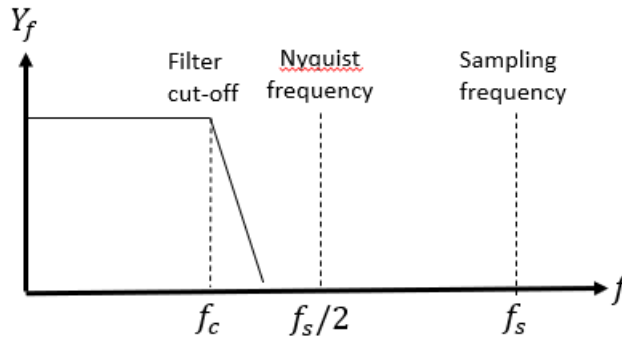


Figure 1.5: Filter design

1.5 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 disc as the beam can rotate at 360 degrees. Make sure the beam cannot hit the table when rotating by placing the QUBE-Servo 2 at the edge of your desk.

1.5.1 Control performance requirements

The performance requirements for the **angular servo 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%
	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 disc 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 disc.

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:

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

1.5.2 Recording of the step response experiment

1. Open the file *Step_resp_Qube.mdl* 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.

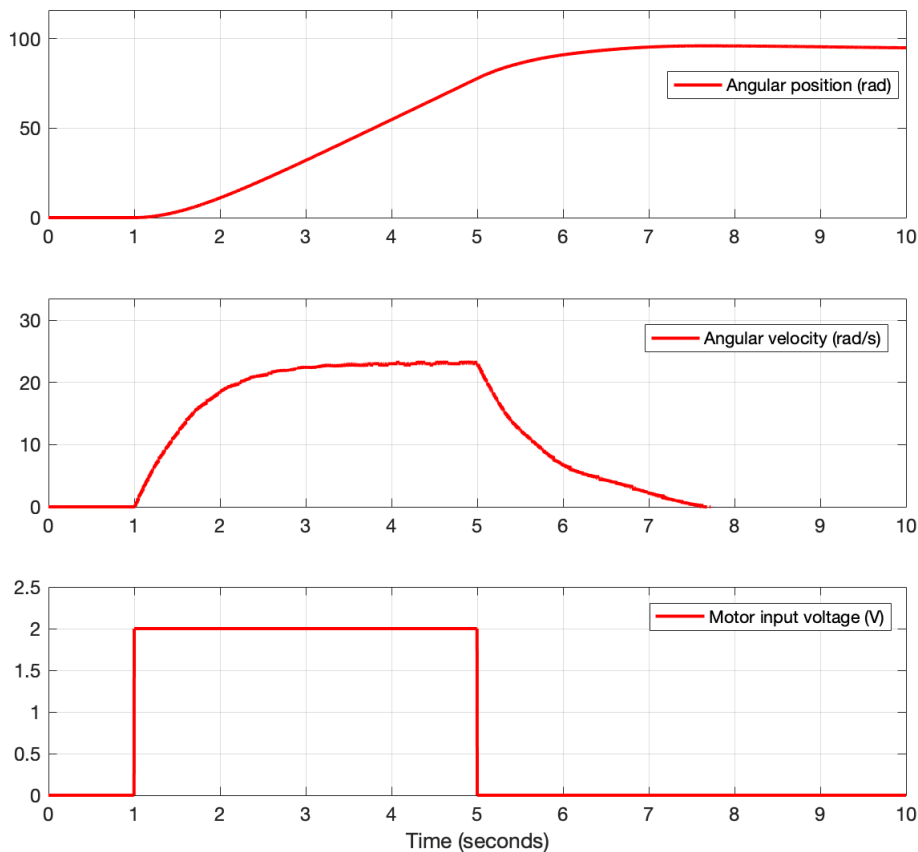


Figure 1.6: Beam angular position and velocity responses to a positive and negative step input sent to the motor voltage

1.5.3 Data-driven model learning and validation

It is possible to use very basic methods to determine the parameters of a first-order Laplace transfer function model from the positive step response for example. As we saw last year, there now exist more advanced model learning/identification methods available in Matlab toolboxes like the CONTSID toolbox developed by the research team of CRAN hosted at Polytech Nancy which determine the parameters of a (continuous-time) Laplace transfer function model directly from measured input/output data. We will make use of the CONTSID algorithms here.

To use these powerful data-driven high-fidelity model learning algorithms, follow the steps :

- Download the CONTSID toolbox from its website¹ to the local folder of your PC.
- Add the CONTSID folder to the Matlab path. If this is not possible, add all the files of CONTSID folder in your working directory.
- Run the file `ident_via_contsid.m` in Matlab (if you use your own laptop, the Matlab System Identification and Control toolboxes must be installed for the CONTSID to work).
- Compare the measured and model angular velocity responses.

¹www.cran.univ-lorraine.fr/contsid

- Note the numerical values of the estimated model steady-state gain K and time-constant T .

1.5.4 Design of a PD control

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

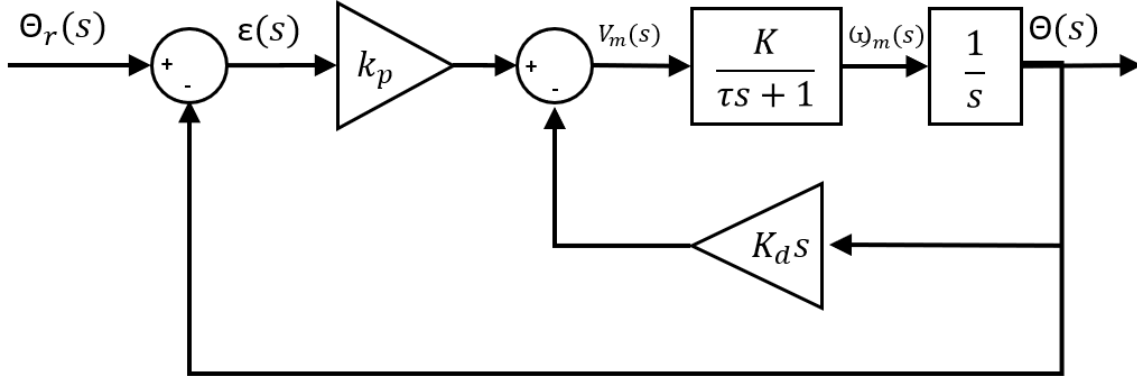


Figure 1.7: PV control

The main advantage of having the derivative action on the output is to avoid the spike effect of a step change on the setpoint.

In the intern loop from the Figure 1.7, we have the relationship:

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

with

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

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

From Figure 1.7, the control input then becomes :

$$U(s) = k_p(R(s) - Y(s)) - k_d s Y(s)$$

or

$$V_m(s) = k_p(\Theta_r(s) - \Theta(s)) - k_d s \Theta(s)$$

Finally, we can calculate the closed-loop transfer function and determine the values of k_p and k_d by identification with the canonical second-order transfer function.

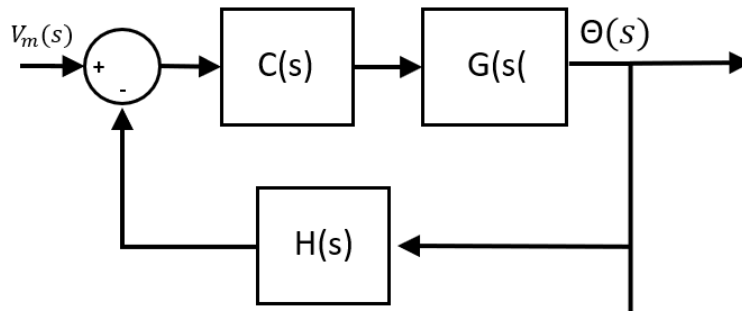


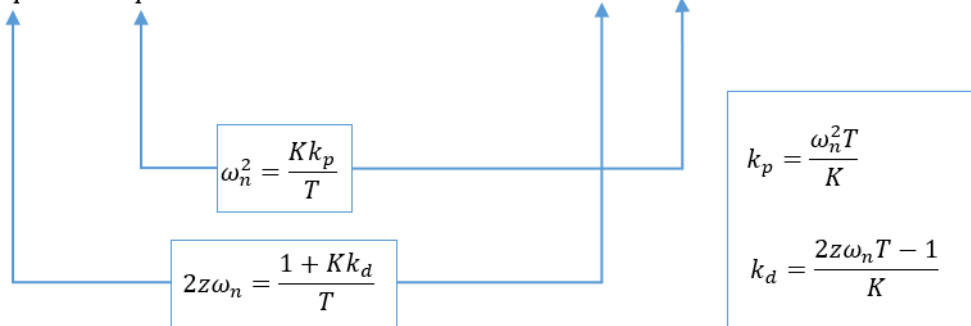
Figure 1.8: Intern loop

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 previously obtained in 1.5.3.
3. Enter the specification values of the first overshoot in % from the control performance requirements and the peak-time.
4. Run the .m file to calculate the gains.
5. Open the file `q_qube2_pd_control_tune.mdl` in Simulink.
6. Click on **Run** to simulate and observe the response.
7. Are the requirements satisfied ?
8. 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.

1.6 Image-based ball position measurement

In the first part, we designed a PD controller for the angular position of the beam. Usually, in classical control theory the information of the position of the ball is directly provided by a sensor.

The specificity of this lab is to design an image-based control. We will use image processing to extract the position information from the camera images.

The different tasks can be designed as follows:

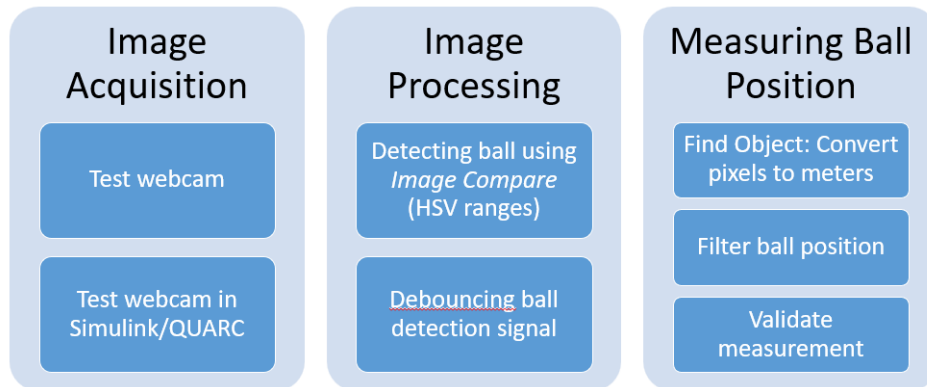


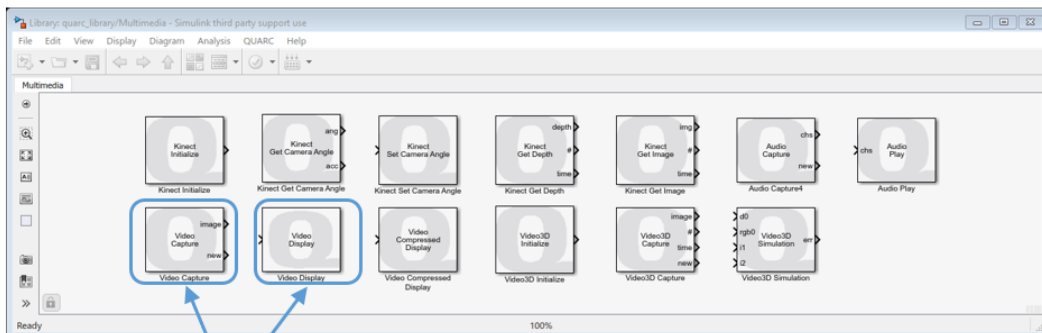
Figure 1.9: Ball position measurements tasks

1.6.1 Image acquisition



Figure 1.10: Image acquisition

The QUARC library provides video acquisition blocks as shown in Figure 1.11.



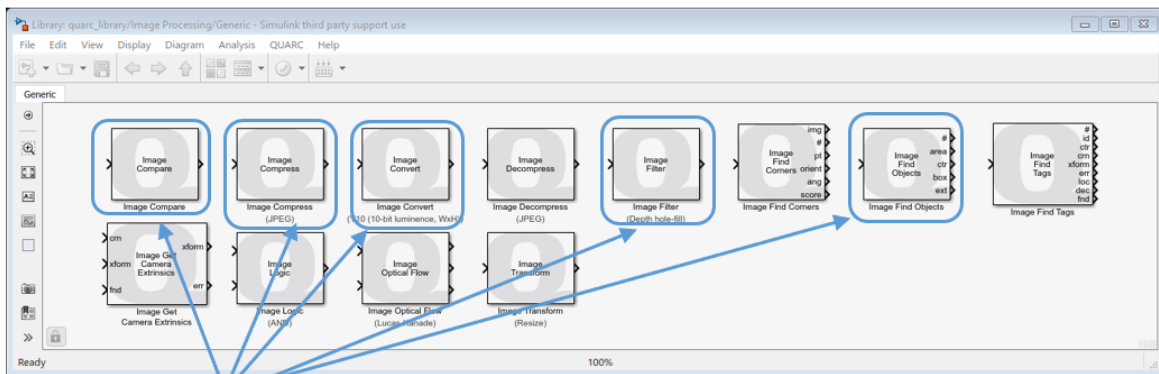
- **Video Capture** block to acquire video from webcam.
- **Video Display** block to view video feed.

Figure 1.11: Image and video acquisition QUARC blocks

1. Connect the camera to the USB port of the computer.
2. Open the file `q_camera_test_tune.mdl` in Simulink.
3. Open the *Image Acquisition* subsystem by double click on it.
4. Select the device and the resolution of 640*480 pixels in the *Video Capture* Window.
5. Click on **Monitor & Tune** in the Hardware panel to run the test.
6. Observe the output video by double click on the *Video Display* block.
7. By now you should see the video capture of the camera. Adjust the position of the camera to make sure the beam length fits within the width of the camera view;

1.6.2 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 1.12: 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 1.13: 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.

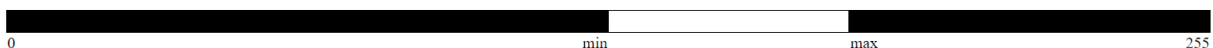


Image Filter The *Image Filter* Block filters an image or a proportion of an image in order to reduce noise. To filter only a portion of the image, a rectangular region of interest is defined. The image may be filtered within the region of interest or outside the region of interest. The location of the 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.

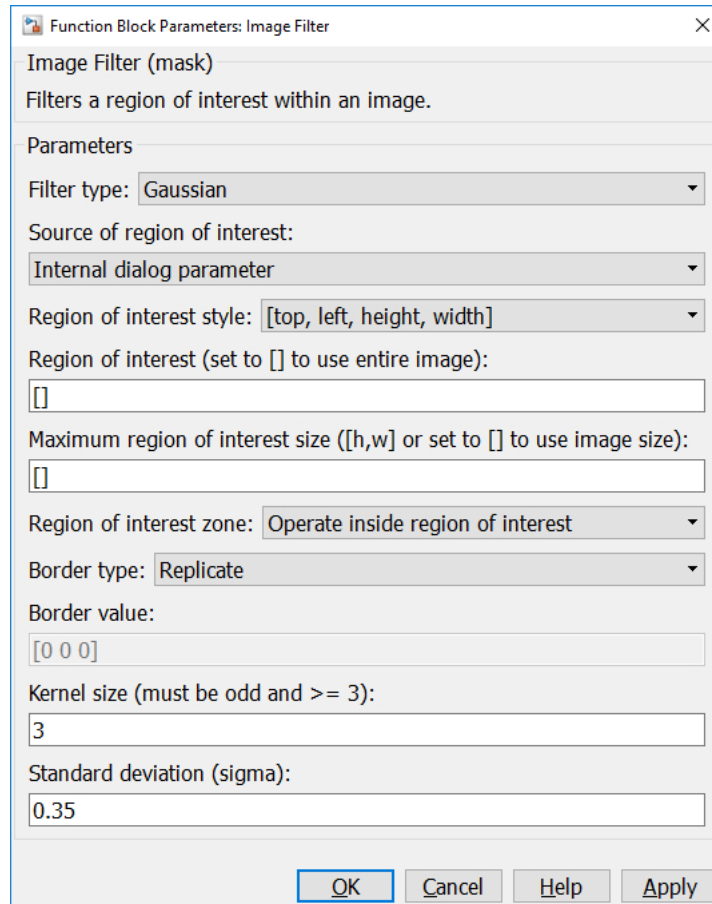


Figure 1.14: Image Filter parameters

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 neighbourhood or a *Low-pass* filter.

1.6.3 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.

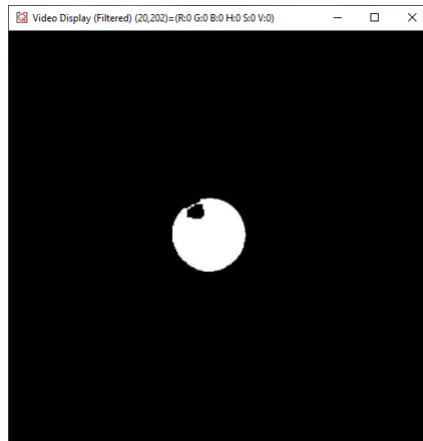


Figure 1.15: Filtered image after image processing

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.

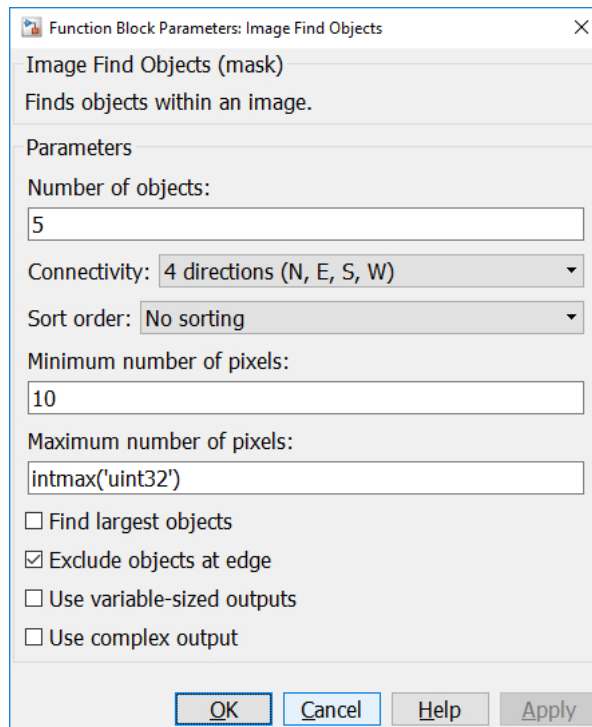


Figure 1.16: Image Find Objects dialog box

1.6.4 Use of image compare block to detect ball object

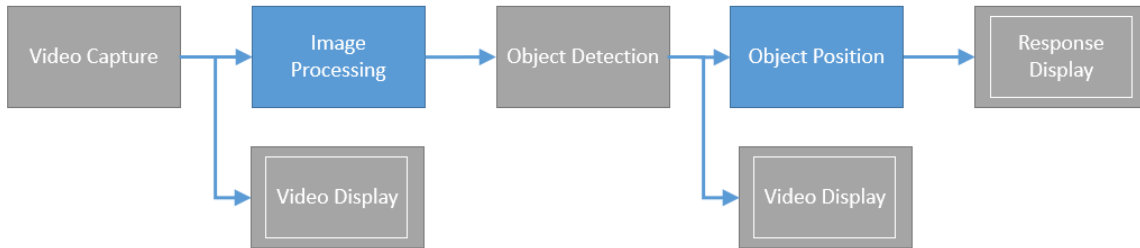


Figure 1.17: Ball detection and position measure from the camera

1. Open the file `q_camera_detect_object_tune.mdl` in Simulink.
2. Run and look at the HSV (Hue, Saturation, Value) ranges in the raw video for the ball.
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.

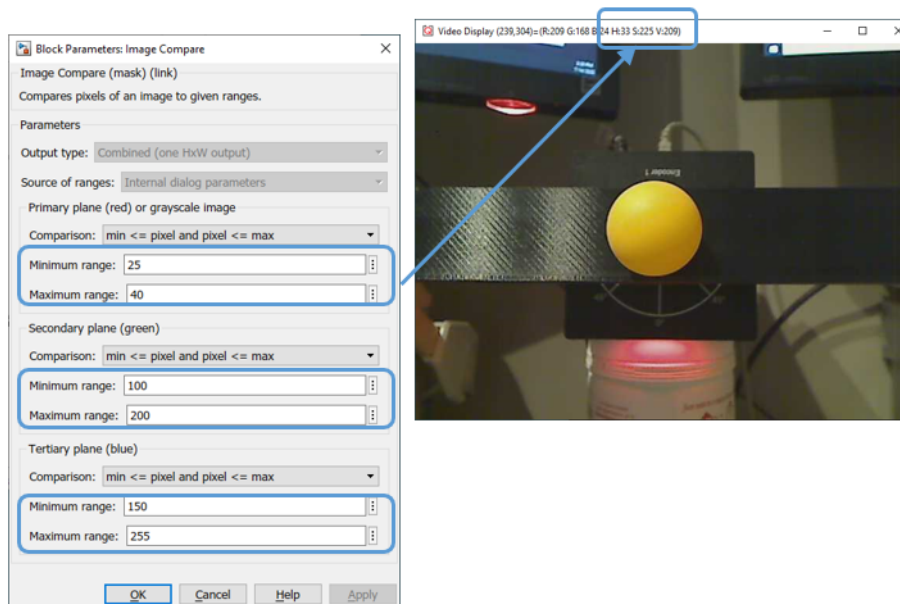


Figure 1.18: Ball detection with Image Compare

1.6.5 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.

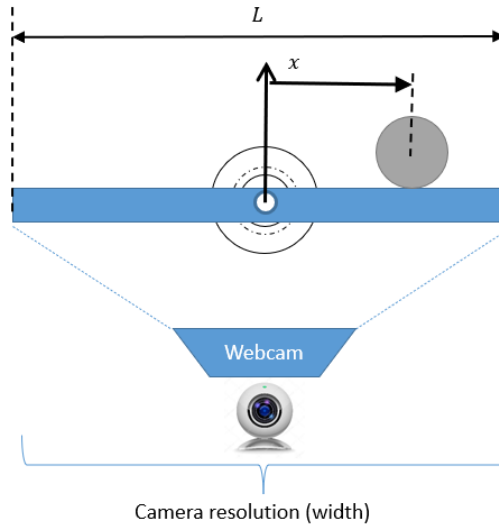


Figure 1.19: Convert object detection into position

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

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

1. Open the file `q_camera_measure_ball_tune.mdl` 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 so it match the formula of your Ball & Beam system.
5. Open the file `cd_filter_cutoff.m`.
6. Adjust the filter cut-off frequency according to camera rate (30 ips) and the Shannon sampling theorem. Refer to the reminder part.
7. Click on Monitor & Tune in the hardware panel to run the test in Simulink.
8. Verify that the correct position is measured.
9. Move the ball and make sure the signal is not noisy.
10. Based on the same principle, you can try to add the measurement in the y axe (row).

1.7 Ball & Beam cascade control design

1.7.1 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

1.7.2 Transfer function model

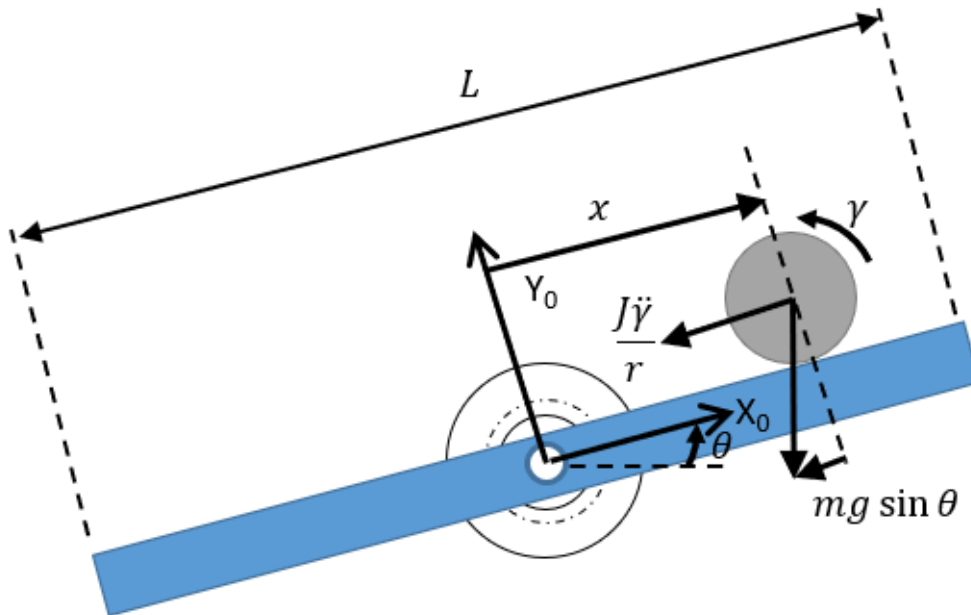


Figure 1.20: Ball & Beam Modeling

From Figure 1.20, applying Newton's law of motion, we have

$$m\ddot{x} + \frac{J\ddot{\gamma}}{r} - mg \sin \theta = 0 \quad (1.4)$$

$$m\ddot{x} + \frac{J\ddot{x}}{r^2} = mg \sin \theta \quad (1.5)$$

$$m\ddot{x} + \frac{2m}{3}\ddot{x} = mg \sin \theta \quad (1.6)$$

$$\ddot{x} = \frac{3}{5}g \sin \theta \quad (1.7)$$

J is the moment of inertia which for a thing 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

$$s^2X(s) = \frac{3}{5}g\Theta(s) \tag{1.8}$$

$$\Rightarrow X(s) = \frac{K_{bb}}{s^2}\Theta(s) \tag{1.9}$$

1.7.3 Control design

The ideal ball & beam control is a simple PD control of the ball position.

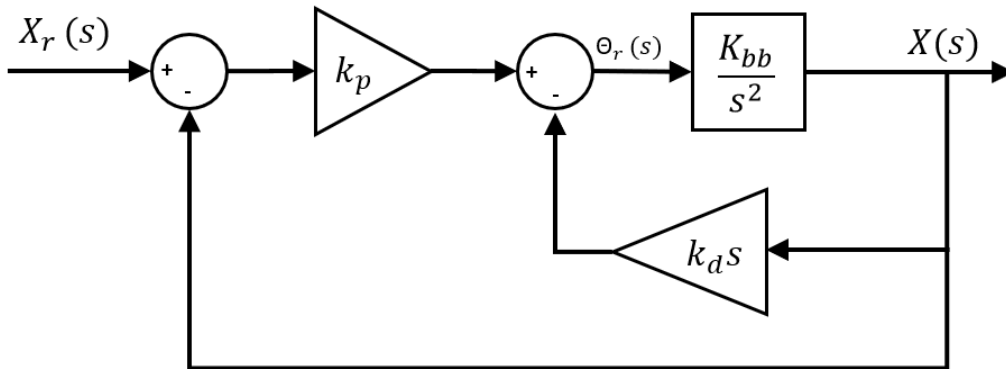


Figure 1.21: Ideal ball & beam control

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

In the intern loop from the Figure 1.20, we have the relationship :

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

with

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

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

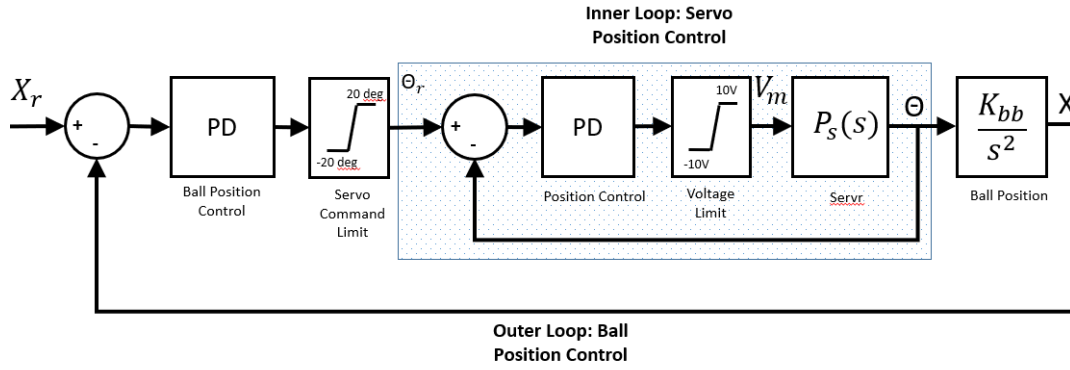


Figure 1.22: Ball & beam full cascade control

We use cascade control as it is used when there is more than one measurement and only one control variable available. The full Ball & Beam cascade control is represented in Figure 1.22. Finally the full closed-loop transfer function:

$$\begin{aligned} \Theta_r(s) &= k_p(X_r(s) - X(s)) - k_d s X(s) \\ \Rightarrow \frac{X(s)}{X_r(s)} &= \frac{k_p K_{bb}}{s^2 + k_d K_{bb} s + k_p K_{bb}} \end{aligned} \quad (1.10)$$

We then find k_p and k_d .

$$\frac{X(s)}{X_r(s)} = \frac{\omega_n^2}{s^2 + 2z\omega_n s + \omega_n^2} \quad (1.11)$$

$$k_p = \frac{\omega_n^2}{K_{bb}} \quad \text{and} \quad k_d = \frac{2z\omega_n}{K_{bb}} \quad (1.12)$$

1.7.4 Control performance in simulation

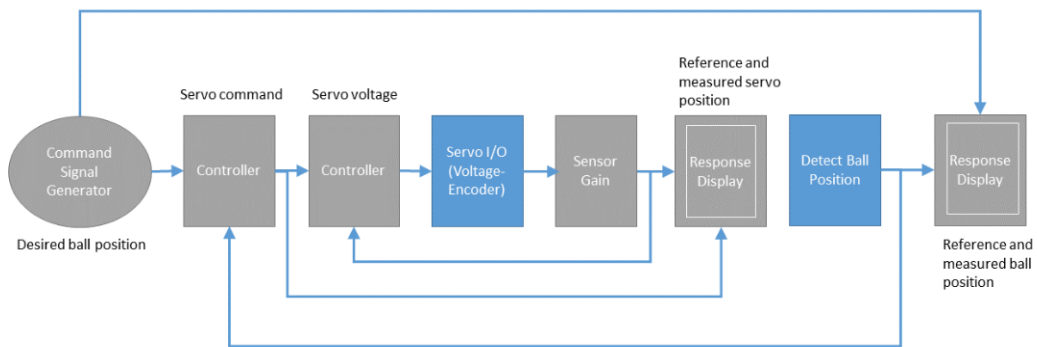


Figure 1.23: Ball & Beam closed-loop scheme

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.mdl` in Simulink.
4. Simulate the response of the system.

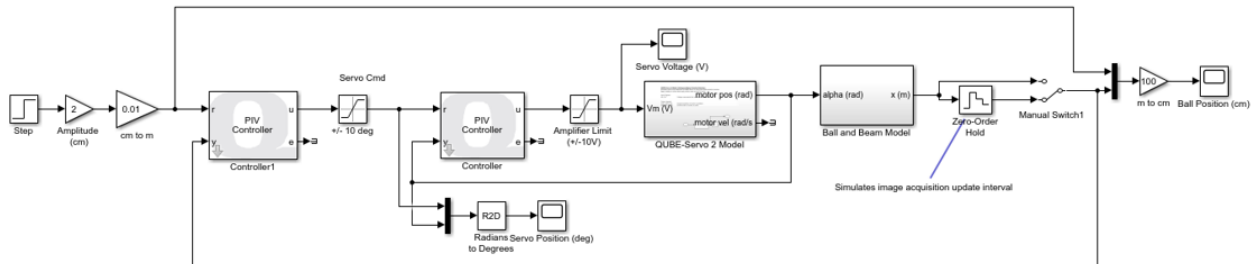


Figure 1.24: Ball & beam control simulation

1.8 Ball & Beam cascade control implementation

1.8.1 Validation of the control on the real system

1. Open the file `q_ball_beam_control_tune.mdl` in Simulink.
2. Click on **Monitor & Tune** in the Hardware panel to run the test.
3. Observe the response from a $-2/2\text{cm}$ reference.
4. Try to stabilize the ball at the position 0cm .

1.8.2 Control tuning

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

If the ball is difficult to stabilize

1. Verify that the ball position measurement is correct.
2. Ensure the ball position measurement is not 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 the beam is initially at 0 angle and beam ends at edges of camera view.
2. Lower settling time (t_s).
3. Add an integral gain : from 0.02 and increasing it by 0.01 . Setting it to high make the response 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 an audible 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 ball tracking, but increase noise and make system less stable (usually noticeable at 40 Hz).

1.9 Troubleshooting

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.

1.10 Summary and reflections

Summarize and reflect on what you have seen and learned in this lab.