Chapter 5

Control systems

Three different types of controllers are considered in this work, namely PID, LQR, and MPC. PID control does not provide active swing-damping control of the payload, but it is used in the training data stage discussed in Chapter 4. LQR is a popular and well-known optimal control technique. In previous work by [4] and [5], an LQR implementation with a parameter estimator was designed for active swing-damping control of a multirotor with an unknown suspended payload. This LQR architecture was shown to be an effective swing-damping controller and will be applied in this work as the baseline controller.

A data-driven system identification method was introduced in Chapter 4 to estimate a linear model of unknown dynamics. The models generated by this method will be used in an MPC for active swing-damping control of the multirotor and payload system. The proposed MPC architecture will ~~therefore~~ be compared to the baseline LQR controller. These swing-damping control architectures are summarised in Table 5.1, where each controller is paired with a system identification method.

Table 5.1: System identification techniques paired with the corresponding controllers.

System identification		**Controller**
Model type	Algorithm	
White-box	RLS mass estimator, and FFT cable length estimator	LQR
Black-box	DMDc, or HAVOK	MPC

In this chapter, a MATLAB/Simulink simulation environment will be introduced to test the proposed control architectures. An overview of the different controllers will be given and the design process of each controller will be explained. The controllers will then be tested in simulations with different system parameters and disturbances. Finally, the simulation results will be shown and discussed.

5.1. Simulation Environment

The controllers in this chapter are tested with a MATLAB/Simulink simulation environment. This is used rather than the SITL and Gazebo simulation environment because it allows us to iterate designs faster. Simulink provides a graphical interface for control system design which enables us to change the control system design with ease. In contrast, the SITL and Gazebo simulation environment requires text-based control laws with C++ and requires ROS nodes to interface an MPC with PX4. This requires a lot more development time for control system design than the graphical tools in Simulink. The SITL and Gazebo simulation also has a longer runtime per simulation, which further adds to the development time of the iterative control system design process.

The quadrotor and suspended payload system is modelled in Simulink with the differential equations derived in Chapter 3. The resulting controllers are also applied in Simulink. Using the cascade PID control architecture (discussed in the sections below), this simulation environment was verified against practical data with and without a payload. The plots in Figure 3.1, Figure 3.2, and Figure 3.3 from Chapter 3 show how well the simulations match the actual system dynamics. The controllers will therefore be designed, tested and compared using this simulation environment.

5.2. Cascaded PID control

PID control is a popular linear control technique that applies a control signal proportional to the error signal, the integral of the error signal, and the derivative of the error signal. This is the control technique used in the default PX4-Autopilot multirotor controllers [18]. PX4-Autopilot was chosen as the multirotor flight-stack because it is open-source and widely used in industry and research. The PID implementation of PX4 does not provide active swing-damping of the payload, however, it is used in the system identification flight stage discussed in Chapter 4.

The default PX4 control architecture consists of multiple cascaded PID controller loops. This is divided into two main sections, the inner-loop attitude controllers and the outer-loop translational controllers. Figure 5.1 shows a high-level overview of the PX4 control architecture without showing state feedback.

The setpoint vectors in Figure 5.1 are denoted by $\boldsymbol{X}_{sp}$ for position, $\boldsymbol{V}_{sp}$ for velocity, $\boldsymbol{A}_{sp}$ for acceleration, $\boldsymbol{q}_{sp}$ for the attitude quaternion, $\boldsymbol{\Omega}_{sp}$ for angular rate, and $\boldsymbol{T}_{sp}$ for motor thrust. The virtual actuator commands are denoted by $\delta_{A_{sp}}$, $\delta_{E_{sp}}$, $\delta_{R_{sp}}$, and $\delta_{T_{sp}}$, for the virtual aileron, elevator, rudder, and thrust commands.

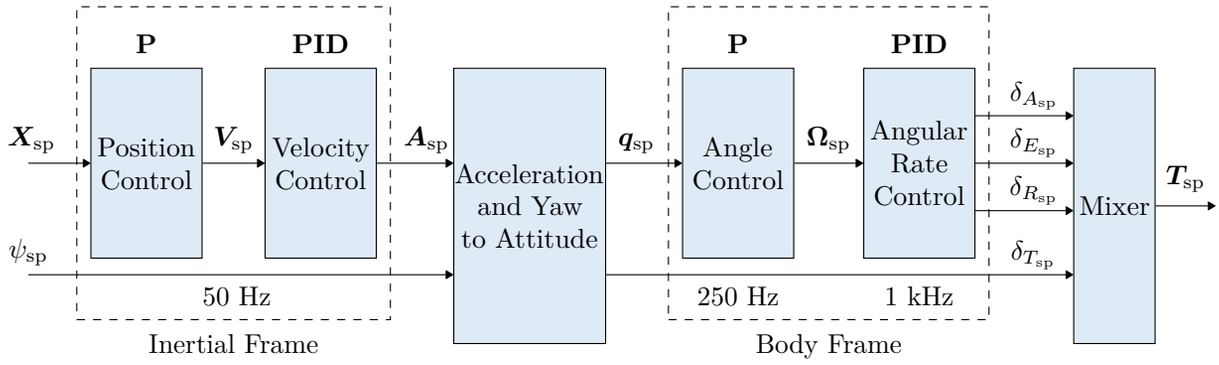


Figure 5.1: Cascaded PID control architecture of PX4 [2]

The inner-most control element is the mixer, which converts virtual actuator commands to actuator thrust commands. The attitude controller includes the angle and angular rate controllers, which send commands to the mixer. The translational controller consists of the position and velocity controllers, which send commands to the attitude controller. The rate of each controller is also shown in Figure 5.1. Note that the outer-loop controllers deal with slow dynamics and therefore run at slower rates than the inner-loop controllers.

5.2.1. Angular rate controller

Three separate linear PID controllers are used to control angular rates in the pitch, roll, and yaw ~~axis~~ axes of the multirotor body frame. The angular rate controller receives angular rate estimates from the PX4 state estimator and outputs virtual actuator commands to the mixer. Figure 5.2 shows a diagram of the PX4 angular rate controller.

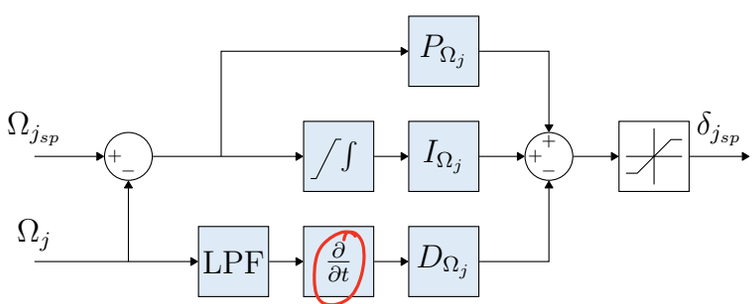


Figure 5.2: Angular rate controller diagram [2]

THIS WRONG, NOT PARTIAL DERIVATIVE dy/dt. CHANGE TO

In Figure 5.2, the j subscript denotes a specific element in the vectors $\boldsymbol{\Omega}_{sp}$ and $\boldsymbol{\Omega}$, and WHAT IS THIS? DO NOT THINK IT IS DEFINED the corresponding virtual actuator, where $j = \{1, 2, 3\}$. Some elements in the controller improve the practical ~~control~~ implementation, but have a negligible effect on the design on the gains, namely:

- A Low Pass Filter (LPF) is added in the derivative path to reduce the effect of sensor noise.
- The derivative path is implemented on the plant output signal, instead of the error signal to eliminate ~~the~~ derivative kick.

- Saturation is applied to the integral path to eliminate integral wind-up.
- The control signal is saturated to avoid dangerously large setpoint commands.

These effects are also described in detail by [4].

Classical control theory was used by [27] to design the controller gains of the practical multirotor, Honeybee. ~~This~~ The same process is ~~also~~ explained in detail by [4]. The controller gains were designed for a transient response that is as fast as possible while retaining fast disturbance rejection, and minimal overshoot. It was determined by [27] that the default PX4 angular rate controller gains of the ZMR250 airframe provide excellent control for Honeybee and satisfy these design requirements.

DO THE READER KNOW WHAT HONEYBEE IS AT THIS STAGE?

For a 1 rad/s step response, the pitch rate controllers result in a 3.6 % overshoot, 0.024 s rise time, 0.8 s ~~for a~~ 2 % settling time, and 138 rad/s bandwidth [27]. These gains are well suited for Honeybee and will also be implemented in this work. The default roll-rate and yaw-rate controller gains are also implemented for the same reason.

5.2.2. Angle controller

The pitch, roll, and yaw angles are controlled by the angle controller in the body frame. A quaternion based controller is implemented by PX4 for angle control based on work by [28]. The structure and design of this controller is well explained by [27], [4], and [5] and is only briefly discussed here.

For the control law proposed by [28], the error quaternion is calculated as:

$$\boldsymbol{q}_e = \boldsymbol{q}^{-1} \cdot \boldsymbol{q}_{sp}, \tag{5.1}$$

where $\boldsymbol{q}_e$ is the quaternion error, $\boldsymbol{q}$ is the current attitude quaternion of the multirotor, and $\boldsymbol{q}_{sp}$ is the attitude quaternion setpoint. The resulting control law is given as:

IS THE INNER TERMS OF THE QUATERNION DEFINED IN CHAPTER 3?

$$\boldsymbol{\Omega}_{sp} = 2\boldsymbol{P}_q \operatorname{sgn}(q_{0_e})\boldsymbol{q}_{v_e}, \qquad \operatorname{sgn}(q_{0_e}) = \begin{cases} 1, & q_{0_e} \geq 0 \\ -1, & q_{0_e} < 0 \end{cases}, \tag{5.2}$$

where $\boldsymbol{\Omega}_{sp}$ is a vector of the roll, pitch, and yaw angular rate setpoints, $\boldsymbol{P}_q$ is a vector of the corresponding proportional gains, q_{0_e} is the error of the quaternion magnitude component, and $\boldsymbol{q}_{v_e}$ is the error of the vector component of the quaternion. The implementation of this control law is illustrated in Figure 5.3 for a single element of $\boldsymbol{\Omega}_{sp}$.

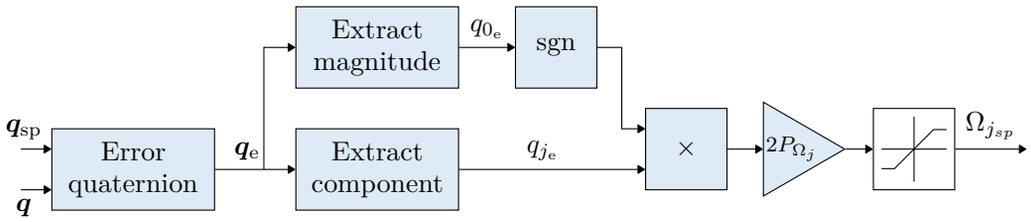


Figure 5.3: Quaternion based angle controller diagram [2]

In Figure 5.3, the j subscript denotes a specific element in the vectors $\boldsymbol{q}_e$, $\boldsymbol{P}_q$, and $\boldsymbol{\Omega}_{sp}$, where $j = \{1, 2, 3\}$. The attitude quaternion setpoint is taken as input, the current attitude is received from the PX4 estimator, and the resulting angular rate setpoint is produced as the control signal.

IT IS NOT RELEVANT TO THE ANGLE CONTROLLER WHETHER THE VELOCITY CONTROLLER HAS AN INTEGRATOR.

An integrator is not required in this controller, because it is placed between the velocity and angular rate controllers, which both include integrators to reject steady-state disturbances [4]. Therefore only a proportional term is applied in this control law. The proportional gain was designed for Honeybee by [27] for a faster transient response than with the default ZMR250 gain. This results in a 1 rad step response with a 0 % overshoot, 0.3 s rise time, 0.47 s ~~for a~~ 2 % settling time, and 11 rad/s bandwidth [27]. This has a large time-scale separation from the angular rate controller, which has a bandwidth of 138 rad/s. A fast angle response is also desired in this work, hence the same gains designed by [27] will be used.

RATHER SAY "LITTLE TO NO OVERSHOOT"

5.2.3. Translational controller

The translational controller consists of the position and velocity controllers. Position control is not required for training data in this work, therefore it will not be discussed in this section. A diagram of the North velocity controller is shown in Figure 5.4. The West and Down velocity controllers duplicate the same configuration.

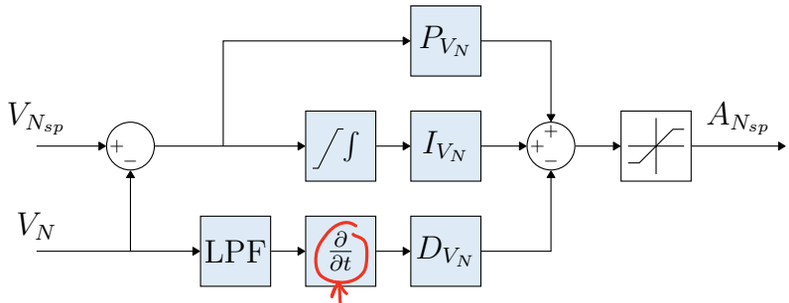


Figure 5.4: Velocity controller diagram [2]

SEE PREVIOUS NOTE

In order to drive the multirotor to a given velocity setpoint, the velocity controller commands an acceleration setpoint in the inertial frame. This acceleration setpoint is trans-

YOU SOMETIMES USE "QUADROTOR", TRY AND BE CONSISTENT

formed to an attitude setpoint which is used by the angle controller. This transformation is based on work done by [29].

Recall from Chapter 4, that the cascaded PID controller is used for velocity step inputs in the training data flight stage for system identification. The system identification methods produce linear models, which are used in swing damping controllers to minimise the payload angles during flight. The velocity controller gains used for Honeybee by [27] result in aggressive velocity responses, which produces large payload swing angles. Such large payload angles are undesirable for safe flights. Therefore the velocity controller gains are redesigned for a slower transient response and smaller payload swing angles.

The derivation of the transfer function, $G_{V_N}(s)$, of the North velocity dynamics of a multirotor with a suspended payload was derived by [5] and is described here briefly. Firstly, the non-linear dynamics were derived with Lagrangian mechanics as described in Chapter 4. The equations were then linearised around hover with the small-angle approximation. The linearised equations for the longitudinal or North velocity dynamics can be presented in the state-space form as,

$$\dot{\mathbf{X}}_{long} = \mathbf{A}_{long}\mathbf{X}_{long} + \mathbf{B}_{long}\mathrm{U}_{long} \text{ and} \tag{5.3}$$

$$\mathrm{Y}_{long} = \mathbf{C}_{long}\mathbf{X}_{long}, \tag{5.4}$$

where

$$\mathbf{X}_{long} = \begin{bmatrix} V_N & \dot{\theta} & \theta \end{bmatrix}^T, \tag{5.5}$$

$$U_{long} = A_N, \tag{5.6}$$

$$\mathbf{A}_{long} = \begin{bmatrix} 0 & \frac{c_\theta}{l m_Q} & \frac{g m_p}{m_Q} \\ 0 & -\frac{c_\theta (m_Q + m_p)}{l^2 m_Q m_p} & -\frac{g(m_Q+m_p)}{l m_Q} \\ 0 & 1 & 0 \end{bmatrix}^T, \tag{5.7}$$

$$\mathbf{B}_{long} = \begin{bmatrix} 1 & -\frac{1}{l} \end{bmatrix}^T, \tag{5.8}$$

$$\mathbf{C}_{long} = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}. \tag{5.9}$$

For the definition of these symbols, refer to Chapter 4. The input and output of the plant are $\mathrm{U}_{long} = A_N$ and $\mathrm{Y}_{long} = V_N$, respectively. Note that the actual input of the plant is $A_{N_{sp}}$, however, it is assumed that $A_{N_{sp}} \approx A_N$ due to the large time-scale separation between the velocity controller and attitude controller. The transfer function can therefore

be calculated as,

$$G_{V_N}(s) = \frac{V_N(s)}{A_N(s)} = \mathbf{C}_{long}\left(s\mathbf{I} - \mathbf{A}_{long}\right)^{-1}\mathbf{B}_{long} \tag{5.10}$$

$$G_{V_N}(s) = \frac{s^2 + \frac{c_\theta}{m_p l^2}s + \frac{g}{l}}{s\left[s^2 + \frac{c_\theta(m_Q + m_p)}{m_Q m_p l^2}s + \frac{g(m_Q + m_p)}{m_Q l}\right]} \tag{5.11}$$

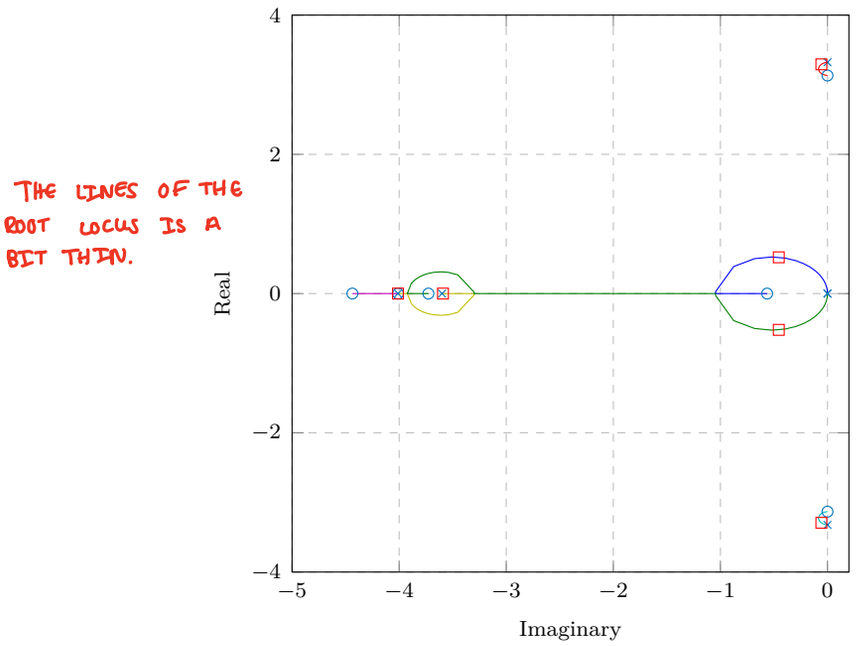
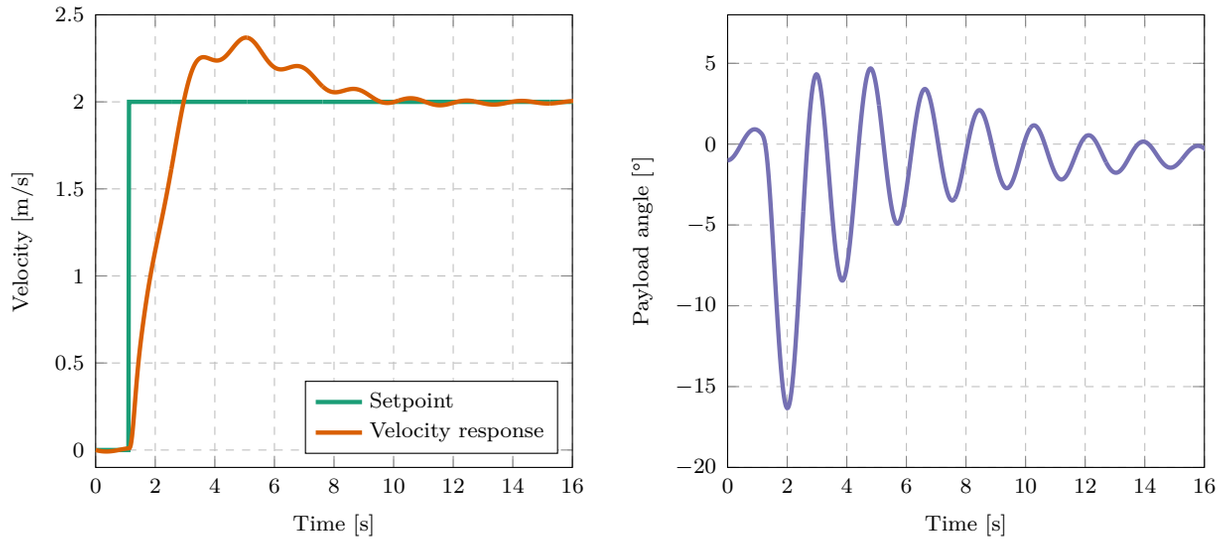


Figure 5.5: Root locus plot of the North velocity dynamics including PID controller

A payload with $m_p = 0.2\,\text{kg}$ and $l = 1\,\text{m}$ is considered for the controller design. Figure 5.5 shows a root locus plot of the closed loop system which includes the PID controller. The gains were tuned in an iterative process to produce a slow step response that stimulates the payload dynamics enough for system identification. These gains were also tested iteratively with different payload parameters in the ranges, $0.1 \leq m_p \leq 0.3$ kg and $0.5 \leq l \leq 2$ m, to establish safe flights with different payloads.

Figure 5.6 shows a velocity step response for the resulting cascaded PID controller with a specific payload. Note that the response results in a large overshoot, however, this aids in keeping the payload angles low. The initial swing angle is quite large, however, this attenuates to lower angles quickly enough for stable flight. The velocity oscillations are clearly visible in Figure 5.6a. The current PID controller does not provide an adequate way of actively damping the payload oscillations, hence an active swing damping controller is required.



(a) North velocity **(b)** Payload angle

Figure 5.6: PID velocity step response ($l = 1\,\text{m}$, $m_p = 0.2\,\text{kg}$)

5.3. LQR

LQR is a popular optimal control technique which has been shown to be an effective swing-damping controller for multirotors with a suspended payload [4] [5] [30]. The Linear Quadratic Gaussian (LQG) technique combines this optimal control technique with a full-state estimator when all state variables cannot be measured [31]. An LQG was effectively used by [5] for swing-damping control of a multirotor and suspended payload system. However, to focus on the controller performance without considering state estimation, it is assumed that full-state feedback is available in this work. Therefore, an LQR for North velocity control with full-state feedback is presented in this section. This controller structure can be duplicated for East velocity control.

An LQR does not inherently apply integral action and does not achieve zero steady-state tracking error in the presence of disturbances. Therefore the state vector is augmented with an integral state of the velocity error, $\mathcal{V}_N$, such that:

$$\dot{\mathcal{V}}_N = V_{N_{sp}} - V_N \tag{5.12}$$

The North velocity dynamics were derived and linearised in Chapter 3 to produce a state-space model which will be used to design the LQR. It is assumed that all model parameters are known before a flight, except the payload parameters, l and m_p, which are estimated in the system identification phase. The augmented state space model which includes the integral state is given by:

$$\dot{\boldsymbol{x}}_A = \boldsymbol{A}_A \boldsymbol{x}_A + \boldsymbol{B}_A \boldsymbol{u}_A, \tag{5.13}$$

where

$$\boldsymbol{x}_A = \begin{bmatrix} \mathcal{V}_N & V_N & \theta & \dot{\theta} \end{bmatrix}^T, \tag{5.14}$$

$$\boldsymbol{u}_A = \begin{bmatrix} A_{N_{sp}} \end{bmatrix}, \tag{5.15}$$

$$\boldsymbol{A}_A = \begin{bmatrix} 0 & -1 & 0 & 0 \\ 0 & \frac{-c_{air}}{m_Q} & \frac{m_p \cdot g}{m_Q} & \frac{c_\theta}{(l \cdot m_Q)} \\ 0 & 0 & 0 & 1 \\ 0 & \frac{c_{air}}{(m_Q \cdot l)} & \frac{(m_p + m_Q) \cdot g}{(m_Q \cdot l)} & \frac{-c_\theta \cdot (m_p + m_Q)}{(l^2 \cdot m_Q \cdot m_p)} \end{bmatrix}, \text{ and} \tag{5.16}$$

$$\boldsymbol{B}_A = \begin{bmatrix} 0 \\ \frac{1}{m_Q} \\ 0 \\ \frac{-1}{(m_Q \cdot l)} \end{bmatrix}. \tag{5.17}$$

The LQR control law is defined as:

$$\boldsymbol{u}_A = \boldsymbol{K}_{lqr}(\boldsymbol{x}_{A_{sp}} - \boldsymbol{x}_A), \tag{5.18}$$

where $\boldsymbol{K}_{lqr}$ is the LQR gain, and $\boldsymbol{x}_{A_{sp}}$ is the augmented state vector setpoint. Only $V_{N_{sp}}$ has a non-zero value, hence $\boldsymbol{x}_{A_{sp}} = \begin{bmatrix} 0 & V_{N_{sp}} & 0 & 0 \end{bmatrix}^T$.

Furthermore, the LQR considers the cost function:

$$J(\boldsymbol{u}_A) = \int_0^\infty \left(\boldsymbol{X}_A^T \boldsymbol{Q} \boldsymbol{X}_A + \boldsymbol{u}_A^T \boldsymbol{R} \boldsymbol{u}_A \right) dt, \tag{5.19}$$

where $\boldsymbol{Q}$ is the state weighting matrix, and $\boldsymbol{R}$ is the input weighting matrix. The LQR gain, $\boldsymbol{K}_{lqr}$, is therefore determined by substituting Equation 5.18 into Equation 5.19 and minimising the cost function.

The LQR control performance can be manually tuned by changing the state and input weights in $\boldsymbol{Q}$ and $\boldsymbol{R}$ respectively. The weighting of each state variable can be determined according to tracking importance. For example, if the weighting of V_N is small in comparison to the weighting of θ, the controller will produce a slow velocity response with small payload swing angles. For the values in $\boldsymbol{R}$, if the weighting of an input variable is large, the controller will output lower control values for that variable.

The general design requirements of the LQR are to produce a fast velocity response with zero steady-state error while damping the payload oscillations quickly. The priority is to produce a smooth trajectory with minimal oscillation, however a reasonably fast response time is still required. An iterative tuning approach was used to determine the $\boldsymbol{Q}$ and $\boldsymbol{R}$

matrix entries that produce a good performance. The final LQR weightings were selected as:

$$\boldsymbol{Q} = diag([0.1 \quad 10 \quad 0 \quad 100]), \quad \boldsymbol{R} = 13. \tag{5.20}$$

Note that the weight of the payload angle variable is 0. This is because the payload angle will have a non-zero steady-state value at constant velocity, V_N, due to aerodynamic drag. The payload angle is therefore damped by placing a heavy weighting on the derivative of the payload angle instead.

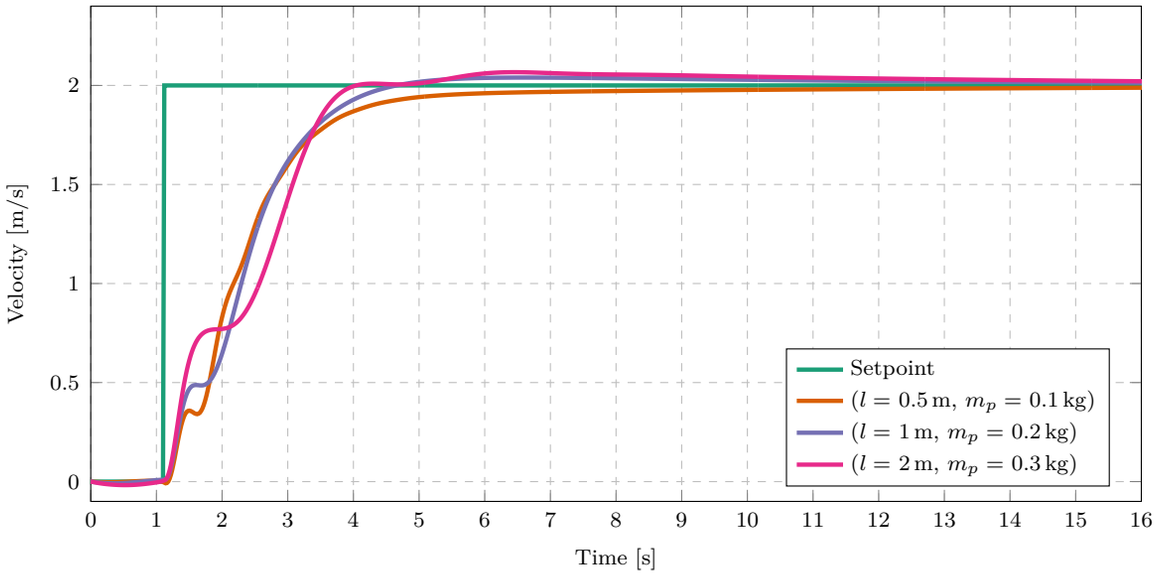


Figure 5.7: LQR velocity step responses with different payloads

Figure 5.7 shows a plot of the simulated LQR velocity response with different payload parameters. It is clear that the controller damps the payload angles well and produces a smooth trajectory with different payloads. For each different payload, the payload parameters are estimated, the state-space model is populated, and the LQR gain is calculated using the same weighting matrix values. The controller performance, including disturbance rejection, will be further discussed in Section 5.5.

5.4. MPC

Model Predictive Control (MPC) refers to a control system approach that determines the control action at each time-step by solving an open-loop optimal control problem over a finite prediction horizon [32]. MPC does not refer to a specific algorithm implementation, but rather to the general control system approach.

Figure 5.8 shows the structure of a typical MPC implementation. As a high-level overview, the MPC receives the measured output vector, $\boldsymbol{y}$, and the output setpoint $\boldsymbol{y}_{sp}$, and

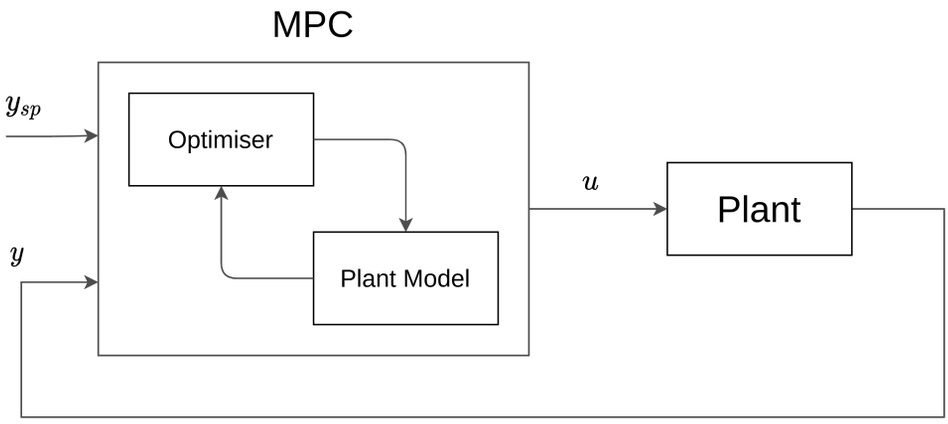


Figure 5.8: Diagram of the structure of a typical MPC

determines the control action, $\boldsymbol{u}$, to drive the values in $\boldsymbol{y}$ to the values $\boldsymbol{y}_{sp}$. An optimiser uses an internal plant model to determine an optimal control sequence over a prediction horizon [32]. Therefore $\boldsymbol{y}_{sp}$ may be replaced by a target trajectory for the prediction horizon. Only the first control action of the optimal sequence is executed and the optimisation is re-calculated at every time-step.

THIS IS NOT ALWAYS TRUE, IT IS A DESIGN DECISION TO ONLY USE THE FIRST CONTROL INPUT BEFORE REPLANNING.

Each specific MPC implementation is dependant on the plant model representation [33]. In this section, an overview will be given of the specific MPC implementation used in this work. The MPC objective function will be explained and the design process to tune the controller will be discussed. It will also be discussed how integral action is achieved with the MPC. Finally, the control response of the tuned MPC will be shown and discussed for the ~~a~~ simulated system.

5.4.1. Receding horizon

As stated before, an MPC considers an open-loop optimal control problem over a finite prediction horizon [32]. Using a plant model for predictions, an optimiser determines the optimal control sequence that will minimise an objective function over the prediction horizon. MPC is also referred to as receding horizon control because the control sequence is determined for the next prediction horizon period at every time-step.

Figure 5.9 illustrates the receding horizon concept for a Single Input Single Output (SISO) system. One of the main objectives of the optimiser is to minimise the error between the predicted trajectory and the target trajectory. Starting at time-step k, a prediction horizon of N_p time-steps is considered, and the optimiser suggests a controller decision of N_c unique control values. N_c is referred to as the control horizon and is subject to the condition, $N_c \leq N_p$. The control sequence, or controller decision, at time-step k is denoted

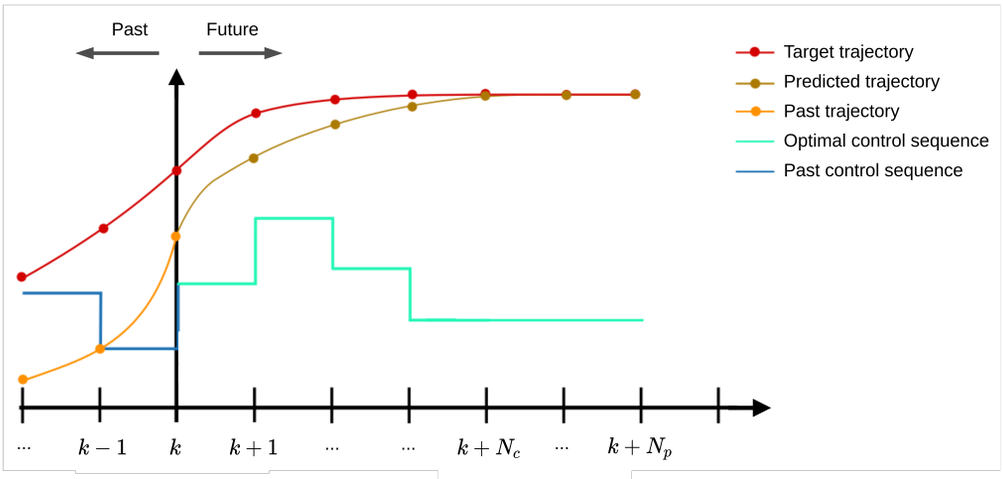


Figure 5.9: Illustration of the receding horizon of an MPC (adapted from [3])

by,

$$\boldsymbol{z}_k{}^T = [\boldsymbol{u}(k|k)^T \quad \boldsymbol{u}(k+1|k)^T \quad \ldots \quad \boldsymbol{u}(k+p-1|k)^T], \tag{5.21}$$

which minimizes a specific objective function over the prediction horizon. The controller decision produces the predicted trajectory when applied to the plant model. Note that only the first control action, $\boldsymbol{u}(k|k)$, will be executed and a new controller decision will be determined at time-step $k+1$. Also note that if $N_c < N_p$, the remaining control actions are set to,

HAVE ALREADY INDICATED THAT THIS IS A DESIGN DECISION.

$$\boldsymbol{u}(i|k) = \boldsymbol{u}(N_c|k), \quad i > N_c. \tag{5.22}$$

5.4.2. Plant model

An important characteristic of MPC is that it uses a separately identifiable plant model in the control optimisation process [33]. An estimated model from the data-driven techniques discussed in Chapter 4 will be used as the plant model for the proposed MPC architecture. DMDc produces a discrete, linear state-space model of the system dynamics. Hence, an MPC implementation that corresponds to such a model will be applied. The following state-space model representation will be used,

$$\boldsymbol{x}_{mpc}(k+1) = \boldsymbol{A}_{mpc}\boldsymbol{x}_{mpc}(k) + \boldsymbol{B}_{mpc}\boldsymbol{u}_{mpc}(k), \tag{5.23}$$

where $\boldsymbol{A}_{mpc}$ is the system matrix and $\boldsymbol{B}_{mpc}$ is the input matrix applied by the MPC. $\boldsymbol{x}_{mpc}(k)$ is the state vector and $\boldsymbol{u}_{mpc}(k)$ is the input vector at time-step k for this state space model. It is assumed that full-state feedback is available, therefore $\boldsymbol{y}_{mpc} = \boldsymbol{x}_{mpc}$.

DMDc applies multiple delay-coordinates to account for input delay and state delay in the system. In Section 4.4, it was shown that the adapted DMDc algorithm produces three matrices, $\boldsymbol{A}_{dmd}$, $\boldsymbol{A}_d$, and $\boldsymbol{B}_{dmd}$. However, the MPC requires a single system matrix, $\boldsymbol{A}_{mpc}$.

Therefore the DMDc system is converted into:

$$\begin{bmatrix} \boldsymbol{x}_{dmd}(k+1) \\ \boldsymbol{d}(k+1) \end{bmatrix} = \begin{bmatrix} \boldsymbol{A}_{dmd} & \boldsymbol{0} \\ \boldsymbol{I}_d & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x}_{dmd}(k) \\ \boldsymbol{d}(k) \end{bmatrix} + \begin{bmatrix} \boldsymbol{B}_{dmd} \\ \boldsymbol{0} \end{bmatrix} \boldsymbol{u}_{dmd}(k), \tag{5.24}$$

$$\boldsymbol{x}_{mpc}(k+1) = \boldsymbol{A}_{mpc} \cdot \boldsymbol{x}_{mpc}(k) + \boldsymbol{B}_{mpc} \cdot \boldsymbol{u}_{mpc}(k), \tag{5.25}$$

where $\boldsymbol{I}_d$ is the identity matrix that links the corresponding entries in $[\boldsymbol{x}_{dmd}(k) \quad \boldsymbol{d}(k)]^T$ to $\boldsymbol{d}(k+1)$. This produces large state-space matrices with many output variables (represented in $\boldsymbol{d}(k+1)$) that are necessary for state predictions but do not require setpoint tracking. To ignore these variables in the control optimisation, they are assigned a zero weight in the MPC objective function. The objective function will be further discussed in the sections below.

Recall from Section 4.6.5 that $\dot{\theta}$ is not used in the estimated model. However, as discussed in Section 5.3, it is better for a controller to minimise $\dot{\theta}$, rather than θ. This is because θ has a non-zero steady-state value during a velocity step response due to aerodynamic drag. The state vector is therefore augmented with the $\dot{\theta}$ variable, such that the state vector is rather defined by,

$$\boldsymbol{x}_{mpc}{}^T = \begin{bmatrix} V_N & \theta & \dot{\theta} & \boldsymbol{d} \end{bmatrix}, \tag{5.26}$$

where $\boldsymbol{d}$ is the delay state vector defined in Equation 4.4. The $\boldsymbol{A}_{mpc}$ matrix is also augmented with a Backwards Euler numerical differentiation equation, such that,

$$\dot{\theta}(k) = \frac{1}{T_s}\theta(k) - \frac{1}{T_s}\theta(k-1). \tag{5.27}$$

In this way, a weight can be applied to the $\dot{\theta}$ variable in the MPC objective function to control this variable. $\boldsymbol{B}_{mpc}$ is augmented with zeros so that the state space matrix dimensions agree and so that $\boldsymbol{u}_{mpc}(k)$ does not directly influence $\dot{\theta}(k)$.

5.4.3. Algorithm implementation

A specific algorithm needs to be selected or designed to implement MPC in this work. There are numerous open-source methods available for this purpose. An extensive list of options is provided in the survey by [34] and the most promising ones are summarised here. A custom MPC implementation can be developed in MATLAB or C++ with the aid of software packages like CVXGEN [35], ACADO [36], YALMIP [37], Multi-Parametric Toolbox [38], and do-mpc [39]. Other open-source MPC implementations are also available as ROS packages from work done by [40] or [41].

The Simulink implementation of an MPC from the Model Predictive Control Toolbox™ [42] was selected for this work. It was selected because it integrates well with our simulation environment in Simulink and it specifically uses a discrete state-space plant model. The Model Predictive Control Toolbox™ also supports C++ code generation for stand-alone ROS nodes. Therefore, it can integrate with the SITL implementation of PX4 and it can be run on a companion computer for practical implementations.

The Model Predictive Control Toolbox™ solves the control optimisation problem as a Quadratic Program (QP) at each time interval [42]. To do this, it applies an active-set QP solver using the KWIK algorithm from [43]. This QP usually consists of three features, namely,

- the objective function,
- the constraints, and
- the controller decision

The objective function provides a scalar value that quantifies the controller performance. The controller decision is the set of $\boldsymbol{u}_{mpc}$ values determined by the QP solver that minimises the objective function. The constraints are conditions that the controller decision should satisfy, such as bounds on $\boldsymbol{x}_{mpc}$, $\boldsymbol{u}_{mpc}$, and $\Delta\boldsymbol{u}_{mpc}$ values.

A powerful advantage of MPC is that constraints are easily included in the optimal control implementation. However, constraints are not necessary for this work and will not be applied. This is advantageous for practical implementations because unconstrained MPC is less computationally complex than constrained MPC.

The objective function used by the Model Predictive Control Toolbox™ is documented well by the corresponding user manual [42], and an overview of this implementation will be presented. The objective function consists of the sum of three terms that each quantify a specific aspect of the control performance, and is calculated as,

$$J(\boldsymbol{z}_k) = J_y(\boldsymbol{z}_k) + J_u(\boldsymbol{z}_k) + J_{\Delta u}(\boldsymbol{z}_k), \tag{5.28}$$

where $\boldsymbol{z}_k$ is the controller decision at time-step k. The three scalar performance measures are denoted as $J_y(\boldsymbol{z}_k)$ for output setpoint tracking, $J_u(\boldsymbol{z}_k)$ for Munipulated Variable (MV) tracking, and $J_{\delta u}(\boldsymbol{z}_k)$ for MV move suppression. Each performance measure includes weights that balance the competing objectives of the different terms. These weights need to be manually tuned for a desired controller performance.

Output setpoint tracking

The performance measure of the output setpoint tracking is calculated as,

$$J_y(\boldsymbol{z}_k) = \sum_{j=1}^{n_y} \sum_{i=1}^{N_p} \left\{ w^y{}_j \left[r_j(k+i|k) - y_j(k+i|k) \right] \right\}^2, \tag{5.29}$$

where the symbols are denoted as,

k	Current control interval time-step.
n_y	Number of output variables.
N_p	Prediction horizon.
$y_j(k+i\|k)$	Predicted value of j^{th} output variable at i^{th} time-step from k.
$r_j(k+i\|k)$	Reference value of j^{th} output variable at i^{th} time-step from k.
$w^y{}_j$	Tuning weight for j^{th} output variable.

The controller receives the reference values, $r_j(k+i|k)$, for the prediction horizon starting at time-step k. Using the internal plant model, the predicted output values, $y_j(k+i|k)$, is determined based on the controller decision, $\boldsymbol{z}_k$. The values of N_p and $w^y{}_j$ are design choices that are constant controller specifications. The value of n_y are also constant and are determined from the plant model.

Manipulated variable tracking

In some control applications, it is desirable to keep specific MV variables close to a target value. In the multirotor use case, lower MV values are preferred because this corresponds to lower energy use. The MV target values are therefore set. The performance measure of manipulated variable tracking is calculated as,

$$J_u(\boldsymbol{z}_k) = \sum_{j=1}^{n_u} \sum_{i=0}^{N_p-1} \left\{ w^u{}_j \left[u_j(k+i|k) - u_{j,sp}(k+i|k) \right] \right\}^2, \tag{5.30}$$

where the symbols are denoted as,

n_u	Number of manipulated variables.
$u_j(k+i\|k)$	Control decision of j^{th} MV at i^{th} time-step from k.
$u_{j,target}(k+i\|k)$	Target value of j^{th} MV at i^{th} time-step from k.
$w^u{}_j$	Tuning weight for j^{th} MV.

The desired $u_{j,target}(k+i|k)$ values can be received for the prediction horizon starting at time-step k. However, for our use case, all $u_{j,target}(k+i|k)$ values are constant and zero. The value of n_u is fixed by the plant model. The $w^u{}_j$ values are also constant and are determined as a design decision.

Manipulated variable increment suppression

Large increments or moves in the MV values are often undesirable for good control performance. In the multirotor use case, large increments of the acceleration MVs result in aggressive jerks which may cause the system to go beyond the accurate domain of the linear approximation model. High frequency moves in the acceleration MVs may also cause jittery flight dynamics because acceleration setpoint changes correspond to attitude changes. The performance measure of MV tracking is used to penalise increments in the MVs. This is calculated as,

$$J_{\Delta}u(\boldsymbol{z}_k) = \sum_{j=1}^{n_u} \sum_{i=0}^{N_p-1} \left\{ w^{\Delta u}{}_j \left[u_j(k+i|k) - u_j(k+i-1|k) \right] \right\}^2, \tag{5.31}$$

where the symbols are denoted as,

$w^{\delta u}{}_j$ Tuning weight for movement in the j^{th} MV.

The values of $w^{\Delta u}{}_j$ are constant and are determined as a design choice.

It is important to note the similarities and differences between the LQR and MPC objective functions. The LQR implementation described in Section 5.3, did not included penalisation for MV increments. However, if $J_{\Delta}u(\boldsymbol{z}_k)$ is removed from Equation 5.28, the MPC and LQR optimiser consider the same variables.

The LQR optimisation corresponds to solving the unconstrained MPC optimisation problem for $N_p = \infty$. However, the LQR optimisation is run only once to determine the LQR gain, whereas the MPC optimisation is re-run at every time-step. Also note that the LQR uses a continuous-time model, but the MPC considered in this work uses a discrete-time model.

5.4.4. Integral action

A simple implementation of predictive control with multiple output variables does not inherently apply integral action or disturbance rejection. For the multirotor and suspended payload use case, MPC control without integral action results in a non-zero steady-state error of the multirotor velocity, due to wind disturbance and inaccuracies in modelling the drag.

Different methods have been proposed to apply integral action with an MPC. A commonly used method involves applying an integrator to the control action determined by the MPC [44]. In this implementation, the MPC determines the optimal control action increment, $\Delta u_k{}^*$, and calculates the control action, $u_k = u_{k-1} + \Delta u_k{}^*$ which is then applied. Hence, integral action is applied to the plant input. This method is also described by [32].

Another commonly used strategy involves estimating an input disturbance that influences an output variable in the plant model [44]. In this way, integral action is applied to the plant output. This integral action strategy will be applied in this work. Since integral action is required for V_N in the North velocity controller, a disturbance model that influences V_N is augmented to the input matrix.

The resulting state-space matrix is,

$$\boldsymbol{x}_{mpc}(k+1) = \boldsymbol{A}_{mpc}\boldsymbol{x}_{mpc}(k) + \begin{bmatrix}\boldsymbol{B}_{mpc} & \boldsymbol{B_{ud}}\end{bmatrix}\begin{bmatrix}\boldsymbol{u}_{mpc}(k) \\ \hat{u}_{ud}(k)\end{bmatrix}, \tag{5.32}$$

where $\boldsymbol{B_{ud}}$ is the input disturbance model and $\hat{u}_{ud}$ is the estimated input disturbance value. The input disturbance model is designed as,

$$\boldsymbol{B_{ud}} = \begin{bmatrix}b_{ud} & 0 & 0 & \boldsymbol{0}\end{bmatrix}^T, \tag{5.33}$$

such that an input disturbance only influences V_N in the state vector,

$$\boldsymbol{x_{mpc}}^T = \begin{bmatrix}V_N & \theta & \dot{\theta} & \boldsymbol{d}\end{bmatrix}. \tag{5.34}$$

The variable b_{ud} is a tunable value that quantifies the effect of the input disturbance on V_N. This variable will be tuned in Section 5.4.5

The specific value of the non-zero matrix entry in $\boldsymbol{B_{ud}}$ has only a slight effect on the control performance, hence the iterative tuning process for this value was simple. The value of $\hat{u}_{ud}(k)$ is estimated by the default Kalman filter estimator from the Model Predictive Control Toolbox™. This filter is based on the state-space model from Equation 5.32. The value of $\hat{u}_{ud}(k)$ is then used in the QP solver to determine the optimal control action of the MPC.

It was determined from simulations with different payload parameters and disturbances that the MPC with the default input disturbance estimator provides acceptable controller performance without additional tuning. Hence, the default Kalman filter is used in the final control implementation. Zero steady-state error for velocity tracking was achieved for different payloads and different input disturbances, showing that integral action is achieved. The simulation results showing the MPC integral action will be shown and discussed in Section 5.5.

5.4.5. Tuning

An MPC can easily be tuned for a range of different design requirements. The same general design requirements will be applied to the MPC as to the LQR in Section 5.3, namely, to produce a fast velocity response with zero steady-state error while damping the payload oscillations quickly. The MPC is firstly tuned to have a similar response time to the LQR so that the controller performances can be roughly compared. Thereafter, it is tuned to produce a trajectory that is as smooth as possible.

The MPC parameters that are determined in the controller design are, N_p, N_c, $\boldsymbol{w}^y$, $\boldsymbol{w}^u$, $\boldsymbol{w}^{\Delta u}$, and b_{ud}. T_s is fixed by the system identification phase, since the sample time of the discrete model and the controller should match.

MATH FONT

In the controller tuning process, a large value of ${w^y}_j$ corresponds to aggressive control of the j^{th} output variable, because the tracking error of that variable will be heavily penalised. In contrast, small values of ${w^u}_j$ or ${w^{\Delta u}}_j$, correspond to aggressive manoeuvres, because the control values are not heavily penalised in the objective function.

The computational complexity of the QP problem increases significantly with larger values of N_p [45]. The computational complexity also increases with larger values of N_c. Therefore the smallest values of N_p and N_c that still provide acceptable controller performance will be used. In the tuning process, the initial values were set to $T_p = T_c = 2 \times t_p$ where $T_p = N_p \times T_s$, $T_p = N_p \times T_s$, and t_p is the peak-time of the velocity step response with a PID controller. The objective function weights were then tuned for a desired controller performance.

THEY ARE THE SAME

The objective function weights were iteratively tuned for the desired control performance in a similar way to the LQR. All the weights corresponding to the delay-coordinates are set to zero. Due to the non-zero steady-state value of θ, the corresponding weight is also set to zero. Hereafter, the value of $T_p = T_c$ is incrementally decreased until a noticeable change in the controller performance. T_p is fixed at the smallest value before this change occurs. T_c is then further decreased until a noticeable change in performance.

The last variable to be tuned is b_{ud}, which influences the disturbance rejection performance of the controller. A large value of b_{ud} results in a fast disturbance rejection performance, however it also produces in a large overshoot. Starting at an initial guess of $b_{ud} = 1$, the value is iteratively tuned for a consistent performance with a range of different payloads and wind disturbances.

The final designed controller configuration parameters are shown in Table 5.2. Note that the weights in $\boldsymbol{w}^y$ correspond to the variables in, $[V_N \quad \theta \quad \dot{\theta}]$, the weight in $\boldsymbol{w}^u$ corresponds to the variable $A_{N_{sp}}$, and the weight in $\boldsymbol{w}^{\Delta u}$ corresponds to the variable $\Delta A_{N_{sp}}(k) = A_{N_{sp}}(k) - A_{N_{sp}}(k-1)$.

Table 5.2: MPC configuration parameters.

Parameter	Value
N_p	166
N_c	116
T_s	0.03 s
$\boldsymbol{w}^y$	$[2 \quad 0 \quad 10]$
$\boldsymbol{w}^u$	[0.1]
$\boldsymbol{w}^{\Delta u}$	[10]
b_{ud}	0.1

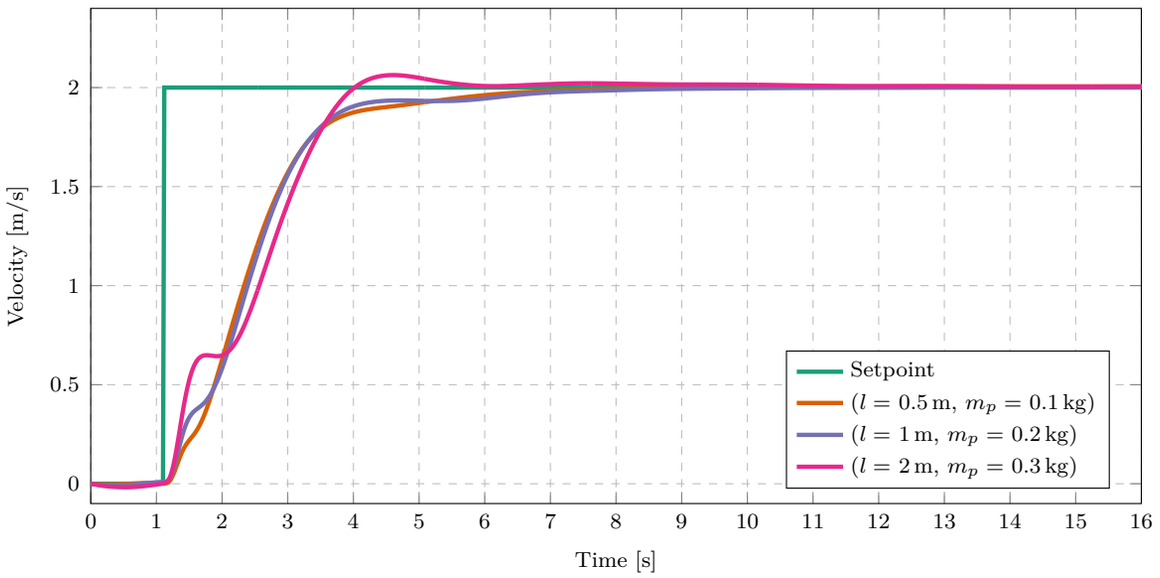


Figure 5.10: MPC velocity step responses with different payloads

Figure 5.10 shows a plot of the simulated MPC velocity response with different payload parameters. It is clear that the controller damps the payload angles well and produces a smooth trajectory with different payloads. For each different payload, a DMDc model is first estimated. Thereafter the MPC is simulated with the same controller configuration defined in Table 5.2. ~~The controller performance, including disturbance rejection, will be further discussed in Section 5.5.~~

5.5. Implementation and results

After the system identification phase, active swing damping control can be applied to the multirotor and payload system. The control architectures are summarised in Table 5.1 by pairing the system identification techniques with the appropriate controllers. In this work, the *MPC architecture* refers to the entire control implementation, which includes the data-driven system identification method and the MPC. Likewise, the *LQR architecture* includes the white-box modelling, the parameter estimation methods, and the resulting LQR determined from the system identification model. These control architectures will be tested in simulation and their results will be shown and compared in this section.

5.5.1. Simple suspended payload

The modelling assumptions of the white-box model discussed in Chapter 3 defines a point-mass suspended with a rigid cable which is attached to the CoM of the multirotor. This is a simplistic suspended payload model but represents the dynamics of many practical payloads well. In this section, the simulated payload model complies will all these assumptions. The simulation model used in this section was verified with practical data in Section 3.7. This is also the payload model used for simulations with an LQR controller by [4] and [5]

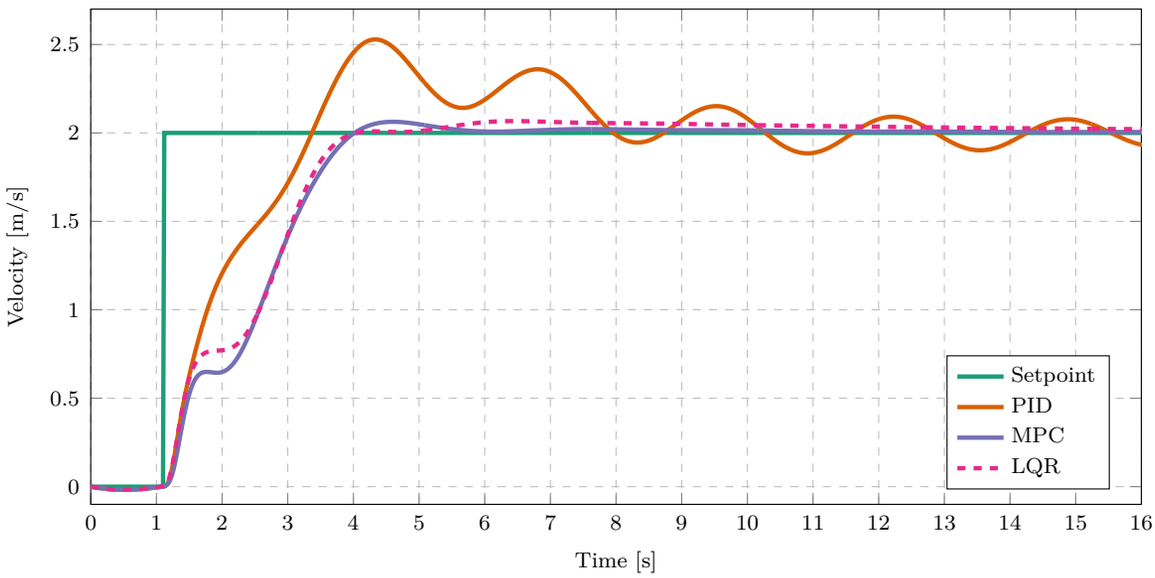


Figure 5.11: Velocity step response comparison of different controllers ($l = 2\,\text{m}$, $m_p = 0.3\,\text{kg}$)

From simulation results, it appears that both the MPC and LQR effectively damp the payload oscillations while controlling the velocity of the multirotor. Figure 5.11 shows the velocity step responses of the MPC, LQR and PID controllers for a multirotor with a suspended payload. From Figure 5.11 it is clear that both the LQR and MPC controllers

actively damp the velocity oscillation caused by the swinging payload. The PID controller does not consider the payload angle, hence the oscillations are not damped as strongly.

For the MPC and LQR, the respective models were first generated in the training phase of the simulation. Thereafter, the MPC and LQR were manually and iteratively tuned to produce a step response with a similar response time and overshoot. The PID response shown uses the same controller gains used in the training phase.

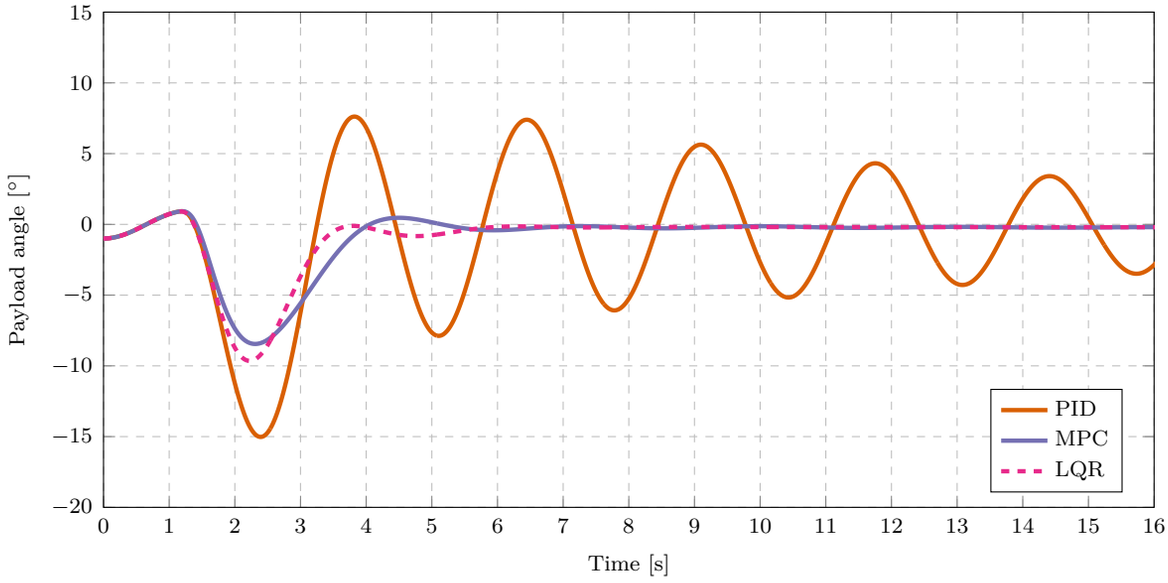


Figure 5.12: Payload angle comparison of different controllers ($l = 2\,\text{m}$, $m_p = 0.3\,\text{kg}$)

Figure 5.12 shows the payload angle data of the velocity step response. Both the MPC and LQR damp the payload angle well and the oscillations cease after only two or three swings. In this case, the MPC response results in a smaller initial swing angle, however, this is dependant on the specific tuning of each controller. The LQR can also be tuned to produce a similar swing angle.

Figure 5.13 shows the acceleration setpoint commanded by the two controllers for this step response. This is probably due to the inherent similarity between the controller implementations as discussed in Section 5.4. The similarity in the acceleration setpoint responses also show that the energy expended in a velocity steps are roughly equal for these two controller implementations. However, this is also highly dependant on the weightings used in optimisation problem of both controllers. Both controllers also produce a non-zero steady-state setpoint as expected, which is required to counter aerodynamic drag.

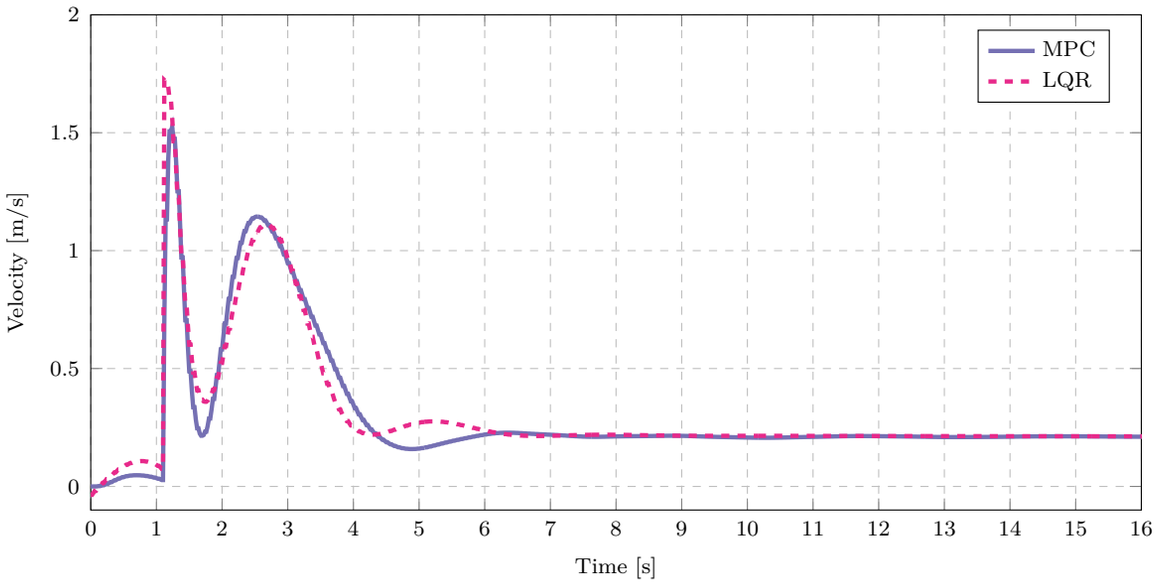


Figure 5.13: Acceleration setpoint commanded by different controllers for a velocity step input ($l = 2\,\mathrm{m}$, $m_p = 0.3\,\mathrm{kg}$)

5.5.2. Different payload parameters

The system identification and control implementations are required to perform well with different unknown payload parameters. Therefore, numerous flights with a range of different payload were simulated, the respective models were trained and the controllers were implemented. Figure 5.14 and Figure 5.15 show velocity step responses with LQR and MPC implementations with two payloads flights. Both the parameter estimation with LQR implementation, and the DMDc with MPC implementation, handle flights with different cable lengths and payload masses well. In each flight, LQR and MPC damp the payload oscillations and control the multirotor velocity well.

The controllers were not specifically tuned for each simulation. Instead, the same controller parameters were used for these simulations as for the simulations in Section 5.5.1. This shows that each control architecture is adaptable to different payload parameters without manual intervention.

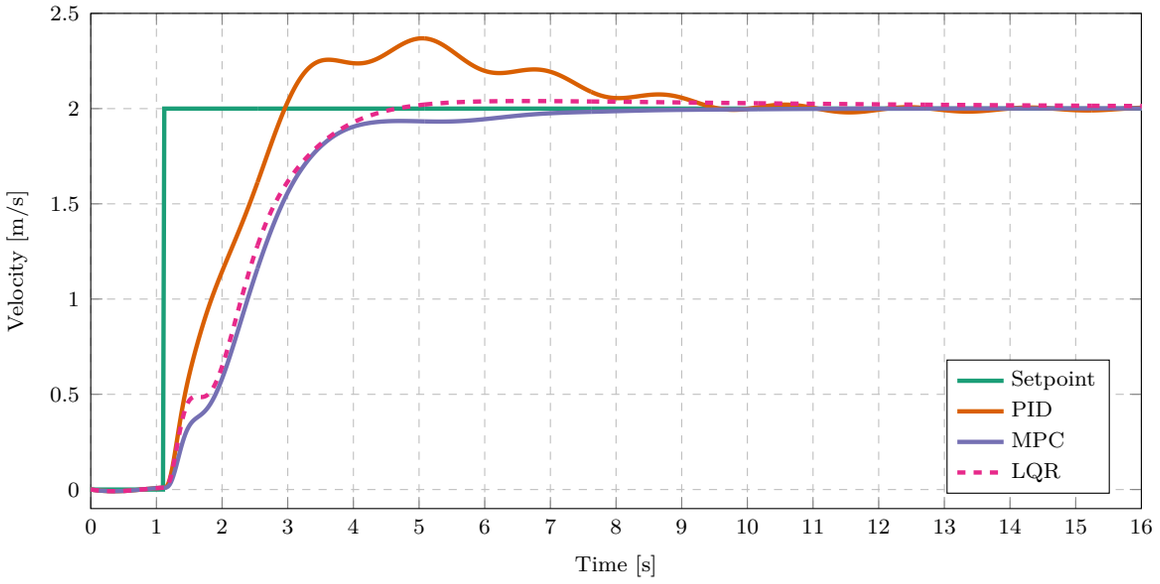


Figure 5.14: Velocity step response comparison of different controllers ($l = 1\,\mathrm{m}$, $m_p = 0.2\,\mathrm{kg}$)

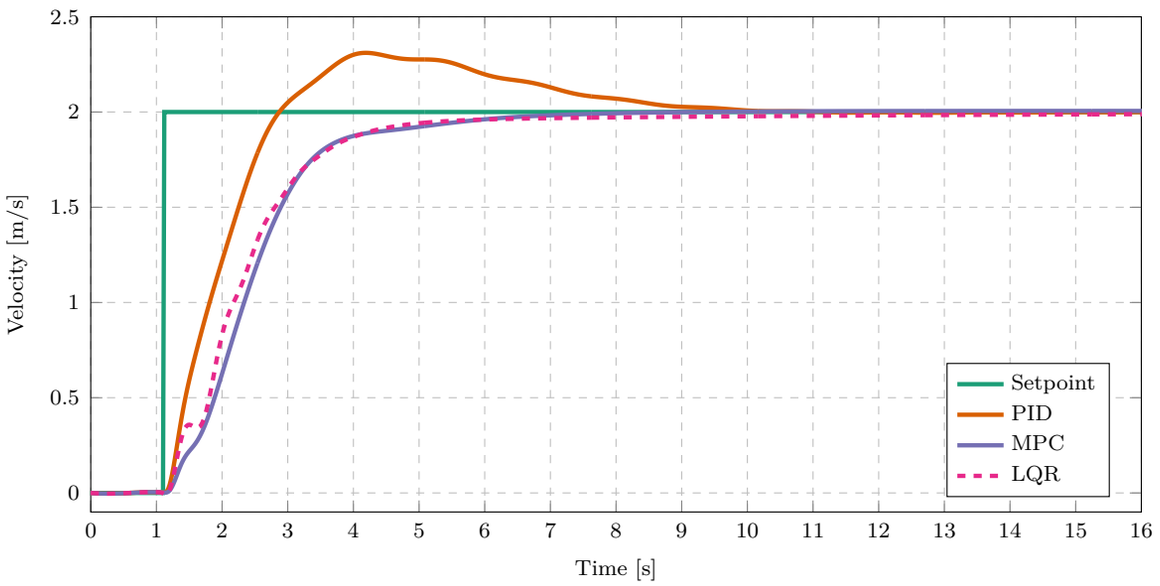


Figure 5.15: Velocity step response comparison of different controllers ($l = 0.5\,\mathrm{m}$, $m_p = 0.1\,\mathrm{kg}$)

5.5.3. Wind disturbance

For zero steady-state error with a practical system, a controller needs to apply some form of disturbance rejection. Practical systems experience unmeasured disturbances and other deviations which are not accounted for by the plant model. For example, a mean force applied by wind could prevent zero steady-state tracking error of the multirotor velocity without disturbance rejection. As discussed in Section 5.3, an integral state variable was added to the LQR plant model for integral action of the multirotor velocity tracking. As discussed in Section 5.4, an unmeasured input was added to the MPC plant model with a disturbance estimator to apply integral action to the multirotor velocity.

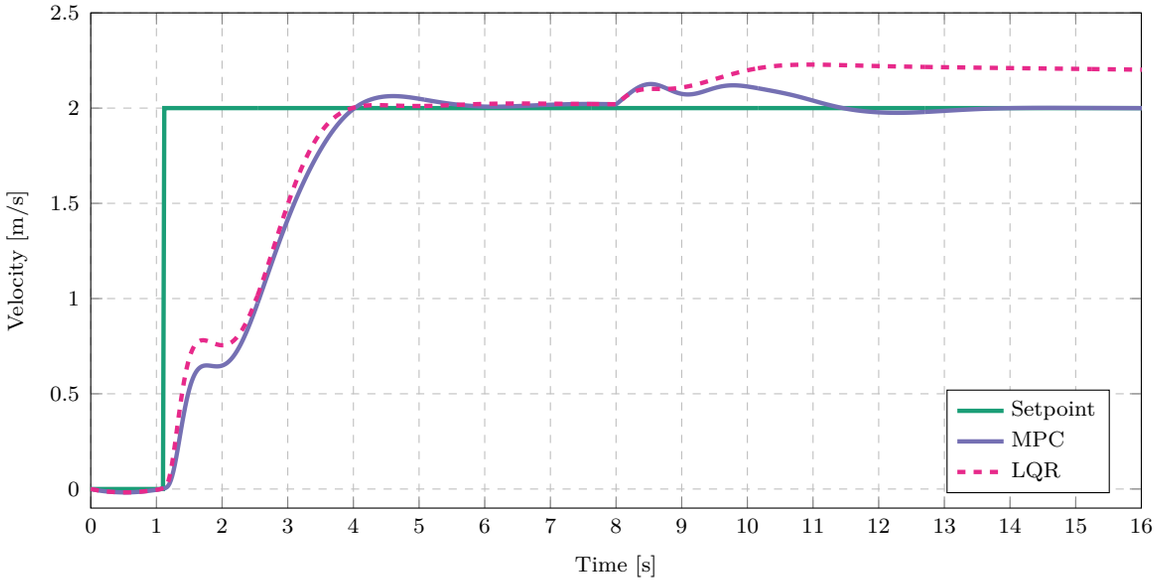


Figure 5.16: Effect of an unmeasured step input disturbance. ($l = 2\,\text{m}$, $m_p = 0.3\,\text{kg}$)

Figure 5.16 shows the responses of the controllers from Section 5.5.1 with a constant wind disturbance starting at 8 s. At Time = 8 s, a wind speed of 2 m/s is applied to the simulation model as an unmeasured step input. This mostly affects the multirotor velocity because the wind causes a greater drag force on the multirotor, hence a larger acceleration setpoint is required to maintain a constant velocity. For both system identification approaches, the models were trained without wind.

It appears that the MPC shows better disturbance rejection than the LQR when using the controller parameters which were tuned for good performance in Section 5.5.1. This is primarily because the weighting of the integral variable in the LQR optimisation was minimised to reduce overshoot. The integral weighting can be increased to improve integral action at the expense of increasing overshoot in the velocity response.

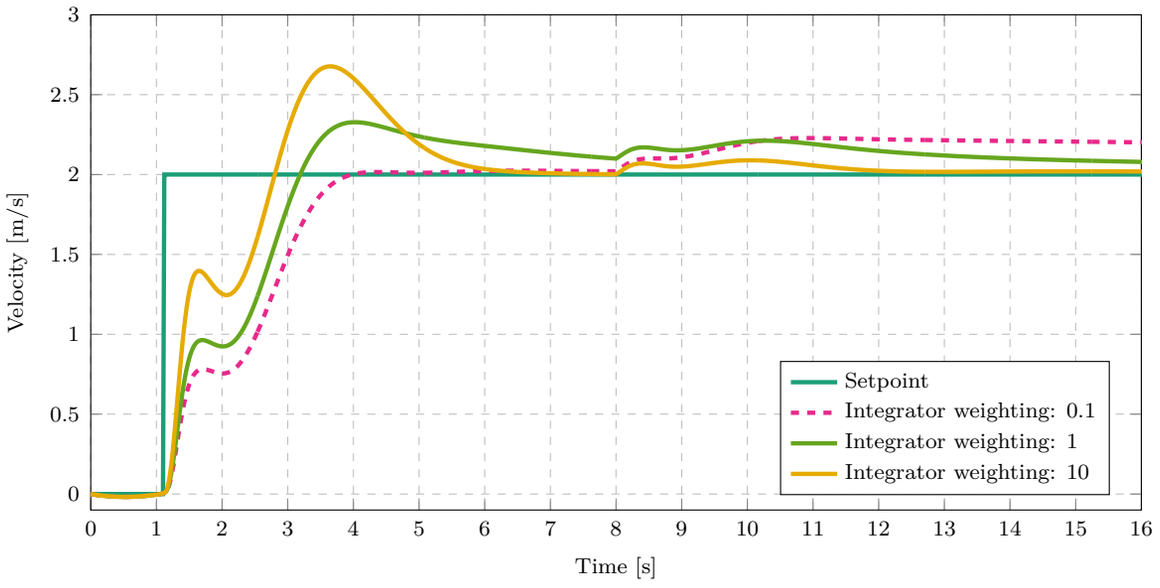


Figure 5.17: Different LQR responses for different integrator gains ($l = 2\,\text{m}$, $m_p = 0.3\,\text{kg}$).

Figure 5.17 shows the LQR responses with different integral state weightings. The other state variable weights are kept constant for each response. It is clear that the settling time and disturbance rejection of the LQR improves for larger integral state weighting. However, the overshoot increases significantly because of the integrator build-up at the start of the response.

In contrast to the LQR, the MPC shows good disturbance rejection while maintaining a low overshoot. This is because the disturbance estimator applies integral action which depends on the deviation of the actual dynamics from the plant model. whereas the LQR applies integral action proportional to the integral of the tracking error. Therefore the MPC implementation produces less integrator build up which results in a lower overshoot.

5.5.4. Dynamic payload

As discussed in Section 4.6.9, some payloads have dynamics that differ significantly from a suspended rigid mass. In this work, these payloads are referred to as dynamic payloads. An example of such a payload is an elongated payload, which can be represented by a double pendulum model.

In Section 4.6.9, the proposed system identification techniques were tested on simulated flight data with such a payload. For the white-box system identification approach, it was shown that the white-box model captures the dynamics of the dominant frequency of the oscillating payload but ignores the higher frequency dynamics. For the black-box approach, a prediction model was generated from a set of training data. This model accurately predicted the multirotor and payload dynamics, including the low and high-frequency

dynamics, of a set of testing data.

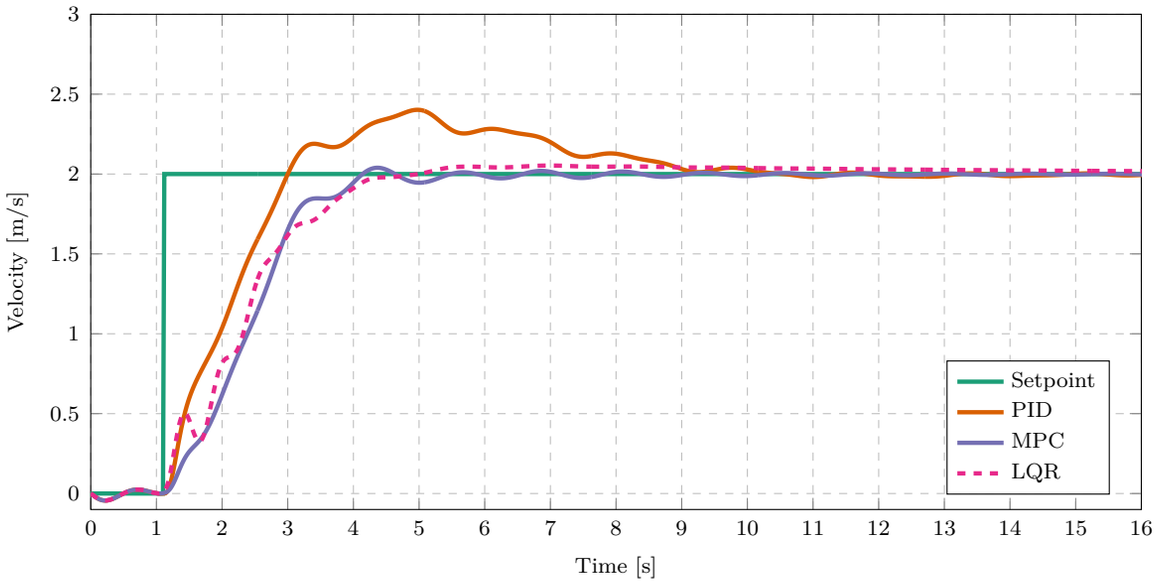


Figure 5.18: Velocity step response comparison of different controllers ($l = 2\,\mathrm{m}$, $m_p = 0.3\,\mathrm{kg}$)

For the simulations in this section, accurate system identification models were generated as described in Section 4.6.9 and used in the MPC and LQR controllers. Figure 5.18 shows the resulting controller responses with a dynamic payload. It appears that the velocity responses of both the LQR and the MPC are less smooth than with a simple suspended payload and show small velocity oscillations.

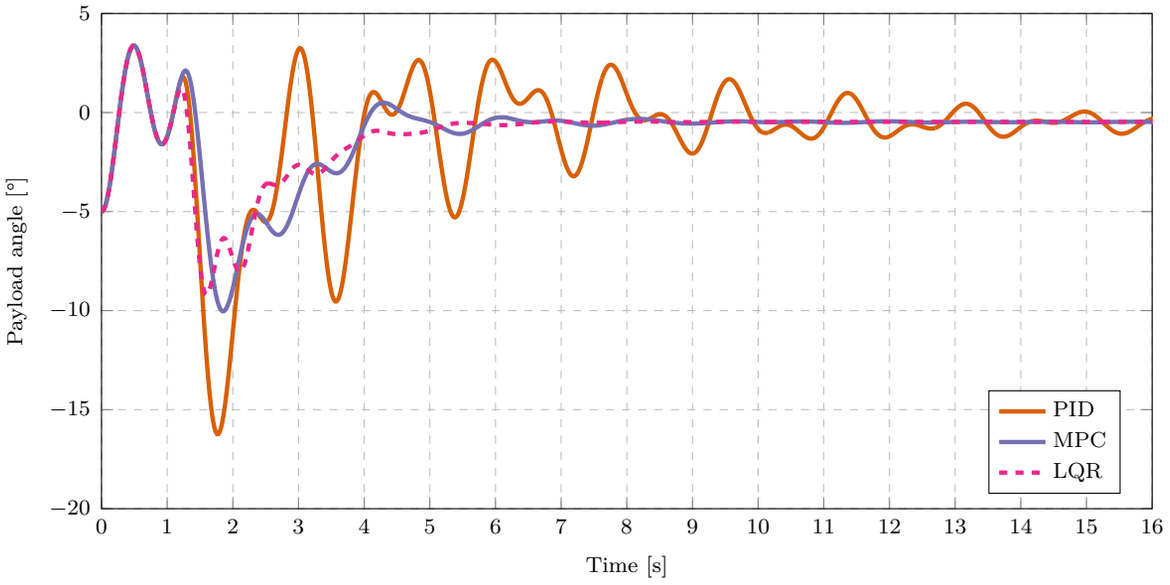


Figure 5.19: Payload angle comparison of different controllers ($l = 2\,\mathrm{m}$, $m_p = 0.3\,\mathrm{kg}$)

Figure 5.19 shows the suspension cable angle for a velocity step response. It appears that the LQR and MPC damp the payload oscillations with a similar response time. However,

the superimposed, high-frequency oscillations are smaller in the LQR response than in the MPC response. The LQR naively damps the payload oscillations because its controller gain is determined from the same dynamical model as for a simple suspended payload. Because the angle of the payload relative to the cable is not measured or considered in the LQR plant model, it does not directly damp these high-frequency oscillations. The MPC should account for the superimposed frequency and provide a smoother response than the LQR, since the black-box model includes the double pendulum dynamics in its prediction model. Therefore it is expected that the MPC optimiser determines a smooth trajectory that damps both the low and high-frequency oscillations.

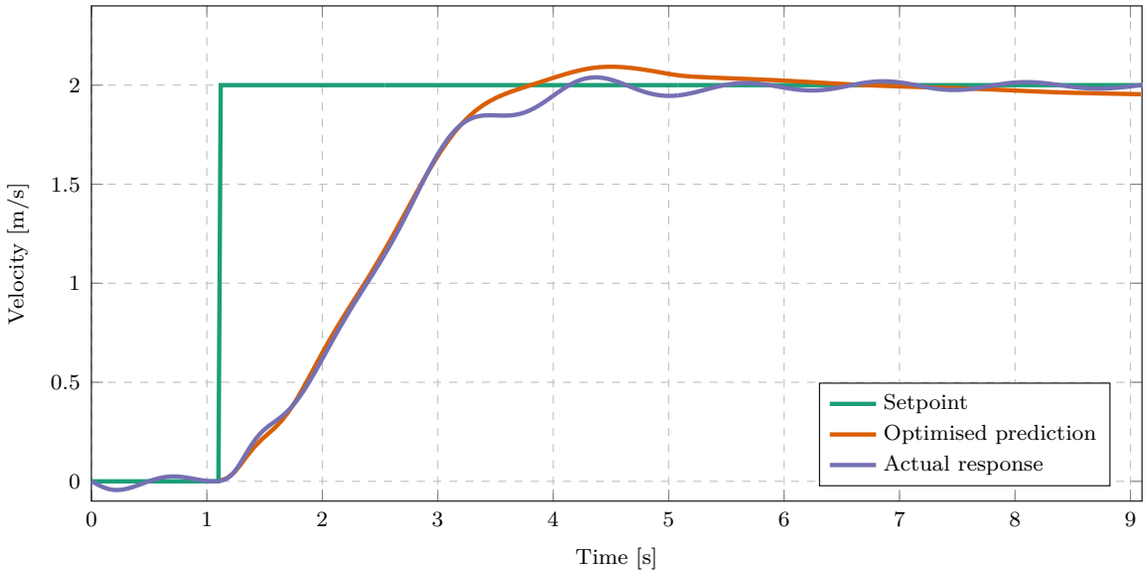


Figure 5.20: Optimised prediction and actual velocity response of the MCP with a dynamic payload

However, the MPC does not provide a smooth velocity or payload angle response. Even though the MPC generates a smooth optimised trajectory with the plant model, the actual response of the simulated system differs from this prediction. Figure 5.20 shows the predicted velocity of the MPC optimiser, given the velocity setpoint and initial condition at Time = 1.1 s. The actual simulated response, resulting from replanning at every time-step with the MPC, is also shown in Figure 5.20. For the first part of the velocity response, the the actual response matches the optimised trajectory well. However, after Time = 3.2 s, the predicted dynamics is noticeably different from the actual response to the optimised input sequence.

Figure 5.21 shows the predicted and actual payload angle response for this simulation, starting at the same time-step. The MPC optimiser also determined a smooth trajectory for the payload angle, but the actual response differs significantly from this trajectory. Even though the black-box model predictions accurately matched the testing data, Figure 5.20

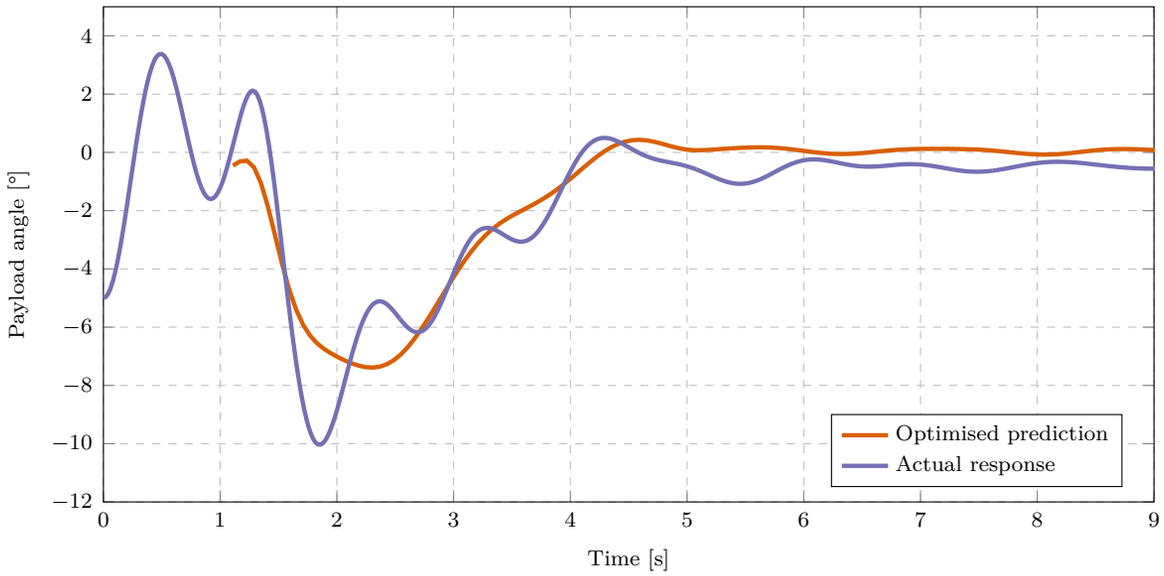


Figure 5.21: Optimised prediction and actual payload angle response of the MCP with a dynamic payload

and Figure 5.21 show that the model is not an accurate approximation of the simulated system for all values of the state and input vectors.

The estimated model provides accurate predictions in the domain of state and input vectors considered in the training data. However, the MPC generates trajectories that are beyond this domain of training data. The multirotor with a simple suspended payload represents a mildly non-linear system, hence the linear approximation was effective for control with an MPC. However, a double pendulum system reveals highly non-linear dynamics with multiple fixed points. This system also includes an unmeasured state variable which adds complexity to dynamics.

From the results in Figure 5.20 and Figure 5.21 it appears that the data-driven linear model results in acceptable control with an MPC. However, the actual dynamics do not follow the optimised trajectory of the MPC and the MPC does not conclusively outperform the LQR implementation.

5.5.5. Change in unexpected system parameters

The control architectures have been shown to be adaptable to variations in the payload parameters. However, changing other system parameters may affect the performance of the different controllers. As mentioned in Chapter 4, a disadvantage of the white-box system identification approach used by the LQR, is that parameter estimation techniques need to be manually designed for each unknown parameter. In the specific implementation, the LQR model assumes that the multirotor mass is known. Hence, changing the mass of the multirotor is detrimental to the accuracy of plant model and therefore affects the

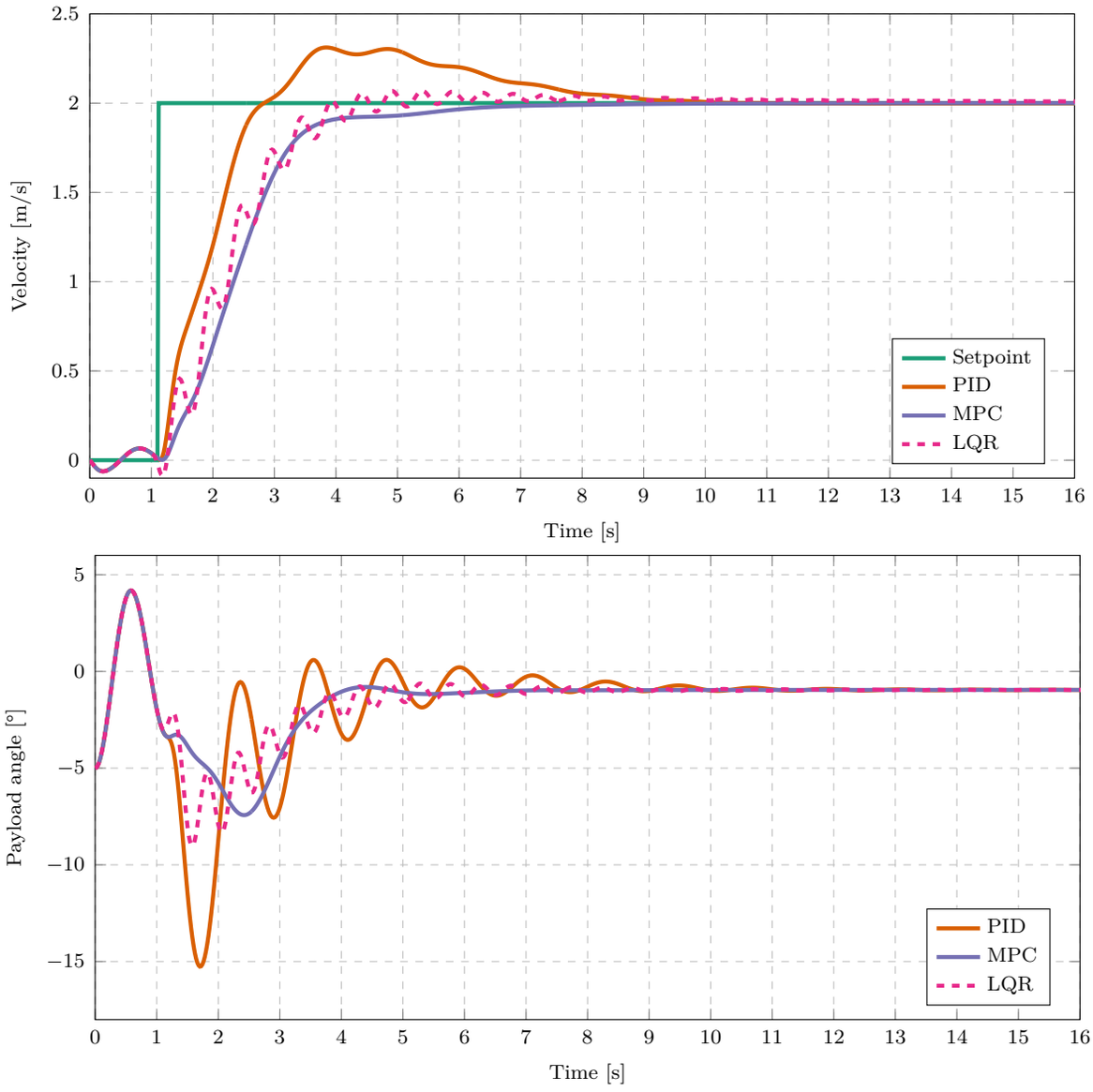


Figure 5.22: Velocity step responses with the multirotor mass decreased by 0.25 kg ($l = 0.5\,\text{m}$, $m_p = 0.3\,\text{kg}$)

LQR performance. In contrast, the data-driven system identification method for the MPC plant model does not rely on such modelling assumptions.

Simulations were performed with an altered multirotor mass to demonstrate how the control architectures handle changes in other system parameters. Figure 5.22 shows the velocity step responses of the PID, MPC and LQR implementations with an altered multirotor mass. For these simulations, the original multirotor mass, $m_Q = 0.796\,\text{kg}$, was decreased by $0.250\,\text{kg}$, resulting in a new multirotor mass of, $m_Q = 0.546\,\text{kg}$. The same system identification processes were naively followed as in the previous sections, without prior knowledge of the change in m_Q. The same tuned controller parameters were also used.

In Figure 5.22 it appears that the LQR results in lower payload oscillations than the PID controller but induces higher frequency oscillations. This results in a jittery velocity response with the LQR ~~and is undesirable for a multirotor flight~~. The LQR control performance has degraded because the dynamics of the LQR plant model differs significantly from the actual dynamics. In contrast, the MPC still results in a smooth velocity profile and damps the payload oscillations effectively, as in previous simulations. This is expected since the system identification model used by the MPC included the effect of the changed mass by estimating the entire model without considering individual parameters.

It should be noted that another mass estimator can be implemented to estimate m_Q in a flight stage before the payload is added. However, this involves manually redesigning the system identification procedure for each new unknown system parameter such as m_Q. In these simulations, it was shown that changing non-estimated system parameters in the white-box approach can be detrimental to the control performance. Unlike the white-box approach, the black-box approach handles changes in different system parameters well without prior knowledge of these parameters.

5.6. Conclusion

From simulations without wind disturbances, ~~it was shown that~~ the MPC and LQR architectures deliver similar control performances for a range of different payload parameters. Both controllers result in a similar response time and velocity overshoot, and the payload angle is damped well by both controllers. Therefore, in the absence of wind, the control performance does not conclusively differentiate between the LQR and MPC architectures for a simple suspended payload. As expected, the PID controller does not provide acceptable control of the multirotor with a suspended payload and does not actively damp the payload oscillations.

Both the LQR and the MPC architectures handle different payload parameters well. Even though the parameter estimation techniques (used with the LQR) did not consider the mass of the multirotor (m_Q), the LQR architecture still provided acceptable swing damping control with small changes in m_Q. However, it was shown that the LQR architecture produces an undesirable control performance for large changes in m_Q. Therefore, parameter estimation procedure will need to be redesigned to account for such changes in other system parameters. However, the data-driven approach does not rely on modelling assumptions, hence the MPC still provides good control performance for different values of m_Q.

Both the LQR and the MPC effectively rejected the unmeasured input disturbance caused by wind, resulting in zero steady-state tracking error for the multirotor velocity. However,

the LQR implementations which are tuned for effective disturbance rejection result in large velocity overshoots due to the integrator build-up. The MPC implementation applies integral action with a disturbance estimator and achieves zero steady-state error without increasing the velocity overshoot.

For simulations with a dynamic payload, both the LQR and MPC effectively applied swing damping control. However, ~~there~~ the trajectories were not as smooth as with the simple suspended payload. Even though the simulated dynamics differed significantly from the simple suspended payload model used by the LQR, the LQR still managed to damp the payload oscillations ~~quickly~~ effectively.

ONE PARAGRAPH

The MPC also managed to damp the payload oscillations, but did not conclusively outperform the LQR. Even though the estimated model used by the MPC showed good prediction accuracy with the given testing data set, the actual system response did not follow the predicted trajectory of the MPC well. It appears that the optimised trajectory of the MPC is beyond the domain where the linear model provides an accurate approximation of the non-linear dynamics. Therefore, an improved data-driven system identification model, which provides an accurate approximation of the dynamics for a larger domain, is required for improved MPC control of such a dynamic payload.

The advantage of the LQR architecture is that it is computationally simple in comparison to an MPC. However, the LQR architecture is designed for a specific system configuration and only accounts for changes in specific system parameters. In contrast, the MPC architecture provides a good general solution for different system configurations without considering individual parameters and without a priori modelling.

I REALY LIKE THIS PARAGRAPH.