Lab 6: Embedded MPC design, implementation and validation

MCHA3500

Semester 2 2019

Introduction

In this lab, you will design, implement and validate a model-predictive control (MPC) system for a Type I robot on your STM32F4 microcontroller using an incremental encoder to measure the wheel angle and a potentiometer to measure the pendulum angle.

The tasks should be completed and demonstrated to a marker **before the end of lab session you are enrolled in**. Once you complete the tasks, call the lab demonstrator to start your assessment. The lab is worth 3% of your course grade and is graded from 0–3 marks.

Background

By linearising the Lagrangian model of the robot, you should have obtained a model in the following form:

$$\mathbf{M}\ddot{\mathbf{q}} + \mathbf{D}\dot{\mathbf{q}} + \mathbf{K}\mathbf{q} = \boldsymbol{\tau}, \tag{1}$$

where $\mathbf{q} = \begin{bmatrix} x \\ \theta \end{bmatrix}$ are the position coordinates, $\mathbf{M}$, $\mathbf{D}$ and $\mathbf{K}$ are the mass, damping and stiffness matrices, respectively, for the linearised model, and $\boldsymbol{\tau} = \begin{bmatrix} F \\ 0 \end{bmatrix}$ are the generalised input forces for each degree of freedom, where F is the force acting on the cart. This model can be written in state-space form by choosing states $\mathbf{q}$ and $\mathbf{v} \triangleq \dot{\mathbf{q}}$ and then rearranging (1) to yield,

$$\begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\mathbf{q}} \end{bmatrix} = \begin{bmatrix} -\mathbf{M}^{-1}\mathbf{D} & -\mathbf{M}^{-1}\mathbf{K} \\ \mathbf{I} & \mathbf{0} \end{bmatrix} \begin{bmatrix} \mathbf{v} \\ \mathbf{q} \end{bmatrix} + \begin{bmatrix} \mathbf{M}^{-1}\mathbf{e} \\ \mathbf{0} \end{bmatrix} F, \tag{2}$$

where $\mathbf{e} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Let the force on the cart be produced by a torque applied to a wheel with radius r via an *ideal* gearhead permanent magnet DC motor with gyrator constant K, armature resistance R, armature inductance L and gear ratio N. It is assumed that the motor, gearbox and wheel masses and inertias have already been reflected and lumped with the cart mass, and need not be modelled separately.

The force acting on the cart is given by

$$F = \frac{NK}{r} I, \tag{3}$$

where I is the armature current, which follows the dynamics

$$\dot{I} = \frac{1}{L}\left(U_a - RI - \frac{NK}{r} v\right), \tag{4}$$

where U_a is the applied armature voltage.

Substituting (3) into the linearised cart-pole dynamics (2) and including the motor dynamics (4) yields the following linear state-space model:

$$\underbrace{\begin{bmatrix} \dot{\mathbf{v}} \\ \dot{\mathbf{q}} \\ \dot{I} \end{bmatrix}}_{\dot{\mathbf{x}}} = \underbrace{\begin{bmatrix} -\mathbf{M}^{-1}\mathbf{D} & -\mathbf{M}^{-1}\mathbf{K} & \frac{NK}{r}\mathbf{M}^{-1}\mathbf{e} \\ \mathbf{I}_{2\times 2} & \mathbf{0}_{2\times 2} & \mathbf{0}_{2\times 1} \\ -\frac{1}{L}\mathbf{e}^{\mathsf{T}} & \mathbf{0}_{1\times 2} & -\frac{R}{L} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{v} \\ \mathbf{q} \\ I \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} \mathbf{0}_{2\times 1} \\ \mathbf{0}_{2\times 1} \\ \frac{1}{L} \end{bmatrix}}_{\mathbf{B}} \underbrace{U_a}_{u}, \tag{5}$$

which we will consider as the plant model for the purposes of designing a control system.

There are two output models we need to consider in designing an output feedback controller for this system:

- $y_r = \mathbf{C}_r\mathbf{x} + D_r u$: the output that we wish to regulate, and
- $\mathbf{y}_m = \mathbf{C}_m\mathbf{x} + \mathbf{D}_m u$: the output measured from the available sensors.

The output we wish to regulate is the cart velocity, therefore the control output model should be

$$y_r = \dot{x} = \underbrace{\begin{bmatrix}1 & 0 & 0 & 0 & 0\end{bmatrix}}_{\mathbf{C}_r} \underbrace{\begin{bmatrix} v \\ \omega \\ x \\ \theta \\ I \end{bmatrix}}_{\mathbf{x}} + \underbrace{0}_{D_r}\underbrace{U_a}_{u}. \tag{6}$$

For the observer output model, we will assume that measurements of the wheel angle and pendulum angle are available and therefore

$$\mathbf{y}_m = \begin{bmatrix} \frac{1}{r}x \\ \theta \end{bmatrix} = \underbrace{\begin{bmatrix} 0 & 0 & \frac{1}{r} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathbf{C}_m} \underbrace{\begin{bmatrix} v \\ \omega \\ x \\ \theta \\ I \end{bmatrix}}_{\mathbf{x}} + \underbrace{\begin{bmatrix} 0 \\ 0 \end{bmatrix}}_{\mathbf{D}_m}\underbrace{U_a}_{u}. \tag{7}$$

Observer

The observer can be used to estimate the state at time step $k+1$ given information from measurements taken up to an including time step k. This estimate is denoted $\boldsymbol{\mu}_{k+1|k}$, where the first subscript indicates the time step of the estimate, and the second subscript indicates the time index of the latest measurement taken into account.

The predicted state estimate $\boldsymbol{\mu}_{k+1|k}$ can be updated recursively as follows:

$$\begin{aligned} \boldsymbol{\mu}_{k+1|k} &= \mathbf{A}_d\boldsymbol{\mu}_{k|k-1} + \mathbf{B}_d u_k + \mathbf{L}_o(\mathbf{y}_k - \mathbf{C}_m\boldsymbol{\mu}_{k|k-1} - D_m u_k) \\ &= \underbrace{(\mathbf{A}_d - \mathbf{L}_o\mathbf{C}_m)}_{\mathbf{A}_{\text{obs}}}\boldsymbol{\mu}_{k|k-1} + \underbrace{(\mathbf{B}_d - \mathbf{L}_o\mathbf{D}_m)}_{\mathbf{B}_{\text{obs}}} u_k + \mathbf{L}_o\mathbf{y}_{m,k}, \end{aligned} \tag{8}$$

where $\mathbf{L}_o$ is the observer gain and $\mathbf{y}_{m,k}$ is the measured sensor output at time step k.

Model-predictive control

Assume we have the discrete-time control model

$$\begin{aligned} \mathbf{x}_{k+1} &= \mathbf{A}_d\mathbf{x}_k + \mathbf{B}_d\mathbf{u}_k, \\ \mathbf{y}_k &= \mathbf{C}_r\mathbf{x}_k + \mathbf{D}_r\mathbf{u}_k, \end{aligned}$$

where $\mathbf{y}_k$ is the output to be regulated at time step k.

To design a controller for this system, we propose the following cost function:

$$\mathcal{V} = \underbrace{\sum_{k=1}^{\infty} \frac{1}{2}(\mathbf{y}_k - \mathbf{y}_k^*)^\mathsf{T}\mathbf{Q}_y(\mathbf{y}_k - \mathbf{y}_k^*)}_{\text{tracking error penalty}} + \underbrace{\sum_{k=1}^{\infty} \frac{1}{2}(\mathbf{u}_k - \mathbf{u}_{k-1})^\mathsf{T}\mathbf{R}_u(\mathbf{u}_k - \mathbf{u}_{k-1})}_{\text{actuator change penalty}}, \tag{9}$$

where we can tune $\mathbf{Q}_y$ to penalise the output error and we can tune $\mathbf{R}_u$ to penalise the change in actuator command from its setting at the previous time step.

To do this, we will introduce the following state transformation:

$$\underbrace{\begin{bmatrix}\mathbf{x}_{k+1}\\ \mathbf{u}_k\end{bmatrix}}_{\mathbf{z}_{k+1}} = \underbrace{\begin{bmatrix}\mathbf{A}_d & \mathbf{B}_d\\ \mathbf{0} & \mathbf{I}\end{bmatrix}}_{\mathbf{A}_z}\underbrace{\begin{bmatrix}\mathbf{x}_k\\ \mathbf{u}_{k-1}\end{bmatrix}}_{\mathbf{z}_k} + \underbrace{\begin{bmatrix}\mathbf{B}_d\\ \mathbf{I}\end{bmatrix}}_{\mathbf{B}_z}\underbrace{(\mathbf{u}_k - \mathbf{u}_{k-1})}_{\Delta\mathbf{u}_k}, \tag{10a}$$

$$\mathbf{y}_k = \underbrace{\begin{bmatrix}\mathbf{C}_r & \mathbf{D}_r\end{bmatrix}}_{\mathbf{C}_z}\underbrace{\begin{bmatrix}\mathbf{x}_k\\ \mathbf{u}_{k-1}\end{bmatrix}}_{\mathbf{z}_k} + \underbrace{\mathbf{D}_r}_{\mathbf{D}_z}\underbrace{(\mathbf{u}_k - \mathbf{u}_{k-1})}_{\Delta\mathbf{u}_k}. \tag{10b}$$

By substituting (10b) into (9), the cost function can be written as

$$\mathcal{V} = \sum_{k=1}^{\infty} \frac{1}{2}(\mathbf{z}_k - \mathbf{z}_k^*)^\mathsf{T}\mathbf{Q}_z(\mathbf{z}_k - \mathbf{z}_k^*) + \frac{1}{2}\Delta\mathbf{u}_k^\mathsf{T}\mathbf{R}_z\Delta\mathbf{u}_k + (\mathbf{z}_k - \mathbf{z}_k^*)^\mathsf{T}\mathbf{S}_z\Delta\mathbf{u}_k, \tag{11}$$

where

$$\begin{aligned}
\mathbf{Q}_z &= \mathbf{C}_z^\mathsf{T}\mathbf{Q}_y\mathbf{C}_z,\\
\mathbf{S}_z &= \mathbf{C}_z^\mathsf{T}\mathbf{Q}_y\mathbf{D}_z,\\
\mathbf{R}_z &= \mathbf{D}_z^\mathsf{T}\mathbf{Q}_y\mathbf{D}_z + \mathbf{R}_u,\\
\mathbf{z}_k^* &= \begin{bmatrix}\mathbf{x}_k^*\\ \mathbf{u}_{k-1}^*\end{bmatrix} = \begin{bmatrix}\mathbf{N}_x\mathbf{y}_k^*\\ \mathbf{N}_u\mathbf{y}_{k-1}^*\end{bmatrix}, \quad k = 1, 2, \ldots, N\\
\mathbf{z}_{N+1}^* &= \begin{bmatrix}\mathbf{x}_{ss}^*\\ \mathbf{u}_{ss}^*\end{bmatrix} = \begin{bmatrix}\mathbf{N}_x\mathbf{y}_\infty^*\\ \mathbf{N}_u\mathbf{y}_\infty^*\end{bmatrix},
\end{aligned}$$

and $\mathbf{N}_x$ and $\mathbf{N}_u$ satisfy

$$\begin{bmatrix}\mathbf{A}_d - \mathbf{I} & \mathbf{B}_d\\ \mathbf{C}_r & \mathbf{D}_r\end{bmatrix}\begin{bmatrix}\mathbf{N}_x\\ \mathbf{N}_u\end{bmatrix} = \begin{bmatrix}\mathbf{0}\\ \mathbf{I}\end{bmatrix}.$$

The cost (11) can be written as the sum of the first N terms and a terminal cost

$$\begin{aligned}
\mathcal{V} = &\frac{1}{2}(\mathbf{z}_{N+1} - \mathbf{z}_{N+1}^*)^\mathsf{T}\mathbf{Q}_f(\mathbf{z}_{N+1} - \mathbf{z}_{N+1}^*)\\
&+ \sum_{k=1}^{N} \frac{1}{2}(\mathbf{z}_k - \mathbf{z}_k^*)^\mathsf{T}\mathbf{Q}_z(\mathbf{z}_k - \mathbf{z}_k^*) + \frac{1}{2}\Delta\mathbf{u}_k^\mathsf{T}\mathbf{R}_z\Delta\mathbf{u}_k + (\mathbf{z}_k - \mathbf{z}_k^*)^\mathsf{T}\mathbf{S}_z\Delta\mathbf{u}_k.
\end{aligned} \tag{12}$$

The terminal penalty $\mathbf{Q}_f$ can be found by solving the associated algebraic Riccati equation,

$$\mathbf{Q}_f = \mathbf{A}_z^\mathsf{T}\mathbf{Q}_f\mathbf{A}_z - (\mathbf{A}_z^\mathsf{T}\mathbf{Q}_f\mathbf{B}_z + \mathbf{S}_z)(\mathbf{B}_z^\mathsf{T}\mathbf{Q}_f\mathbf{B}_z + \mathbf{R}_z)^{-1}(\mathbf{B}_z^\mathsf{T}\mathbf{Q}_f\mathbf{A}_z + \mathbf{S}_z^\mathsf{T}) + \mathbf{Q}_z.$$

The cost (12) can be written compactly as

$$\mathcal{V} = \frac{1}{2}(\mathfrak{z} - \mathfrak{z}^*)^\mathsf{T}\mathcal{Q}_z(\mathfrak{z} - \mathfrak{z}^*) + \frac{1}{2}\Delta\mathfrak{u}^\mathsf{T}\mathcal{R}_z\Delta\mathfrak{u} + (\mathfrak{z} - \mathfrak{z}^*)^\mathsf{T}\mathcal{S}_z\Delta\mathfrak{u}, \tag{13}$$

where

$$\mathfrak{z} = \begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \vdots \\ \mathbf{z}_N \\ \mathbf{z}_{N+1} \end{bmatrix}, \quad \mathfrak{z}^* = \begin{bmatrix} \mathbf{z}_1^* \\ \mathbf{z}_2^* \\ \vdots \\ \mathbf{z}_N^* \\ \mathbf{z}_{N+1}^* \end{bmatrix}, \quad \mathcal{Q}_z = \begin{bmatrix} \mathbf{Q}_z & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{Q}_z & \ddots & & \vdots \\ \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & & \ddots & \mathbf{Q}_z & \mathbf{0} \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} & \mathbf{Q}_f \end{bmatrix}, \quad \mathcal{S}_z = \begin{bmatrix} \mathbf{S}_z & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{S}_z & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \vdots & & \ddots & \mathbf{S}_z \\ \mathbf{0} & \cdots & \cdots & \mathbf{0} \end{bmatrix},$$

and

$$\Delta\mathfrak{u} = \begin{bmatrix} \mathbf{u}_1 - \mathbf{u}_0 \\ \mathbf{u}_2 - \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_N - \mathbf{u}_{N-1} \end{bmatrix}, \quad \mathcal{R}_z = \begin{bmatrix} \mathbf{R}_z & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_z & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & \mathbf{R}_z \end{bmatrix}.$$

The augmented state vector over $N+1$ steps is given by

$$\underbrace{\begin{bmatrix} \mathbf{z}_1 \\ \mathbf{z}_2 \\ \mathbf{z}_3 \\ \vdots \\ \mathbf{z}_{N+1} \end{bmatrix}}_{\mathfrak{z}} = \underbrace{\begin{bmatrix} \mathbf{I} \\ \mathbf{A}_z \\ \mathbf{A}_z^2 \\ \vdots \\ \mathbf{A}_z^N \end{bmatrix}}_{\mathcal{A}_z} \underbrace{\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_0 \end{bmatrix}}_{\mathbf{z}_1} + \underbrace{\begin{bmatrix} \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ \mathbf{B}_z & \ddots & & \vdots \\ \mathbf{A}_z\mathbf{B}_z & \mathbf{B}_z & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{A}_z^{N-1}\mathbf{B}_z & \cdots & \mathbf{A}_z\mathbf{B}_z & \mathbf{B}_z \end{bmatrix}}_{\mathcal{B}_z} \underbrace{\begin{bmatrix} \mathbf{u}_1 - \mathbf{u}_0 \\ \mathbf{u}_2 - \mathbf{u}_1 \\ \vdots \\ \mathbf{u}_N - \mathbf{u}_{N-1} \end{bmatrix}}_{\Delta\mathfrak{u}}, \tag{14}$$

where the control change vector over N steps is given by

$$\underbrace{\begin{bmatrix} \mathbf{u}_1 - \mathbf{u}_0 \\ \mathbf{u}_2 - \mathbf{u}_1 \\ \mathbf{u}_3 - \mathbf{u}_2 \\ \vdots \\ \mathbf{u}_N - \mathbf{u}_{N-1} \end{bmatrix}}_{\Delta\mathfrak{u}} = \underbrace{\begin{bmatrix} \mathbf{0} & -\mathbf{I} \\ \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} \\ \vdots & \vdots \\ \mathbf{0} & \mathbf{0} \end{bmatrix}}_{\mathcal{E}} \underbrace{\begin{bmatrix} \mathbf{x}_1 \\ \mathbf{u}_0 \end{bmatrix}}_{\mathbf{z}_1} + \underbrace{\begin{bmatrix} \mathbf{I} & \mathbf{0} & \cdots & \cdots & \mathbf{0} \\ -\mathbf{I} & \mathbf{I} & \ddots & & \vdots \\ \mathbf{0} & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \cdots & \mathbf{0} & -\mathbf{I} & \mathbf{I} \end{bmatrix}}_{\mathcal{W}} \underbrace{\begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \\ \mathbf{u}_3 \\ \vdots \\ \mathbf{u}_N \end{bmatrix}}_{\mathfrak{u}}. \tag{15}$$

Substituting (15) into (14) yields

$$\begin{aligned} \mathfrak{z} &= \mathcal{A}_z\mathbf{z}_1 + \mathcal{B}_z\Delta\mathfrak{u} \\ &= \mathcal{A}_z\mathbf{z}_1 + \mathcal{B}_z(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}) \\ &= (\mathcal{A}_z + \mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u}, \end{aligned} \tag{16}$$

and substituting (15) and (16) into (13) yields

$$\begin{aligned}
\mathcal{V} &= \frac{1}{2}(\mathfrak{z}-\mathfrak{z}^*)^\mathsf{T}\mathcal{Q}_z(\mathfrak{z}-\mathfrak{z}^*) + \frac{1}{2}\Delta\mathfrak{u}^\mathsf{T}\mathcal{R}_z\Delta\mathfrak{u} + (\mathfrak{z}-\mathfrak{z}^*)^\mathsf{T}\mathcal{S}_z\Delta\mathfrak{u} \\
&= \frac{1}{2}((\mathcal{A}_z+\mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u} - \mathfrak{z}^*)^\mathsf{T}\mathcal{Q}_z((\mathcal{A}_z+\mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u} - \mathfrak{z}^*) \\
&\quad + \frac{1}{2}(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u})^\mathsf{T}\mathcal{R}_z(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}) + ((\mathcal{A}_z+\mathcal{B}_z\mathcal{E})\mathbf{z}_1 + \mathcal{B}_z\mathcal{W}\mathfrak{u} - \mathfrak{z}^*)^\mathsf{T}\mathcal{S}_z(\mathcal{E}\mathbf{z}_1 + \mathcal{W}\mathfrak{u}) \\
&= \frac{1}{2}\mathfrak{u}^\mathsf{T}\underbrace{\mathcal{W}^\mathsf{T}(\mathcal{B}_z^\mathsf{T}\mathcal{Q}_z\mathcal{B}_z + \mathcal{B}_z^\mathsf{T}\mathcal{S}_z + \mathcal{S}_z^\mathsf{T}\mathcal{B}_z + \mathcal{R}_z)\mathcal{W}}_{\mathcal{H}}\mathfrak{u} \\
&\quad + \Big(\underbrace{\mathcal{W}^\mathsf{T}\left(\left((\mathcal{B}_z^\mathsf{T}\mathcal{Q}_z + \mathcal{S}_z^\mathsf{T})(\mathcal{A}_z+\mathcal{B}_z\mathcal{E}) + (\mathcal{B}_z^\mathsf{T}\mathcal{S}_z + \mathcal{R}_z)\mathcal{E}\right)\mathbf{z}_1 - (\mathcal{B}_z^\mathsf{T}\mathcal{Q}_z + \mathcal{S}_z^\mathsf{T})\mathfrak{z}^*\right)}_{\mathfrak{g}}\Big)^\mathsf{T}\mathfrak{u} \\
&\quad + \text{const},
\end{aligned}$$

The linear coefficient, $\mathfrak{g}$ depends on the predicted state and previous input $\mathbf{z}_1 = \left[\begin{smallmatrix}\mathbf{x}_1\\ \mathbf{u}_0\end{smallmatrix}\right]$ and desired state trajectory $\mathfrak{z}^*$, and can be factorised as follows:

$$\mathfrak{g} = \underbrace{\begin{bmatrix}\mathcal{W}^\mathsf{T}\left((\mathcal{B}^\mathsf{T}\mathcal{Q}+\mathcal{S}^\mathsf{T})(\mathcal{A}_z+\mathcal{B}\mathcal{E}) + (\mathcal{B}^\mathsf{T}\mathcal{S}+\mathcal{R})\mathcal{E}\right) & -\mathcal{W}^\mathsf{T}(\mathcal{B}^\mathsf{T}\mathcal{Q}+\mathcal{S}^\mathsf{T})\end{bmatrix}}_{\mathbf{\Gamma}}\underbrace{\begin{bmatrix}\mathbf{z}_1\\ \mathfrak{z}^*\end{bmatrix}}_{\mathbf{k}}. \tag{17}$$

The matrices $\mathcal{H}$ and $\mathbf{\Gamma}$ can be computed offline and stored for later use. When we run the controller online, we compute $\mathfrak{g} = \mathbf{\Gamma k}$ and then solve the following optimisation problem: pedictive controller actions can be computed as

$$\begin{aligned}
\mathfrak{u}^* &= \arg\min_{\mathfrak{u}} \frac{1}{2}\mathfrak{u}^\mathsf{T}\mathcal{H}\mathfrak{u} + \mathfrak{g}^\mathsf{T}\mathfrak{u}, \\
&\text{s.t.} \quad \mathbf{L}\,\mathfrak{u} \le \mathbf{m},
\end{aligned}$$

where $\mathbf{L}$ and $\mathbf{m}$ are chosen to implement the necessary input and state constraints (refer to lecture slides for details).

Merge the lab template

1. Open SourceTree and open your `workspace` directory (using **File** → **Open...**)
2. Click the **Fetch** button in the toolbar, ensure that **Fetch from all remotes** is checked and click **OK**. If you expand the **REMOTES** pane on the left-most bar of SourceTree, you should see a new remote branch **origin** → **lab6**.
3. Check that the current branch is **master**, which is indicated in bold in the left pane under the **BRANCHES** section. If **master** isn't the current branch, double-click it to switch to it.
4. Switch to the **History** view (click **WORKSPACE** → **History** on the left pane) and locate the commit associated with the `origin/lab6` branch in the commit history. Click this commit to highlight it and click the **Merge** button in the toolbar (or right click the commit and click **Merge...** in the context menu). Ensure that **Commit merged changes immediately** is

unchecked and click **OK** to attempt the merge. Depending on how you solved the previous labs, you may have one or more merge conflicts. All changes except merge conflicts are automatically staged, which you can check by switching to the **File status** view (click **WORKSPACE** → **File status** on the left pane). This view shows *staged* files in the top pane and *unstaged* files in the bottom pane.

5. If there any merge conflicts, these will be indicated with a warning triangle. Right click a conflicted file and click **Resolve Conflicts** → **Launch External Merge Tool**. This will open your configured mergetool with either 3 or 4 panels, depending on the tool. Work through the lines containing merge conflicts and select whether you want to accept the `LOCAL` chunks (i.e., on `master`), the `REMOTE` chunks (i.e., on `origin/lab6`) or the `BASE` chunks[1] (common ancestor) into the `MERGED` output. Save and close each file edited by the mergetool to let SourceTree know you have resolved the conflicts.

Tip

If you make a mistake, you can abort the merge at the terminal using

Terminal

```
nerd@basement ~/MCHA3500/workspace/robot $ git merge --abort
```

and then restart the merge within SourceTree.

Once all conflicts are resolved, confirm that all changes are staged, i.e., no changes appear in the **Unstaged files** area within SourceTree.

6. Before committing the merge, check all unit tests pass from the terminal.

Terminal

```
nerd@basement ~/MCHA3500/workspace/robot $ make test
[snip]
----------------------
[#] Tests 0 Failures 1 Ignored
OK
```

If necessary, fix any *other* issues that were introduced by the merging operation and stage these changes along with those staged during the merge.

Tip

You can stage changes in SourceTree by clicking the checkbox or plus sign next to each file in the **Unstaged files** area, or by dragging them into the top pane.

7. Click the commit message textbox in the bottom pane to expand the commit message editor. It should already be populated with a default merge message, e.g.,

```
Merge commit origin/lab6 into master
```

[1] Only for 4-panel mergetools

Click the **Commit** button to commit the merge.

1 Unconstrained MPC (1 mark)

Design and implement your observer and MPC controller in MATLAB/SIMULINK as follows:

1) Open `robot/matlab/Type_I_DC_MPC.mdl` in SIMULINK and complete the `Cart-pole` subsystem using your solution from Lab 2. You can copy and paste the internal blocks or the whole subsystem.

2) Open `robot/matlab/mpc_design_Type_I_DC.m` in MATLAB and continue with the steps below.

3) Compute the observer gain, $\mathbf{L}_o$, by computing the discrete-time steady-state Kalman gain as follows:

```
Qo     = ???;                          % Process noise (should be 5x5)
Ro     = ???;                          % Measurement noise (should be 2x2)
[~,Lo] = kalmd(ss(A, [B eye(5)], Cm, [Dm zeros(2,5)]), Qo, Ro, T);
```

Assume the measurement noise for both the encoder and potentiometer are dominated by quantisation effects.

Info

`kalmd` directly computes the discrete-time Kalman gain for a given continuous-time system. If we had used `kalman` with the discrete-time plant parameters $\mathbf{A}_d$ and $\mathbf{B}_d$, then $\mathbf{Q}_o$ and $\mathbf{R}_o$ are interpreted as process and measurement noise *covariances* in a discrete-time model. We ~~covered this in great detail~~ very briefly touched on this concept in the Week 4 lecture.

4) Review the observer and MPC controller implementation in SIMULINK. For the cart velocity reference, we will use the following signal:

$$v^*(t) = \begin{cases} 0\,\mathrm{m\,s^{-1}} & t < 10\,\mathrm{s}, \\ 1\,\mathrm{m\,s^{-1}} & 10\,\mathrm{s} \leq t < 20\,\mathrm{s}, \\ 0\,\mathrm{m\,s^{-1}} & t \geq 20\,\mathrm{s}. \end{cases} \tag{18}$$

5) Choose an appropriate sampling time, T, and control horizon, N, to suit the dynamics of the plant.

Hint

The sampling rate needs to be fast enough to stabilise the dynamics of the system and meet the desired closed-loop bandwidth (from reference to output). In MPC, we generally want as long a control horizon (in time) as is computationally feasible. This can be achieved by increasing the number of steps in the horizon and/or decreasing the sampling rate. The latter also affords more time to complete the calculation. On the other hand, since the plant is unstable, long term open-loop predictions will be poor beyond a few time constants of the fastest unstable pole, since small errors in the initial state lead to exponentially divergent state trajectories.

The time constants associated with the unstable pole(s) can be computed as follows:

```
lambda = eig(A);                % Pole locations
wn = abs(lambda);               % Natural frequencies [rad/s]
zeta = -cos(angle(lambda));     % Damping ratios [-]
taus = 1./(wn.*zeta);           % Time constants [s]
taus(real(lambda) > 0)          % Time constants of unstable poles [s]
```

6) Adjust $\mathbf{Q}_y$ and $\mathbf{R}_u$ to tune the closed-loop response such that the demanded armature voltage remains within $-12\,\text{V} \leq U_a \leq 12\,\text{V}$ for the entire simulation.

7) Once you are happy with the performance of your control system, stage and commit your changes to your local git repository.

2 Add input constraints (1 mark)

Achieve a more aggressive closed-loop response while ensuring the demanded control actions do not exceed saturation limits as follows:

1) Implement the input constraints

$$-12\,\text{V} \leq u_k \leq 12\,\text{V}, \quad k = 1, 2, \ldots, N,$$

where u_k is the demanded armature voltage at k time steps in the future. To do this, express the inequality constraints in the form

$$\mathcal{L}\mathfrak{u} \leq \mathfrak{m},$$

and implement these in MATLAB using the `Aineq` and `bineq` variables.

2) Adjust the control tuning to achieve an aggressive response and show that demanded armature voltage remains within the feasible set.

Note
It may be necessary to adjust the control horizon to ensure stability.

3) Once you are happy with the performance of your control system, stage and commit your changes to your local git repository.

3 Hardware-in-the-loop verification (1 mark)

In this section, you will perform a hardware-in-the-loop test using the **dSPACE DS1104 R&D Controller Board**. This test will verify the input/output behaviour of your controller on the embedded target. The dSPACE hardware is ideally suited for developing controllers, providing an interface for connecting a simulated plant in MATLAB/Simulink to your microprocessor in real-time. The microcontroller will receive measurements generated to accurately imitate real sensor outputs. Your

controller will provide its control action by generating an analog voltage from 0 V to 3.3 V using the on-board DAC of the STM32F446. This control output corresponds to the application of a 0 V to 12 V voltage to the brushed DC motor of the plant.

The existing `encoder` and `potentiometer` module will be used to measure the wheel angle ϕ and pendulum angle θ. The additional hardware initialisation necessary to perform the HIL test is already implemented in the provided modules.

The `mpc_task` module runs the control system in a 100Hz loop using an `osTimer`. The simulator and controller are synchronised to start and stop using an *active-low*[2] GPIO trigger on **PA12**.

The callback function associated with the MPC task performs the following steps in sequence:

- Emit stored control action ($\mathbf{u}_0$).
- Read sensors.
- Run one step of the observer to obtain $\boldsymbol{\mu}_{1|0}$.
- Compute $\mathfrak{g}$ online via matrix-vector multiplication.
- Solve the QP associated with the MPC problem and store $\mathbf{u}_1$ to be used on the next control step.
- Prints the time required to complete the previous steps to serial.

Complete the MPC implementation on your STM32F446 as follows:

1) Convert the measurement from the ADC to measure θ rad. The full scale of the measurement in LSBs is provided as an `enum` defined in the header of the `potentiometer` module. Assume the range 0 V to 3.3 V corresponds to the range $-120°$ to $120°$ on the potentiometer.
2) Convert the measurement from the quadrature encoder to measure the wheel angle, $\phi = \frac{1}{r}x$ rad. Assume there are 2048 events per mechanical revolution.
3) Update the enumerated values in `src/MPC.h` to reflect the correct dimensions of the control horizon (`MPC_N_CONTROL`) and number of general inequality constraints (`MPC_N_INEQ`).

Automation tip

Manually copying and pasting values from MATLAB into your C source is time consuming and error prone[citation needed]. Although it is not required for this lab, you might consider automating the population of these `enum` values from MATLAB.

4) Run the controller by entering `MPC start` in CoolTerm and determine the run time of each control calculation.

Note

If the control horizon is too long, the computational time required may exceed the time available within the controller task. In addition, the memory storage required may exceed the RAM available on the STM32F446.

[2] dSPACE pins have internal 5V pull-up resistors, defaulting their state to high

If necessary, reduce the control horizon and/or the sampling rate to ensure the control calculation can be performed within one sample period.

5) Connect your STM32 to the dSPACE headcrab.

> **Warning**
> Do not touch any of the other connectors on dSPACE system.

If a simulation was previously running, this will reset the plant simulation and will transition to a **PAUSE** state, waiting for your controller to start.

6) In CoolTerm use `MPC start` to start the controller. This will also synchronise the plant simulation to restart from initial conditions.

7) Observe the plots of the real-time simulation in the ControlDesk software on the lab computer. If successful, the pendulum angle θ will stabilise to approximately 0 rad. This should (approximately) look like Figure 1.

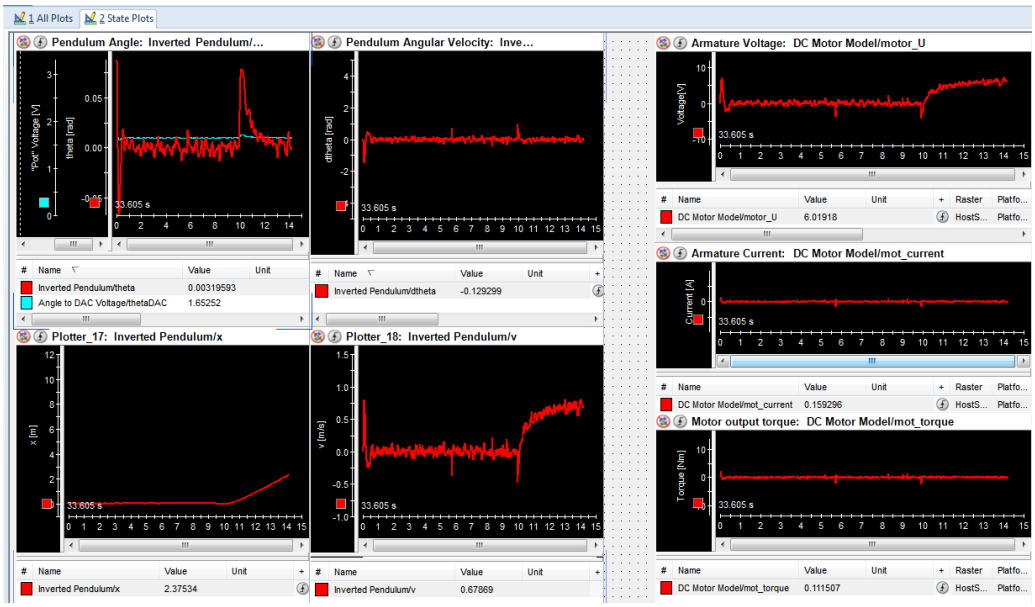

Figure 1: State plots of stable HIL simulation in dSPACE ControlDesk

8) Once you have validated your control design in hardware, demonstrate this to your tutor.

9) Stage and commit your changes to your local git repository.

Recommendations

Here are some suggestions on what you should work on next:

- Refactor the `MPC_run` function within the MPC module to handle multiple inputs and multiple outputs.

- Implement the actuator saturation constraints directly as *bound constraints* instead of general inequality constraints.
- Automate the transfer of MPC `enum` parameters MATLAB to native C as suggested in the **Automation tip** box.
- Consider implementing disturbance estimation by augmenting the observer with a model of an input disturbance and then implement feedfoward disturbance rejection in your controller. The observer states then become $\begin{bmatrix} \boldsymbol{\mu} \\ \hat{d} \end{bmatrix}$, where $\hat{d}$ is the estimated input disturbance.
- (DC motors only) Implement your control allocation function on the microcontroller and then add the simulation model of the *non-ideal* motor and gearbox in SIMULINK and connect it to the existing plant. Verify that this operates correctly.