Recap: Robust MPC

- Robust MPC solvers the following Optimal Control problem as follows:

$$J_0(\bar{x}_0) = \min_{X,U,Z} \quad \hat{J}_N(\bar{x}_N) + \sum_{i=0}^{N-1} \bar{x}_i^\top Q \bar{x}_i + u_i^\top R u_i$$

Nominal stage cost

$$\text{s.t.} \quad \bar{x}_{i+1} = A\bar{x}_i + Bu_i, \qquad \forall i \in \{0, 1, 2, ...\}$$

Nominal dynamics

$$\bar{x}_i \in \mathcal{X} \ominus \mathcal{R}_i, \qquad \forall i \in \{1, 2, ...\}$$

$$u_i = K\bar{x}_i + z_i, \qquad \forall i \in \{0, 1, 2, ...\}$$

z_i is effectively the control decision

$$u_i \in \mathcal{U} \ominus (K \circ \mathcal{R}_k), \qquad \forall i \in \{0, 1, 2, ...\}$$

$$x_0 = \bar{x}_0$$

$$x_N \in \mathcal{X}_f \ominus \mathcal{R}_N$$

Robust Invariant Set

Constraint Robustification

Adding Learning to Robust MPC

- We will now add a “learning-based” component to our model:
  - (1) The goal is to “reduce” the conservativeness of our robust formulation
  - (2) We can use Machine Learning to improve our dynamics model

- Let’s restart by recalling the linear dynamics:

$$x_{k+1} = Ax_k + Bu_k + w_k$$

- We will add some non-linear function approximation $h(x_k, u_k)$ in order to approximate w_k
  - That way we can try to “predict” the effect of uncertainty before it is manifested

- Then the dynamics would then become:

$$x_{k+1} = Ax_k + Bu_k + h(x_k, u_k)$$

Learning Dynamics

- Another interpretation for $h(x_k, u_k)$ is to capture the “complex” components of the system dynamics:

$$\underbrace{x_{k+1} - Ax_k + Bu_k}_{\boxed{\text{(linear) model mismatch}}} = w_k$$

- So $h(x_k, u_k)$ is an approximation of this mismatch.

- In this case, we say, the linear model is the **nominal model:**

$$\bar{x}_{k+1} = A\bar{x}_k + Bu_k$$

- In many applications the function $h(x_k, u_k)$ is also called the **Oracle Function.**

Learning-Based Model Predictive Control (LBMPC)

- As always, the LBMPC solves a N-step lookahead in a receding horizon fashion:

$$
\begin{aligned}
J_0(\bar{x}_0) = \min_{X,U,Z} \quad & \hat{J}_N(\tilde{x}_N) + \sum_{i=0}^{N-1} \tilde{x}_i^\top Q \tilde{x}_i + u_i^\top R u_i \\
\text{s.t.} \quad & \tilde{x}_{i+1} = A\tilde{x}_i + Bu_i + h(\tilde{x}_i, u_i), \quad \forall i \in \{0, 1, 2, ...\} \\
& \bar{x}_{i+1} = A\bar{x}_i + Bu_i, \quad \forall i \in \{0, 1, 2, ...\} \\
& u_i = K\bar{x}_i + z_i, \quad \forall i \in \{0, 1, 2, ...\} \\
& \bar{x}_i \in \mathcal{X} \ominus \mathcal{R}_i,\, u_i \in \mathcal{U} \ominus (K \circ \mathcal{R}_k), \qquad \forall i \in \{0, 1, 2, ...\} \\
& x_0 = \bar{x}_0 \\
& \bar{x}_N \in \mathcal{X}_f \ominus \mathcal{R}_N
\end{aligned}
$$

Learning-Based Model Predictive Control (LBMPC)

- Note that the key distinction here is that we keep two models:

Nominal Model

$$\bar{x}_{k+1} = A\bar{x}_k + Bu_k$$

- Enforce robust constraints.
- So, for any possible mismatch, the true system remains feasible:

$$x_{k+1} = A\bar{x}_k + Bu_k + w_k \in \mathcal{X}$$

Learned Model

$$\tilde{x}_{k+1} = A\tilde{x}_k + Bu_k + h(\tilde{x}_k, u_k)$$

- No constraints enforced on the learned model.
- The objective function is based on the learned model:

$$\hat{J}_N(\tilde{x}_N) + \sum_{i=0}^{N-1} \tilde{x}_i^\top Q\tilde{x}_i + u_i^\top Ru_i$$

LBMPC Properties

- It turns out that we can still ensure the two main properties, even with the Oracle:
  - **(1) Recursive Feasibility**
  - **(2) (Robust) Asymptotic Stability**

- We need that $\mathcal{X}_f$ to be a robust control invariant set, associated with the LQR control law.

- Our terminal cost function approximation will be given as before:

$$\hat{J}_N(\bar{x}_N) = \bar{x}_N^\top P \bar{x}_N$$

- Lastly the function $h_k(x_k, u_k)$ needs to be continuous and we needs to satisfy:

$$h(x_k, u_k) \in \mathcal{W}$$

So the approximation needs to bounded and be contained into the uncertainty polytope

LBMPC Properties

- The proofs follow the lines we covered last lecture:
  - Check the paper: “Provably Safe and Robust Learning-Based Model Predictive Control, Aswani et al. 2012” for the full proofs.

- Instead, we will focus our analyzes to designing an appropriate oracle function.

- For the MPC results to hold the function $h(x_k, u_k)$ needs to be:
  - (1) Continuous
  - (2) Bounded

- We will add (3) differentiable to the requirements (although it is not needed for the proofs)

- The reason to add differentiable is to allow numerical solvers to take gradients in order to solve the optimal control problem.

Parametric Oracles

- The first type of Oracles are the typical parametric functions:
  - The function $h(x_k, u_k)$ is defined by a parameter vector θ_k

- Suppose we were able to run the system in a simulator, applying some control sequence and obtaining the following trajectory:

$$(x_0, u_0, x_1, u_1, \ldots, x_{N-1}, u_{N-1}, x_N)$$

- Then we obtaining our oracle function, by solving the typical Regression Problem:

$$\theta^* \in \arg\min_{\theta} \Big\{ \sum_{i=0}^{N-1} (Y_i - h(x_i, u_i; \theta))^2 \Big\}$$

- Where:

$$Y_i = x_{i+1} - (Ax_i + Bu_i)$$

Example: Linear Oracles

- The oracle function can be itself a linear function:

$$h(x_k, u_k; \theta_k) = F_k x_k + G_k u_k$$

- In this case, the oracle has a nice interpretation as being a correction to the dynamics

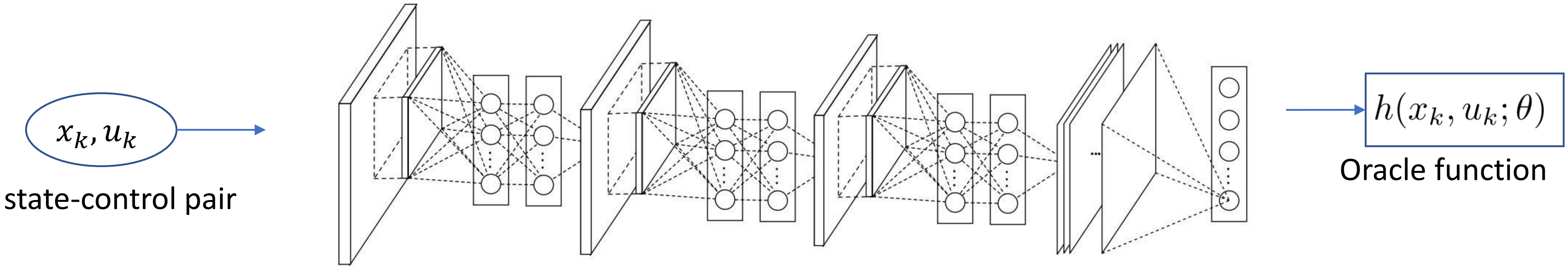
- The learned model then becomes:

$$\tilde{x}_{k+1} = (A + F_k)\tilde{x}_k + (B + G_k)u_k$$

- Where $\theta_k = (F_k, G_k)$. So we can see that oracle corrects the linear nominal model by essentially "updating" the matrices A and B.

Example: DNN's

- The oracle function can be given by a Deep Neural Network (DNN):



- And the DNN would be training in a similar fashion as we saw for model-free methods.

- Hence, LBMPC algorithm would alternate between two steps:
  - **(1)** Prediction step: updating the oracle
  - **(2)** Feedback step: Solving the optimal control problem

Non-Parametric Oracles

- The second type of Oracles are the ones that do not rely on parameters.

- One of such oracles is a kernel-based oracle called the **Nadararya-Watson Oracle**:

$$h(x_k, u_k) = \frac{\sum_{i=0}^{N-1}(x_{i+1} - Ax_i - Bu_i) \quad \mathcal{K}\left(h^{-2}\left\|\begin{bmatrix}x_k \\ u_k\end{bmatrix} - \begin{bmatrix}x_i \\ u_i\end{bmatrix}\right\|^2\right)}{\lambda + \sum_{i=0}^{N-1}\mathcal{K}\left(h^{-2}\left\|\begin{bmatrix}x_k \\ u_k\end{bmatrix} - \begin{bmatrix}x_i \\ u_i\end{bmatrix}\right\|^2\right)}$$

- Where $\mathcal{K}(\cdot)$ is a kernel function, and λ is some regularization hyper-parameter

Example: Kernel-based Oracles

- The idea of using kernels in LBMPC is very nice, because it does not assuming any prior form for the un-modelled dynamics.

- So the oracle essentially computes an withed average based on the kernels:

$$h(x_k, u_k) = \frac{\sum_{i=0}^{N-1} Y_i \mathcal{K}(z_i)}{\lambda + \sum_{i=0}^{N-1} \mathcal{K}(z_i)}$$

- Where

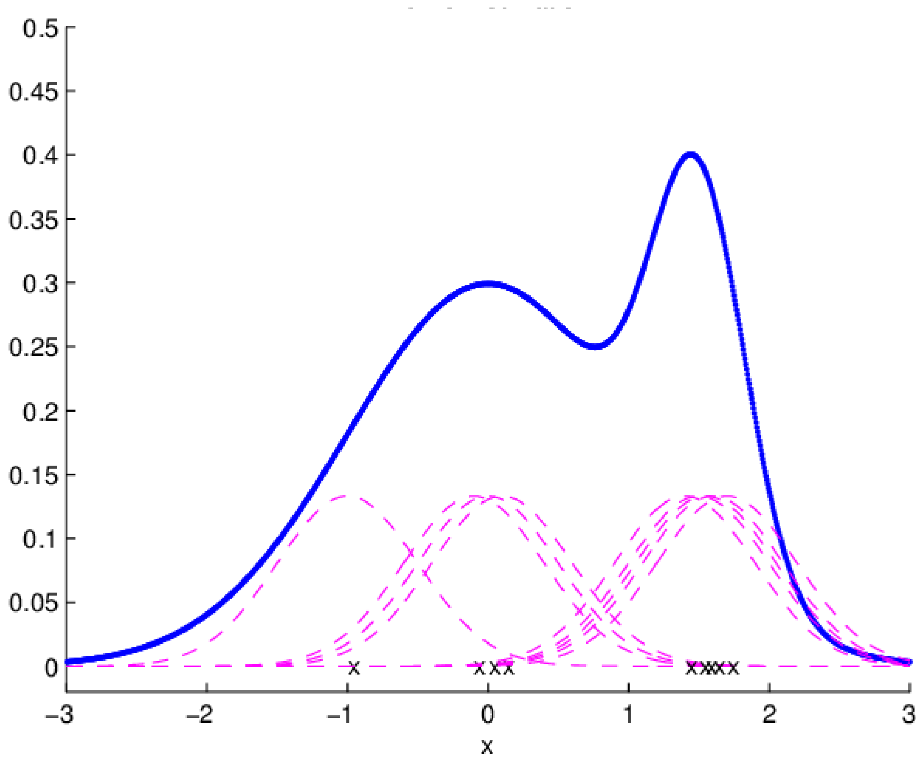
$$Y_i = x_{i+1} - (Ax_i + Bu_i) \qquad \mathcal{K}(z_i) = \mathcal{K}\left(h^{-2} \left\| \begin{bmatrix} x_k \\ u_k \end{bmatrix} - \begin{bmatrix} x_i \\ u_i \end{bmatrix} \right\|^2 \right)$$

Example: Kernel-based Oracles

- For example we can use Gaussian Kernels:

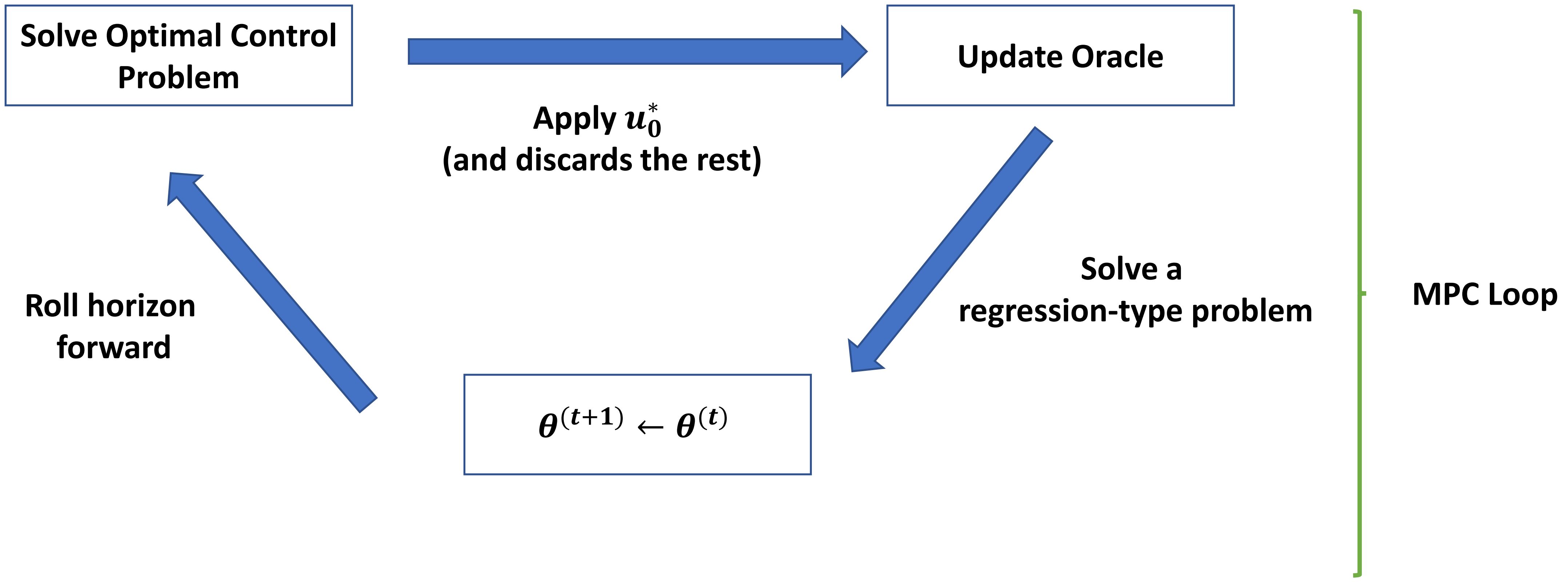
$$\mathcal{K}(z_i) = \frac{1}{\sqrt{2\pi}} \exp\left(- \left\| \begin{bmatrix} x_k \\ u_k \end{bmatrix} - \begin{bmatrix} x_i \\ u_i \end{bmatrix} \right\|^2 \right)$$

- We can represent this weighted-average by a figure:



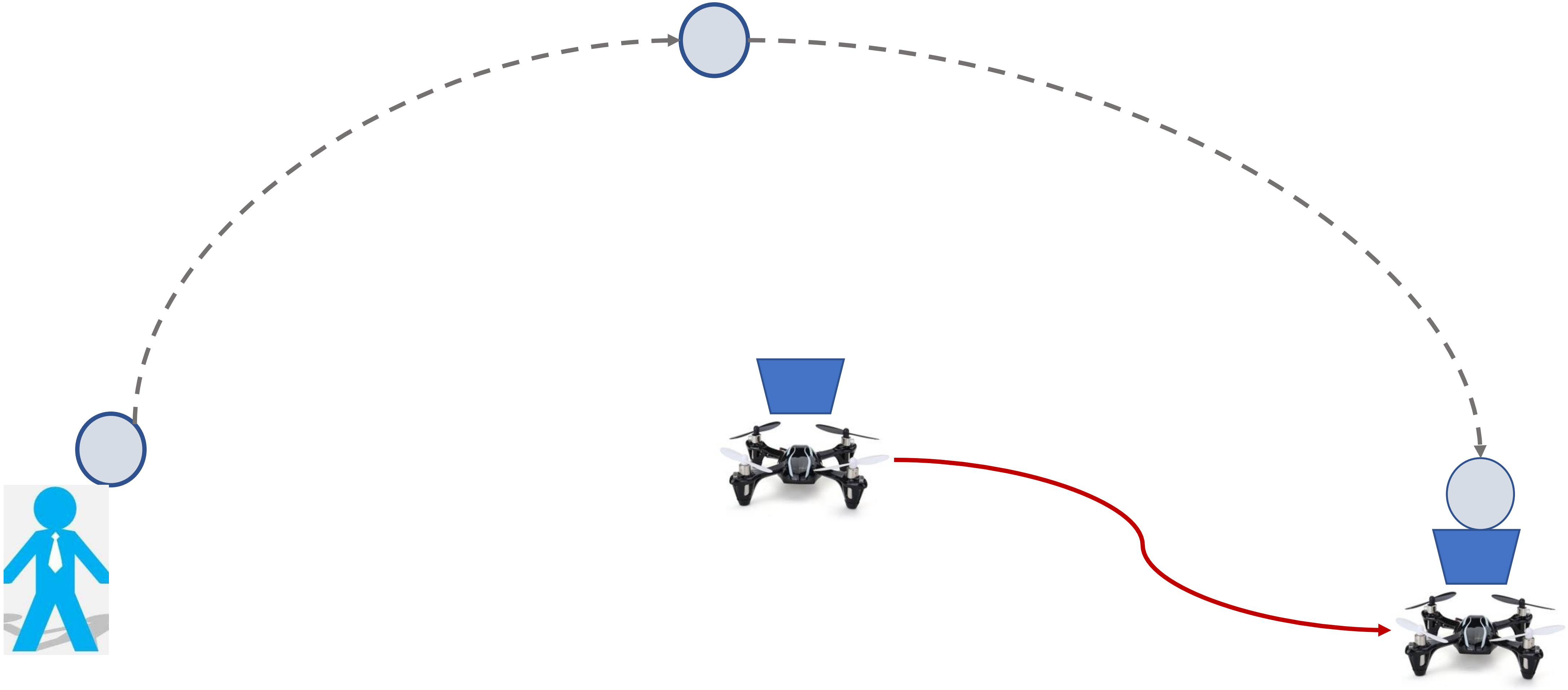
Learning-Based Model Predictive Control (LBMPC)

- We can represent the LBMPC in the following scheme:



Example of LBMPC: Quadrotor Flight Control

- We illustrate an application of LBMPC with a Ball-Catching experiment by a quadrotor:



Quadrotor Model

- The quadrotor drone can be modelled a linear system where the states are 3D-positition, their time-derivatives, the rotation angles and their derivatives.
    - (x_N, x_E, x_D) are the positions
    - (ψ, θ, ϕ) are the rotation angles (yaw-pitch-roll).
    - The rotation ψ angle is held fix, for reference.
- It is common to work with two sets of reference frames:
    - (1) body-fixed frame
    - (2) inertial frame
- For ease of presentation, let’s abstract the reference frames and just focus on the resulting linear system.
    - For a full description of the underlying physics we refer to: “Learning-Based Model Predictive Control on a Quadrotor: Onboard Implementation and Experimental Results. Bouffard et al.”

Quadrotor Model

- The linear dynamics are obtained by discretization of the continuous system, with steps $\Delta t = 0.025s$. And are given as follows, each horizontal axis:

$$x_{k+1} = Ax_k + Bu_k = \begin{bmatrix} 1 & 0.025 & 0.003 & 0 \\ 0 & 1 & 0.245 & 0 \\ 0 & 0 & 0.797 & 0.023 \\ 0 & 0 & -1.798 & 0.977 \end{bmatrix} x_k + \begin{bmatrix} 0 \\ 0 \\ 0.01 \\ 0.9921 \end{bmatrix} u_k$$

- For the vertical axis there is the effect the effect of gravity which changes the system matrices. For that component the linear dynamic is given by:

$$x_{k+1} = Ax_k + Bu_k = \begin{bmatrix} 1 & 0.025 \\ 0 & 1 \end{bmatrix} x_k + c_T \begin{bmatrix} 0.0003 \\ 0.025 \end{bmatrix} u_k + b_z$$

Quadrotor Model

- The full system is defined by the concatenation of the three axis:

$$A = \mathrm{blkdiag}(A, A, A_z) \in \mathbb{R}^{10\times 10}$$
$$B = \mathrm{blkdiag}(B, B, B_z) \in \mathbb{R}^{10\times 3}$$
$$b = \begin{bmatrix} 0 & 0 & b_z \end{bmatrix}^\top$$

- And the **nominal system** evolves as:

$$\bar{x}_{k+1} = A\bar{x}_k + Bu_k + b$$

- The control inputs u_k are the thrust executed along each axis.

Quadrotor Model

- Since we are using a linear model to approximate the flight dynamics we will add a linear oracle. The oracle will be time-varying in order to correct for mismatches in the linear model:

$$h(x_k, u_k) = F x_k + G u_k + v$$

- We can interpret this as (A, B) being the linearization (e.g.: derivate) information along some reference pair (x, u). And (F, G) are correction when we move from (x, u) to (x_k, u_k).

- Then the **learned model** becomes:

$$\tilde{x}_{k+1} = (A + F)\tilde{x}_k + (B + G)u_k + b + v$$

Ball in free flight model

- The ball model is taken to represent the dynamics of ball being thrown in the air by a human.

- The ball falls dues to gravity and spins, due to the human throwing it.

- We consider a linear model as well that incorporates air drag suffered by the ball while it is flight:

$$x_{k+1} = \mathrm{blkdiag}(A_b, A_b, A_b)x_k + b_F$$

- Where b_F is an empirical offset vector that is dependent on air resistance

- And A_b is a double integrator, discretized for each axis: $A_b = \begin{bmatrix} 1 & 0.025 \\ 0 & 1 \end{bmatrix}$

Training the Oracle

- In the practical experiments, the authors used onboard sensors to estimate the quadrotor and the ball positions.

- That means that the state x_k is not fully observed.

- In particular, the sensors are only able to estimate the actual positions and angles, with the associate derivatives not being observable.

- So we can write the sensor information as:

$$y_k = Cx_k + \epsilon$$

- Where ϵ is some white noise vector.

Training the Oracle

- In a practical situation such as this. We need to resort to system identification techniques in order to infer the state values from the observations.

- One such technique is the Extended Kalman Filter (EKF).
  - The reference paper implements this in their practical experiments.

- We have not covered this topic in the course. So for this presentation let's suppose we do have the full system state observation.
  - That is, we are able to fully compute the nominal state $\bar{x}_k$.

- Then in this case, we can estimate the oracle parameters via the typical regression step:

Training the Oracle

- The regression step is then as follows:

$$\theta^* \in \arg\min_{\theta} \Big\{ \sum_{i=0}^{N-1} (Y_i - h(x_i, u_i; \theta))^2 \Big\}$$

- Where $\theta = (F, G, v)$ and:

$$Y_i = x_{i+1} - (Ax_i + Bu_i)$$

- Note that as the system progress we keep adding more and more "data points" to this regression problem.
  - We can let the quadrotor fly in many simulation runs in order to collect a sizeable data set of transitions (x_i, u_i, x_{i+1}).

LBMPC for the quadrotor

- The LBMPC problem for the quadrotor is as follows:

$$
\begin{aligned}
J_0(\bar{x}_0, \theta_0) = \min_{X,U,Z} \quad & (\tilde{x}_N - x_s)^\top P(\tilde{x}_N - x_s) + \sum_{i=0}^{N-1} (\tilde{x}_i - x_s)^\top Q(\tilde{x}_i - x_s) + (u_i - u_s)^\top R(u_i - u_s) \\
\text{s.t.} \quad & \tilde{x}_{i+1} = (A + F_0)\tilde{x}_i + (B + G_0)u_i + b + v_0, \quad \forall i \in \{0, 1, 2, ...\} \\
& \bar{x}_{i+1} = A\bar{x}_i + Bu_i, \quad \forall i \in \{0, 1, 2, ...\} \\
& u_i = K\bar{x}_i + z_i, \quad \forall i \in \{0, 1, 2, ...\} \\
& \bar{x}_i \in \mathcal{X} \ominus \mathcal{R}_i,\ u_i \in \mathcal{U} \ominus (K \circ \mathcal{R}_k), \qquad \forall i \in \{0, 1, 2, ...\} \\
& x_0 = \bar{x}_0 \\
& \bar{x}_N \in \mathcal{X}_f \ominus \mathcal{R}_N
\end{aligned}
$$

- Where as usual, $\mathcal{X}_f$ is a robust control-invariant set associated with the LQR version of the problem. And the constraints are polyhedral sets

LBMPC for the quadrotor

- The state-control pair (x_s, u_s) used as a reference trajectory are the desired set-point of the quadrotor:
  - x_s is the predicted landing location for the ball.
  - u_s is the control that keeps the quadrotor stationary at x_s

- We can compute u_s by solving the following system of equations:

$$x_s = (A + F)x_s + (B + G)u_s + b + v_k$$

- Note that u_s may not be a feasible control. It does not need to be, it is only taken as a reference.

- We use the learned model in order to obtain the set-point reference.

Experiments with LBMPC

- The authors implemented the LBMPC algorithm in an onboard computer:
  - 1.6GHz Intel Atom N260 CPU
  - 1 GB of Ram
  - WiFi communications to Vicon MX motion capture system to estimate vehicle and ball positions
- The planning horizon for the MPC is N = 15.
- Commands are issued at the rate of 40Hz.
- The optimal control problem faced by the LBMPC at every planning stage is a Quadratic Program (so quadratic objective, linear constraints).

Experiments with LBMPC

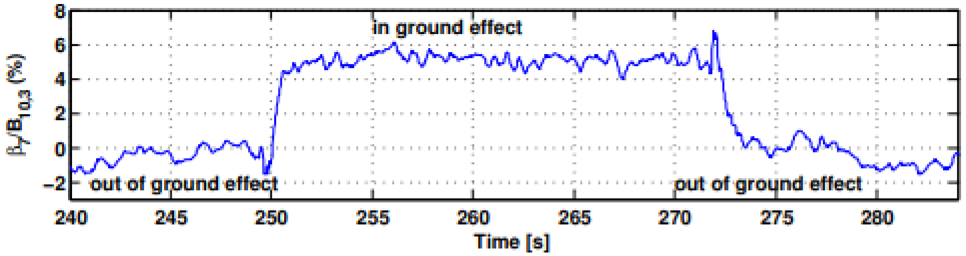
- There were two main sets of experiments:
  - (1) Flight close to the ground
  - (2) Flight to an alternating set-point reference

- The first experiment is interesting because it is designed to show how the oracle can "learn" the aerodynamical effect that ground has on the quadrotor:
  - If the vehicle hover very close to the ground, it subject to additional lift due to the air being "reflected" back to the vehicle.

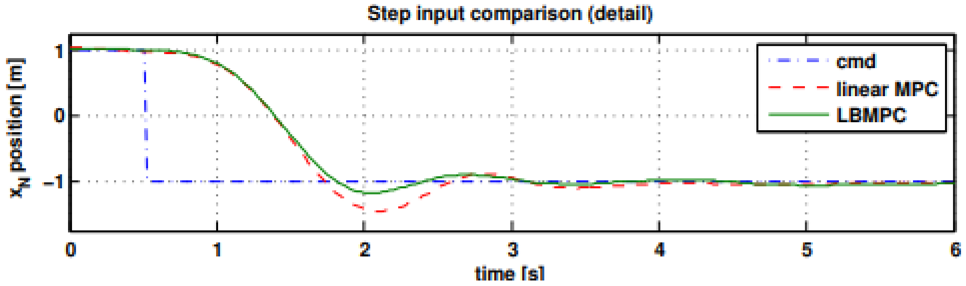
  - It is very important effect when considering "soft landing":
    - Landing smoothly without turning off the engines.

Ground effects experiment

- The results can be summarized as:



(Change in one of the oracle components)



(smoother stabilizing controller)

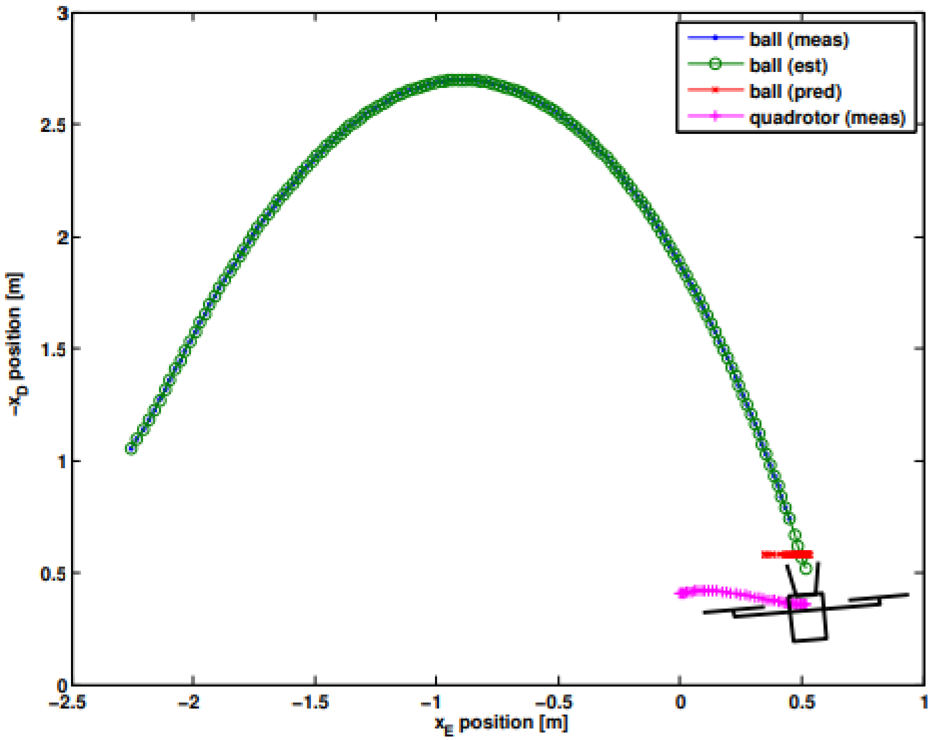
(figure taken from Bouffard, et al)

Experiments with LBMPC

- The second experiment involves ball catching.
- The (ping-pong sized) ball is thrown high in the air.
- The quadrotor need to catch it, slightly above ground (50cm).
- This is a challenging task, since the quadrotor has about 1 second to predict where the ball is going to land and to make the control decisions.
- The quadrotor continually updates the set point reference of the predicted landing point

Ball-catching experiment

- The following illustrate one instance of this experiment:



(figure taken from Bouffard, et al)

- And the results are also available in vide: (https://www.youtube.com/watch?v=dL_ZFSvLXlU)

Learning-Based Model Predictive Control (LBMPC)

- There is a technical detail about the LBMPC problem that is worth mentioning.
- Let's restate the problem again:

$$J_0(\bar{x}_0, \theta^{(0)}) = \min_{X,U,Z} \quad \hat{J}_N(\tilde{x}_N) + \sum_{i=0}^{N-1} \tilde{x}_i^\top Q \tilde{x}_i + u_i^\top R u_i$$

$$\begin{aligned}
\text{s.t. } & \tilde{x}_{i+1} = A\tilde{x}_i + Bu_i + h(\tilde{x}_i, u_i; \theta^{(0)}), \quad \forall i \in \{0,1,2,...\} \\
& \bar{x}_{i+1} = A\bar{x}_i + Bu_i, \quad \forall i \in \{0,1,2,...\} \\
& u_i = K\bar{x}_i + z_i, \quad \forall i \in \{0,1,2,...\} \\
& \bar{x}_i \in \mathcal{X} \ominus \mathcal{R}_i,\ u_i \in \mathcal{U} \ominus (K \circ \mathcal{R}_k), \qquad \forall i \in \{0,1,2,...\} \\
& x_0 = \bar{x}_0 \\
& \bar{x}_N \in \mathcal{X}_f \ominus \mathcal{R}_N
\end{aligned}$$

Learning-Based Model Predictive Control (LBMPC)

- Note that the optimization problem depends not only on the initial state $\bar{x}_0$ but also on the oracle parameter vector θ_0.

- In an abstract representation, we can write that problem as follows:

$$J_0(\bar{x}_0, \theta^{(0)}) = \min_{X, U, Z} \quad F(X, U, Z)$$
$$\text{s.t. } (X, U, Z) \in G(\theta_0, \bar{x}_0)$$

- Where we highlight that the feasible region depends on θ_0.

Learning-Based Model Predictive Control (LBMPC)

- This is something we have encountered before, but never really addressed.

- Since our MPC algorithm is a **model-based** algorithm, one question to ask is whether we are able to learn in fact the true dynamics, if we use a “rich” enough oracle function.

- Namely suppose there exist a function $h^*(x, u)$ such that:

$$x_{i+1} = Ax_i + Bu_i + h^*(x_i, u_i)$$

- And let $J^*(\bar{x})$ be the Optimal value function if we solved the Optimal Control problem with h^* and starting from $\bar{x}$. Is it true that $J\left(\bar{x}, \theta^{(t)}\right) \to J^*(\bar{x})$, as $t \to \infty$?

Convergence of Approximate Optimization

- Recall the simple least squares problem where we wish to solve the following problem:

$$\theta^* = \arg\min_\theta \mathbb{E}\big[(y - x^\top \theta)^2\big]$$

- We typically cannot solve this problem due to the expectation. And we resort to solve, instead, a Sample Average Approximation (SAA):

$$\hat{\theta}_N = \arg\min_\theta \frac{1}{N}\sum_{i=1}^{N}(y_i - x_i^\top \theta)^2$$

- We are concerned if $\hat{\theta}_N \to \theta^*$.

- For this least-squares problem, this is true due to the Uniform Law of Large Numbers.

Convergence of Approximate Optimization

- But in our LBMPC, we are solving a constrained optimization problem that changes over time, due to both initial condition and parameter vector.

- So it is not trivial that the Law of Large Numbers would hold in this case.

- Let $F_n(x)$ be some function at stage n, with parameter vector θ_n.

- We say that a function F_n epi-converges to another function F if and only if, at each point x:

$$\liminf_n F_n(x_n) \geq f(x), \forall x_n \to x$$

$$\limsup_n F_n(x_n) \leq f(x), \exists x_n \to x$$

Convergence of Approximate Optimization

- This definition may be not intuitive. An intuitive explanation is say that F_n epi-converges to F if the epigraph of F_n converges to F.

- In addition if we minimize both functions over a bounded non-empty set X, it follows that:

$$V_n \to V \qquad V_n = \min_{x \in X}\{F_n(x)\} \qquad V = \min_{x \in X}\{F(x)\}$$

- And each approximated problem: $V_n = \min_{x \in X}\{F_n(x)\}$

- Is feasible and form a bounded sequence where the set of optimal solutions will also converge. In the sense that:

$$\limsup_n(\arg\min F_n(x)) \subseteq \arg\min f(x)$$

Convergence of LBMPC: Overview

- It turns out that we can apply this notion of epi-convergence to the LBMPC and prove that

- Using certain types of oracles such as:
  - Linear Oracles
  - Nadaraya-Watson (kernel-based) oracles

- We will converge, in the sense that the oracle will estimate accurately the model mismatches between reality and the nominal model.

- These proofs are a very technical and require a lot of groundwork.

- We present this very high-level view just to highlight the theoretical guarantees that LBMPC enjoy when employing function approximations.