EE 290-005 Integrated Perception, Learning, and Control **Spring 2021**

Lecture 14: Model-based RL & Model Predictive Control

Scribes: Shuxiao Chen, Prasanth Kotaru, Zhongyu Li, Bike Zhang

Presented by Simon Zhai, Haozhi Qi, Aravind Srinivas, and Mihaela Curmei

14.1 KeyPoint Affordances for Category-Level Robotic Manipulation (kPAM)

14.1.1 Overview

It is very important to extract useful information from images captured by a camera in order for a robot to accurately manipulate the objects in the scene, especially semantically. Paper *KeyPoint Affordances for Category-Level Robotic Manipulation* [4] (kPAM) aims at detecting the three-dimensional (3D) semantic key points of an object given an image containing it. After the key points are semantically extracted from the image, information such as the pose of the object is inferred and used for the planning stage of the robot. For example as shown in Figure 14.1, such a task can be hanging a mug on a rack where the bottom, top and handle of the mug along with their position information are obtained from the image, as well as the relative position and pose of the branches of the rack. The robotic arm is then planned to grab the handle of the mug and match it with one of the branches.

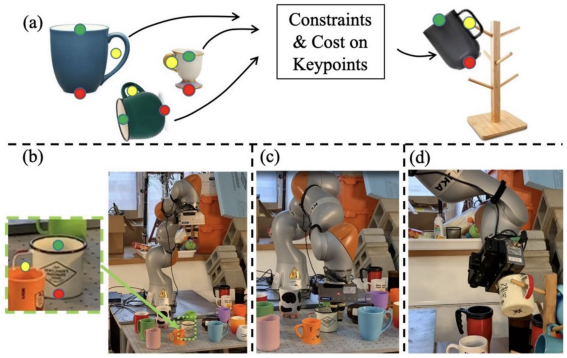


Figure 14.1: The mug and rack task using kPAM. Green, red and yellow points stand for the top, bottom and handle of the mug.

14.1.2 Literature review

Most of the previous methods specify the desired configuration as a target 6-degree-of-freedom (6DOF) pose. The pose of the object is then used for planning. The disadvantages are mainly concentrated on these two points. Firstly, they often fail to capture large intra-category shape variation, for instance, the policy trained on some seen mugs may not be capable on unseen new types of mugs. Secondly, only the knowledge of the pose of the target is not sufficient for completing the task. For example, knowing the pose and size of a

coffee mug relative to some canonical mug is not enough for the robot to successfully hang it on a rack by its handle. A nice demonstration of the two disadvantages can be found in Figure 14.2.

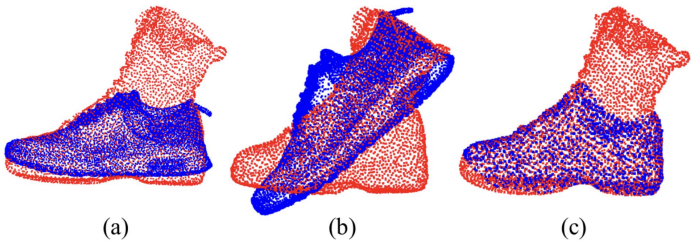


Figure 14.2: Three point clouds of the alignment of different shoes, showing that the previous methods are mostly object-dependent. Given 3D point clouds of some shoes, the robot may not necessarily be able to manipulate if the shoes are of different types. The robot may place the shoe at (a) different poses for a sneaker of a boot and (b) sometimes the misalignment of the point clouds can cause totally different configuration. Finally, (c) shows a legitimate alignment of the point clouds of different shoes.

14.1.3 Methodology

An intuitive way to mark the object is to place semantic 3D key points as the representation. The overall pipeline proposed starts from instance segmentation of the scene, meaning that the target objects are filtered from the background. Next, semantic 3D key points are placed on the segmented objects. This is followed by an optimization-based robot action planning step to generate a manipulation plan. Eventually, the geometric grasping and robot action are sent to execution.

This pipeline provides the user with an interpretable specification of the manipulation, and in the meantime, generalises the existing pose-based manipulation methods. An overview of the manipulation formulation can be found in Figure 14.3.

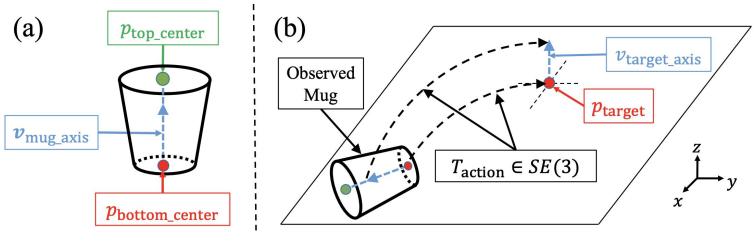


Figure 14.3: An overview of the manipulation formulation using the 'put mugs upright on the table' task as an example: (a) a category-level key point detector is trained to produce two key points: $p_{\text{bottom-center}}$ and $p_{\text{top-center}}$. (b) Given an observed mug, its two key points on bottom center and top center are detected. The rigid transform T_{action}, which represents the robotic pick-and-place action, is solved to move the bottom center of the mug to the target location p_{target} and align the mug axis with the target direction $v_{\text{target-axis}}$.

A category-level pick-and-place pipeline is also introduced in the work. This pipeline is shown in Figure 14.4.

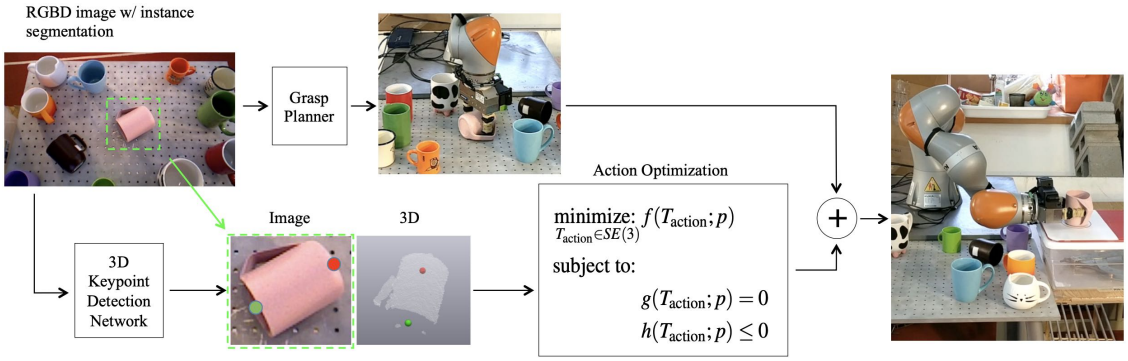


Figure 14.4: An overview of the category-level pick-and-place pipeline using the manipulation formulation. Given an RGBD image with instance segmentation, the semantic 3D key points of the objects in question are detected. We then feed these 3D key points into an optimisation-based planning algorithm to compute the robot pick-and-place actions, which is represented by a rigid transformation T_{action}. Finally, we use an object-agnostic grasp planner to pick up the object and apply the computed robot action.

14.1.4 Experimental results

The author did experiments with manipulation of shoes and mugs. Illustrative graphs are shown in Figure 14.5 and quantitative results are listed in Figure 14.6. The video of the experiments can be found ***here***.

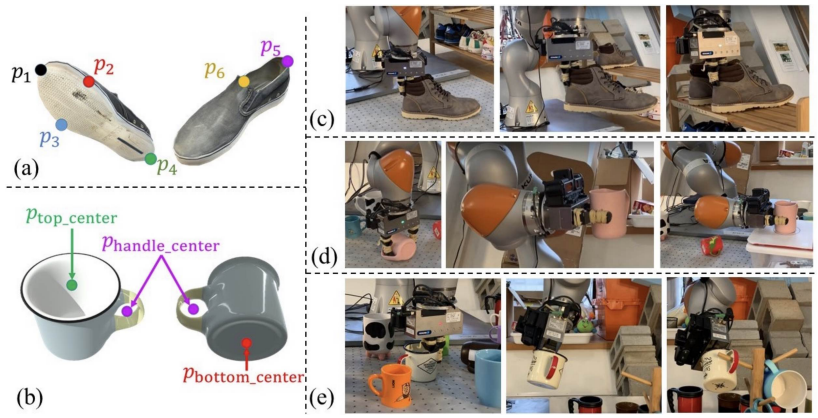


Figure 14.5: An overview of the experiments. (a) and (b) are the semantic key points used for the manipulation of shoes and mugs. Three manipulation tasks are used to evaluate the proposed pipeline: (c) put shoes on a shelf; (d) put mugs on a mug shelf; (e) hang mugs on a rack by the mug handles.

(a) Test Shoes

(b) Test Mugs

Object Type	# train objects	# scenes	# images
Shoe	10	43	39,403
Mug	21	74	70,094

(c) Training dataset statistics

# test objects	# Trials	Placed on shelf	Heel Error (cm)	Toe Error (cm)
20	100	98%	1.09 ± (1.29)	4.34 ± (3.05)

(d) Shoes on Rack

Initial Orientation	# test objects	# Trials	Placed upright on shelf	< 3cm error	< 5cm error
Upright	40	80	100%	97.5%	100%
Horizontal	19	38	97.3%	89.4%	94.7%

(e) Mugs on Shelf

Mug Size	# test objects	# Trials	Success Rate
Regular	25	100	100%
Small	5	20	50%

(f) Mugs on Rack

Figure 14.6: Quantitative results from the 3 hardware experiments. (a) and (b) show some of the test objects for the experiments. (c) statistics of the training data (d) We report the average heel and toe errors (along the horizontal direction) from their desired locations as well as the standard deviation. (e) The reported errors for the mug on shelf task are the distance from the bottom center key point to the target location of that key point in the optimization program. (f) reports success rates for the mug on rack task for different sized mugs. Mugs with handles having either height or width less than 2cm are classified as small (more details in supplementary material). A trial was deemed successful if the mug ended up hanging on the rack by the mug handle.

14.2 Keypoints into the Future[5]

How does this paper relate to the previous paper?

- While supervised learning is used for detecting the key points in the previous work, where as in [5] it is self supervised.
- In the previous paper, the task is completed by manually specifying the target keypoints and solving an optimization problem. In [5], a prediction model is learned and a model predictive control is to follow a demonstrated task .

14.2.1 An Overview

Step one, is to learn a semantically meaningful object keypoints, for instance labeled blue. (see 14.7). The next step is to collect random interaction data between the robot and the object. This interaction is

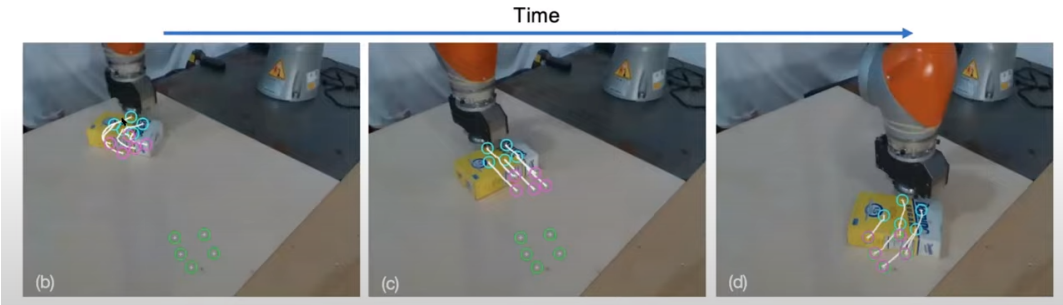


Figure 14.7: Shows semantically meaningful object keypoints. Blue keypoints refer to the initial pose and green refer to the final goal pose.

recorded as training data. A prediction model is learned using the object's key points and the robot's action; for instance, the robot moving the object from one point to another, as shown in the Figure 14.8. i.e., a forward prediction model is learned on the keypoints.

Figure 14.8: Random interactions between the robot and the object to collect the training data.

—

Prof. Malik: Is this similar to Pulik's work on learn to poke by poking[1] but at the level of keypoints?

Yes, but without an inverse model.

—

Finally, A single demonstration is used to specify the task. During testing, the learned model uses the model predictive control to achieve the goal state (where the demonstration defines the goal state.)

14.2.2 How to get keypoints?

How are the keypoints generated, since it is not supervised? Note, this process is not specified in [5] but from the previous literature [3] of the same authors.

- Build a dense visual correspondence model (dense SIFT)
- Extract "reliable" and spatially separated" sparse keypoints.

The pipeline is similar to traditional computer vision, where SIFT features exist for the pixels and extracting meaningful keypoints from these SIFT features. Here, a learning-based method is used instead of SIFT.

14.2.3 The Dense Correspondence Model

Multiple RGB-D cameras, with known extrinsic and intrinsic camera matrices, are used to capture different objects' images from different angles. The images and the depth information are used to construct a mesh model for the objects. This process helps in identifying the pixel's location on the 3D mesh. Contrastive

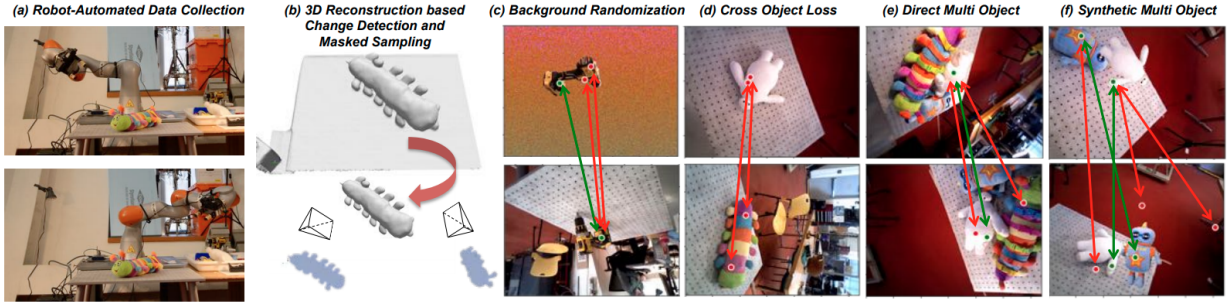


Figure 14.9: The Dense Correspondence Model. (a) Multiple RGBD images at various angles are captured. (b) 3D mesh of the object is reconstructed and (c) Contrastive Learning where the loss function decreases the loss between matching points and maximizes the loss between non-matching points.

learning applies a convolution network on a pair of images to reduce the loss function,

$$\begin{aligned} \mathcal{L}_{\text{matches}}\ (I_a, I_b) &= \frac{1}{N_{\text{matches}}} \sum_{N_{\text{mathes}}} D\left(I_a, u_a, I_b, u_b\right)^2 \\ \mathcal{L}_{\text{non-matches}}\ (I_a, I_b) &= \frac{1}{N_{\text{non-matches}}} \sum_{N_{\text{non-mathhthes}}} \max\left(0, M - D\left(I_a, u_a, I_b, u_b\right)\right)^2 \\ \mathcal{L}\left(I_a, I_b\right) &= \mathcal{L}_{\text{matches}}\ (I_a, I_b) + \mathcal{L}_{\text{non-matches}}\ (I_a, I_b) \end{aligned} \tag{14.1}$$

where the loss function $\mathcal{L}_{\text{matches}}$ reduces the distances between the matches and maximizes $\mathcal{L}_{\text{non-matches}}$ the distance between non-matching points. Finally, the learned object descriptor can be visualized as shown in Figure 14.10. To visualize, the feature is projected to a 3D dimensional space, if the pixels are in the same color, they are the same points in the 3D space.

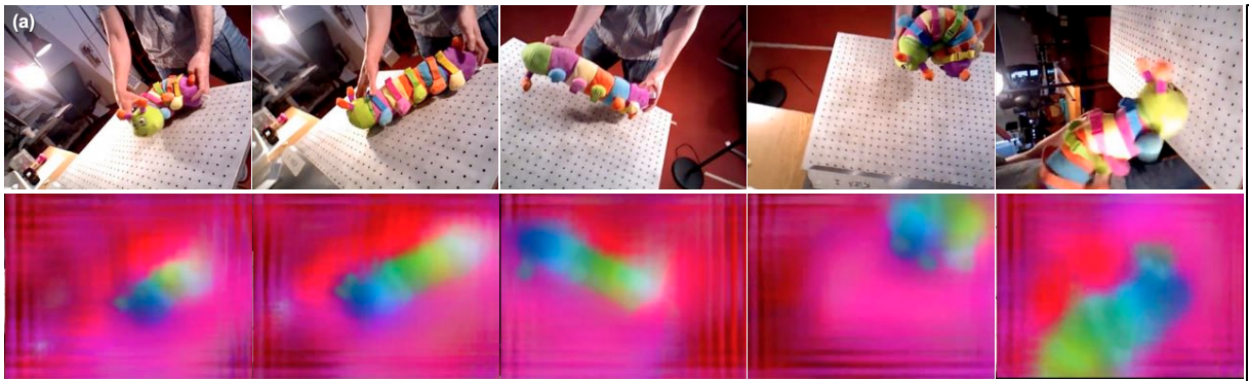


Figure 14.10: Learned object descriptor for the Contrastive Learning, and the descriptor are consistent across significant deformation

Prof. Malik: Why chose keypoints as the representation of an object? Traditionally, keypoints were used in computer vision to solve the correspondence problem. We have the same object from different views, say for structure from motion or stereo, we need to have correspondence and then triangulate. We need to have a way of specifying the pose of the object, and one way of doing it is through these keypoints. However, it is just one choice, but a more fundamental way of representing it is, say, a mesh representation of an object, and the mesh has a canonical coordinate system. Therefore every vertex on the mesh has a specific physical meaning on the object; if we have that, why go the keypoints. Keypoints are a way to represent the mesh, but the mesh is a complete representation or alternately 6DOF pose for minimal representation. Regarding keypoints as an end-goal is strange because it is neither a minimal representation nor a complete representation.
Objects in an image can be detected with varying degrees of information. A bounding box around the object is the simplest method of detecting an object but does convey enough information as it does not capture the orientation of the object. The next level of sophistication is segmenting the object, i.e., identifying the pixel corresponding to the object, but this method will not suffice because it still does not capture the orientation. Thus, a higher level of identification is required, which enables us to estimate the orientation/pose of the object from the image. Thus, the choice of mesh, keypoints, or point cloud comes into play to represent the structure.
Question: Would the keypoints be the minimal representation need to account for the deformation in the objects?
Yes, for deformable objects, keypoints might be a good way of representing the objects.
Prof. Mailk: To given an extremely deformable object, consider the human body. The usual keypoints would be the various joints, left shoulder, right elbow, etc., which is not considered a complete representation of a human body. Mesh is a better(full) representation; for instance, is ¡¡smpl¿¿ representation, Angjoo Kanazawa [2] has work done on this. That is like a canonical representation of the entire body shape, at the mesh level, where every vertex has a meaning corresponding to the human body. Of course, smpl is a specialized case for human bodies. However, for simpler objects keypoints can be an approximation.
Question: Is it possible to learn the low-level representation, such as the keypoints, instead of the representation itself?

14.2.4 Keypoints extraction

- Given the dense image descriptor, how to extract meaningful keypoints?
- From the interaction data, get spatially spread yet confident matches.

Figure 14.11: Visualization of the learned visual-correspondence model on a reference image (left) and target image(right). Colored numbers in the target image represent the probability that the detected correspondence is valid. Green reticle shows a valid correspondence with a high confidence score, red reticle shows a case where no correspondence exists in the target image due to occlusion, hence the low confidence probability. Confidence heatmap shown in bottom right.

The correspondence model is essentially the feature representation for each of the pixels. Different points on the object are randomly sampled, and each of the sampled objects is compared to other objects in the training set and run a visual correspondence model as shown in the Figure. Similarity values and confidence levels are used to compare the query image and the training set for occluded feature points. Finally, the following conditions are observed when extracting the keypoints,

- ensuring that keypoints are not occluded
- points are randomly sampled
- keypoints are not clustered together.

14.2.5 Learning the forward model

- A Neural network with supervised learning
- Given input image, extract keypoints

Once the feature points are extracted, interaction between the robot and the object is considered at each item step. A neural network is model is constructed with inputs consisting of the keypoint locations z_t and the robot actions a_t to predicted the keypoint location at the next time step, as shown in the equation below,

$$\hat{\boldsymbol{z}}_{t+1} = \hat{f}_{\theta_{dyn}}\left(\boldsymbol{z}_{t-l:t}, \boldsymbol{a}_t\right). \tag{14.2}$$

A network with the following loss function to minimize the distance between the predicted and ground truth keypoints locations is considered,

$$\mathcal{L}_{\text{dynamics}} = \sum_{h=1}^{H} \|\hat{z}_{t+h} - \boldsymbol{z}_{t+h}\|_2^2, \quad \hat{\boldsymbol{z}}_{t+h+1} = \hat{f}_{\theta_{\text{dyn}}}\left(\hat{\boldsymbol{z}}_{t-l:t}, \boldsymbol{a}_{t+h}\right), \quad \hat{\boldsymbol{z}}_t = \boldsymbol{z}_t. \tag{14.3}$$

A trajectory specified by the keypoint movement is given, a model predictive control and cross entropy method are used to reach the goal state using online planning.

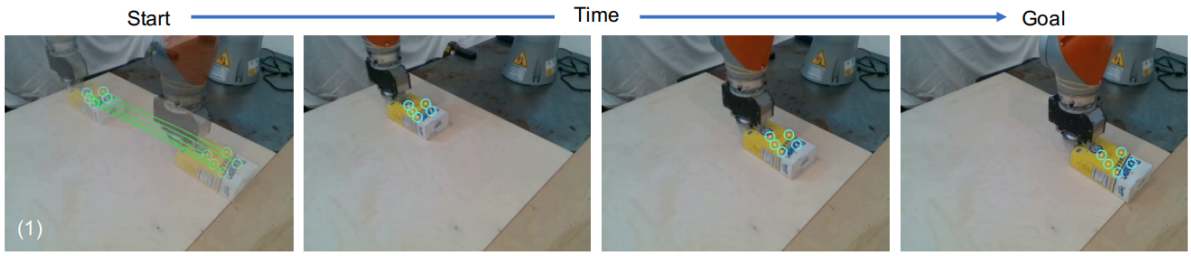
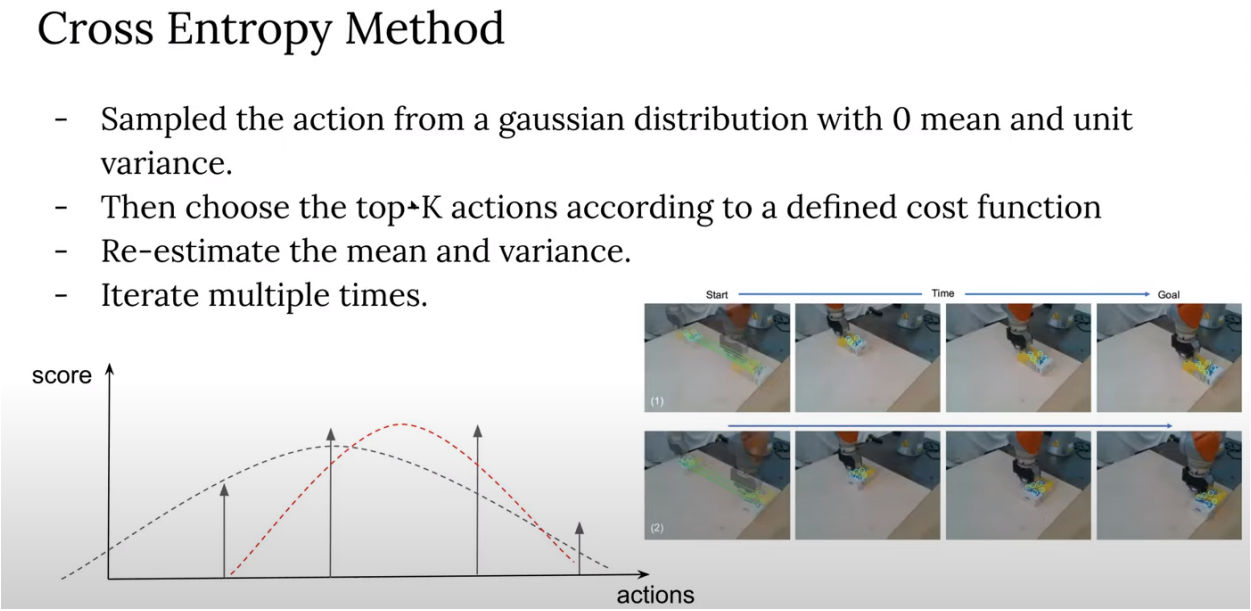


Figure 14.12: The demonstration trajectory used for the hardware experiments. The left image of each row shows the starting position blended with the goal position. The SDS keypoints are shown in teal for each frame. Thegreen lines show the paths followed by the keypoints moving from the starting position to the final position. The right image of each row shows the final/goal position.

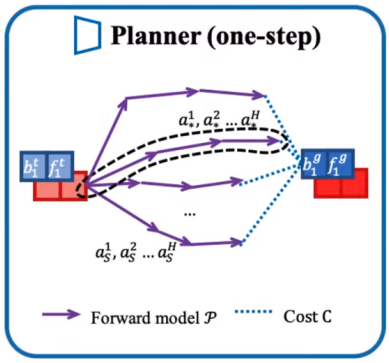
14.2.6 Cross Entropy Method



14.3 MPC Planning

MPC Planning

- Long term prediction is not accurate
- Do multi-step forward, estimate cost
- Only take one step action
- Re-planning



MPC is further discussed in the later sections.

14.3.1 Results

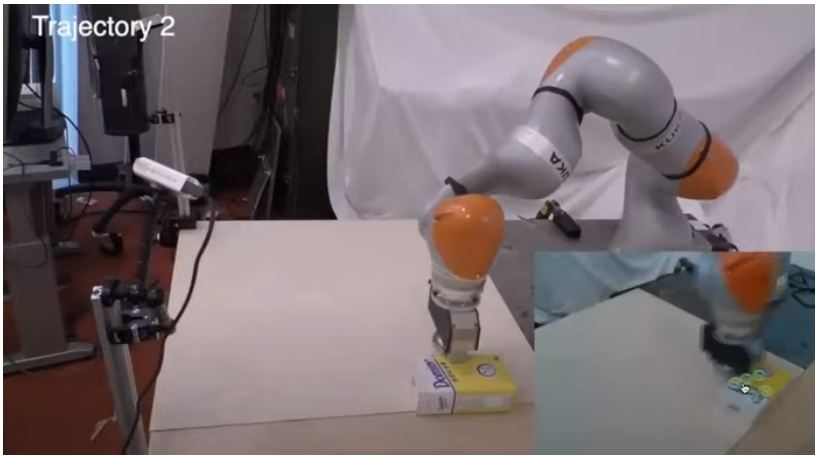


Figure 14.13: As shown in the bottom right is the demonstration to specify the task and the trajectory is executed by the robot.

Prof Ma: It is very much based on sample paths to do sort of MPC. How do they compute the cost? Do they back-propagate the end points?
Task is defined by a demonstration and the trajectory is recorded at each time point and they optimize on this.
Prof. Ma: Do they need to record the pre executed trajectory?
Each task needs to be specified and in the process few demonstrations are recorded. 10 minutes is the training required between the robot and the object.

14.4 Model Predictive Control[8]

14.4.1 Introduction to MPC

Model Predictive Control (MPC) is to solve an optimal control problem with input and state constraints with system dynamics. A typical MPC, as shown in Fig. 14.14, is designed to solve repeatedly online: the solution of MPC provides states and input trajectories with multi-step preview under constraints and only exerts the first step input, and MPC replans using the same schematic by using the measured states to get the next control input. The challenge of MPC is to solve the MPC online with high-frequent replans, which is a big burden for computing resources. And there are some successful experiments on drones to using the MPC.

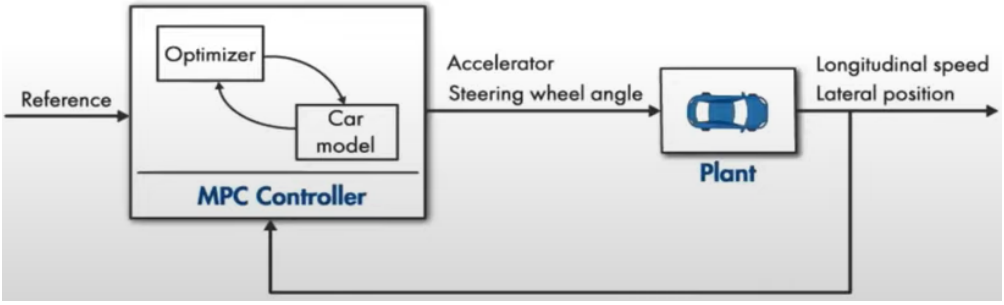


Figure 14.14: A MPC for longitude speed and lateral position regulation for an autonomous car. The MPC can be solved for a trajectory with different horizon, and the car only exerts the very first input. For the next control step, the states of the car is updated, and the MPC with the same formulated is solved again, and keeps this process.

The MPC is also known as receding horizon control. And for the state estimation, a state estimator, such as Kalman Filter (KF) for a linear system, Extended KF and particle filter for nonlinear system, or combining the learning-based preception such as keypoint extraction, can be applied. In order to tune the MPC, adjusting the predictive horizon and cost could affect the MPC results.

14.4.2 Model-Based Reinforcement Learning

Recent work on Model-Based Reinforcement Learning (MBRL) which typically learns the dynamics model can be cooperated with MPC as illustrated in Fig. 14.15, where the policy to execute is a MPC and transition is the learned dynamics model in the Markov chain. This method can be also referred to Learning MPC (LMPC) [7].

A modified MBRL is to do backpropagte through the learned dynamics model into the policy to optimize the policy, and applies the policy output and gets the next states, and do the process to collect state transition pair and update the learned dynamics model. An evolution of MBRL is:

- v0.5 collect random samples, train dynamics, and plan. The pro is that this is simple without iterating procedure, while may suffers from distribution problem.
- v1.0 iteratively collect data, replan, collect data. The pro is simple and is able to solve distribution mismatch, while the open-loop plan may perform poorly.
- v1.5 iteratively collect data using MPC which replans at each step. The pro is robust to small modelling errors while this method is computationally expansive and requires an existing MPC.

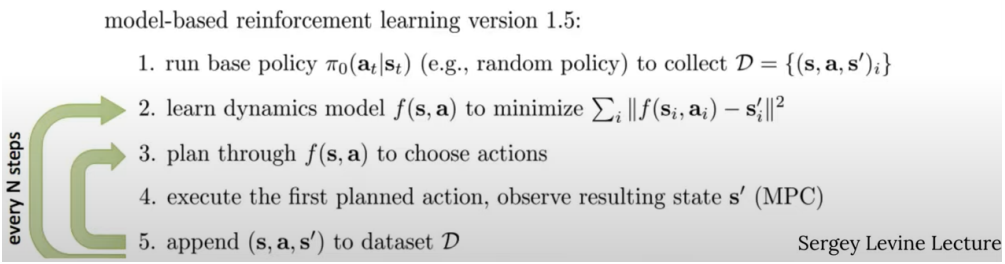


Figure 14.15: A model-based reinforcement learning framework that can be used with MPC. The dynamics model of the system is obtained by learning the collected observed states and predicted states, and the dynamics model can be represented by a linear model or a simple neural network. MPC can be used to obtain the action at the current states, and collect a transition pair and repeat the process.

v2.0 backpropagate directly into policy. The pro is computationally cheap during the runtime while is numerically unstable.

14.4.3 Pros and Cons of MPC

In conclusion, compared to MBRL, the MPC is good at being a flexible general technique and being able to handle nonlinear models, approximate models, and explicit constraints and soft penalties, while having preview capability.

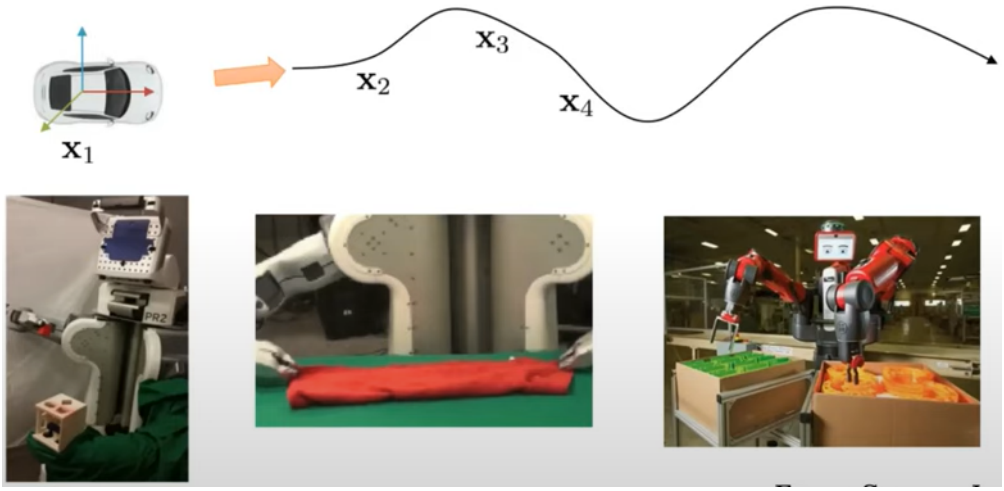


Figure 14.16: The Weakness of MPC. MPC is not good at the manipulation tasks where the robot needs to pick up soft mats or fruits. This is because that soft contact is hard to model and this may cause issue to MPC because there is a huge modeling error.

However, MPC needs to solve an optimization problem online, which takes time. Moreover, if MPC needs to run on a embedded device, it should also consider the memory limitations on hardware. There is a trade-off between the accuracy of MPC and solving time and memory budget. If MPC needs to replan fast, the model used should be simple, such as a linear model, and with less horizon to have less memory. But a linear system could not represent the full-dynamics of a general nonlinear system, and reducing horizon loses the preview properties. People can use a complex nonlinear model to solve in MPC, but this will increase the

computing time and consumed memory dramatically and may cause the MPC computationally infeasible online. And the target state is encoded by adding a terminal cost not explicit constraints in order to have feasibility of the optimization. Moreover, MPC requires a dynamics model, but having a precise model is hard. Although MPC is able to solve a certain range of unknown dynamics, such as an adaptive MPC, but could increase the design complexity. It is also challenging to use MPC to deal with manipulation tasks with soft contacts as shown in Fig. 14.16.

There are also several ways to improve MPC, such as to 1) truncate time horizon, 2) truncate reduced state space, 3) using adaptive MPC to deal with modeling error, 4) to pre-solve the problem in offline regime and formulate a look-up table for online control policy, which is refer to Gain scheduled MPC or Explicit MPC, 5) solve the optimization with a warm start by using proper initial guess, 6) to use parameterized MPC as introduced in the following section.

14.5 Parametrized Model Predictive Control

14.5.1 Standard MPC Overview

Standard model predictive control is widely used in industry, but it is hard to solve efficiently since it has lots of decision variables, and the number of decision variables scales with prediction horizons. There are some ways to alleviate this issue, such as truncated time horizon, truncated state space, adaptive MPC, full offline presolve, warm start, and parametrized MPC, etc.

14.5.2 Parametrized MPC Formulation

Today, in order to solve the above challenges, we will introduce parametrized model predictive control which simplifies the computational cost [8]. At each sampling time, parametrized MPC will

1. learn a parametric description of input and state trajectories,
2. solve an approximate optimization problem for an infinite horizon,
3. apply predicted input to the system for the duration of the sampling interval.

Before diving into the trajectory tracking problem, we first start with a simplified stabilizing problem. An infinite-horizon stabilizing model predictive control formulation is given below

$$J_\infty := \inf \int_0^\infty l(x(t), u(t), t)dt \tag{14.4}$$
$$\dot{x}(t) = Ax(t) + Bu(t) \tag{14.5}$$
$$x(0) = x_0 \tag{14.6}$$
$$x(t) \in \mathcal{X} \tag{14.7}$$
$$u(t) \in \mathcal{U} \tag{14.8}$$

Here, we have an infinite horizon cost J_∞ for some loss function $l(x(t), u(t), t)$ which is assumed to be convex and differentiable (14.4). We consider linear time invariant (LTI) system (14.5). Initial condition is given in (14.6), and state and input constraints are defined in (14.7) and (14.8).

The goal of parametrization of input and state trajectories is to approximate $u(t)$ and $x(t)$ as linear combinations of some basis functions $\tau(t) \in L^2([0, \infty), \mathbb{R}^s)$. $\tilde{u}$ is a linear combination of some basis functions parametrized by η_u. Similarly, $\tilde{x}$ is a linear combination of some basis functions parametrized by η_x.

$$\tilde{u}(t) := (I_m \otimes \tau(t))^T \eta_u, \quad \eta_u \in \mathbb{R}^{ms}, \tag{14.9}$$
$$\tilde{x}(t) := (I_n \otimes \tau(t))^T \eta_x, \quad \eta_x \in \mathbb{R}^{ns}, \tag{14.10}$$

where m is the input dimension, and n is the state dimension. s represents the number of basis functions.

The basis functions are defined as follows

$$\tau_i(t) \in e^{\lambda t}\text{span}\{1, t, \ldots, t^{s-1}\}. \tag{14.11}$$

We would like these basis functions to have some properties: 1) these basis functions are orthonormal 2) $\dot{\tau}(t) = M_\lambda \tau(t)$. The second property is actually a time-shift property, which is the key to prove recursive feasibility and stability. Using these basis functions, we can reformulate the dynamics as

$$\dot{\tilde{x}}(t) = A\tilde{x}(t) + B\tilde{u}(t). \tag{14.12}$$

This dynamics expression has τ inside, however, we want the reformulated dynamics only parametrized by η. Through a detour to the calculus of variation, we can get the following dynamics

$$(A \otimes I_s - I_n \otimes M_\lambda^T)\eta_x + (B \otimes I_s)\eta_u = 0. \tag{14.13}$$

After presenting the reformulation of state, input, and dynamics, we have the parametrized MPC formulation as follows

$$J_s := \min \int_0^\infty l((I_n \otimes \tau(t))^T \eta_x, (I_m \otimes \tau(t))^T \eta_u, t)dt \tag{14.14}$$

$$(A \otimes U_\tau - I_n \otimes U_\tau M_\lambda^T)\eta_x + (B \otimes U_\tau)\eta_u = 0 \tag{14.15}$$

$$(I_n \otimes \tau(0))^T \eta_x = x_0 \tag{14.16}$$

$$(I_n \otimes \tau(t))^T \eta_x \in \mathcal{X} \tag{14.17}$$

$$(I_m \otimes \tau(t))^T \eta_u \in \mathcal{U} \tag{14.18}$$

If we use a larger set of basis function, we can get a lower infinite horizon cost

$$J_s \geq J_{s+1} \geq \cdots \geq J_\infty. \tag{14.19}$$

14.5.3 Results

This paper considers a task that is balancing a pole on a quadrotor while tracking a figure 8 trajectory. There are a few added difficulties, e.g., disturbances, unmodeled dynamics. The formulation using parametrized MPC is

$$\tilde{x}(t) := (I_n \otimes \tau(t))^T \eta_x + x_{\text{ss}}, \tag{14.20}$$

$$\tilde{x}_{\text{ref}}(t) := (I_n \otimes \tau(t))^T \eta_{x,\text{ref}} + x_{\text{ss}}, \tag{14.21}$$

$$\tilde{u}(t) := (I_m \otimes \tau(t))^T \eta_u + u_{\text{ss}}, \tag{14.22}$$

$$\tilde{u}_{\text{ref}}(t) := (I_m \otimes \tau(t))^T \eta_{u,\text{ref}} + u_{\text{ss}}, \tag{14.23}$$

$$\tilde{v}(t) := (I_v \otimes \tau(t))^T \eta_v + v_{\text{ss}}. \tag{14.24}$$

where x_{ss} and u_{ss} are reference trajectory and input, and $\tilde{v}(t)$ represents disturbance.

The trajectory generation result is shown in Fig. 14.17. In order to improve trajectory approximation, the proposed method replaces single approximation by N piecewise approximation. The influence of multiple intervals in the trajectory generation is shown in Fig. 14.18. The parametrized MPC can also handle disturbance well. Fig. 14.19 shows how a single disturbance's component is estimated and approximated, and Fig. 14.20 describes the trajectory tracking performance with disturbance estimates.

14.5.4 Strength and Limitations of Parametrized MPC

Having introduced the parametrized MPC formulation and presented the results, we now summarize the strength and limitations of the parametrized model predictive control.

The strength of parametrized MPC is as follows [6, 9]

1. Parametrized MPC has recursive feasibility guarantee.
2. Parametrized MPC has inherent stability guarantee. There is no need for terminal cost / terminal state constraints.

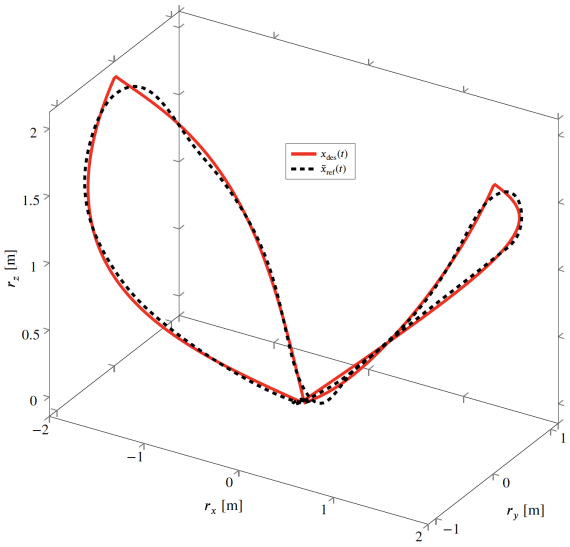


Figure 14.17: Result of the trajectory generation procedure. A three-dimensional figure-eight (in solid red), with sharp edges, is approximated by a feasible trajectory (in dashed black) that is parametrized with basis functions and steady-state offsets.

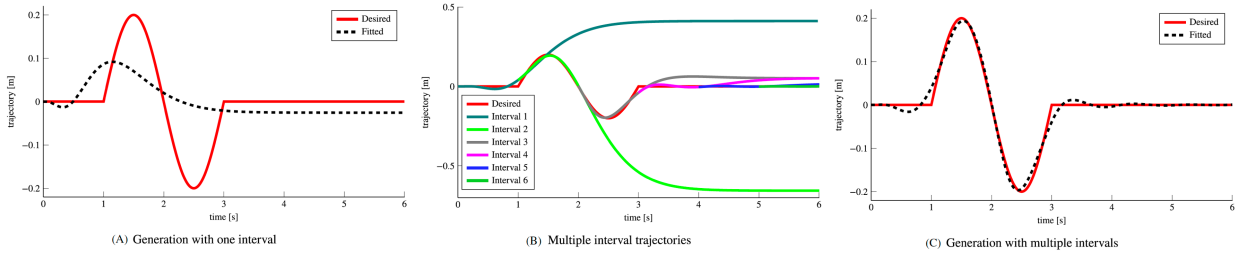
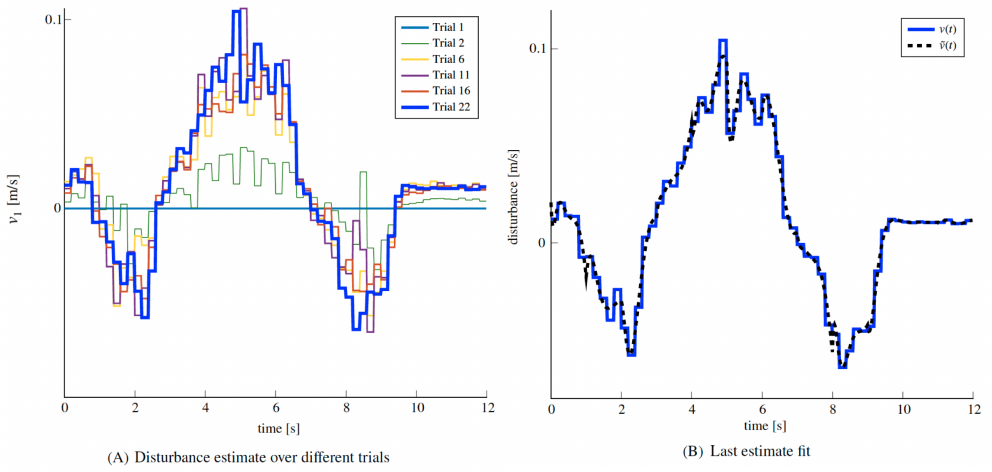


Figure 14.18: Influence of multiple intervals in the trajectory generation procedure. Because of the limited representation power of the basis functions, it is not possible to fit arbitrary long trajectories, see (A). To overcome this problem, it is proposed to split the trajectory over multiple intervals, with different parametrizations for each of these, see (B). Merging the different intervals trajectories results in a more accurate fit, as shown in (C)

3. Parametrized MPC leads to an optimization problem with typically fewer variables compared to the discrete-time finite-horizon description of standard MPC.

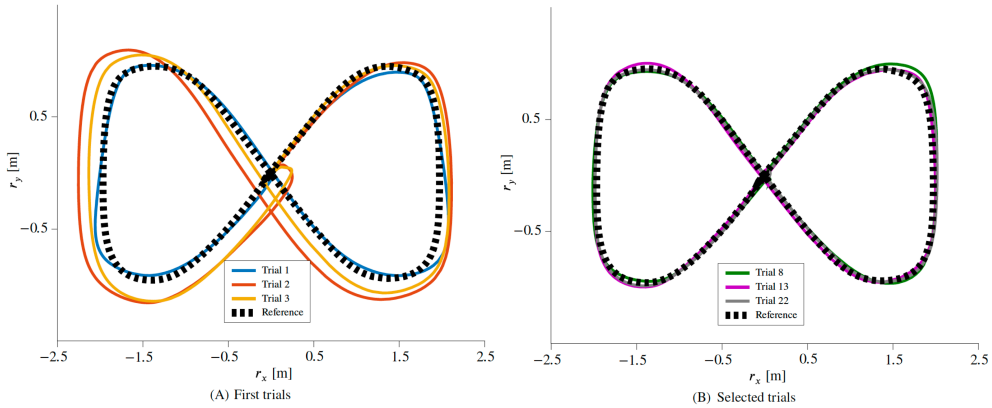
The limitations of parametrized MPC is given below

1. All derivations so far only work with LTI dynamics.
2. There is no formal analysis of approximation quality for input and state trajectories.



(A) Disturbance estimate over different trials (B) Last estimate fit

Figure 14.19: The above plots show how a single disturbances component is estimated and approximated. In (A), the first disturbance vectors component is shown over different trials. The plot (A), where the last estimate is shown in blue (thick), indicates convergence of the disturbance trajectory. In (B), the last estimate is compared with the approximation (parametrized by the basis functions) obtained from the trajectory generation procedure.



(A) First trials (B) Selected trials

Figure 14.20: Trajectory tracking performance. The feasible reference trajectory (in dashed black) is tracked for 22 subsequent trials. The disturbance estimate improves over the trials, yielding better tracking performance. The first three trials are shown in (A), while some of the other trials are shown in (B).

References

[1] P. Agrawal, A. Nair, P. Abbeel, J. Malik, and S. Levine, "Learning to poke by poking: experiential learning of intuitive physics," in *Proceedings of the 30th International Conference on Neural Information Processing Systems*, 2016, pp. 5092–5100.

[2] F. Bogo, A. Kanazawa, C. Lassner, P. Gehler, J. Romero, and M. J. Black, "Keep it smpl: Automatic estimation of 3d human pose and shape from a single image," in *European conference on computer vision*. Springer, 2016, pp. 561–578.

[3] P. R. Florence, L. Manuelli, and R. Tedrake, "Dense object nets: Learning dense visual object descriptors by and for robotic manipulation," *arXiv preprint arXiv:1806.08756*, 2018.

[4] L. Manuelli, W. Gao, P. Florence, and R. Tedrake, "kpam: Keypoint affordances for category-level robotic manipulation," *arXiv preprint arXiv:1903.06684*, 2019.

[5] L. Manuelli, Y. Li, P. Florence, and R. Tedrake, "Keypoints into the future: Self-supervised correspondence in model-based reinforcement learning," *arXiv preprint arXiv:2009.05085*, 2020.

[6] M. Muehlebach and R. D'Andrea, "Parametrized infinite-horizon model predictive control for linear time-invariant systems with input and state constraints," in *2016 American Control Conference (ACC)*. IEEE, 2016, pp. 2669–2674.

[7] U. Rosolia and F. Borrelli, "Learning model predictive control for iterative tasks. a data-driven control framework," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 1883–1896, 2017.

[8] C. Sferrazza, M. Muehlebach, and R. D Andrea, "Learning-based parametrized model predictive control for trajectory tracking," *Optimal Control Applications and Methods*, vol. 41, no. 6, pp. 2225–2249, 2020.

[9] C. Sferrazza, M. Muehlebach, and R. D'Andrea, "Trajectory tracking and iterative learning on an unmanned aerial vehicle using parametrized model predictive control," in *2017 IEEE 56th Annual Conference on Decision and Control (CDC)*. IEEE, 2017, pp. 5186–5192.