Chance Contraint Update

Drone Model

The system equations of the drone are given as a combination of the nominal model and Gaussian process (GP) model:

$$f(\mathbf{x}, \mathbf{u}) = f_n(\mathbf{x}, \mathbf{u}) + f_{\mathrm{GP}}(\mathbf{z}) \tag{1}$$

where $f_{\mathrm{GP}}(\mathbf{z}) \sim \mathcal{N}(\boldsymbol{\mu}(\mathbf{z}), \boldsymbol{\Sigma}(\mathbf{z}))$ is the GP model capturing the wind dynamics with mean $\boldsymbol{\mu}(\mathbf{z})$ and covariance $\boldsymbol{\Sigma}(\mathbf{z})$. The feature vector $\mathbf{z}$ is selected from the state vector $\mathbf{x}$. In this case, as a wind disturbance map is trained, the feature vector is given by the position of the quadrotor, such that $\mathbf{z} = [x, y]^T$.

Nominal Model

The nonlinear nominal drone model can be derived from Newton-Euler equations of motion (EOM) and is summariyed below:

$$\dot{\mathbf{p}} = \mathbf{v} \tag{2a}$$
$$m\dot{v}_x = T_d\left(\sin(\psi)\sin(\phi) + \cos(\phi)\sin(\theta)\cos(\psi)\right) - k_D v_x \tag{2b}$$
$$m\dot{v}_y = T_d\left(-\sin(\phi)\cos(\psi) + \cos(\phi)\sin(\theta)\sin(\psi)\right) - k_D v_y \tag{2c}$$
$$m\dot{v}_z = T_d\left(\cos(\phi)\cos(\theta)\right) - mg - k_D v_z \tag{2d}$$
$$\dot{\phi} = \frac{1}{\tau_\phi}(k_\phi \phi_d - \phi) \tag{2e}$$
$$\dot{\theta} = \frac{1}{\tau_\theta}(k_\theta \theta_d - \theta) \tag{2f}$$
$$\dot{\psi} = \dot{\psi}_d \tag{2g}$$

The quadrotor dynamics model can be linearized around its hovering condition with $\dot{v}_z = 0$. Defining the state vector $\mathbf{x} = \left[\mathbf{p}^T, \mathbf{v}^T, \phi, \theta, \psi\right]^T$ and the input vector $\mathbf{u} = \left[\phi_d, \theta_d, \dot{\psi}_d, T_d\right]^T$ the linearized system can be written as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{v_x} \\ \dot{v_y} \\ \dot{v_z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -k_D* & 0 & 0 & 0 & g & 0 \\ 0 & 0 & 0 & 0 & -k_D* & 0 & -g & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -k_D* & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\phi} & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\theta} & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -\frac{1}{\tau_\psi} \end{bmatrix} \begin{bmatrix} x \\ y \\ z \\ v_x \\ v_y \\ v_z \\ \phi \\ \theta \\ \psi \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ \frac{k_\phi}{\tau_\phi} & 0 & 0 & 0 \\ 0 & \frac{k_\theta}{\tau_\theta} & 0 & 0 \\ 0 & 0 & \frac{k_\psi}{\tau_\psi} & 0 \end{bmatrix} \begin{bmatrix} \phi_d \\ \theta_d \\ \psi_d \\ T_d \end{bmatrix} \tag{3}$$

with the mass-normalized drag coefficient $k_D* = \frac{k_D}{m}$.

Gaussian Process Model

To model wind disturbances two GPs are trained, one GP for the wind disturbance in x-direction and one for the wind disturbance in y-direction. This is assuming that the two outputs, i.e. the two wind disturbances, of each GP are independent of each other.

As the wind disturbance is acting as a force on the quadrotor, this results in a change of speed in both x- and y- direction. As the wind disturbance cannot directly be measured, it is be derived from the difference of the prediction of the velocity at the next state and the actual measured velocity of the next state:

$$\begin{aligned} d_{v_x} &= \frac{\hat{v}_x - v_x}{\Delta T} \\ d_{v_y} &= \frac{\hat{v}_y - v_y}{\Delta T} \end{aligned} \tag{4}$$

The GPs are then trained with the input being the position of the quadrotor $\mathbf{z} = [x, y]^T$ and the output being the measured wind disturbances d_{v_x} and d_{v_y}. The resulting disturbance vector, containing the two trained GP, is the following:

$$\mathbf{d}(\mathbf{z}) = \begin{bmatrix} d_{v_x}(\mathbf{z}) \sim \mathcal{N}\left(\mu^{d_{v_x}}(\mathbf{z}), \Sigma^{d_{v_x}}(\mathbf{z})\right) \\ d_{v_y}(\mathbf{z}) \sim \mathcal{N}\left(\mu^{d_{v_y}}(\mathbf{z}), \Sigma^{d_{v_y}}(\mathbf{z})\right) \end{bmatrix} \tag{5}$$

To map the disturbance vector to the correct states, we define:

$$B_d = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 1 & 0 \\ 0 & 1 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix}. \tag{6}$$

The resulting GP model is:

$$f_{\text{GP}}(\mathbf{z}) = B_d \mathbf{d}(\mathbf{z}) \tag{7}$$

Local Model Predictive Contouring Control

The extended drone model is incorporated into a local model predictive contouring controller (LMPCC) as described here. The LMPCC formulation is the following:

$$J^* = \min_{\boldsymbol{x}_{0:N}, \boldsymbol{u}_{0:N-1}, \theta_{0:N-1}} \sum_{k=0}^{N-1} J(\boldsymbol{x}_k, \boldsymbol{u}_k, \theta_k) + J(\boldsymbol{x}_N, \theta_N)$$

$$\text{s.t.} \quad \boldsymbol{x}_{k+1} = f(\boldsymbol{x}_k, \boldsymbol{u}_k),\ \theta_{k+1} = \theta_k + v_k \tau \tag{8}$$

$$\mathcal{B}(\boldsymbol{x}_k) \cap \left(\mathcal{O}^{\text{static}} \cup \mathcal{O}_k^{\text{dyn}}\right) = \emptyset \tag{9}$$

$$\boldsymbol{u}_k \in \mathcal{U},\ \boldsymbol{x}_k \in \mathcal{X}\ \boldsymbol{x}_0, \theta_0 \text{ given.} \tag{10}$$

State and uncertainty propagation

Due to the representation of the nonlinear disturbance map by a Gaussian process, the predicted states are given as stochastic distributions. Evaluating the posterior of the GP at the next uncertain state input, however, is computationally intractable. Instead, the posterior distribution is approximated by a Gaussian distribution, i.e.

$$x_{k+1} \sim \mathcal{N}\left(\mu_{k+1}^x, \Sigma_{k+1}^x\right) \tag{11}$$

with approximate inference using linearization. For the update of the state mean this simply results in:

$$\mu_{k+1}^x = f(\mu_k^x, u_k) + B_d \mu_k^d(\mu_k^z) \tag{12}$$

For the propagation of the uncertainty, the first-order taylor approximation of the next state estimate is used, which results in the update equation for the uncertainty:

$$\begin{aligned} \mu_{k+1}^x &= f(\mu_k^x, u_k) + B_d \mu_k^d(\mu_k^z) \\ \Sigma_{k+1}^x &= [AB_d] \begin{bmatrix} \Sigma_k^x & \Sigma_k^{xd} \\ \Sigma_k^{dx} & \Sigma_k^d \end{bmatrix} [AB_d]^T. \end{aligned} \tag{13}$$

where A is the system matrix of the linearized quadrotor dynamics and $\nabla \mu_k^d\left(\mu_k^x, u_k\right)$ is the derivative of the GP with respect to it's input state vector evaluated at the mean state vector.

Cost

The stage cost of the LMPCC is

$$J\left(\mathbf{z}_k, \mathbf{u}_k, \theta_k\right) := J_{\text{tracking}}\ \left(\boldsymbol{x}_k, \theta_k\right) + J_{\text{speed}}\ \left(\boldsymbol{x}_k, \boldsymbol{u}_k\right) + J_{\text{input}}\ \left(\boldsymbol{u}_k\right). \tag{14}$$

For tracking of the reference path, a contour and lag error are defined and combined in an error vector: $\boldsymbol{e}_k := \left[\tilde{\epsilon}^c\left(\boldsymbol{x}_k, \theta_k\right), \tilde{\epsilon}^l\left(\boldsymbol{x}_k, \theta_k\right)\right]^{\mathrm{T}}$. These define the distance to the reference path, given the progress along the path θ_k. The tracking cost is:

$$J_{\text{tracking}}(\boldsymbol{x}_k, \theta_k) = \boldsymbol{e}_k^T Q_\epsilon \boldsymbol{e}_k, \tag{15}$$

where Q_ϵ is a design weight. To make progress along the path, a cost is defined that penalizes the deviation of the drone speed v_k from a reference velocity v_{ref}, i.e.,

$$J_{\text{speed}}\ (\boldsymbol{x}_k, \boldsymbol{u}_k) = Q_v \left(v_{\text{ref}}\ - v_k\right)^2 \tag{16}$$

with Q_v a design weight. Additionally, the inputs are penalized with

$$J_{\text{input}}\ (\boldsymbol{x}_k, \theta_k) = \boldsymbol{u}_k^T Q_u \boldsymbol{u}_k \tag{17}$$

, where Q_u is a design weight.

The stage cost is evaluated at the mean of the Gaussian state distribution.

Obstacle avoidance constraints

For obstacle avoidance, two types of constraints are considered: Ellipsoidal constraints and Gaussian constraints. Both types of constraints assume static, circular obstacles and take into account the uncertainty of the state distribution.

For obstacle avoidance, the relevant uncertainties are the uncertainties in the positional states. We therefore extract the uncertainty matrix:

$$\Sigma_{k,\text{pos}} = \begin{bmatrix} \sigma_x & \sigma_{xy} \\ \sigma_{yx} & \sigma_{yy} \end{bmatrix} \tag{18}$$

from the propagated uncertainty at each state k.

Ellipsoidal constraints

For the ellipsoidal constraints, an ellipsoidal uncertainty bound is constructed around the quadrotor, given the uncertainty matrix and a confidence interval χ, which represents the collision probability. The obstacle avoidance constraint at each state is:

$$c_k^{\text{obst},j}(\boldsymbol{z}_k) = \begin{bmatrix} \Delta x_k^j \\ \Delta y_k^j \end{bmatrix}^{\text{T}} R(\psi)^{\text{T}} \begin{bmatrix} \frac{1}{\alpha^2} & 0 \\ 0 & \frac{1}{\beta^2} \end{bmatrix} R(\psi)\ \begin{bmatrix} \Delta x_k^j \\ \Delta y_k^j \end{bmatrix} > 1 \tag{19}$$

,

where Δx_k^jandΔy_k^j is the distance between the quadrotor and the obstacle. $R(\psi)$ is the rotation matrix, describing the rotation of the uncertain ellipsoid and $\alpha = a + r_{\text{obst}} + r_{\text{quad}}$, $\beta = a + r_{\text{obst}} + r_{\text{quad}}$, where a and b the major and minor axes of the ellipsoid. The rotation and major and minor axes of the ellipsoid can be found using a singular value decomposition of the uncertainty of the quadrotor position $\Sigma_{k,\text{pos}}$. The axes are enlarged according to the collision probability χ.

Chance constraints

The chance constraints are defined as a collision probability:

$$\Pr\left(\mathbf{x}^k \notin \mathcal{C}_o^k\right) \geq 1 - \delta_o, \forall o \in \mathcal{I}_o. \tag{20}$$

The collision region is given as an enlarged half space of the actual collision region:

$$\tilde{\mathcal{C}}_o := \left\{\mathbf{x} \mid \mathbf{a}_o^T \left(\mathbf{p} - \mathbf{p}_o\right) \leq b\right\}, \tag{21}$$

where $\mathbf{a}_o = (\hat{\mathbf{p}} - \hat{\mathbf{p}}_o) / \|\hat{\mathbf{p}} - \hat{\mathbf{p}}_o\|$ with quadrotor position $\mathbf{p}$, obstacle position $\mathbf{p}_o$ and $b = r_{\text{quad}} + r_{\text{obst}}$. The chance constraints can be reformulated into deterministic ones using that:

$$\Pr\left(\mathbf{a}^T\mathbf{x} \leq b\right) = \frac{1}{2} + \frac{1}{2}\,\text{erf}\left(\frac{b - \mathbf{a}^T\hat{\mathbf{x}}}{\sqrt{2\mathbf{a}^T\sum\mathbf{a}}}\right). \tag{22}$$

This results in deterministic chance constraint of the form:

$$\mathbf{a}_o^T\left(\hat{\mathbf{p}} - \hat{\mathbf{p}}_o\right) - b \geq \text{erf}^{-1}\left(1 - 2\delta_o\right)\sqrt{2\mathbf{a}_o^T\left(\Sigma_{\text{pos}}\right)\mathbf{a}_o} \tag{23}$$

Comparison

While the ellipsoidal constraints define a hard bound on the probability, the chance constraints should be less conservative. A comparison for both cases should demonstrate that behavior.

The Issue

Working with FORCESPRO, there are two ways of including the uncertainty into the constraints:

1. Passing on the uncertainty as a runtime parameter to the solver. In that case, the solver assumes that the uncertainty is constant at each prediction step, and hence cannot optimize for the uncertainty. The uncertainty is computed outside of the solver based on the previous state and the uncertain prediction model in 13).

2. Defining the uncertain dynamics as part of the dynamic equations used in the equality constraints. As the uncertainty is now a dynamic state the solver also has to evaluate it's derivative and can optimize for the uncertain states. The uncertainty is passed on to the constraints from the state vector.

While the first method is causing no issues in the solver, it might not be the right way of taking the uncertainty into account as the dynamic behavior of the uncertainty based on the states is neglected. However, the question remains whether the solver can be optimizing for the uncertainty if it is only considered in the constraints or if defining it as a runtime parameter is sufficient.

The second method results in unwanted behavior of the solver, often times running into issues finding a feasible solution, evaluating the functions or it's derivatives. This is most likely caused by the form of the constraints using square roots and the solver evaluating these square roots at negative values while optimizing for the uncertainty. Adding slack variables and additional constraints can help overcome some of these issues, but still leaves the solver very fragile running into feasibility issues every now and then, making it very unreliable for actual flight.

Planning

Given time constraints, for now I will move on with the first option to generate some results. I believe that this method should be accurate enough to demonstrate the idea of using GP to model disturbances and how to include those into the LMPCC formulation. I will do a comparison of both ellipsoidal and chance constraints hoping that chance constraints will turn out to be less conservative in this scenario.

If time remains, we will look into the second option trying to solve the solver issues. Unfortunately they are present for both type of constraints. One idea is to add a regularization term. Another idea is to solve a cascaded optimization problem, solving first for a feasible initial guess given the constraints and then passing that on to the solver in a second optimization step, hoping that this will stabilize the solver.