[Technical Report] Fastron for Dynamic Uncertain Environments *

Saikiran Komatineni, Yuheng Zhi

Nov 2021

1 Summary

Fastron is a learning-based algorithm that models the robot configuration space such that it can be used as a proxy collision detector instead of traditional geometric collision checkers that are generally computationally intensive. The authors proposed an active learning strategy to perform efficient collision checking in dynamic environments. It predicts the values of support vectors at time step $t+1$ given the support vectors, with corresponding labels, at time step t. Let S_t be the set of support vectors at time step t. S_{t+1} is obtained with the equation $S_{t+1} = kernel\_perceptron(S_t, exploit(S_t), explore())$. The current heuristic for $exploit()$ samples vectors within a Gaussian region around S_t. This does not incorporate knowledge of the motion of obstacles and the *exploit* may fail at high obstacle speeds. This motivated us to propose the new heuristic that determines $exploit(S_t)$ using information from obstacle motion. The proposed exploitation step translates the motion of obstacles in the environment ${}_{\Delta}\mathbf{x}_t$ to motion of support vectors${}_{\Delta}S_t$.

In reality, there is always uncertainty associated with ${}_{\Delta}\mathbf{x}_t$ caused by noise in observation sensors, perception, and tracking. We are exploring two methods to translate this uncertainty in the workspace to the configuration space i) formulating SV as random variables and using sparse bayesian kernels to determine the output distribution ii) using the gaussian process technique to approximate the collision checking function with corresponding probability distribution. This step was motivated by the many benefits of obtaining a probabilistic collision measure including the introduction of safety parameters for cautious path planning.

*Please note that this report only documents the method of the project and that this work is still ongoing

2 Method: Efficient Sampling for Exploitation in Active Learning

The novelty in the Fastron algorithm lies in the idea that it is sufficient to use a kernel perceptron with sparse support points to determine the collision status of arbitrary configurations. A valid trajectory from the start point to an end point where the robot lies in the collision-free region throughout the motion can then be determined more efficiently. This approach has limitations in dynamic environments where obstacles can be in motion because, although the original algorithm fine-tunes its model on a set of configurations sparsely selected by a heuristic, it does not account for any changes in location of support points caused by obstacle motion. This is more evident in the exploitation stage, especially when obstacles move quickly resulting in the support points no longer being a good estimate of the C-space (assuming the C-space is sampled at the same frequency). Although increasing sampling frequency can improve the model's robustness obstacle motion, it can be very computationally expensive.

The proposed approach here is to develop a more efficient exploitation heuristic that will substantially reduce the number of samples needed to fine-tune the model. The heuristic is based on approximate inverse kinematics. For each support point, the heuristic calculates the configuration that places the robot at the same pose, relative to the nearest obstacle, before and after the obstacle motion. If initially, the robot is in-collision with the obstacle, then the heuristic estimates the configuration at the next time step such that the robot's collision centroid remains the same for the same kinematic point on the robot's body. Similarly, if the robot is initially in collision-free, the updated configuration is determined such that the robot is in the same position from the moving obstacles' frame of reference. We can then perform support point matching between the initial and updated configuration.

2.1 Mathematical Derivations

An underlying assumption we make for this heuristic is that for each support point, we can find its equivalent configuration in the next timestep. If we find such an equivalent configuration, it is highly likely that point is a support point at the next timestep. It remains to define *equivalent*. We define two kinds of equivalent configurations as shown Fig. 1.

- For an in-collision support configuration, an equivalent configuration should collide with the obstacle at the same point in *the obstacle frame* with the same point on the robot's kinematic chain as the original support point. This point is called an *equivalent target*, which can be the centroid of the region that is in collision.
- For a collision-free support configuration, the *equivalent target* can be defined as the closest point on the robot to the obstacle, instead of the centroid of the collided region. Again, this point will move with the *obstacle*. Alternatively, we can create such a concept called the *generalized* rigid

body of a frame T, defined as a set of points expanding the universe and moves with T. Then we can also say the equivalent target is also the point on the generalized rigid body of T_{obs}^{t} at the location of the closest point on the robot's kinematic chain to the obstacle, in the considered support configuration. An equivalent configuration should reach the same equivalent target in the generalized rigid body of $T_{obs}^{t+\Delta t}$ with the same point on the robot's kinematic chain as the original support configuration.

We will then estimate the corresponding changes in joint angles in order to reach the equivalent targets using approximate inverse kinematics.

Define $\mathbf{x} = [x, y, z, r, p, \gamma]^T$ as the pose of a point on the robot's kinematic chain (note: this point is not necessarily the end effector, but any point on the kinematic chain) in the workspace at time step t, which superposes the equivalent target before and after the obstacle moves. Define the robot configuration as $\boldsymbol{\theta} \coloneqq [\theta_0, \theta_1, \ldots, \theta_n]^T$ and the forward kinematics model as f. The forward kinematics equation is as follows:

$$f(\boldsymbol{\theta}) = \mathbf{x} \coloneqq [x, y, z, r, p, \gamma]^T \tag{1}$$

We can take derivative on both sides.

$$\nabla_t f(\boldsymbol{\theta}(t)) = \nabla_t \mathbf{x}(\mathbf{t}) \tag{2}$$

$$\frac{df(\boldsymbol{\theta}(t))}{d\boldsymbol{\theta}(t)} \frac{d\boldsymbol{\theta}(t)}{dt} = \dot{\mathbf{x}} = [\dot{x}, \dot{y}, \dot{z}, \dot{r}, \dot{p}, \dot{\gamma}]^T \tag{3}$$

By rearranging terms and defining the Jacobian we have:

$$J = \frac{df(\boldsymbol{\theta}(t))}{d\boldsymbol{\theta}(t)} = \begin{bmatrix} \frac{\partial f_1}{\partial \theta_1} & \cdots & \frac{\partial f_1}{\partial \theta_n} \\ \vdots & \vdots & \vdots \\ \frac{\partial f_6}{\partial \theta_1} & \cdots & \frac{\partial f_6}{\partial \theta_n} \end{bmatrix} = \frac{\dot{\mathbf{x}}}{\dot{\boldsymbol{\theta}}} \tag{4}$$

$\dot{\mathbf{x}}$ represents the rate of change of state for each obstacle and $\dot{\boldsymbol{\theta}}$ represents the corresponding rate of change of joint angles. Our objective is to determine the change in $\boldsymbol{\theta}$ with a small observed change in $\mathbf{x}$. Derivation of the inverse Kinematics (given ${}_{\Delta}\mathbf{x}$, determine) is shown below: ${}_{\Delta}\boldsymbol{\theta}$

The forward kinematics of the manipulator arm with one joint ($\boldsymbol{\theta}_j$) can be represented using the Taylor series representation:

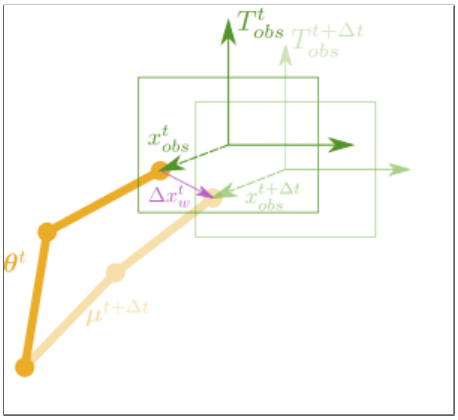
$$f_i(\boldsymbol{\theta}_t) = f_i(\boldsymbol{\theta}_0) + \frac{f_i'(\boldsymbol{\theta}_0)}{1!}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0) + \frac{f_i''(\boldsymbol{\theta}_0)}{2!}(\boldsymbol{\theta}_t - \boldsymbol{\theta}_0)^2 + \ldots \tag{5}$$

For a small change in $\mathbf{x}_i$, this can be approximated as:

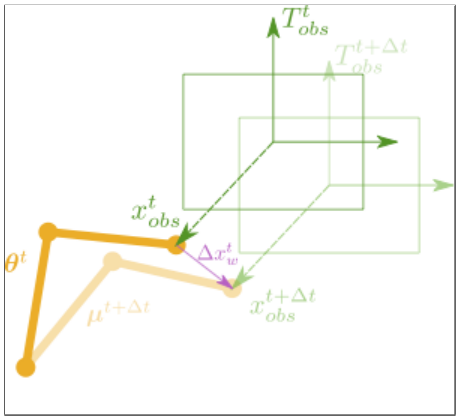
$${}_{\Delta}\mathbf{x}_i = f_i(\boldsymbol{\theta}_t) - f_i(\boldsymbol{\theta}_0) \approx f_i'(\boldsymbol{\theta}_0){}_{\Delta}\boldsymbol{\theta} \tag{6}$$

$${}_{\Delta}\mathbf{x} \approx J(\boldsymbol{\theta}_0){}_{\Delta}\boldsymbol{\theta} \tag{7}$$

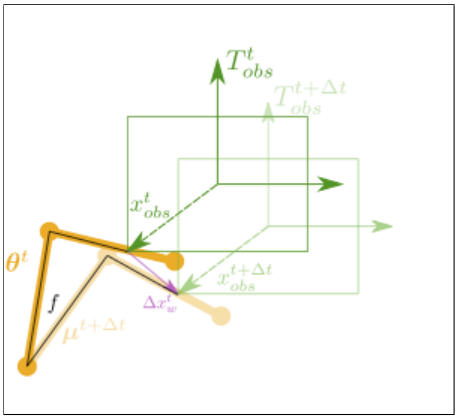
$${}_{\Delta}\boldsymbol{\theta} \approx J(\boldsymbol{\theta}_0)^{+}{}_{\Delta}\mathbf{x} \tag{8}$$



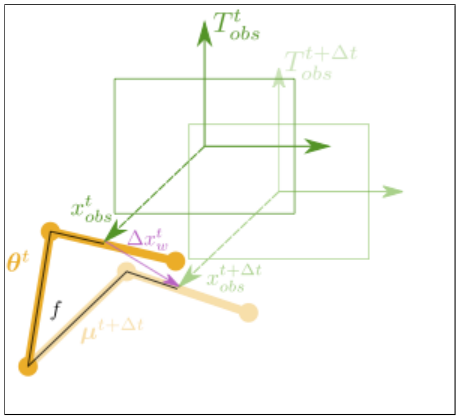
(a) For an in-collision support configuration, the target point is defined as either the deepest penetration point or the center of the penetrated region on the rigid-body obstacle.



(b) For a collision-free support configuration, the target point is defined as the point on the obstacle's *generalized* rigid body at the location of the closest point of the robot to the obstacle.



(c) Another in-collision support configuration when the equivalent target is NOT at the position of the end effector.



(d) Another collision-free support configuration when the equivalent target is NOT at the position of the end effector.

Figure 1: Notation illustrations. For both kinds of configurations, the point on the forward-kinematic chain, x^t, is updated by changing the robot configuration $\boldsymbol{\theta}^t$ to $\boldsymbol{\mu}^{t+\Delta t}$ so that it approximately remains the same in the obstacle frame T^t_{obs} (i.e., $x^t_{obs} \approx x^{t+\Delta t}_{obs}$). This requires x^t_w to be moved by $\Delta x^t_w = x^{t+\Delta t}_w - x^t_w$ in the world frame. In a small timestep, $\Delta x^t_w \approx \dot{x}^t_w \Delta t$, where $\dot{x}^t_w$ is equivalent to the world-frame velocity of the constant target point in the obstacle frame. Also in a small timestep, the amount of change needed in the robot configuration is subject to $\dot{x}^t_w = J\dot{\theta}^t$, $\Delta\theta^t \approx \dot{\theta}^t \Delta t$. The approximate equivalent configuration $\boldsymbol{\mu}^{t+\Delta t} = \boldsymbol{\theta}^t + \Delta\theta$ is used as the mean of a set of Gaussian samples for exploitation in active learning.

In this project we are interested in estimating ${}_{\Delta}\boldsymbol{\theta}$ given a noisy ${}_{\Delta}\mathbf{x}$. An observation model captures the environment states using a sensor on board the robot manipulator arm. A trajectory prediction model predicts the probabilistic motion of all obstacles in the environment.

We can guess the locations of the equivalent configurations of current support points in the next timestep using $\dot{\boldsymbol{\theta}}$, and take samples for *exploitation* in active learning.

$$\Theta_{exploitation} := \{\boldsymbol{\theta} \,|\, \boldsymbol{\theta} \sim N(\boldsymbol{\theta}_{support} + \dot{\boldsymbol{\theta}}_{support}\Delta t, \sigma^2 I)\}. \tag{9}$$

The exploration samples $\Theta_{exploration}$ follow the original random heuristic. Ground-truth labels Y_{active} will be queried for $\Theta_{active} = \Theta_{exploitation} \cup \Theta_{exploration} \cup \Theta_{support}$. The final step is to finetune/re-train the model on the dataset $(\Theta_{active}, Y_{active})$.

3 Method: Trajectory Prediction using Unscented Kalman Filter (UKF)

One way to correct for error in observations and make trajectory predictions is to use the Kalman Filter (KF). The KF uses prior knowledge of the environment state (which includes states of all obstacles in the environment) and observations to predict the environment state up to a certain horizon T. It is noteworthy that the discretization of the support vectors along the time dimension may not align with that of the environment. Let f_P be the frequency at which support vector predictions are required to be made and f_O be the frequency at which obstacles are observed in the environment. Case (i) $f_P <= f_O$: interpolation can be used to estimate observations at the time step needed. Case (ii) $f_P > f_O$: Kalman Filter can be used to predict environment states at intermediate time steps where observations are not available.

The environment state is formally defined as $E = [e_1, e_2, \ldots, e_o]$ where e_i is the state of obstacle i and o is the total number of obstacles in the environment. $e_i = [x, y, z, r, p, \gamma]^T$ where (x, y, z) and (r, p, γ) represent the position and orientation respectively.

We specifically apply the Unscented Kalman Filter to each obstacle separately and make the assumption that the obstacles are independent of each other. For the remainder of the write-up we use the definition $X :=$ unobserved state of the system (for simplicity X only represents one obstacle)and $Y :=$ observed signal. Non-linear state-space representation is as follows[1]:

$$\begin{aligned} X(k+1) &= F[X(k)] + \omega(k) \\ Y(k) &= H[X(k)] + \nu(k) \end{aligned} \tag{10}$$

[1]Notation is inspired by [3]

Detected obstacles' dynamics are assumed to be known and the motion model representation is shown in figure 2 [3]. $G(z)$ is a known transfer function that approximates the obstacle dynamics. States that the Kalman filter will estimate include: obstacle setpoints (r_ρ speed in the xy-plane, r_ψ heading in the xy-plane, r_z altitude), output of model $G(z)$ (y_p speed in xy-plane, y_ψ heading in xy-plane, altitude z), obstacle position in space (x, y, z), internal states of $G(z)$ (X_m). These can be summarized as:

$$X_r = [r_\rho, r_\psi, r_z]^T \tag{11}$$
$$X_y = [y_p, y_\psi, z]^T \tag{12}$$
$$X_p = [x, y]^T \tag{13}$$
$$\tag{14}$$
$$X = [X_r^T, X_m^T, X_y^T, X_p^T]^T \tag{15}$$

Equations for State Estimation and Object Dynamics:

$$\text{State Estimation} \qquad X_r(k+1) = X_r(k) + \omega_r(k) \tag{16}$$
$$\text{Object Dynamics} \qquad X_y(k+1) = G(z)X_r(k) \tag{17}$$

where $\omega_r = [\omega_\rho, \omega_\psi, \omega_z]^T$ is a random vector consisting of 3 independent, zero-mean, white noise Gaussian Random Variables (GRV).

Motion equations:

$$x(k+1) = x(k) + T_s y_p(k) cos(y_\psi(k)) \tag{18}$$
$$y(k+1) = y(k) + T_s y_p(k) sin(y_\psi(k)) \tag{19}$$
$$X_p(k+1) = F_p[X_y(k), X_p(k)] \tag{20}$$

T_s is the sampling period. Up to this point equations for EKF and UKF are the same. For UKF, instead of using Taylor series approximation of equation 20 unscented transform is used to preserve non-linearity of F_p. In UKF, the state is redefined $X_k^a = [X_k^T, \omega_k^T, \nu_k^T]^T$. UKF algorithm is shown below [5]:

$$\hat{X_0} = E[X_0] \tag{21}$$
$$\Sigma_0 = E[(X_0 - \hat{X_0})(X_0 - \hat{X_0})^T] \tag{22}$$
$$\hat{X_0}^a = E\begin{bmatrix}\hat{X_0^T} & 0 & 0\end{bmatrix}^T \tag{23}$$
$$\hat{\Sigma_0}^a = E[(X_0^a - \hat{X_0}^a)(X_0^a - \hat{X_0}^a)^T] = \begin{bmatrix}\Sigma_0 & 0 & 0\\ 0 & \Sigma_\omega & 0\\ 0 & 0 & \Sigma_\nu\end{bmatrix} \tag{24}$$

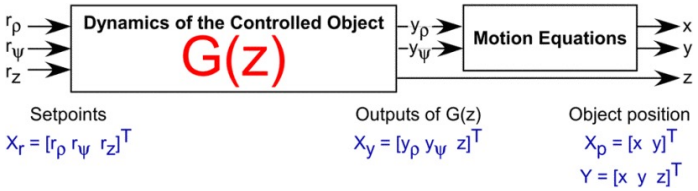


Figure 2: Motion model of one observed obstacle

For $k \in 1, \ldots, \infty$:

Calculate Sigma points:

$$\chi^a_{k-1} = \left[\hat{X^a_{k-1}} \quad \hat{X^a_{k-1}} \pm \sqrt{(L+\lambda)\Sigma^a_{k-1}}\right] \tag{25}$$

Prediction Step: computing sigma points, mean, covariance, corrected observation, observation mean:

$$\chi^x_{k|k-1} = F[\chi^x_{k-1}, \chi^v_{k-1}] \tag{26}$$

$$\hat{X}^-_k = \sum_{i=0}^{2L} W^{(m)}_i \chi^x_{i,k|k-1} \tag{27}$$

$$\Sigma^-_k = \sum_{i=0}^{2L} W^{(c)}_i [\chi^x_{i,k|k-1} - \hat{X}^-_k][\chi^x_{i,k|k-1} - \hat{X}^-_k]^T \tag{28}$$

$$\mathcal{Y}_{k|k-1} = H[\chi^x_{k|k-1}, \chi^n_{k-1}] \tag{29}$$

$$\hat{Y}^-_k = \sum_{i=0}^{2L} W^{(m)}_i \mathcal{Y}_{i,k|k-1} \tag{30}$$

Update Step: updating Covariance, Unobserved state

$$\Sigma_{\overline{Y}_k \overline{Y}_k} = \sum_{i=0}^{2L} W^{(m)}_i [\mathcal{Y}_{i,k|k-1} - \hat{Y}^-_k][\mathcal{Y}_{i,k|k-1} - \hat{Y}^-_k]^T \tag{31}$$

$$\Sigma_{X_k Y_k} = \sum_{i=0}^{2L} W^{(c)}_i [\chi_{i,k|k-1} - \hat{X}^-_k][\mathcal{Y}_{i,k|k-1} - \hat{Y}^-_k]^T \tag{32}$$

$$\kappa = \Sigma_{X_k Y_k} \Sigma^{-1}_{\overline{Y}_k \overline{Y}_k} \tag{33}$$

$$\hat{X}^-_k = \hat{X}^-_k + \kappa(Y_k - \hat{Y}^-_k) \tag{34}$$

$$\Sigma_k = \Sigma^-_k - \kappa \Sigma_{\hat{Y}_k \hat{Y}_k} \kappa^T \tag{35}$$

$\chi^a = \left[(\chi^X)^T \quad (\chi^\omega)^T \quad (\chi^\nu)^T\right]$, λ = scaling parameter, L =dimension of augmented state, Σ_ω =motion model noise covariance, Σ_ν =measurement noise

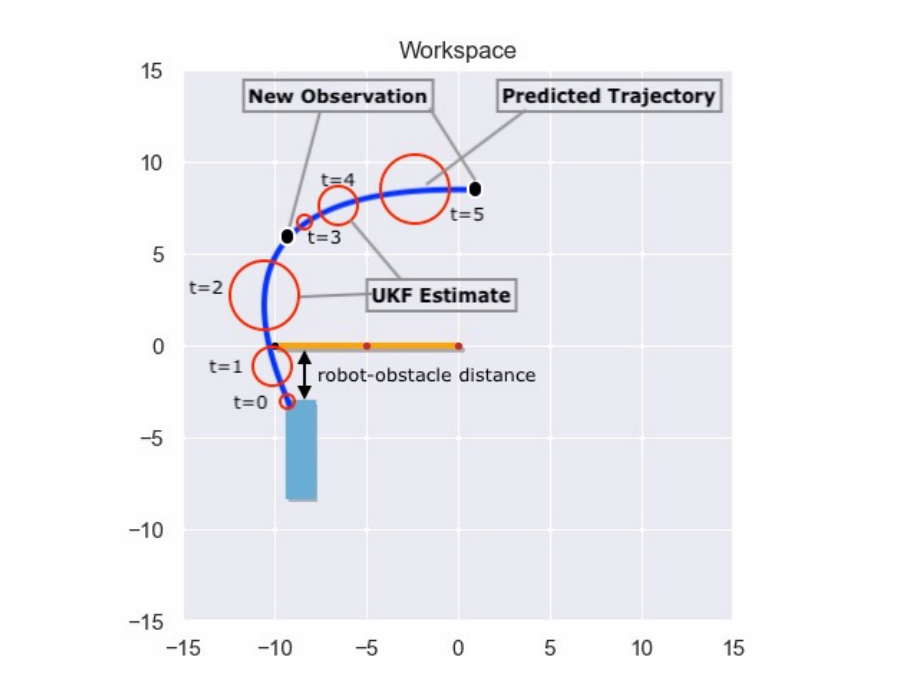


Figure 3: Sample Environment (2D): blue rectangle is the obstacle and robot is shown in orange. The blue curve shows one potential estimated trajectory of the obstacle over 5 time steps. The black points show new observations (note that these points may not necessarily fall on the blue line). The red circles represent the covariance of the estimated obstacle states. Closest distance between the obstacle and the robot is also shown.

covariance, W_i =sigma points weights.
The update step will be computed at each time step where an observation is available and the prediction step will be computed according to the frequency f_P. Trajectory of an obstacle up to horizon T can be obtained from $\mathcal{Y}$ by repeatedly computing the prediction step up to $\mathcal{Y}_{T|T-1}$.

3.1 Exploitation heuristic:

d_t is the shortest distance between the robot and the closest obstacle at time step t. ${}_{\Delta}\mathbf{o}_t := \hat{Y}^-_{t+1} - \hat{Y}^-_t$ is the change in the *closest* obstacle's position from time t to $t+1$. ${}_{\Delta}\mathbf{x}_t :=_{\Delta} \mathbf{o}_t + d_t$ and using equation 8 we obtain ${}_{\Delta}\theta_t$ which is the required change in robot configuration at time step t. Due to the uncertainty in the estimated trajectory, ${}_{\Delta}\mathbf{o}_t$ is a random variable (RV) resulting in ${}_{\Delta}\mathbf{x}_t$ also being RV's.

One progression we are exploring following from the linearity of equation 8 is to formulate ${}_{\Delta}\theta_t$ as an RV[2]. Let s_t be one support vector at time step t. According to the heuristic, $s_{t+1} = s_t +_{\Delta} \boldsymbol{\theta}$. Since ${}_{\Delta}\boldsymbol{\theta}$ is an RV, s_{t+1} is also an RV. Figure 3 shows a sample environment with a hypothetical estimated obstacle trajectory.

We now have a set of support vectors for each time step which are random variables. The next step is to compute a probability for collision at each time step.

4 Method: Probabilistic Collision Checking

The original Fastron algorithm [1] uses a sparse set of support vectors to classify a new test point as either collision free or in collision $\in \{-1, 1\}$. The objective of this section is to extend this approach to obtain associated probabilities for each classification. There are many benefits to this including:

- In robot motion planning, it can be beneficial to introduce a safety parameter which determines how cautious a planned path will be. The collision probability can be directly related to the safety parameter where the lower the probability, the safer the planned path.
- In some cases, collision classification may only be a small part of an overall decision to be made. In such cases classification output can be combined with other outputs to form a cohesive decision.
- Although not applicable to collision checking, when performing multi-category classification, selection can be done based on maximum posterior probability over all the classes. This is the Bayes optimal decision when loss is equal for all classes.
- As [2] pointed out, unknown regions in robotics are typically labeled as free, disallowing us from distinguishing between unknown and free regions. This distinction may be desirable in some scenarios.

The probabilistic extension of Fastron follows the formulation of Relevance Vector Machines (RVM) [4] which is a probabilistic version of support vector machines (SVM). [2] have demonstrated the combined application of RVM and sparse bayesian kernels to obtain probabilistic occupancy grid mapping.

They show that the predictive distribution of a test point $\mathbf{x}$ is [3]:

$$p(y|\mathbf{x}, \boldsymbol{\xi}) \approx \int p(y|\mathbf{x}, \boldsymbol{\omega}) p(\boldsymbol{\omega}|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{\omega} \tag{36}$$

[2]Note that this step is still preliminary and we are working on the feasibility of this step

[3]To follow the steps to get to the predictive distribution equation, please review section V.A in Thai, et. al's paper [2]

$$p(y|\mathbf{x}, \boldsymbol{\xi}) \approx \int p(y|\mathbf{x}, \boldsymbol{\omega}, S) p(\boldsymbol{\omega}, S|\mathbf{y}, \mathbf{X}, \boldsymbol{\mu}, \boldsymbol{\Sigma}) d\boldsymbol{\omega} dS \tag{37}$$

y is the target $\in -1, +1$, $\boldsymbol{\xi}$ is a hyperparameter that describes the distribution of the weights $\boldsymbol{\omega}$. The first term in the integral is the data likelihood and the second term is the weight posterior. The authors use a LaPlace approximation to determine $\boldsymbol{\mu}, \boldsymbol{\Sigma}$.

Our work concerns the first integral term. It is defined as $p(y|\mathbf{x}, \boldsymbol{\omega}) := \sigma(y(\Phi_{\mathbf{x}}^T \boldsymbol{\omega} + b))$ where $\Phi_{\mathbf{x}}^T := \begin{bmatrix} k_1(\mathbf{x}) & k_2(\mathbf{x}) & \dots & k_M(\mathbf{x}) \end{bmatrix}^T = \begin{bmatrix} k(\mathbf{x}, \mathbf{x}_1) & k(\mathbf{x}, \mathbf{x}_2) & \dots & k(\mathbf{x}, \mathbf{x}_M) \end{bmatrix}^T \in \mathcal{R}^M$ where k is the kernel function. We are interested in finding the density function of $p(y|\mathbf{x}, \boldsymbol{\omega})$ when $\mathbf{x}$ is a random variable.

4.1 Probabilistic Fastron

Before we proceed further, let us review the 3 key steps from the active learning algorithm:

1. **Training:**
   - Input: training data: $\mathcal{X} = [(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \dots, (\boldsymbol{x}_N, y_N)$
   - Output: support vector set: $\mathcal{S} = [s_1, s_2, \dots, s_m]$
   - Operation: iteratively update the weights of the kernel matrix and drop redundant support vectors
2. **Support Vector computation:**
   - Input: support vector set at time step t: $\mathcal{S}_t$
   - Output: support vector set at time step t+1: $\mathcal{S}_{t+1} = [s_1, s_2, \dots, s_{m*}]$
   - Operation:
     - Previous Approach: sample from $\mathcal{N}(s_j, \sigma) \forall j \in \{1, 2, \dots, m*\}$
     - Second approach:
       (a) ${}_{\Delta}z$ =translation in closest obstacle
       (b) ${}_{\Delta}s_j = InvKin({}_{\Delta}z)$
       (c) sample from $\mathcal{N}(s_j +_{\Delta} s_j, \sigma) \forall j \in \{1, 2, \dots, m*\}$
3. Use a geometric collision checker to compute labels for each support vector in $\mathcal{S}$

There are several challenges that need to be overcome before we can treat support vectors as random variables:

Step 3 Computing labels: we can no longer treat labels as deterministic boolean values as support vectors are themselves random. The simplest solution to this is to sample multiple points from each SV distribution and use a geometric collision checker to compute their labels; we can then use the monte carlo method to obtain probabilities for each label. A better approach is to formulate the labels as bernoulli RVs.

Step 1 Since both, the training samples and support vectors are gaussian RVs, the kernel matrix will have to be modified. Consider a simple linear kernel function $k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{x}^T \boldsymbol{x}'$. In this case the result of the kernel operation is also a gaussian RV, however this step is more complex for non-linear kernels. In such cases, an unscented transform can be used to approximate the mean and covariance of the kernel function.

- The hypothesis $F =$ is also a GRV. To check for misclassifications, rather than checking $yF \leq 0$, we can perform the same computation with the expectations of each variable. (Using expectation is not ideal here, an alternative method should be explored). The remainder of the training step can be maintained as is while treating $K, F, \boldsymbol{x}, y$ as RVs.

Step 2 The final step is to use random SVs to predict the their future locations. Note that part (c) of the operation simply translates to the addition of two GRVs original support vectors and translation in support vectors are both gaussian.

With this approach we can train a probabilistic version of the Fastron algorithm. The next step is to modify the equation to compute the data likelihood (the first term in the integral in equation 37). [2] defined the data likelihood as:

$$p(\boldsymbol{y}, \mathcal{X}, \boldsymbol{\omega}) = \prod_{l=1}^{N} \sigma(F(\boldsymbol{x}_l))^{\frac{1+y_l}{2}} (1 - \sigma(F(\boldsymbol{x}_l)))^{\frac{1-y_l}{2}} \tag{38}$$

Since $F(\mathbf{x}) := \Phi_{\mathbf{x}}^T \boldsymbol{\omega} + b$ is a GRV as well, $p(y|\mathbf{x}, \boldsymbol{\omega})$ is the result of a non-linear transform applied to a GRV. By change of variables, we can represent this probability density (with respect to $\mathbf{x}$) $p_{\mathbf{x}}(F(\mathbf{x})$ with another variable $a := \sigma(F)$ such that $p_{\mathbf{x}}(F(\mathbf{x}))\delta\mathbf{x} \approx p_a(a)\delta a$ where $\delta\mathbf{x}$ and δa are small changes in $\mathbf{x}$ and a respectively.

$$p_a(a) = p_{\mathbf{x}}(F(\mathbf{x})) \left| \frac{dF(\mathbf{x})}{da} \right|^{-1} \tag{39}$$

Both the first and second terms of the integration in equation 36 are now gaussian probability desnities and the integration is done over a joint gaussian distribution which can be simplified into the form:

$$p(y|\boldsymbol{x},\boldsymbol{\xi}) = \sigma(\frac{y(\Phi_x^T\mu + b)}{\sqrt{1+\Phi_x^T\Sigma\Phi_x}}) \tag{40}$$

References

[1] Nikhil Das, Naman Gupta, and Michael Yip. "Fastron: An online learning-based model and active learning strategy for proxy collision detection". In: *Conference on Robot Learning*. PMLR. 2017, pp. 496–504.

[2] Thai Duong, Michael Yip, and Nikolay Atanasov. "Autonomous Navigation in Unknown Environments with Sparse Bayesian Kernel-based Occupancy Mapping". In: *arXiv preprint arXiv:2009.07207* (2020).

[3] Carole G. Prevost, Andre Desbiens, and Eric Gagnon. "Extended Kalman Filter for State Estimation and Trajectory Prediction of a Moving Object Detected by an Unmanned Aerial Vehicle". In: *2007 American Control Conference*. 2007, pp. 1805–1810. DOI: 10.1109/ACC.2007.4282823.

[4] Michael E Tipping and Anita C Faul. "Fast marginal likelihood maximisation for sparse Bayesian models". In: *International workshop on artificial intelligence and statistics*. PMLR. 2003, pp. 276–283.

[5] E.A. Wan and R. Van Der Merwe. "The unscented Kalman filter for nonlinear estimation". In: *Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No.00EX373)*. 2000, pp. 153–158. DOI: 10.1109/ASSPCC.2000.882463.