Infinite-Horizon Stochastic Optimal Control

Ryosuke Kato

I. Introduction

Trajectory tracking problem is how to track a moving reference point in continuous space in the minimal cost. This problem is categorized as infinite-horizon stochastic optimal control. The main difficulties of this problem are that i) inifinite-horizon, ii) stochastic, and iii) continuous space. Usually, we do not have sufficient resources to find the exact solution of this problem due to these difficulties. Therefore, it is important to relax these difficulties by some methods and find a sub-optimal solution. Receding-horizon certainty equivalent control (CEC) control relaxes i) and ii) and generalized policy iteration (GPI) relaxes i) and iii).

II. Problem Statement

This project will consider safe trajectory tracking for a ground differential-drive robot. Consider a robot whose state $\mathbf{x}_t := (\mathbf{p}_t, \theta_t)$ at discrete-time $t \in \mathbb{N}$ consists of its position $\mathbf{p}_t \in \mathbb{R}^2$ and orientation $\theta_t \in [-\pi, \pi)$. The robot is controlled by a velocity input $\mathbf{u}_t := (v_t, \omega_t)$ consisting of linear velocity $v_t \in \mathbb{R}$ and angular velocity (yaw rate) $\omega_t \in \mathbb{R}$. The discrete-time kinematic model of the differential-drive robot obtained from Euler discretization of the continuous-time kinematics with time interval $\tau > 0$ is

$$
\begin{aligned}
\mathbf{x}_{t+1} &= \begin{bmatrix} \mathbf{p}_{t+1} \\ \theta_{t+1} \end{bmatrix} \\
&= f(\mathbf{x}_t, \mathbf{u}_t, \mathbf{w}_t) \\
&= \begin{bmatrix} \mathbf{p}_t \\ \theta_t \end{bmatrix} + \begin{bmatrix} \tau cos(\theta_t) & 0 \\ \tau sin(\theta_t) & 0 \\ 0 & \tau \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} + \mathbf{w}_t \\
& \quad t = 0, 1, 2, ...
\end{aligned}
$$

where $\mathbf{w}_t \in \mathbb{R}^3$ models the the motion noise with Gaussian distribution $\mathcal{N}(\mathbf{0}, diag(\sigma)^2)$ with standard deviation $\sigma = [0.04, 0.04, 0.004] \in \mathbb{R}^3$. The motion noise is assumed to be independent across time and of the robot state $\mathbf{x}_t$. The kinematics model above defines the probability density function $p_f(\mathbf{x}_{t+1}|\mathbf{x}_t, \mathbf{u}_t)$ of $\mathbf{x}_{t+1}$ conditioned on $\mathbf{x}_t$ and $\mathbf{u}_t$ as the density of a Gaussian distribution with mean $\mathbf{x}_t + \mathbf{G}(\mathbf{x}_t)\mathbf{u}_t$ and covariance $diag(\sigma)^2$. The control input $\mathbf{u}_t$ of the vehicle is limited to an allowable set of linear and angular velocities $\mathcal{U} := [0, 1] \times [-1, 1]$.

The objective is to design a control policy for the differential-drive robot in order to track a desired reference position trajectory $\mathbf{r}_t \in \mathbb{R}^2$ and orientation trajectory $\alpha_t \in [-\pi, \pi)$ while avoiding collisions with obstacles in the environment. There are two circular obstacles $\mathcal{C}_1$ centered at $(-2, -2)$ with radius 0.5 and $\mathcal{C}_2$ centered at $(1, 2)$ with radius 0.5. Let $\mathcal{F} := [-3, 3]^2 \backslash (\mathcal{C}_1 \cup \mathcal{C}_2)$ denote the free space in the environment. The environment, the obstacles, and the reference trajectory are illustrated in Fig. 1.

It will be convenient to define an error state $\mathbf{e}_t := (\tilde{\mathbf{p}}_t, \tilde{\theta}_t)$, where $\tilde{\mathbf{p}}_t := \mathbf{p}_t - \mathbf{r}_t$ and $\tilde{\theta}_t := \theta_t - \alpha_t$ measure the position and orientation deviation from the reference trajectory, respectively. The equations of motion for the error dynamics are:

$$
\begin{aligned}
\mathbf{e}_{t+1} &= \begin{bmatrix} \tilde{\mathbf{p}}_{t+1} \\ \tilde{\theta}_{t+1} \end{bmatrix} \\
&= g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t) \\
&= \begin{bmatrix} \tilde{\mathbf{p}}_t \\ \tilde{\theta}_t \end{bmatrix} + \begin{bmatrix} \tau cos(\tilde{\theta}_t + \alpha_t) & 0 \\ \tau sin(\tilde{\theta}_t + \alpha_t) & 0 \\ 0 & \tau \end{bmatrix} \begin{bmatrix} v_t \\ \omega_t \end{bmatrix} \\
& \quad + \begin{bmatrix} \mathbf{r}_t - \mathbf{r}_{t+1} \\ \alpha_t - \alpha_{t+1} \end{bmatrix} + \mathbf{w}_t
\end{aligned}
$$

We formulate the trajectory tracking with initial time τ and initial tracking error $\mathbf{e}$ as a discounted infinite-horizon stochastic optimal control problem:

$$
\begin{aligned}
V^*(\tau, \mathbf{e}) &= \min_{\pi} \mathbb{E}\left[\sum_{t+\tau}^{\infty} \gamma^t \left(\tilde{\mathbf{p}}_t^T \mathbf{Q} \tilde{\mathbf{p}}_t + q(1 - cos(\tilde{\theta}_t))^2 + \tilde{\mathbf{u}}_t^T \mathbf{R} \tilde{\mathbf{u}}_t\right)\right] \\
& s.t. \quad \mathbf{e}_{t+1} = g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{w}_t), \quad \mathbf{w}_t \sim \mathcal{N}(\mathbf{0}, diag(\sigma)^2), \\
& \qquad t = \tau, \tau + 1, ... \\
& \qquad \mathbf{u}_t = \pi(t, \mathbf{e}_t) \in \mathcal{U} \\
& \qquad \tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F}
\end{aligned}
$$

where $\mathbf{Q} \in \mathbb{R}^{2\times 2}$ is a symmetric positive-definite matrix defining the stage cost for deviating from the reference position trajectory $\mathbf{r}_t, q > 0$ is a scalar defining the stage cost for deviating from the reference orientation trajectory α_t, and $\mathbf{R} \in \mathbb{R}^{2\times 2}$ is a symmetric positive-definite matrix defining the stage cost for using excessive control effort. We will compare two different approaches for solving the problem in (3): (a) receding-horizon certainty equivalent control (CEC) and (b) generalized policy iteration (GPI).

III. Technical Approach

A. *receding-horizon CEC*

receding-horizon CEC approximates the original problem in the following two points.

- It approximate infinite-horizon as finite-horizon by time scope parameter T.
- It approximate stochastic transition as deterministic transition by ignoring the transition noise.

Thus, the optimal value function of receding-horizon CEC is obtained by the following equation

$$
\begin{aligned}
V^*(\tau, \mathbf{e}) &= \min_{\mathbf{u}_\tau,\dots,\mathbf{u}_{\tau+T-1}} \mathfrak{q}(\mathbf{e}_{\tau+T}) \\
&+ \sum_{t=T}^{\tau+T-1} \gamma^t \left(\tilde{\mathbf{p}}_t^T \mathbf{Q} \tilde{\mathbf{p}}_t + q(1-cos(\tilde{\theta}_t))^2 + \mathbf{u}_t^T \mathbf{R} \mathbf{u}_t\right) \\
s.t. \quad & \mathbf{e}_{t+1} = g(t, \mathbf{e}_t, \mathbf{u}_t, \mathbf{0}), \quad t = \tau, ..., \tau+T-1 \\
& \mathbf{u}_t \in \mathcal{U} \\
& \tilde{\mathbf{p}}_t + \mathbf{r}_t \in \mathcal{F}
\end{aligned}
$$

The receding-horizon CEC problem can be formulized as non-linear program (NLP)

$$
\begin{aligned}
\min\nolimits_{\mathbf{U}} \quad & c(\mathbf{U}, \mathbf{E}) \\
s.t. \quad & \mathbf{U}_{lb} \leq \mathbf{U} \leq \mathbf{U}_{ub} \\
& \mathbf{h}_{lb} \leq \mathbf{h}(\mathbf{U}, \mathbf{E}) \leq \mathbf{h}_{ub}
\end{aligned}
$$

where $\mathbf{U} = [\mathbf{u}_\tau^T, ..., \mathbf{u}_{\tau+T-1}^T]^T$ and $\mathbf{E} = [\mathbf{e}_\tau^T, ..., \mathbf{e}_{\tau+T}^T]^T$. $c(\mathbf{U}, \mathbf{E})$ is the same as the original objective function $V^*(\tau, \mathbf{e})$. The optimizing variable is action (policy) i.e. $\mathbf{U}$ such that $\mathbf{U}_{lb} \leq \mathbf{U} \leq \mathbf{U}_{ub}$. $\mathbf{U}_{lb}$ and $\mathbf{U}_{ub}$ is the same as the scope of linear and angular velocities i.e. $\mathbf{U}_{lb} = [0, -1]$ and $\mathbf{U}_{ub} = [1, 1]$. The constraint $\mathbf{h}_{lb} \leq \mathbf{h}(\mathbf{U}, \mathbf{E}) \leq \mathbf{h}_{ub}$ corresponds avoiding collision. Let **c** be coordinate of the nearest circle to the robot at some time step. Then, non-linear constraint **h(U, E)** can be written as

$$
\begin{aligned}
\mathbf{h}(\mathbf{U}, \mathbf{E}) &= \begin{bmatrix} \|\tilde{\mathbf{p}}_\tau + \mathbf{r}_\tau - \mathbf{c}\|_2 \\ \vdots \\ \|\tilde{\mathbf{p}}_{\tau+T} + \mathbf{r}_{\tau+T} - \mathbf{c}\|_2 \end{bmatrix} \\
\mathbf{h}_{lb} &= [0.5, \cdots, 0.5]^T \\
\mathbf{h}_{ub} &= [\infty, \cdots, \infty]^T
\end{aligned}
$$

In practice, to allow some time margin to change direction of the robot before collision, we set $\mathbf{h}_{lb} = [0.6, \cdots, 0.6]$. We set other parameters as follows

$$
\begin{aligned}
\Delta\tau &= 0.1 \\
q &= 1 \\
T &= 5 \\
\gamma &= 0.9 \\
Q &= \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \\
R &= \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}
\end{aligned}
$$

where $\Delta\tau$ is discretized time step to compute next state. This NLP is solved by CasADi.

B. GPI algorithm

1) state discretization: Generalized policy iteration is a policy iteration algorithm assuming that estimate of value function converges in finite steps in the policy evaluation part. Namely, it approximates infinite-horizon problem by finite-horizon problem as receded-horizon CEC does. The main drawback of GPI algorithm is time-complexity. To tackle with this, first we assume motion of reference point at sufficiently small time step is negligible as shown i.e.

$$
\begin{aligned}
\mathbf{r}_t - \mathbf{r}_{t+1} &\approx 0 \\
\alpha_t - \alpha_{t+1} &\approx 0
\end{aligned}
$$

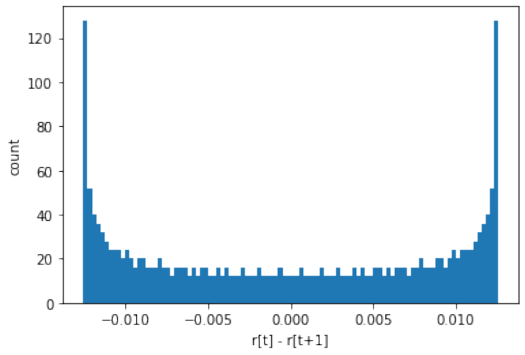

Fig. 1. change of coordinate of a reference point at small time step $\Delta\tau = 0.1$

That is, as shown in Figure 1, the largest absolute value of difference $|r_{x,t} - r_{x,t+1}|$ is around 0.01 if $\Delta\tau = 0.1$ and it is much smaller than the cell size of discretized grid, 0.3 if $n_x = 21$. By ignoring such change of a reference point, we can compute transition without using time τ as state to avoid huge state space. Thus, we use the following four variables to represent the state

- $\tilde{p}_{x,dis}$: discretized x-coordinate of error $\tilde{\mathbf{p}}$
- $\tilde{p}_{y,dis}$: discretized y-coordinate of error $\tilde{\mathbf{p}}$
- $\tilde{\theta}_{dis}$: discretized difference of angle
- α_{dis}: discretized angle of reference point

Let n_x, n_y, n_θ, and n_α be number of these discretized state. We set $n_x = 21$, $n_y = 21$, $n_\theta = 11$, and $n_\alpha = 11$ i.e. $21 \times 21 \times 11 \times 11 = 53,361$ states in total and split each continues value evenly i.e. if we want to discretize $\tilde{p}_x \in [-3, 3]$ into 21, $\tilde{p}_{x,dis}$ becomes $(-3, -2.7, ..., 3)$.

In addition to state, we also discretized policy of robot i.e. $u = [v, \omega]^T$ to 4 v_{dis}s and 4 ω_{dis}s, respectively. Thus, we have $4 \times 4 = 16$-sized action space and the overview of the GPI algorithm is as follows.

Algorithm 1 Generalized Policy Iteration

```
1: Initialize $V_0$
2: for k = 0,1,2,... do
3:     $\pi_{k+1}(\mathbf{x}) = \text{argmin}_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} H[\mathbf{x}, \mathbf{u}, V_k(\cdot)]$
4:     $V_{k+1} = \mathcal{B}^n_{\pi_{k+1}}[V_K] \quad for\ n \geq 1$
```

where $\mathbf{x} = (\tilde{p}_{x,dis}, \tilde{p}_{y,dis}, \tilde{\theta}_{dis}, \alpha_{dis})$ is discretized state and $\mathcal{B}_\pi[V](\mathbf{x})$ is policy evaluation backup operator

$$
\mathcal{B}_\pi[V](\mathbf{x}) = l(\mathbf{x}, \pi(\mathbf{x}) = \gamma \mathbb{E}_{\mathbf{x}' \sim p_f(\cdot|\mathbf{x}, \pi(\mathbf{x}))}[V(\mathbf{x})]
$$

We also set other parameters as follows. These values are the same for all three GPI models.

$$\begin{aligned} \Delta\tau &= 0.1 \\ q &= 1 \\ \gamma &= 0.9 \\ Q &= \begin{bmatrix} 10 & 0 \\ 0 & 10 \end{bmatrix} \\ R &= \begin{bmatrix} 1 & 1 \\ 1 & 1 \end{bmatrix} \end{aligned}$$

2) precomputing: Since we do not use time as state, optimized value function V^* and policy $\pi^*(\mathbf{x})$ are now independent to time. Thus, we can precompute V^* and $\pi^*(\mathbf{x})$ beforehand in time complexity $O(n_x n_y n_\theta n_\alpha)$. Once the precomputing is done, we can find optimal action at each time step in time complexity $O(1)$. This advantage is useful in particular when we want to do online motion prediction.

3) probability simplification: When we compute transition distribution $\mathcal{N}(\mathbf{0}, diag(\sigma)^2)$ in the discretized grid in both policy evaluation and policy improvement parts, density of most cells are approximately zero as shown in Figure 2. Thus, we can simplify density computation by ignoring cells that are far from the mean cell after transition $g(t, \mathbf{e}_t, \mathbf{u}_t, 0)$. We tried two types of simplification: (i) computing density of only the mean cell and set other densities to zero (model I) and (ii) computing density of the mean cell and 26 neighboring cells and set other densities to zero (model II). As discussed in **Results** section, (i) works well but (ii) does not converge. The reason why we consider 26 neighbors (i.e. ± 1 change of not 4 but only 3 variables) is as follows. We also assume discretized angle of a reference point does not change at sufficiently time step $\Delta\tau = 0.1$ and thus motion (transition) is always between two cells in the same discretized angle α_{dis} value. Since transition always occur in the same α_{dis}, we can ignore change of α_{dis} and consider only three other variables ($\tilde{p}_{x,dis}, \tilde{p}_{y,dis}, \tilde{\theta}_{dis}$).

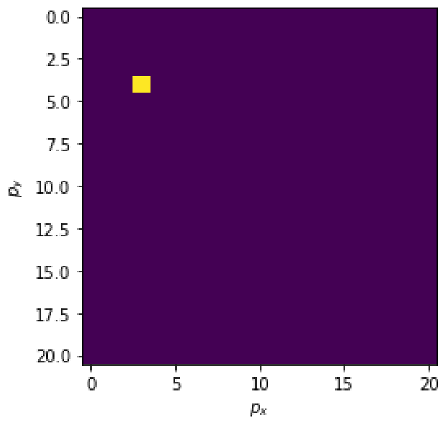

Fig. 2. Probability density represented at 2D (px and py) heat-map of some transition

4) avoiding collision: We implemented the following simple collision avoidance algorithm. If the distance of the robot and the nearest circle is less than some threshold, the robot takes an action which maximized the expected distance from the nearest circle instead of the optimal policy $\pi^*(\mathbf{x})$. We set threshold to 0.75.

Algorithm 2 Collision Avoidance

```
1: for t=0,1,... do
2:     if $\|\tilde{\mathbf{p}} + r_t - c_{nearest}\|_2 \leq$ threshold then
3:         action = $\text{argmax}_u \|p_{t+1}(u) + r_{t+1} - c_{nearest}\|_2$
4:     else
5:         action = $\pi^*(\mathbf{x})$
```

5) vectorization: We also tried vectorization of computation. Namely, we can execute matrix addition and matrix multiplication of multiple states simultaneously using NumPy's broadcasting if the matrix operations are not sequential nor dependent among states. For example, this part of value function

$$\tilde{\mathbf{p}}^T \mathbf{Q} \tilde{\mathbf{p}}$$

can be computed as

$$diag\left(\mathbf{P}_t^T \mathbf{Q}_{large} \mathbf{P}\right)$$

where

$$\mathbf{P} = \begin{bmatrix} p_{x,1} & 0 & \cdots & 0 \\ p_{y,1} & 0 & \cdots & 0 \\ 0 & p_{x,2} & \cdots & 0 \\ 0 & p_{y,2} & \cdots & 0 \\ & \vdots & & \\ 0 & 0 & \cdot & p_{x,n} \\ 0 & 0 & \cdot & p_{y,n} \end{bmatrix}$$

$$\mathbf{Q}_{large} = \begin{bmatrix} Q_{1,1} & Q_{1,2} & 0 & 0 & \cdots & 0 & 0 \\ Q_{2,1} & Q_{2,2} & 0 & 0 & \cdots & 0 & 0 \\ 0 & 0 & Q_{1,1} & Q_{1,2} & \cdots & 0 & 0 \\ 0 & 0 & Q_{2,1} & Q_{2,2} & \cdots & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & Q_{1,1} & Q_{1,2} \\ 0 & 0 & 0 & 0 & \cdots & Q_{2,1} & Q_{2,2} \end{bmatrix}$$

In addition, we can use Scipy's sparse matrix such as Compressed Sparse Column (CSC) matrix to make the matrix operation more efficient. After applying these vectorization techniques, our GPI algorithm of around 50,000 states converges in only several minutes and it seemed promising. However, as **Result** section its performance is not as good as expected. There may be a bug of indexing or some other part of matrix operation and the bug can affect the poor performance. We tried to find the bug as much as possible but we could not find it.

IV. Results

A. Overview

We compared performances of 1 Receding-horizon CEC model and 3 GPI models. The details of 3 GPI models are as follows.

- Model I: When evaluating motion noise $\sim \mathcal{N}(\mathbf{0}, diag(\sigma)^2)$ in the discretized state grid, only mean cell's density is computed. Other densities are set to 0. Namely, we use deterministic motion model instead of stochastic model to compute efficiently.
- Model II: When evaluating motion noise $\sim \mathcal{N}(\mathbf{0}, diag(\sigma)^2)$ in the discretized state grid, density of mean cell and 26 neighboring cells are computed. Other densities are set to 0.
- Model III: Vectorized by NumPy's broadcasting and Scipy's sparse matrix. Other settings are same as Model I.

Overall, a receding-horizon CEC model is the most preferable one in terms of accuracy and efficiency. Model 1 (deterministic GPI model) also achieves high accuracy after precomputing. Though accuracy of model 3 (vectorized model) is poor due to some bug, its time complexity of precomputing is significantly better than not vectorized ones (model 1 and 2) and this implies that once a bug of vectorization is fixed, model 3 can be the most preferable one.

B. Tracking Error

For each model, we computed mean error

$$\frac{1}{n_{step}} \sum_{i=1}^{n_{step}} \|\tilde{\mathbf{p}}_i\|_2$$

as evaluation metrics. The mean error for each model is as follows. Receding-horizon CEC and Model I is better than other models.

- receding-horizon CEC: 1.347
- Model I: 0.890
- Model II: 3.121
- Model III: 8.232

C. Collision Avoidance

Receding-horizon CEC model occasionally have a collision with the circles. 3 GPI models have no collision.

D. Time Complexity

Receding-horizon CEC model is the most efficient one among 4 models. It finishes computing in less than 1 minutes. The precomputing time of 3 GPI models are 1-2 hours (Model 1), around 10 hours (Model 2), and around 3 minutes (Model 3). Since the state spaces of these models are independent to time, they can find optimal actions instantly once the precomputing is done. Thanks to vectorization, Model 3 is much faster than Model 1 and 2 despite of more than 50,000 states.

E. Other remarks: Receding-horizon CEC

Trajectories of receding-horizon CEC is shown in Figure 3. Though it can track the reference well and the computing time is the shortest among all 4 models, the robot sometimes collide with the circles. These collisions are probably due to small $T(T = 5)$ and small lower bound of $h(\mathbf{U}, \mathbf{E}), 0.6$. Since it is deterministic model, the robot can get too close to circle ignoring motion noise and it can be another reason of the collisions.

F. Other remarks: GPI algorithm

1) Model I: computing only mean density: Trajectories of model I as shown in Figure 4. It can track a reference point well and can avoid the circles completely. The negative effect of discretization and ignoring motion noise (i.e. computing only mean cell's density) seems to be limited.

2) Model II: computing mean and 26 neighboring cells' density: Trajectories of model II as shown in Figure 5. The accuracy are low; one robot gets out of the $[-3, 3]^2$-sized area of a reference point, it tends to go to the opposite direction. This can be a negative effect by rounding off continuous space where the robot is too far from a reference. For example, if error of the robot location $\tilde{\mathbf{p}} = [6, -5]^T$, we treated the error as $\tilde{\mathbf{p}} = [3, -3]^T$ by discretization because discretized grid size is $[-3, 3]^2$. Namely, the model fails to compute values around boundary of discretized space.

3) Model III: using broadcasting and sparse matrix: Trajectories of model II as shown in Figure 6. Apparently, its performance is poor. However, since it converges quickly and its trajectory always has a similar pattern, we think there is some implementation bug of vectorization and this bug is the main reason for the poor performance.

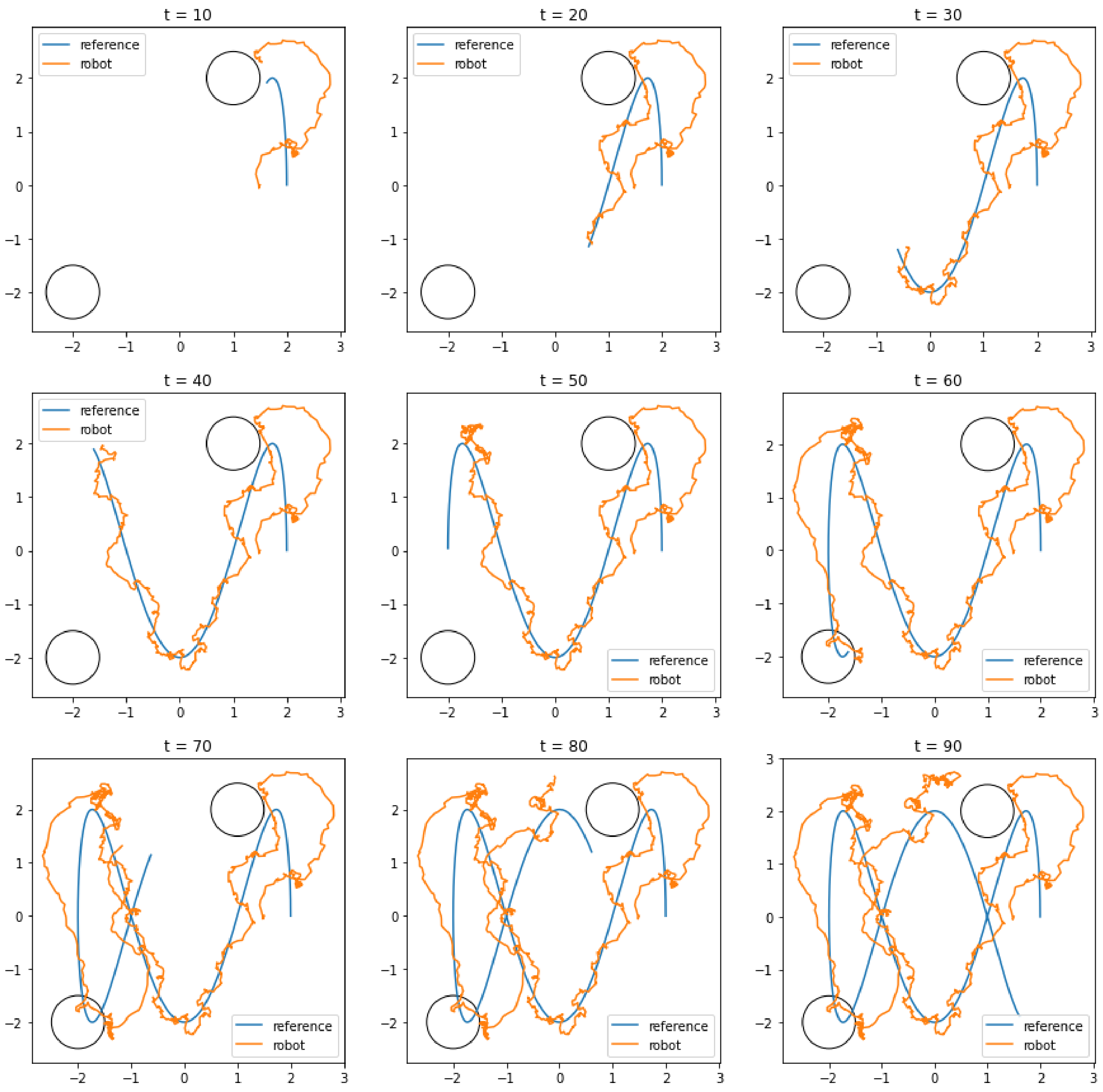

Fig. 3. Trajectories by Receded-horizon CEC

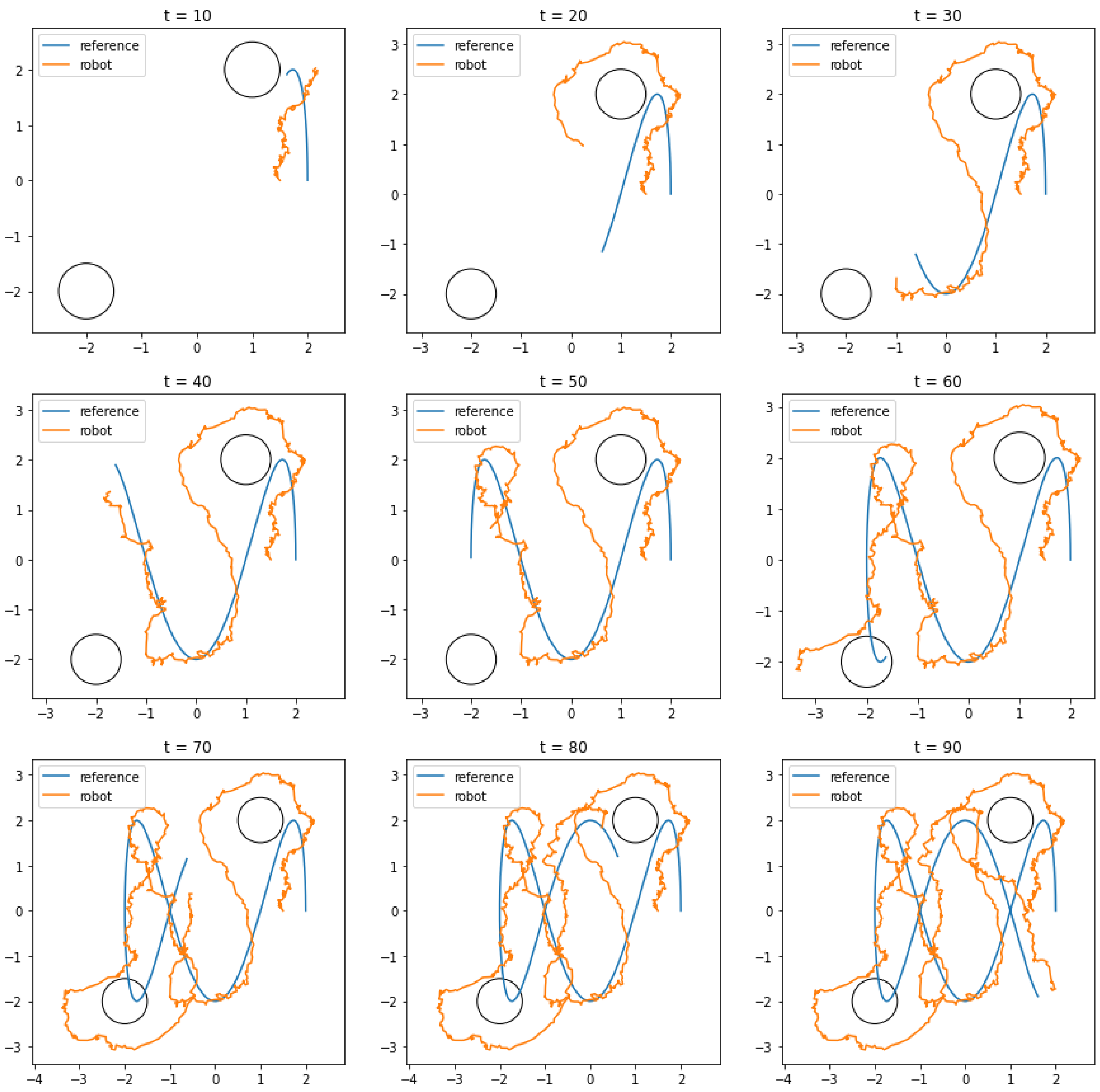

Fig. 4. Trajectories by Model I: computing only mean density

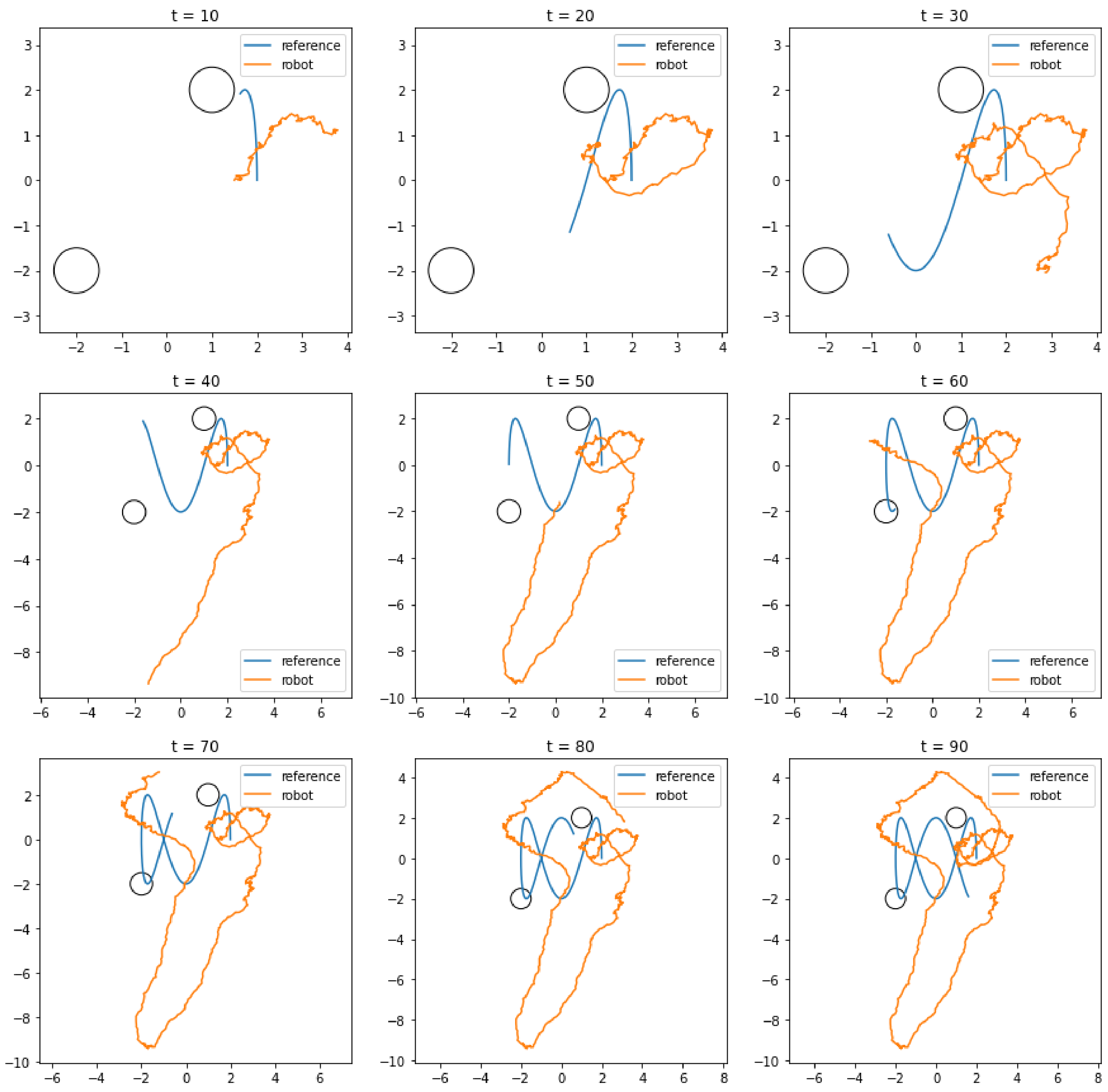

Fig. 5. Trajectories by Model II: computing mean and 26 neighboring cells' density

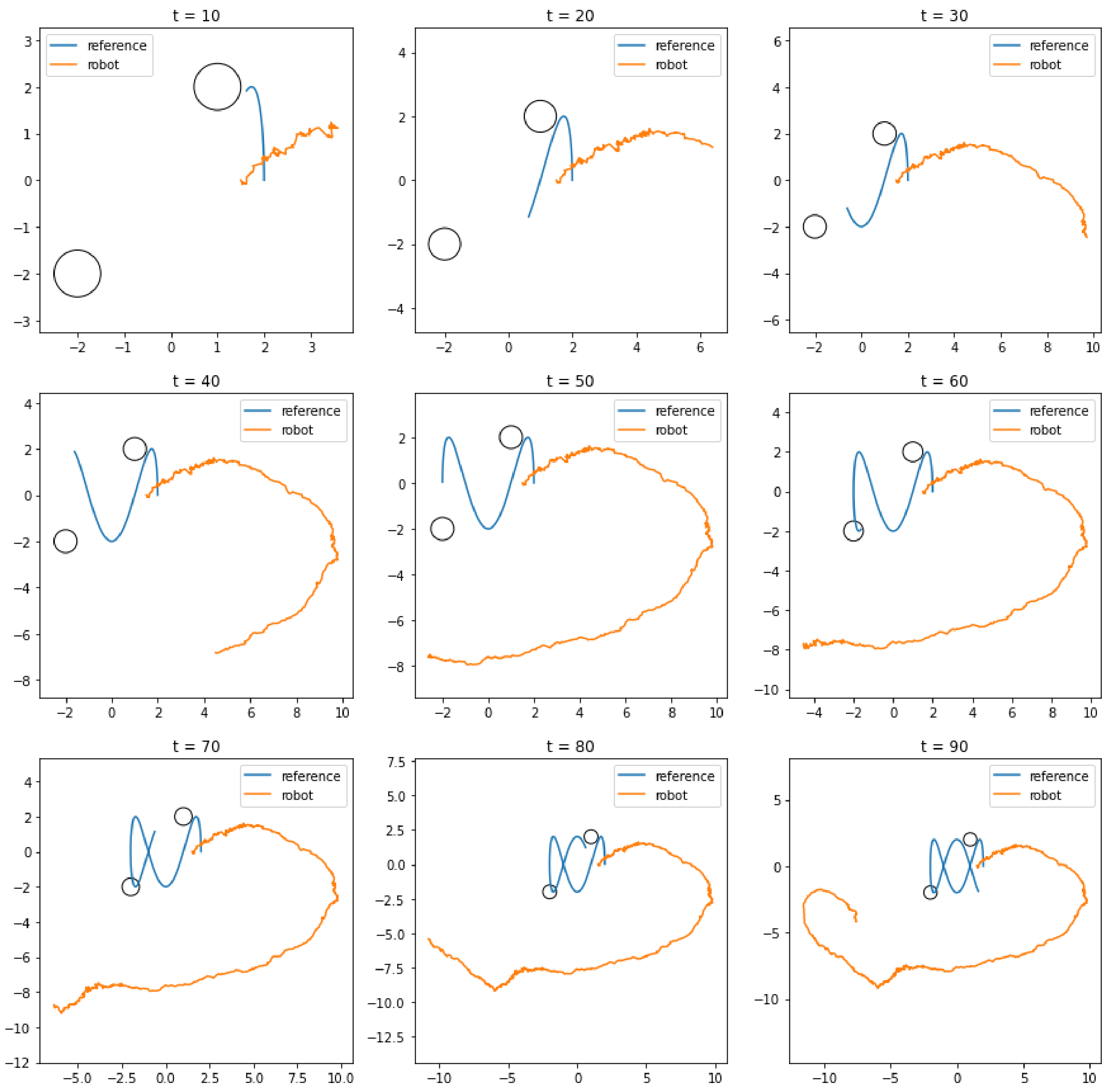

Fig. 6. Trajectories by Model III: using broadcasting and sparse matrix