Chapter 2
Mobile Sensor Trajectory Optimization

2.1 Motivation and the Application Scenario

Thanks to technology advances, large-scale WSNs and mobile WSNs are now more affordable than ever, and WSNs may be useful in many different applications. Many current wired sensor systems in industry are used to monitor lumped parameter systems such as electrical motors. Replacing the wired systems by WSNs can be very attractive, because wireless systems are much easier to install and maintain. In addition, if we deploy a larger number of wireless sensors, the system may be more robust against sensor failures. However, WSNs are not simple replicas of their wired counterparts. There exists a wide class of processes whose behaviors are described by PDEs due to the inherent spatial and temporal variability of their states. These are commonly referred to as DPSs. Since parameters of a DPS are, as the name suggests, distributed, it makes sense to deploy many sensors to measure the parameters at many different positions. This many result in a fine grid and more precise observation than traditional wired sensing systems, which are normally much smaller in scale due to the difficulties of wired connections. Thus, it may be useful to apply WSNs in a variety of scenarios, including the following applications:

- Wildfire monitoring [4]
- Landslide prediction [106]
- Volcano status monitoring [35]
- Diffusive pollution monitoring and control [84, 85]
- Water pollution monitoring [24]
- Chemical plume tracking [107]

2.2 System Identification for DPSs

Currently, DPSs occupy an important place in control and systems theories [108, 109, 110, 111, 112, 113]. One of the basic and most important questions in DPSs is

parameter estimation, which refers to the determination of unknown parameters in the system model from observed data such that the predicted response of the model is close, in some well-defined sense, to the process observations. For that purpose, the system behavior or response is observed with the aid of some suitable collection of discrete sensors, which reside at predefined spatial locations. However, the resulting measurements are incomplete in the sense that the entire spatial state profile is not available. Moreover, the measurements are inexact by virtue of inherent errors of measurement associated with sensing systems and also because of the measurement environment. These factors lead to the question of where to locate sensors to ensure that the information content of the resulting outputs with respect to the distributed state and PDE model is as good as possible.

It is widely accepted that making use of sensors placed in an "intelligent" manner may lead to dramatic gains in the achievable accuracy of the resulting parameter estimates, so efficient sensor location strategies are highly desirable. In turn, the complexity of the sensor location problem implies that there are very few sensor placement methods that are readily applicable to practical situations and these are not well known among researchers. This generates a keen interest in the potential results, as the motivation to study the sensor location problem stems from practical engineering issues. Optimization of air quality monitoring networks is one such issue of interest. One of the tasks of environmental protection systems is to forecast expected levels of pollutant concentrations. To produce such a forecast, a smog prediction model is necessary, which is usually presented in the form of a PDE. As more sensor measurements unavoidably introduce higher energy costs and thereby increase the maintenance budget, we are faced with the problem of how to optimize their locations to obtain the most precise model with a limited number of sensors. Other stimulating applications include, among other things, groundwater modeling, recovery of valuable minerals and hydrocarbons from underground permeable reservoirs, gathering measurement data for calibration of mathematical models used in meteorology and oceanography, automated inspection in static and active hazardous environments where trial-and-error sensor planning cannot be used (e.g., in nuclear power plants), and emerging smart material systems.

The sensor placement problem has been considered from various angles, but the results communicated by most authors are limited to the selection of stationary sensor positions [80, 114, 115]. An intuitively clear generalization is to apply sensors that are capable of continuously tracking points, providing at any given moment the best information about the parameters (such a strategy is usually called continuous scanning). However, communications in this field are rather limited. One piece of work [116] considers the determinant of the FIM associated with the parameters to be estimated as a measure of the identification accuracy and looks for an optimal time-dependent measure, rather than for the trajectories themselves. On the other hand, Uciński [80, 115, 117, 118, 119] apart from generalizations of Rafajłwicz's results, develops some computational algorithms based on the FIM. The problem is then reduced to a state-constrained optimal-control one for which solutions are obtained via gradient techniques capable of handling various constraints imposed on sensor motions. Another piece of work [120] attempted properly formulate and

solve the time-optimal problem for moving sensors, which observe the state of a DPS so as to estimate some of its parameters. Notice that the idea of moving observations has also been applied in the context of state estimation [121, 122, 123, 124], but those results can hardly be exploited in the framework considered here as those authors make extensive use of some specific features of the problem addressed (e.g., the linear dependence of the current state on the initial state for linear systems).

It should be emphasized that technological advances in communication systems and the growing ease in making small, low-power, and inexpensive mobile systems now make it feasible to deploy a group of networked vehicles in a number of environments [9,24,121,125,126]. A cooperative and scalable network of vehicles, each equipped with a single sensor, has the potential to substantially improve the performance of the observation systems. Applications in various fields of research are being developed and interesting ongoing projects include extensive experimentation based on testbeds. The problem to be discussed in this chapter caught our attention while working on our MAS-net experimental platforms [83, 84, 85, 86, 87, 88].

The MAS-net project is proposed to combine the latest sensor network technologies with mobile robotics for an application-oriented high-level task, namely, characterization, estimation, and control of an undesired diffusion process by networked mobile actuators and sensors. One potential solution is to estimate the parameters in a "closed-loop" or "online" approach [127]. This idea can be explained as follows. With the guessed initial values of the unknown parameters, the system starts to drive sensors in an "optimal" trajectory with respect to the parameters. Sensor data are then collected while the sensors are moving. Using the collected data, parameter estimates are improved and the moving sensor trajectories are then updated accordingly. Then, the sensors are driven to follow the newly updated trajectories based on the parameters estimated. Through this "closed-loop" iteration or recursive online adaptation, the estimated parameters converge to the true values of the DPS. This so-called "online" mode was listed as one of the important future research efforts [127].

In this chapter, we focus on the "control for sensing" part of the procedure; that is, given an estimate of the DPS parameters, how to drive the mobile sensors optimally so that the effect of the sensor noise can be minimized. We present a numerical solution for a mobile sensor motion trajectory scheduling problem under nonholonomic constraints as in MASmotes [86], the two-wheeled differentially driven mobile robots, in our MAS-net project. More details about the project are presented in Chap. 1.

From the theoretical perspective, the key challenge is to develop real-time parameter estimation and state estimation of a class of DPSs by a swarm of mobile sensors with nonholonomic constraints and limited communication capability. In addition, mobile actuators (e.g., a mobile robot equipped with a chemical neutralizer dispenser or sprayer) with the same nonholonomic constraints will be added to control the DPS (basically, to reduce the concentration) with the help of the mobile sensors.

The model-based adaptive measurement and control problem in the MAS-net project is formulated in our work [84, 85]. To implement this distributed control

system, the parameter estimation for the DPS is required, and the choice of the best experimental conditions for that purpose is referred to as an OED problem [128, 129, 130, 131].

Recently, the dynamic-sensor-motion scheduling problem has been studied intensively, with many practical considerations such as robust design and collision avoidance [80, 127] using kinematic sensor models. In this chapter, we extend the work in [80] by introducing realistic robot mechanical configurations and solving the problem with RIOTS, a MATLAB® optimal control toolbox.

The rest of this chapter is organized as follows. The formulation of the MAS-net estimation problem is described in Sect. 2.3, in which the dynamic model for differentially driven mobile robots is presented in Sect. 2.3.1 and a module for the diffusion process is presented in Sect. 2.3.2. The objective function for the optimal sensor motion scheduling is described in Sect. 2.3.3. Section 2.3.4 reformulates the problem in the framework of optimal control. In Sect. 2.4, a numerical solution procedure for this problem is presented. The RIOTS [132], a MATLAB® optimal control solver, is described briefly in Sect. 2.4.1 and Sect. 2.4.2 describes a method to incorporate the MATLAB® Partial Differential Equation Toolbox™ [133] and RIOTS. Some illustrative simulation results are presented in Sect. 2.5 with remarks on the results obtained. Section 2.6 concludes this chapter. Further comments on the implementation of the simulation are presented in Appendix A.

2.3 Problem Formulation

2.3.1 The Dynamic Model of Differentially Driven Robots

MASmote [86] is a differentially driven ground mobile robot as illustrated in Fig. 2.1. Its dynamic model can be described by (2.1), where the symbols are defined as follows:

- m: the weight of the robot
- I: the inertia of the robot along the z axis. Note that I is a scalar
- l: the length of the robot's axis
- r: wheel radius. The left and right wheels have the same radius
- b: the edge length of the robot's square chassis. It is assumed that the wheels and the axis are mounted on the square chassis
- α: the yaw angle as shown in Fig. 2.1
- (x, y): the coordinate of the center of the axis
- τ_l,τ_r: the torque applied on the left and right wheel, respectively

$$\begin{bmatrix} m & 0 & 0 \\ 0 & m & 0 \\ 0 & 0 & I \end{bmatrix} \begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{\alpha} \end{bmatrix} + \begin{bmatrix} 2b & 0 & 0 \\ 0 & 2b & 0 \\ 0 & 0 & bl^2/2 \end{bmatrix} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} r\cos(\alpha) & r\cos(\alpha) \\ r\sin(\alpha) & r\sin(\alpha) \\ -rl/2 & rl/2 \end{bmatrix} \begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix}. \quad (2.1)$$

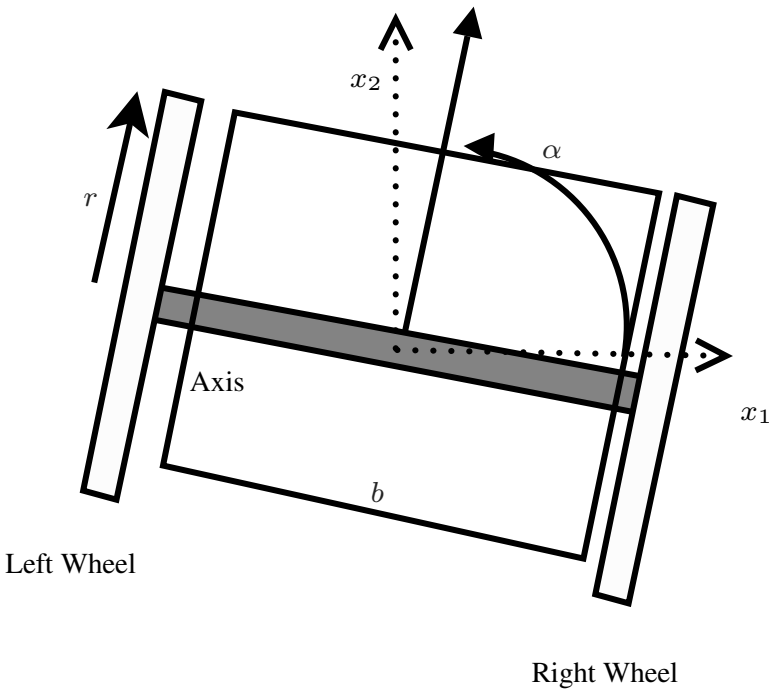

Fig. 2.1 A differentially driven mobile robot

In (2.1), the mobile robot is represented in the form of a second-order system. For convenience, the corresponding state space form can be easily derived by introducing $\mathbf{x}$, the extended system state vector, defined as

$$\mathbf{x} := \begin{bmatrix} x \\ y \\ \alpha \\ \dot{x} \\ \dot{y} \\ \dot{\alpha} \end{bmatrix},$$

and τ is defined as

$$\tau = \begin{bmatrix} \tau_l \\ \tau_r \end{bmatrix}.$$

Note that $\mathbf{x} \neq x$. $\mathbf{x}$ is the state vector, while x is the robot's position on the x-axis. In this chapter, $\mathbf{x}$ is always a function of time, t, so are the state variables, i.e., x, y, α, $\dot{x}$, $\dot{y}$, and $\dot{\alpha}$. For simplicity, the time indices of these state variables are frequently dropped, e.g., $\mathbf{x}(t)$ and $\mathbf{x}$ are interchangeable in this chapter.

To have a compact notation, let us define matrices $\mathbf{A}$ and $\mathbf{B}$ as

$$\mathbf{A} := \begin{bmatrix} 0\,0\,0 & 1 & 0 & 0 \\ 0\,0\,0 & 0 & 1 & 0 \\ 0\,0\,0 & 0 & 0 & 1 \\ 0\,0\,0 & -2b/m & 0 & 0 \\ 0\,0\,0 & 0 & -2b/m & 0 \\ 0\,0\,0 & 0 & 0 & -bl^2/(2I) \end{bmatrix},$$

and

$$\mathbf{B}(\mathbf{x}) := \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ r\cos(\alpha)/m & r\cos(\alpha)/m \\ r\sin(\alpha)/m & r\sin(\alpha)/m \\ -rl/(2I) & rl/(2I) \end{bmatrix}.$$

Thus, the robot dynamics can be written as

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}(\mathbf{x})\tau. \tag{2.2}$$

For simplicity, $\mathbf{B}(\mathbf{x})$ is denoted $\mathbf{B}$ in this chapter.

In the following example, we assume that there are three robots in the network. It is easy to generalize the result to a network with an arbitrary number of robotic sensor nodes. To solve the multi-robot-motion-scheduling problems described in Sect. 2.5, we need to write the dynamics of three robots as a single dynamic system. Denote the states of each robot in (2.2) as $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$, respectively. After defining

$$\mathbf{x}_T := \begin{bmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \mathbf{x}_3 \end{bmatrix}, \quad \mathbf{A}_T = \begin{bmatrix} \mathbf{A}_1 & 0 & 0 \\ 0 & \mathbf{A}_2 & 0 \\ 0 & 0 & \mathbf{A}_3 \end{bmatrix},$$

$$\mathbf{B}_T = \begin{bmatrix} \mathbf{B}_1 & 0 & 0 \\ 0 & \mathbf{B}_2 & 0 \\ 0 & 0 & \mathbf{B}_3 \end{bmatrix}, \text{ and } \tau_T = \begin{bmatrix} \tau_1 \\ \tau_2 \\ \tau_3 \end{bmatrix},$$

where $\mathbf{A}_j, \mathbf{B}_j$ are for the jth robot, the dynamics of all three robots can be written compactly as follows:

$$\dot{\mathbf{x}}_T = \mathbf{A}_T\mathbf{x}_T + \mathbf{B}_T\tau_T. \tag{2.3}$$

2.3.2 The Model of the Diffusion Process

For the purpose of comparison, here we use the same diffusion system model as in Example 4.1 of book [80]. We rewrite it using our notation in the following form:

$$\frac{\partial u(x,y,t)}{\partial t} = \frac{\partial}{\partial x}\left[\kappa(x,y)\frac{\partial u(x,y,t)}{\partial x}\right]$$

$$
\begin{aligned}
&+ \frac{\partial}{\partial y}\left[\kappa(x,y)\frac{\partial u(x,y,t)}{\partial y}\right] \\
&+ 20\exp(-50(x-t)^2), \\
&(x,y) \in \Omega = (0,1)\times(0,1), t \in T, \\
u(x,y,0) &= 0, \quad (x,y) \in \Omega, \\
u(x,y,t) &= 0, \quad (x,y,t) \in \partial\Omega \times T, \\
T &:= \{t | t \in (0,1)\}, \\
\kappa(x,y) &= c_1 + c_2 x + c_3 y, \\
&c_1 = 0.1,\ c_2 = -0.05,\ c_3 = 0.2,
\end{aligned}
$$

where $u(x, y, t)$ is the concentration of the pollution, (x, y) is the spatial coordinate, c_1, c_2, c_3 are the nominal parameters, and t is the time.

2.3.3 The Objective Function for Sensor Motion Scheduling

In this chapter, the aim of the optimization is to reject sensor noise as much as possible. For the ith mobile sensor, its observation is assumed to be the following:

$$z_i(t) = u(\mathbf{x}_i(t), t) + \epsilon(\mathbf{x}_i(t), t),$$

where ϵ is white noise with statistics

$$E\{\epsilon(x,y,t)\} = 0,$$

$$E\{\epsilon(x,y,t)\epsilon(x^*,y^*,t^*)\} = \sigma^2\delta(x-x^*)\delta(y-y^*)\delta(t-t^*).$$

The positions are in the domain of the diffusion process, i.e., $(x, y) \in \Omega$ and $(x^*, y^*) \in \Omega$. δ is the Dirac delta function, and σ is a positive constant.

The objective function is chosen to be the so-called D-optimality criterion defined on the FIM, which will be presented in detail soon. If a measurable random variable x depends on parameter $\mathbf{c}$ and x follows the standard normal distribution, then the FIM [80] is

$$\mathbf{M} = \sum_{j=1}^{n} \int_0^{t_f} p_j \sigma_j^{-2} \left[\frac{\partial u(\mathbf{x}_j(t), t)}{\partial \mathbf{c}}\right] \left[\frac{\partial u(\mathbf{x}_j(t), t)}{\partial \mathbf{c}}\right]^T \mathrm{d}t,$$

where n is the number of sensors; t_f is the duration of the observation; p_j and σ_j are the number of sensors at the position $\mathbf{x}_j(t)$ and the standard deviation of sensor j's noise, respectively; $u(\mathbf{x}_j(t), t)$ is the expected sensor measurement under the noise-free scenario on the time instance t and at the position $\mathbf{x}_j(t)$; the column vector $\mathbf{c}$ is the parameters of interest in the diffusion model. The derivatives of a scalar with respect to a column vector is defined as

$$\frac{\partial u}{\partial \mathbf{c}} = \begin{bmatrix} \frac{\partial u}{\partial c_1} \\ \frac{\partial u}{\partial c_2} \\ \vdots \end{bmatrix},$$

where $u \in \mathbb{R}$, $\mathbf{c} \in \mathbb{R}^m$, and $\mathbf{c} = [c_1, c_2, \cdots, c_m]^T$.

In this chapter, we assume that the sensors are identical, hence $\sigma_i = \sigma_j$ for $i, j \in [1, 2, \cdots, n]$. In addition, no two sensors are placed at the same position, i.e., $p_i = p_j = 1$ for $i, j \in [1, 2, \cdots, n]$. The constant values p_j and σ_j are ignored in the following, since they do not affect the optimization process.

Up to a constant multiplier, the FIM constitutes the inverse of the covariance matrix for the LS estimator, defined by the following criterion:

$$J_1(\mathbf{c}) = \frac{1}{2} \int_0^{t_f} \|z(t) - \hat{u}(\mathbf{x}, t; \mathbf{c})\|^2 \mathrm{d}t. \tag{2.4}$$

The notation $\hat{}$ in (2.4) indicates the predicted value. For n robots, $J_1(c)$ becomes

$$J_1(\mathbf{c}) = \sum_{j=1}^{n} \frac{1}{2} \int_0^{t_f} \|z_j(t) - \hat{u}_j(\mathbf{x}, t; \mathbf{c})\|^2 \mathrm{d}t.$$

Then, the FIM of n robots is defined as follows:

$$\mathbf{M} = \sum_{j=1}^{n} \int_0^{t_f} \left[\frac{\partial u(\mathbf{x}_j(t), t)}{\partial \mathbf{c}}\right] \left[\frac{\partial u(\mathbf{x}_j(t), t)}{\partial \mathbf{c}}\right]^T \mathrm{d}t, \tag{2.5}$$

where the vector $\mathbf{c}$ is defined as follows in this particular diffusion case:

$$\mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}.$$

Note that $\mathbf{x}_j$ is the state vector of the jth robot. Here, $\mathbf{c}$ is the parameter vector in the DPS to be identified, and the partial derivatives are evaluated at $\mathbf{c} = \mathbf{c}_0$, a preliminary estimate of $\mathbf{c}$.

Note that the FIM, $\mathbf{M}$, is a matrix. Thus, there are many metrics that can be defined to indicate the "size" of the matrix. The D-optimality criterion [134] used in this chapter is defined as

$$\Psi(\mathbf{M}) = -\ln \det(\mathbf{M}).$$

Other optimality criteria are applicable but not discussed in this chapter. The comparisons among different criteria are presented in Chap. 3.

The objective function for the MAS-net estimation problem is to minimize $J_2(\mathbf{x}) = \Psi(\mathbf{M})$. Our goal here is to find the optimal control function $\tau \in L_\infty^{2n}[t_0, t_f]$ for n two-wheel differentially driven mobile sensors together with the initial states

$\mathbf{x}(t_0) = \xi \in \mathbb{R}^K$ where $K = 6n$ and $t \in [t_0, t_f] = [0, 1]$, such that $J_2(\mathbf{x})$ is minimized. The notation $L_\infty^n[t_0, t_f]$ represents the space of Lebesgue measurable and bounded functions $[a, b] \to \mathbb{R}^m$.

2.3.4 Problem Reformulation in the Optimal Control Framework

We use RIOTS to solve the proposed problem. More details on RIOTS will be presented later. According to the general optimal control problem formulation in RIOTS [135], our optimal mobile sensor motion scheduling problem can be formulated as follows:

$$\min_{(\tau,\xi)\in L_\infty^{2n}[t_0,t_f]\times\mathbb{R}^K} J(\tau, \xi), \tag{2.6}$$

where

$$J(\tau, \xi) = g_0(\xi, \mathbf{x}(t_f)) + \int_{t_0}^{t_f} l_o(t, \mathbf{x}, \tau)\mathrm{d}t$$

is subject to the following conditions and constraints:

$$\begin{aligned}
&\dot{\mathbf{x}} = h(t, \mathbf{x}, \tau),\\
&\mathbf{x}(t_0) = \xi, \quad t \in [t_0, t_f],\\
&\tau_{j,\min}(t) \le \tau_j(t) \le \tau_{j,\max}(t),\ j = 1, \cdots, n, t \in [t_0, t_f],\\
&\xi_{j,\min}(t) \le \xi_j(t) \le \xi_{j,\max}(t),\ j = 1, \cdots, k, t \in [t_0, t_f],\\
&l_{ti}(t, \mathbf{x}(t), \tau(t)) \le 0, \quad t \in [t_0, t_f],\\
&g_{ei}(\xi, \mathbf{x}(t_f)) \le 0, \quad g_{ee}(\xi, \mathbf{x}(t_f)) = 0,
\end{aligned}$$

where $\tau_{j,\min}(t), \tau_{j,\max}(t), \xi_{j,\min}(t), \xi_{j,\max}(t)$ are upper or lower bounds; $l_{ti}(\cdot)$, $g_{ei}(\cdot)$, $g_{ee}(\cdot)$ are equality or inequality constraints. For our optimal motion scheduling problem, $\dot{\mathbf{x}} = h(t, \mathbf{x}, \tau) = \mathbf{A}\mathbf{x} + \mathbf{B}\tau$ for the single robot case and for the three-robot case $\dot{\mathbf{x}}_T = h(t, \mathbf{x}_T, \tau_T) = \mathbf{A}_T\mathbf{x}_T + \mathbf{B}_T\tau_T$. Here, we define $l_0(\xi, \mathbf{x}(t_f)) = 0$ and $g_0(\xi, \mathbf{x}(t_f)) = \Psi(\mathbf{M})$ to simplify the numerical computation. This technique is called solving an "equivalent Mayer problem" [136]. To understand the equivalent Mayer problem, let us start from the definition of some new notation. $g(\mathbf{x}_i)$ is called the sensitivity function, where

$$g(\mathbf{x}_i, t) := \left[\frac{\partial u(\mathbf{x}_i, t)}{\partial \mathbf{c}}\right]^T.$$

Then, the FIM in (2.5) is

$$\mathbf{M} = \sum_{j=1}^{n} \int_{t_0}^{t_f} g(\mathbf{x}_j(t), t) g^T(\mathbf{x}_j(t), t)\mathrm{d}t. \tag{2.7}$$

Define the Mayer states as

$$\chi_{(i,j)}(t) := \int_{t_0}^{t} \varpi_{(i,j)}(\tau)\mathrm{d}\tau, \tag{2.8}$$

where

$$\varpi_{(i,j)}(t) := \sum_{l=1}^{n} \mathbf{g}_{(i)}(\mathbf{x}_l(t), t)\mathbf{g}_{(j)}(\mathbf{x}_l(t), t).$$

Therefore, $\varpi(t)$ is a matrix, so is $\chi(t)$. Denote $\chi_{\mathrm{dl}}(t)$ the stack vector which stacks all the entries on the diagonal and below the diagonal of the matrix $\chi(t)$ to a vector. For example, if $\chi(t)$ is a 2×2 matrix,

$$\chi_{\mathrm{dl}}(t) = \begin{bmatrix} \chi_{(1,1)}(t) \\ \chi_{(2,1)}(t) \\ \chi_{(2,2)}(t) \end{bmatrix}.$$

Then, the extended Mayer state vector $\tilde{\mathbf{x}}(t)$ can be expressed as

$$\tilde{\mathbf{x}}(t) := \begin{bmatrix} \mathbf{x}(t) \\ \chi_{\mathrm{dl}}(t) \end{bmatrix}.$$

Comparing (2.7) and (2.8), one can easily observe the key idea. That is, $\chi(t_f) = \mathbf{M}$ and $\chi_{\mathrm{dl}}(t)$ contains all the information of $\mathbf{M}$ since $\mathbf{M}$ is symmetric. Ignoring the time indices, after replacing the extended state vector $\mathbf{x}$ with the extended Mayer vector $\tilde{\mathbf{x}}$, we can get $\mathbf{M}$ without explicit integration.

Thus, when considering the equivalent Mayer problem, the models used for RIOTS are as follows:

$$\dot{\tilde{\mathbf{x}}} = \begin{bmatrix} A\mathbf{x} + B\tau \\ \varpi_{\mathrm{dl}} \end{bmatrix},$$
$$\dot{\tilde{\mathbf{x}}}_T = \begin{bmatrix} \mathbf{A}_T\mathbf{x} + \mathbf{B}_T\tau_T \\ \varpi_{\mathrm{dl}} \end{bmatrix}.$$

2.4 Finding a Numerical Solution for the Problem

2.4.1 A Brief Introduction to RIOTS

RIOTS stands for "recursive integration optimal trajectory solver." It is a MATLAB® toolbox designed to solve a very broad class of optimal control problems as defined in (2.6). When executing under MATLAB® script mode, the following configuration files need to be provided: `sys_l.m`, `sys_h.m`, `sys_g.m`, `sys_init.m`, `sys_acti.m`. They are the l_o, h, g_o functions in (2.6) and two initial conditions, respectively. Detailed instructions on how to prepare these files and many sample

problems can be found in [135]. The most important function in this optimal control toolbox is `riots`, explained in detail in [132], p.73.

```
[u,x,f,g,lambda2] = riots([x0,{fixed,{x0min,x0max}}],
  u0,t,Umin,Umax, params,[miter,{var,{fd}}],
  ialg,{[eps,epsneq,objrep,bigbnd]},{scaling},
  {disp},{lambda1}).
```

The parameters useful for understanding our numerical experiments here are as follows:

- `x0`: initial values of $\tilde{\mathbf{x}}$
- `fixed`: a vector to specify which entries in `x0` are fixed and which entries are not. Later in Sect. 2.5, results for two configurations are presented by changing `fixed`, which are cases of "fixed initial states" and "unfixed initial states," respectively. For the first case, the robots' initial conditions, $\mathbf{x}_0$, are fixed. For the second case, χ_{dI} is fixed so that the robots start from the optimal starting positions
- `x0min`, `x0man`: bounds of the initial conditions
- `u0`: initial values of the control functions τ
- `t`: time
- `Umin`, `Umax`: bounds for τ

The definitions of other parameters are described in the manual [132].

2.4.2 Using MATLAB® PDE Toolbox Together with RIOTS

The sensitivity function is generated before the function call of `riots` by MATLAB® PDE Toolbox . The procedure for solving the sensitivity function amounts to finding solutions for the following equations:

$$\begin{cases} \dfrac{\partial u}{\partial t} = \nabla \cdot (\kappa \nabla u) + 20 \exp(-50(x_1 - t)^2), \\ \dfrac{\partial g_{(1)}}{\partial t} = \nabla \cdot \nabla u + \nabla \cdot (\kappa \nabla g_{(1)}), \\ \dfrac{\partial g_{(2)}}{\partial t} = \nabla \cdot (x \nabla u) + \nabla \cdot (\kappa \nabla g_{(2)}), \\ \dfrac{\partial g_{(3)}}{\partial t} = \nabla \cdot (y \nabla u) + \nabla \cdot (\kappa \nabla g_{(3)}), \end{cases}$$

where $\nabla = (\partial/\partial x, \partial/\partial y)$. Note that there are three g functions since there are three parameters c_1, c_2, c_3 in Sect. 2.3.2.

2.5 Illustrative Simulations

2.5.1 Differentially Driven and Omnidirectionally Driven

One type of robot model is a simple kinematic model [80]:

$$\begin{bmatrix}\dot{x}(t)\\ \dot{y}(t)\end{bmatrix} = r\omega(t),\ \begin{bmatrix}x(0)\\ y(0)\end{bmatrix} = \begin{bmatrix}x_0\\ y_0\end{bmatrix}, \tag{2.9}$$

where $\omega(t)$ is the angular speed vector, and r is the radius of the wheels. Obviously, (2.9) is an approximation. In this chapter, we refer to a robot that is subject to the kinematic in (2.9) as a proximal "omnidirectionally driven robot" since the velocity can be set arbitrarily. When the robot is differentially driven, we are interested to see the difference in the optimal sensor motion scheduling. The following four cases are compared first:

- Case 1: omnidirectionally-driven robots starting from a fixed initial state vector. In the context of the MAS-net project, the fixed initial positions are the homes of those robots.
- Case 2: differentially driven robots with a fixed given initial state vector. Moreover, we consider two subcases. Case 2a has an initial yaw angle of 15° and case 2b of -15°.
- Case 3: omnidirectionally driven robots without a fixed initial state vector. We assume that the optimal static sensor location problem is solved first, then use this optimal position obtained as the initial state and seek the optimal sensor trajectories.
- Case 4: the same as in case 3 but using differentially driven mobile robots.

According to the above definitions, Fig. 2.2 shows the results for case 1; Fig. 2.3 for case 2a; Fig. 2.4 for case 2b; Fig. 2.5 for case 3; and Fig. 2.6 for case 4. From these figures, we note the following observations:

- Differentially driven robots are less likely to change the orientation. The optimal mobile sensor trajectories in cases 2 and 4 have smaller curvatures compared with that in cases 1 and 3.
- No matter what the robot dynamics are, the robots tend to move along the same rough direction. This can be observed by comparing cases 1, 2a, 2b and cases 3, 4.
- For multi-robot cases, the final positions of the robots tend to be evenly distributed. A comparison of Fig. 2.3 and Fig. 2.4 is especially interesting. The two figures support each other, as the trend of the trajectories tend to align with each other.

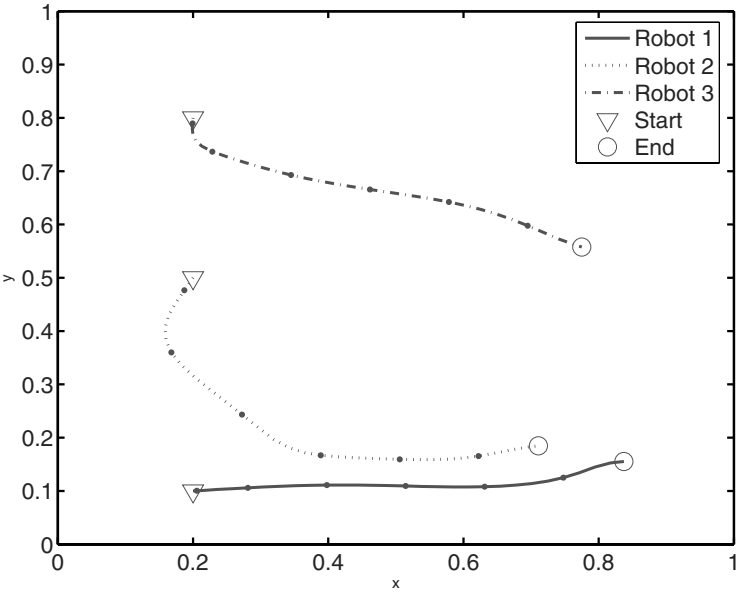

Fig. 2.2 The optimal sensor trajectories of omnidirectionally driven robots (case 1)

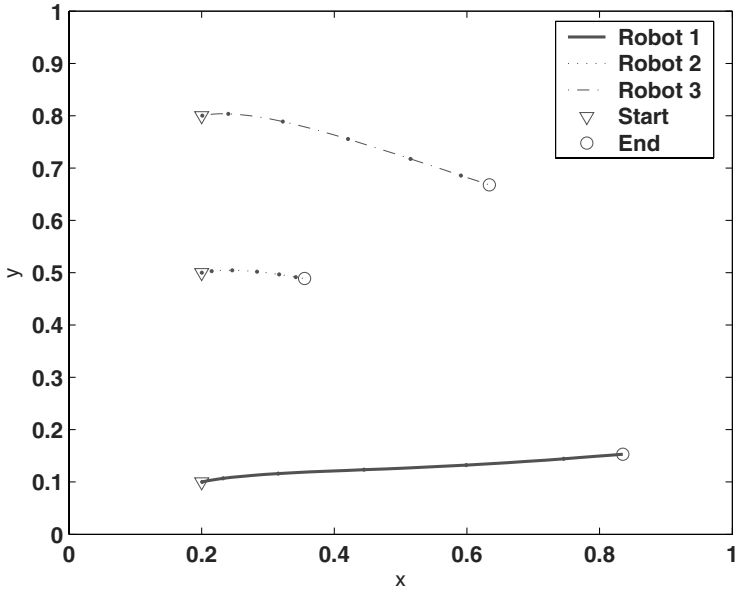

Fig. 2.3 The optimal sensor trajectories of differentially driven robots: 15° initial yaw angle (case 2a)

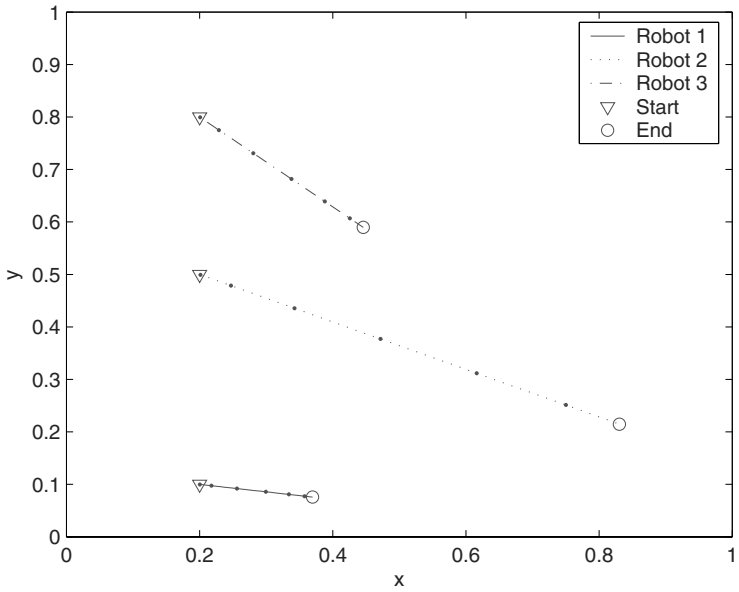

Fig. 2.4 The optimal sensor trajectories of differentially driven robots: -15° initial yaw angle (case 2b)

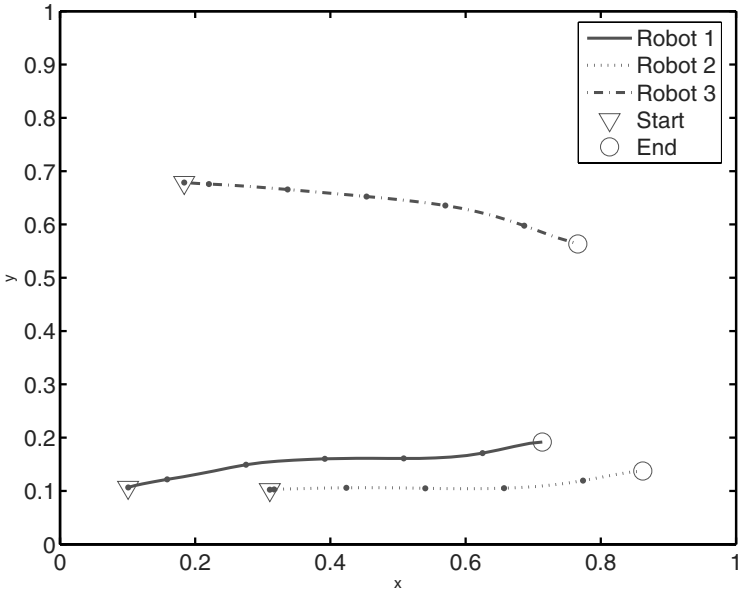

Fig. 2.5 The optimal sensor trajectories of omnidirectionally driven robots using optimal initial conditions (case 3)

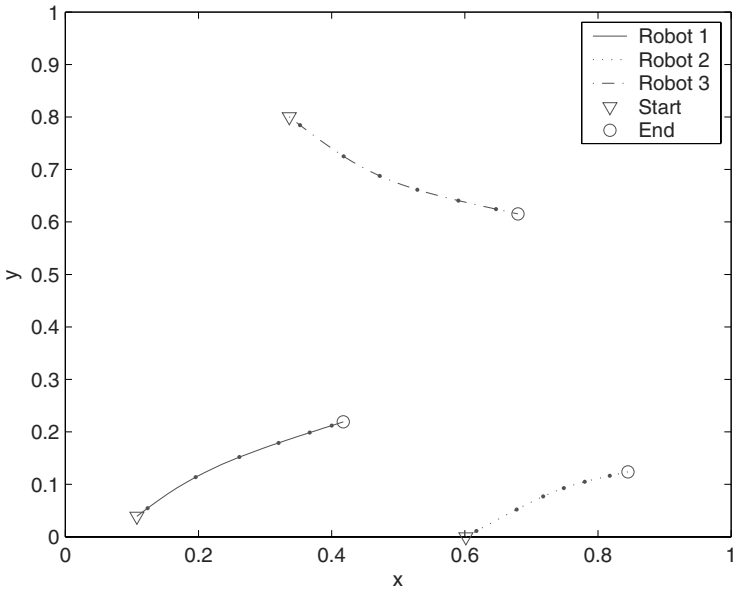

Fig. 2.6 The optimal sensor trajectories of differentially driven robots using optimal initial conditions (case 4)

2.5.2 Comparison of Robots with Different Capabilities

Here we consider two more cases to compare robots with different capabilities.

- Case 5: using a single weak-and-heavy robot, whose weight is 0.5 and the range of its torque for each wheel is ± 10.
- Case 6: using a single strong-and-light robot, whose weight is 0.05 and the range of its torque for each wheel is ± 100.

With the same fixed initial states and the same time interval, the robot in case 5 moves a shorter distance than that in case 6, as seen from Fig. 2.7 and Fig. 2.8. This matches our intuition that it is desirable for the sensors to measure the DPS states at more spatial locations whenever possible. This simulation also implies that mobile sensors are more favorable than static sensors, if the cost issues are ignored. The static sensors can be considered as extremely weak and infinitely heavy robots.

2.5.3 On the Effect of the Initial Orientation

In addition to case 2a and case 2b, the effects of different initial yaw angle is studied in this section. The robots associated with each figure in this section have the same mechanical configurations and the same initial conditions.

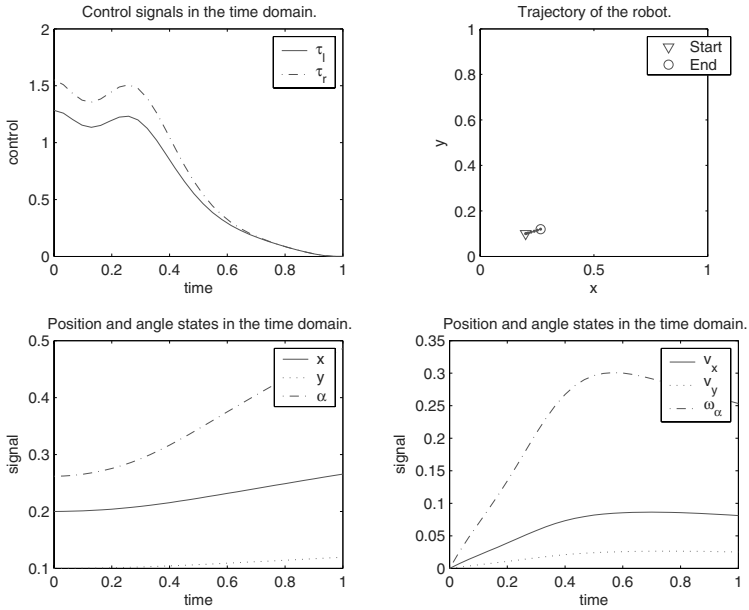

Fig. 2.7 The optimal trajectory of a weak-and-heavy differentially driven robot (case 5)

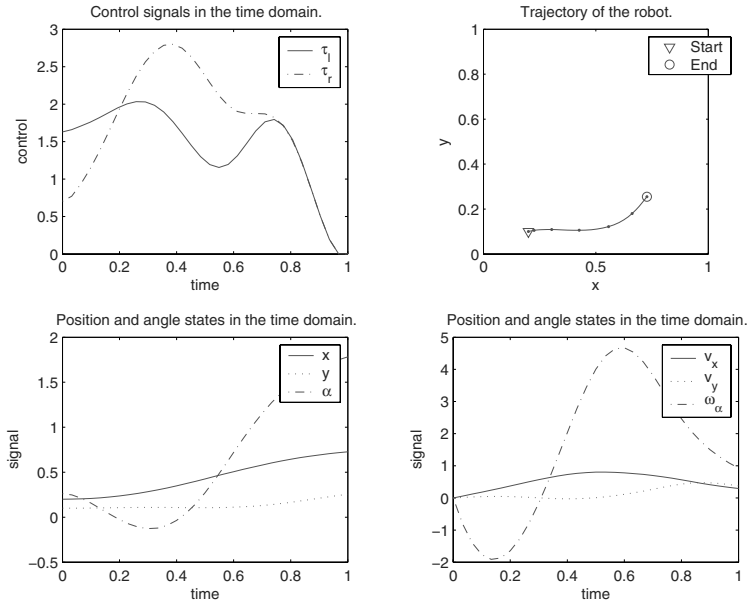

Fig. 2.8 The optimal trajectory of a strong-and-light differentially driven robot: initial yaw angle is 15° (case 6)

Let us compare the following figures:

- Figure 2.3: three robots with 15° initial yaw angle
- Figure 2.4: three robots with -15° initial yaw angle
- Figure 2.8: one robot with 15° initial yaw angle
- Figure 2.9: one robot with -15° initial yaw angle

The initial yaw angle affects the curvature of the optimal trajectory, but does not change the trend of the optimal trajectory. This indicates that the initial yaw angle matters, but is not critical. Figures 2.8 and 2.9 support the above statement. With different initial yaw angles, the two robots starting at the same position have different trajectories, but their final positions are close. For multi-robot cases, the *formation* pattern of the robots tends to be similar.

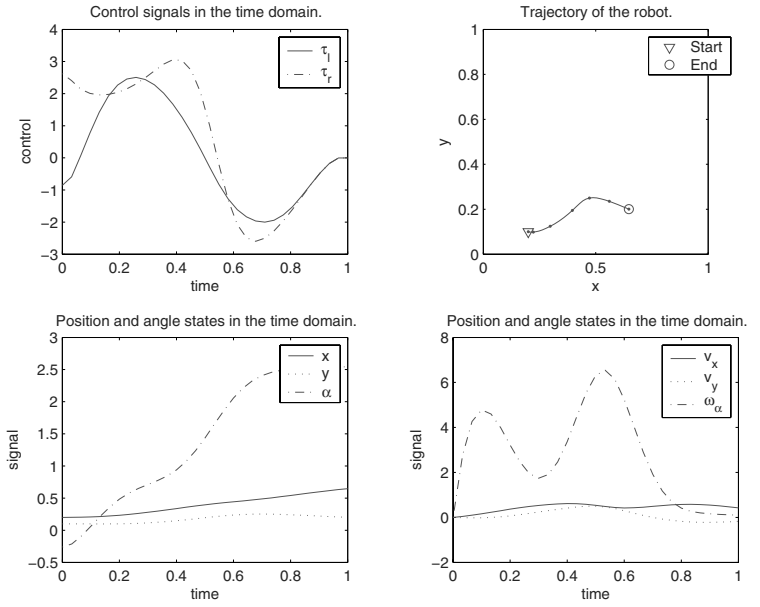

Fig. 2.9 The optimal trajectory of a strong differentially-driven robot: initial yaw angle -15° (case 6)

2.6 Chapter Summary

This chapter presents a numerical procedure for optimal sensor-motion scheduling for diffusing plume observation. Given a DPS with nominal parameters, differentially driven mobile robots move along their optimal trajectories such that the sensor noise effect on the estimation of system parameters is minimized. This optimal

measurement problem is an important module for a potential closed-loop DPS parameter identification algorithm. This chapter reformulates a differentially driven robot dynamic model in the framework of optimal control. Through the combined use of two existing MATLAB® toolboxes for optimal control (RIOTS) and PDEs (MATLAB® Partial Differential Equation Toolbox™), the optimal sensor-motion scheduling problem can be solved numerically. Simulation results and their observations are presented.

Springer

http://www.springer.com/978-1-84882-655-7

Optimal Observation for Cyber-physical Systems
A Fisher-information-matrix-based Approach
Song, Z.; Chen, Y.; Sastry, C.R.; Tas, N.C.
2009, XVIII, 171 p. With online files/update., Hardcover
ISBN: 978-1-84882-655-7