COSMOS: Making Robots and Making Robots Intelligent
Lecture 8: Analysis and simulation of the `Robobrain` model

Jorge Cortés and William B. Dunbar

October 27, 2005

Abstract

In this lecture, we discuss the model for `Robobrain` and analyze its open-loop behavior, i.e., we determine the fixed-points and their stability. We also show motivate the need for feedback control versus open-loop control. The feedback control design of `Robobrain` will be treated in the next lecture.

Contents

1 `Robobrain` model

Recall the `Robobrain` model that we introduced in Lecture 5. To begin, examine the schematic image of an enlarged `Robobrain` robot sitting the middle of a room, given in Figure 1.

From previous discussions, recall that Δ is the sample period and denote the sample times as $t_k = k * \Delta$, $k = 0, 1, 2, \ldots$. Also, x_k is the x position of the robot at time t_k, using the same subscript notation for the other variables. The discrete-time dynamic model of the robot dynamics is given by

$$\begin{cases} x_{k+1} = x_k + \Delta u_k \cos(\theta_k) \\ y_{k+1} = y_k + \Delta u_k \sin(\theta_k) \\ \theta_{k+1} = \theta_k + \Delta v_k \end{cases} \tag{1}$$

Indicated in the figure are the *forward velocity* u and the *turning velocity* v, defined by

$$u = \frac{1}{2} r(\omega_R + \omega_L), \qquad v = \frac{2\pi}{\mathbf{w}} r(\omega_R - \omega_L),$$

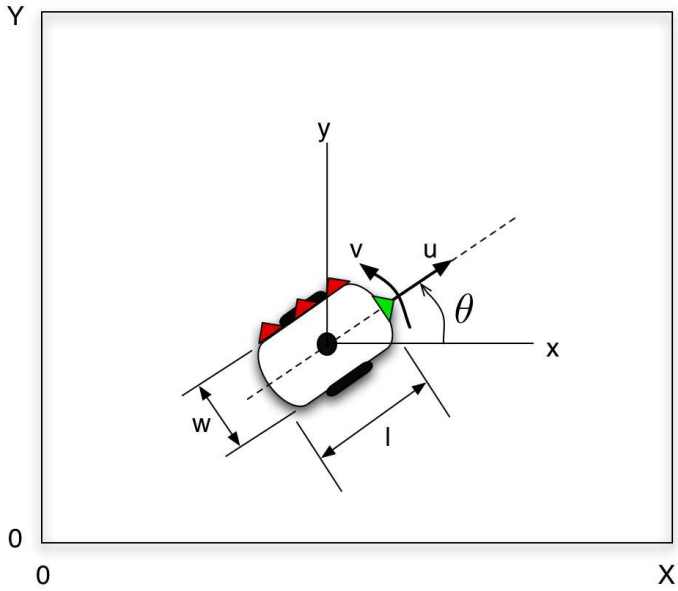

Figure 1: Schematic drawing of an enlarged Robobrain robot operating in a room. The border in the drawing represents the room walls, and the coordinate system shows that the lower left corner is denoted $(x, y) = (0, 0)$ and the upper right corner is denoted $(x, y) = (X, Y)$, with X and Y given by the dimensions of the room.

where **w** is the width of `Robobrain` (the distance between the wheels), and ω_L and ω_R are the angular velocities of the left and right wheels (measured in radians per second).

Task 1.1 Give an example of uncertainty or disturbances that are *not* accounted for in the `Robobrain` model in equation (1).

Task 1.2 In practice, you will be designing u and v, but actually controlling the angular velocities ω_L and ω_R on `Robobrain`. What are the equations for ω_L and ω_R in terms of the controls u and v? Why is it useful to have these formulas?

2 Open-Loop Analysis

In this section, we shall do some analysis on the model of `Robobrain`, such as finding the fixed-points (equilibrium points) and determining their stability.

Task 2.1 What are the equilibrium states $(x_{\text{eq}}, y_{\text{eq}}, \theta_{\text{eq}})$ for the `Robobrain` model in equation (1)? What are the corresponding equilibrium control inputs $(u_{\text{eq}}, v_{\text{eq}})$?

For our purposes, let's define stability as follows:

Definition 2.2 An equilibrium state $(x_{\text{eq}}, y_{\text{eq}}, \theta_{\text{eq}})$ is **(asymptotically) stable** if, by applying the equilibrium control $(u_{\text{eq}}, v_{\text{eq}})$ and starting with (x_0, y_0, θ_0) reasonably close to $(x_{\text{eq}}, y_{\text{eq}}, \theta_{\text{eq}})$, the

state $(x_k, y_k, \theta_k) \to (x_{\text{eq}}, y_{\text{eq}}, \theta_{\text{eq}})$ as k grows larger and larger. In other words, the orbit approaches and eventually reaches the equilibrium state, possibly even in finite time.

Task 2.3 Using your intuition and the previously computed equilibrium values, is any equilibrium state for `Robobrain` stable? Why or why not?

For the purposes of analyzing our control, we will need simulations to determine if our control design gives us what we want, such as stability and good performance.

Task 2.4 Write a MATLAB® function **robobrain.m** that computes the orbit for x_k, y_k and θ_k given the following control signals:

$$u_k = 4, \quad k = 0, 1, 2, ...,$$
$$v_k = 4\sin(2\pi t_k), \quad k = 0, 1, 2, ...$$

Take the initial state to be $(x_0, y_0, \theta_0) = (2, 0, \pi/4)$. Let $\Delta = 0.01$ and compute the orbit for 500 iterations. Create two plots to show your orbit results (you can use the MATLAB® command `subplot` to do this). One plot should show y vs. x, and the other should show θ vs. time (recall that $t_k = \Delta * k$). You should get a plot like the one shown in Figure 2

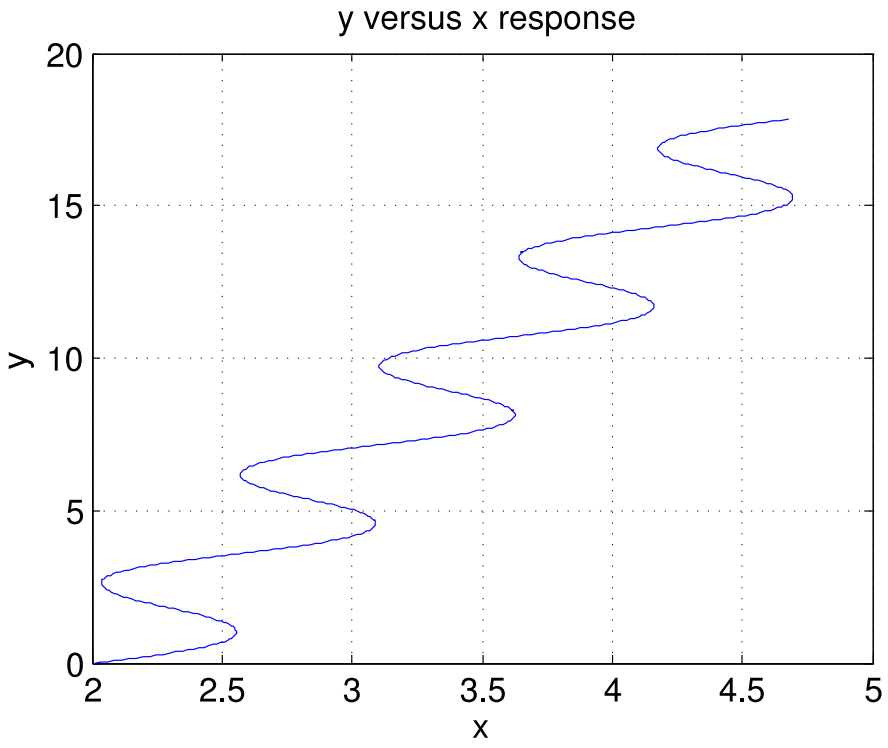

Figure 2: Plot of y vs. x for the simulation conditions stated in Task 2.4.

Note that the control signals are open-loop in the sense that there is no *feedback* from the actual states into the model of `Robobrain`. Instead, we specify our controls ahead of time, without ever

looking during the motion of `Robobrain` to its actual state or trajectory. As you might expect, this has big risks. For one thing, do you think this open-loop strategy will be robust to disturbances? Let's see it. To demonstrate the need for feedback, let's inject a disturbance into the model. Assume that the orbits that you achieved in Task 2.4 are the desired behavior for the robot.

Task 2.5 Write a MATLAB® function **robobrainDist.m** that computes the orbit for $\tilde{x}_k$, $\tilde{y}_k$ and $\tilde{\theta}_k$, where the model for these states is given by

$$\begin{aligned}\tilde{x}_{k+1} &= \tilde{x}_k + \Delta u_k \cos(\tilde{\theta}_k) + 0.1 \sin(\pi \tilde{x}_k / X)\\ \tilde{y}_{k+1} &= \tilde{y}_k + \Delta u_k \sin(\tilde{\theta}_k) + 0.15 \sin(\pi \tilde{y}_k / Y)\\ \tilde{\theta}_{k+1} &= \tilde{\theta}_k + \Delta v_k\end{aligned}$$

Use the same initial condition, control signal and parameter values as in the previous task, and assume $X = Y = 20$. How different are the two orbits? Could there be any physical meaning to this type of uncertainty? How is this example indicative of the need for feedback, given that the desired orbit is the one you obtained with the "disturbance free" model? You should get a plot like the one shown in Figure 3

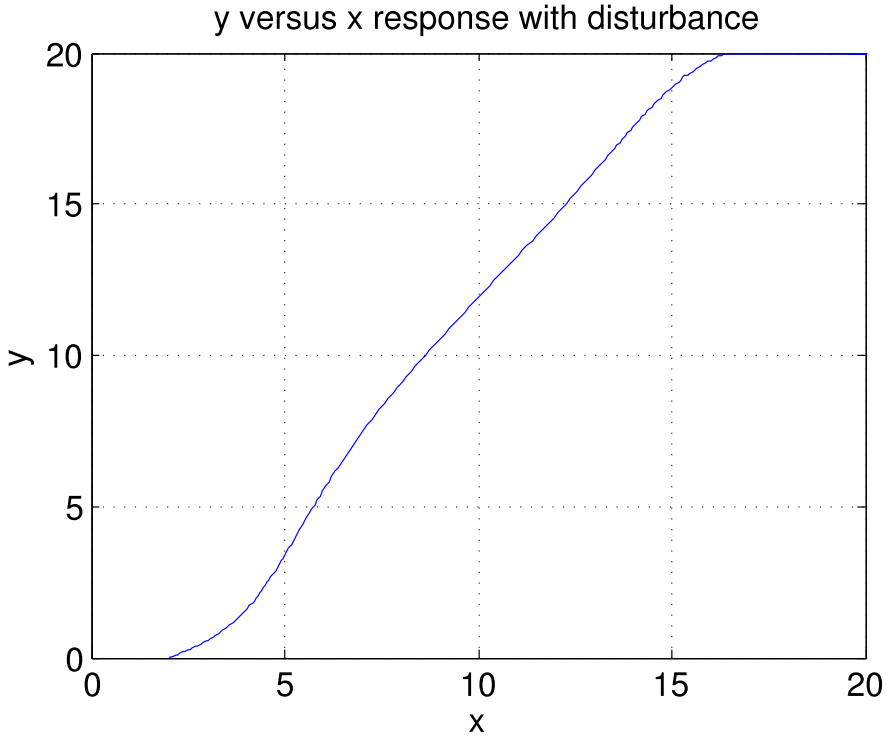

Figure 3: Plot of y vs. x for the simulation conditions stated in Task 2.5.

3 Wall Following Control

The objective of this control design is for the robot to approach any wall in any room or hall way and, upon detecting the wall using IR sensors, turn to follow the wall at a specified separation distance d_{sep}. The robot should achieve wall following keeping the wall on its left.

Each robot is outfitted with four IR sensors, each with range of detection that is a narrow cone in shape. Of the four sensors, two are facing forward and two are facing left. Denote the measured distances to an object from each sensor as follows: $d_{f,l}$ is measured from the front left, $d_{f,r}$ from the front right, $d_{l,b}$ from the left back, and $d_{l,f}$ from the left facing sensor at the front corner of `Robobrain`. An examination of the `Robobrain` platform will clarify where these sensors are placed. From the sensor distance measurements, we define the forward average distance as $d_f = (d_{f,l}+d_{f,r})/2$ and the left average distance as $d_l = (d_{l,b}+d_{l,f})/2$. Since these distance measurements are changing in time, we will define each distance at any update k (at time t_k) by writing $d_{(\cdot)}(t_k)$. So, for example, $d_{l,b}(t_k)$ is the distance from the left back sensor to the wall at time t_k.

In reality, the distance is calculated based on voltage signals from the IR sensor, and these signals are a nonlinear function of the actual distance to the object being detected, requiring calibration.

For simplicity, wall following will be performed in a long hall way. Eventually, "disturbances" will be added in the form of low degree ramps that come out and back from the hall way wall. As the wall is initially approached, up to two IR sensors will detect the wall.

The following algorithms define how the IR distance measurements are used to compute the configuration variables x and θ, assuming a straight long hallway. First, we have the initialization procedure.

Name: `Robobrain` **sensor initialization algorithm**
Goal: Initialize position of `Robobrain` to follow a wall

1: Go forward with control $u = v_{\text{nom}}$ and $v = 0$ until $d_f(t) \approx 2d_{\text{sep}}$. The positive scalar v_{nom} is the nominal wall following velocity, defined to be some percentage of the maximum velocity of a wheel.
2: Turn CW slowly in place with control $u = 0$ and $v = -0.05v_{\text{nom}}$ until $d_{l,f}(t)$ registers wall measurements. Continue to turn slowly, keeping track of the decreasing values of $d_{l,f}(t)$. As soon as $d_{l,f}(t)$ begins to increase, stop.
3: Reset time to be $t_0 = 0$ and define $\theta_0 = \pi/2$ and $x_0 = d_l(t_0)$. Note that $d_l(t_0) = d_{l,b}(t_0) = d_{l,f}(t_0)$.

Task 3.1 How might you rewrite step 2 using $d_{l,b}$ instead of $d_{l,f}$? In step 3, why is it that $d_l(t_0) = d_{l,b}(t_0) = d_{l,f}(t_0)$? Can we determine what y_0 is based on the sensor data? Is there a way to define θ_0 and y_0 and not x_0?

After initialization, we will need the sensors to update the position x_k and the orientation θ_k as k increments in value by $1, 2, 3, \ldots$. These updated states are required for feedback. The following algorithm defines how the states updates are done.

Name: `Robobrain` **sensor update algorithm**
Goal: Initialize position of `Robobrain` to follow a wall

For all $k = 1, 2, 3, ...$, the configurations x_k and θ_k are given as follows:

1: set $\theta_k = \theta_0 + \alpha_k$
2: set $x_k = \frac{1}{2}\left[d_{l,f}(t_k) + d_{l,b}(t_k) + \mathbf{w}\right]\cos(\alpha_k)$, where $\alpha_k = \arctan\left(\frac{d_{l,b}(t_k) - d_{l,f}(t_k)}{\mathbf{l}}\right)$ and $\mathbf{l}$ is the length of `Robobrain`.

Task 3.2 Why are we required to convert IR sensor data into state variable data?

Task 3.3 Write a program that converts the four scalar distance measurements into the x and θ state values using the equations given above. Can anyone derive these equations?