Control for Mobile Robots

Brian Bingham

Revision 5
17 September 2013

Control for Mobile Robots

Preface

These course notes are intended to support and complement the online course *Control of Mobile Robots* developed by Dr. Magnus Egerstedt at the Georgia Institute of Technology. These notes are being developed in the fall semester of 2013 to support a course at the University of Hawaii at Manoa inspired by Dr. Egnerstedt's offering.

The concept of tranlating my course-specific notes into a more general textbook is the result of working with Professor Allen Downey at Olin College. Professor Downey is a prophet of free books and has authored many which he distributes at Green Tea Press (`http://greenteapress.com/`).

If you have suggestions and corrections, please send them to `bsb@hawaii.edu`.

Brian Bingham
Honolulu, HI

Contributors

Part of the fun of developing an open textbook is engaging colleagues and students in improving the content. Below is a list of those that have contributed to the book.

Hurry, be the first to contribute to this book!

Contents

Chapter 1

Linear Systems

1.1 Introduction: Dynamic Models

Our goal is to develop the ability to control dynamic systems, particularly mobile robots. To develop these skills we need to be able to predict how a particular control approach (an algorithm) will perform: Will the control be stable? Will it move the robot fast enough to the right place? Mathematical models provide this predictive capability.

The models that we are concerned with are ones that behave in similar ways to the actual physical systems we wish to control. A good place to start is with models that are linear because they are (relatively) easy to understand and provide a general set of abstractions applicable to a wide variety of physical implementations.

In this chapter we will explore some basic mathematical models and how they can be used to predict the behavior of physical systems. We won't be modeling any mobile robots just yet, but this is the foundation on which we'll build our robot models.

One cautionary note before we get started—all models are wrong. By this we mean that mathematical models are abstract approximations of physical systems. The model never predicts exactly how the actual system will behave under all circumstances. Nevertheless simple models can provide a tremendous amount of insight into how a controller will function enabling us to prototype our ideas using analytical and numerical solutions before we go to the time and trouble of actually implementing a control solution. Just as computer-aided design (CAD) tools allow us to conceive of and refine physical artifacts before we start fabrication, these models allow us to design and test our control solutions before releasing them into the wild.

1.2 First-Order Model

The mathematical models often used for understanding and developing controllers are differential equations. To get started we will consider models that are linear (as opposed to non-linear), ordinary (as opposed to partial) differential equations.

Our first-order model—a first-order, linear, ordinary differential equation—is

$$\frac{dy(t)}{dt} + \frac{1}{\tau}y(t) = f(t), \tag{1.1}$$

where $f(t)$ is the input (forcing function), $y(t)$ is the output (response function) and τ is the time constant. That's it; there is really nothing more to say.

But of course we do have much more to say. We should start by emphasizing that this model is just as wrong as the rest of them, but it can be very useful. There are two reasons that this model is useful:

1. The time-response of this model is sufficiently similar to that of many physical systems to allow us to use this fictitious model (equation) to predict how an actual system might behave.
2. We know (or we will know shortly) how to solve this equation to calculate the output of the model in response to a variety of inputs.

1.2.1 An Example

One example of a physical system that can be approximated by our first-order model is an automobile. Let's consider how the speed of an automobile (the output) responds to changes in the gas petal (the input) as it moves in a constant direction. The sketch if Figure 1.1 illustrates this simplification of a complex physical system. Based on this sketch we can do things like draw a free-

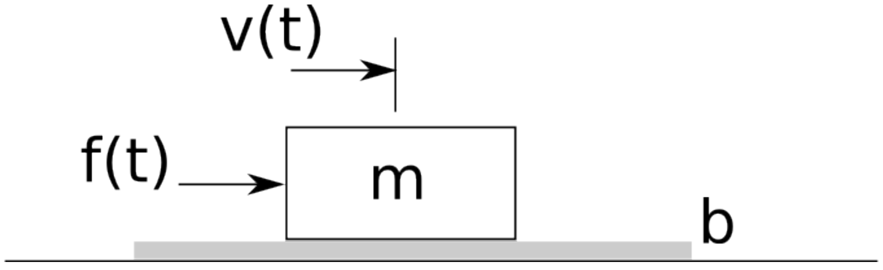

Figure 1.1: Sketch of a simplified automobile model. The point mass (m) moves to the right at a velocity ($v(t)$. The input is a force ($f(t)$) and the drag force ($b(v(t))$ resists the motion.

body-diagram and apply Newtonian principles to arrive at an equation of motion. We might also make an ambitious simplifying assumption that the drag force that resists the motion of the mass is linearly related to the velocity, i.e., $f_{\text{drag}} = b(v(t))$. Then we could come up with an equation of motion

$$m\,(\dot{v}(t)) + b(v(t)) = f(t) \tag{1.2}$$

where m is the mass of the car, $v(t)$ is the speed, $\dot{v}(t)$ is the acceleration, b is the coefficient of linear drag and $f(t)$ is the input force. This equation of motion is our mathematical model This equation has the same form (a first-order, linear, ordinary differential equation) as our first-order model (1.1).

This is meant to be an example of a model that is obviously wrong. A physical automobile is a complex system with many, many degrees of freedom. Our simplified model (1.2) is meant to do just one thing, allow us to predict how, in general, the speed of a car responds to changes in the throttle input.

1.3 Step Response

One useful aspect of our first-order model (1.1) is that the solution to this differential equation is well known for a variety of input functions. A particularly interesting input function is the step input ($\mu(t)$) defined as

$$\mu(t) = \begin{cases} 0 & \text{for}\, t < 0 \\ 1 & \text{for}\, t \geq 0 \end{cases} . \tag{1.3}$$

This function ($\mu(t)$) is also known as the *unit* step function because the amplitude of the step is 1.0. Figure 1.2 illustrates how the value of $\mu(t)$ is zero before $t = 0$ and then instantaneously changes to unity.

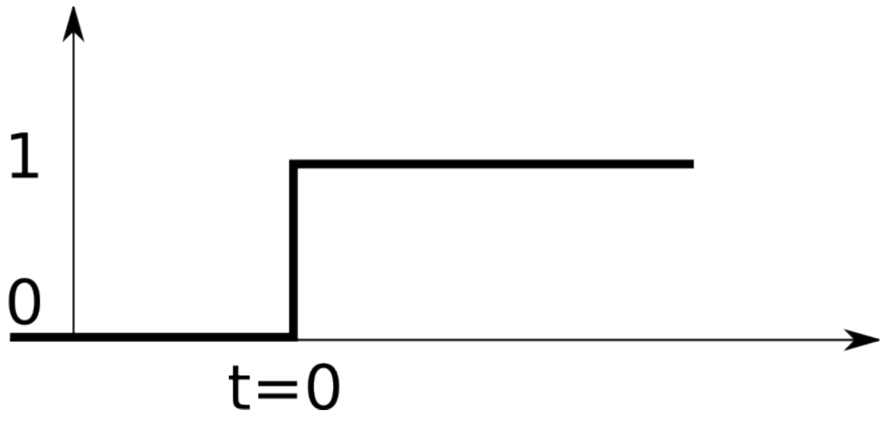

Figure 1.2: Illustration of a step function $\mu(t)$.

Now consider our generic first-order model (1.1) with a step input ($f(t) = A\mu(t)$) and an initial condition of zero ($y(0) = 0$),

$$\frac{dy(t)}{dt} + \frac{1}{\tau}y(t) = A\mu(t). \tag{1.4}$$

(You should notice that the input function ($f(t)$) is a step input with an amplitude of A.) The step response is the solution to this ordinary differential equation which is

$$y(t) = A\tau\left(1 - e^{-t/\tau}\right) \tag{1.5}$$

for $t > 0$. Notice that there are two key parameters of our model step response: the time constant (τ) and the amplitude of the step input (A).

Listing 1.1: Script for first-order response: first_order_response.m

```
% Example of time-response of first-order system
% First Order Model: dy/dt = 1/tau*y = f(t)
tau = 2;  % time constant [s]
% A vector of time values for evaluating the response
```

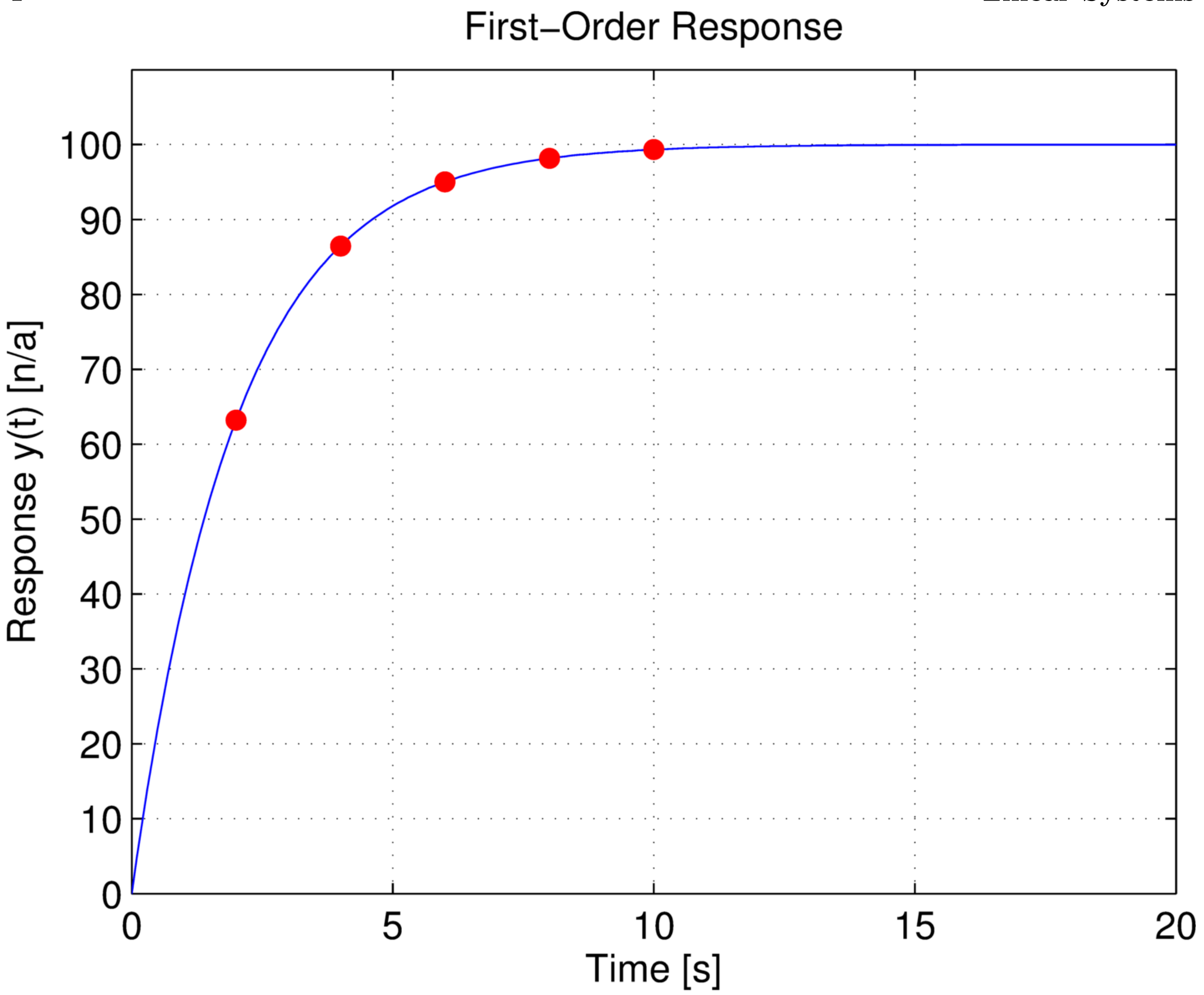

Figure 1.3: Graph of the step response (1.5) with $\tau = 2\,\mathrm{s}$ and $A = 50$. The red dot markers indicate the value of the solution at $t = [\tau, 2\tau, 3\tau, 4\tau, 5\tau]$.

```
t = 0:0.1:20;
% Step Response: f(t)=A*mu(t) , y(0)=0
A = 50;
% Evaluate the solution at each value of t
y = A*tau*(1-exp(-t/tau));
% Plot
figure(1); clf();
plot(t,y,'-')
xlabel('Time [s]')
ylabel('Response y(t) [n/a]')
title('First-Order Response')
grid on
ylim([0 max(y)*1.1])
```

This solution (1.5 is graphed in Figure 1.3 using the MATLAB script in Listing 1.1. This time response to a step input (or step response for short) has a characteristic shape as it rises and approaches the steady-state response exponentially. One important aspect of the response is that it approaches a new constant value, the steady-state response. A simple way to find the steady state value is to set $\dot{y}(t) = 0$ in (1.1). Since we are looking for the "steady-state" value we can anticipate that all rates of change (derivatives) will be zero. Now the steady-state value ($y_{\rm ss}$) is

$$y_{\rm ss} = \tau A. \tag{1.6}$$

Another important characteristic of this model is that the response ($y(t)$) moves exponentially from the initial value to the steady state value. This response is characterized by the the first-order model (τ). If we evaluate time response (1.5) at $t = \tau$ we find that

$$y(t = \tau) = A\tau(0.632) = 0.632(y_{\rm ss})$$

which means that after one time constant has passed the response is 63.2% of the way from the initial value to the steady state value. Take a look at Figure 1.3 to make sure that we did the math correctly!

1.3.1 Car Example—Step Response

Now that we've discussed the time response of our first-order model let's see how this applies to our automobile model. We can rearrange the mathematical model (1.2) so that it looks more like our generic model (1.1) which results in the expression

$$\dot{v}(t) + \frac{1}{m/b}(v(t)) = \frac{F}{m}\mu(t) \tag{1.7}$$

which highlights that the time constant is $\tau = m/b$ and that the steady state speed is $v_{\rm ss} = F/b$. Take a moment to think about this. It means that the larger the mass of our car, the slower the acceleration (because the time constant is larger). You might also notice that the mass has no effect on the steady state speed; our model says that a heavy "car" will reach the same final speed as a light car, it will just get there slower. Finally we might notice that the more drag in our model, the slower our "car" will go for the same input. Listing 1.2 illustrates how we might use MATLAB to graph the solution for specific numerical values.

Listing 1.2: Script for first-order response of our car example: firstorder_car_ex.m

```
% Illustration of First-Order LTI Step Response

% Clear all the variables in the workspace
clear;

% System Parameters
m = 800;         % mass [kg]
b = 225;       % linear drag [Ns/m]
Fstep = 15000;  % amplitude of step force input [N]

% Describe a vector of times for the simulation
```

```
dt = 0.1;    % Time step [s]
Tend = 15;  % End of time horizon for simulation [s]
tt = 0:dt:Tend;

% Analytical Solution
tau = m/b;
% Note that MATLAB can deal with the tt vector in this equation
% and output a vector of velcocities.
Veqn = Fstep/b*(1-exp(-tt/tau));

% Plot the results
figure(1)
clf()
plot(tt,Veqn/0.447,'b.')
xlabel('Time [sec]')  % Always label the axes and include units!
ylabel('Velocity [mph]')
title('First-Order Step Response')
grid on
```

Exercise 1.1 *The step function in (1.3) could be more precisely called the "unit step function" because it has an amplitude of 1. Write an equation (similar to (1.3)) and sketch a graph (similar to Figure 1.2) for the more general step function input $f(t) = A(\mu(t - t_o))$.*

Exercise 1.2 *We justified (1.6) by starting with the model (1.1). Another way to calculate the steady state response is to evaluate the solution (1.5) as $t \to \infty$. Show that using this method yields the same result.*

Exercise 1.3 *We showed that the response of a first-order model to a step function input has an exponential increase characterized by the time constant. Furthermore we showed that after a duration of one time constant beyond the rise in the step input the output will be 63.2% of the way to its steady state value. Figure 1.3 illustrates the value of the response at $t = [\tau, 2\tau, 3\tau, 4\tau, 5\tau]$. Evaluate 1.5 for these values of time and report the results in a two column table where the first column is the ratio t/τ and the second column is the ratio $y(t)/y_{\text{ss}}$.*

Exercise 1.4 *Consider the step response to our car model (1.7) with the following parameters: mass = 800 kg, drag coefficient = 225 $^{Ns}/_{m}$, step input amplitude = 15,000 N. Using MATLAB, create a graph similar to Figure 1.3 to illustrate the step response of our "car". Annotate the graph to show the time constant and the steady state speed. Also, make sure to use appropriate units for each axis.*

Exercise 1.5 *Consider the step response to our car model (1.7) with the parameters given in Exercise 1.4*

- *What is the minimum step input amplitude (F_{min}) that would cause our "car" to go from 0–60 mph?*
- *Based on this minimum step input amplitude, how long does it take for the "car" to go from 0–60 mph? (Hint: the answer to this question can be a number or a sentence!)*
- *If we double F_{min}*
  - *How long does it take for the "car" to go from 0–60 mph?*

- *What is the new top speed (in mph)?*
- *How long does it take for the "car" to achieve this new top speed?*

1.4 Free Response

Another useful time response of our first-order model is the response with no input ($f(t) = 0$) and an initial condition ($y(0) = y_o$). The solution to (1.1) under these conditions is

$$y(t) = y_o\, e^{-t/\tau} \tag{1.8}$$

for $t > 0$. Figure 1.4 illustrates this free response. Just as before the time constant is the key factor that determines the characteristic of the response. Again, at $t = \tau$ the output has fallen by 63.2%.

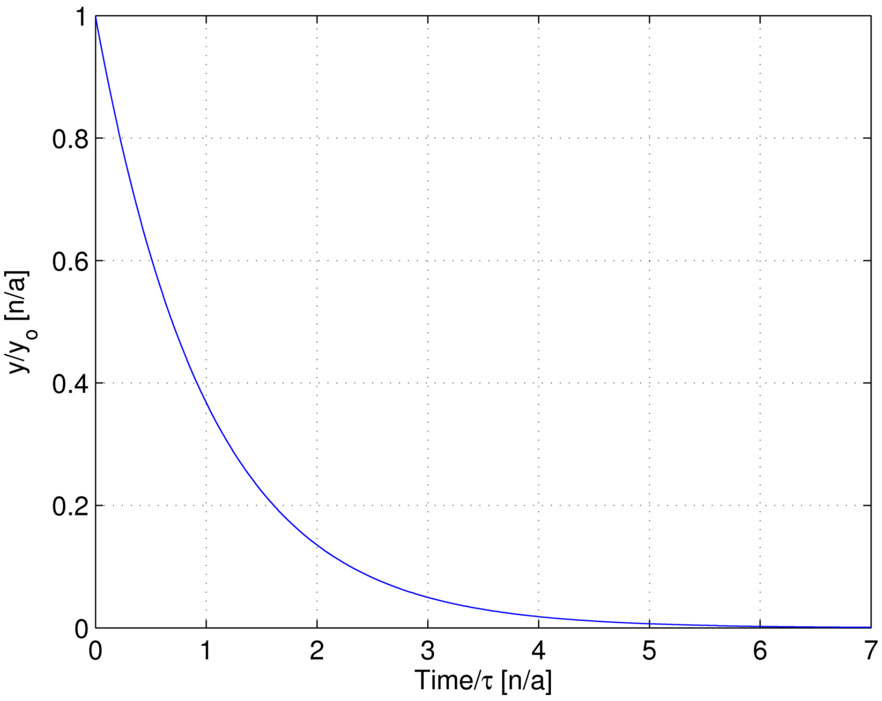

Figure 1.4: Graph of the free response (1.8). Notice that the x and y axes have been normalized so that the output is the ratio of the response and the initial condition and the time is the ratio of time and the time constant.

Exercise 1.6 *Using our automobile model (1.2), write an expression for the free response of this model with appropriate physical parameters. Create a graph of the free response with the initial condition* $v(0) = 60\,\mathrm{mph}$*. For the x-axis of the graph use the units mph and for the y-axis of the graph use the units seconds.*

1.5 Superposition—Step Response with Non-Zero Initial Condition

One the characteristics that make linear system such as our first-order model useful is the principle of superposition which allows determine the solution to a linear model by adding together multiple solutions. As an example, suppose there are two input functions, $f_1(t)$ and $f_2(t)$, and that the corresponding solutions are $y_1(t)$ and $y_2(t)$. If we want to know the response of the system to both inputs, $f(t) = f_1(t) + f_2(t)$, we can simply add the two solutions, $y(t) = y_1(t) + y_2(t)$.

This principle simplifies the task of finding the response to our first-order model when there are both initial conditions and a forcing function. Armed with superposition principle we can determine the step response (the solution to (1.1) with the input $f(t) = A\mu(t)$) of our first-order model when there is a non-zero initial condition ($y(0) = y_o$) by simply adding our two previous solutions, the step response with zero initial condition and the free response with no forcing function:

$$y(t) = y_o\, e^{-t/\tau} + A\tau \left(1 - e^{-t/\tau}\right). \tag{1.9}$$

Another handy aspect of superposition is the scaling property. This means that if we scale (multiply) the input by a factor (a) then the output is simply the response to the original input scaled (multiplied) by the same factor. As an example, consider our first-order car model (1.2) where the input is a force ($f(t)$). If we know that a force of 1,000 N results in a steady state speed of 25 mph then superposition tells us that a force of 2,000 N will result in a final speed of 50 mph.

Exercise 1.7 *Consider our first-order car model discussed above where a 1,000 N input yields a steady state speed of 25 mph. Furthermore, we know that the time constant of this system is 2 s. Again we double the input to from 1,000 N to 2,000 N. What is the time constant of the response with the doubled input?*

Hint: This might be considered a 'trick' question.

1.6 Second-Order Model

We will use second-order model as shorthand for a linear, ordinary, time-invariant second-order differential equation of the form

$$\ddot{y}(t) + 2\zeta\omega_n\dot{y}(t) + \omega_n^2 y(t) = f(t) \tag{1.10}$$

where

- $\ddot{y}(t)$ is the second derivative of $y(t)$ with respect to time,
- $\dot{y}(t)$ is the first derivative of $y(t)$ with respect to time,
- $y(t)$ is the solution to (1.10),
- ζ is the damping ratio,
- ω_n is the undamped natural frequency and
- $f(t)$ is the input, or forcing function.

Just as with our first-order and static (zero-order) models, this model is an abstraction; it never has exactly the same behavior of a physical system, but it has been proven to be a useful approximation for the behavior of a variety of phenomena. To understand the model we'll first present the solution to (1.10) for a few types of input (forcing functions) and then work through some examples of cases where the model is a helpful abstraction of physical systems.

The intent of the following discussion is meant to be a summary and/or review of what you have seen in other courses (Differential Equations, System Dynamics, Feedback Controls, etc.). The discussion should be sufficient for our purposes of modeling and measurement, but it is by no means an exhaustive treatment of analysis of second-order models or harmonic oscillator models.

Also, for now we are only interested in **underdamped** and **undamped** versions of the model (1.10) where $0 \leq \zeta < 1.0$. Another way to say this is that we are only interested in second-order models that have oscillations in their response.

1.6.1 Example: Mass-Spring-Damper

One type of physical system that has similar behavior to our second-order model is the mass-spring-damper system illustrated in Figure 1.5. (Actually, this system is still a bit of an abstraction; we rarely encounter applications that require a point mass attached to a mass-less spring. The abstraction is meant to represent any physical system where there is inertia (mass, kinetic energy), a restoring force (spring, potential energy) and motion resistance (damper, energy dissipation).)

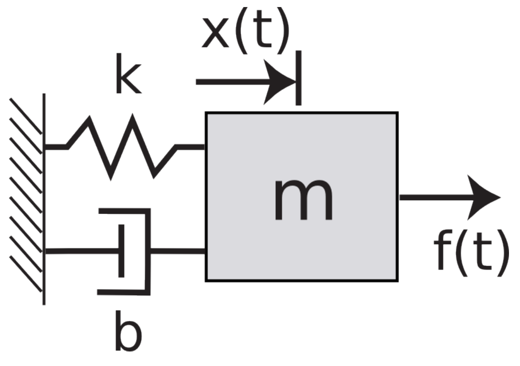

Figure 1.5: Sketch of a mass-spring-damper system.

If you draw a free-body-diagram of this system, you should be able to derive the second-order equation of motion

$$m\ddot{x}(t) + b\dot{x}(t) + kx(t) = f(t) \tag{1.11}$$

where m is the mass, b is the linear damping coefficient, k is the linear spring constant, $x(t)$ is the displacement of the mass and $f(t)$ is the input force.

Exercise 1.8 *Using (1.10) and (1.11), derive expressions for the system parameters (ω_n and ζ) in terms of the physical parameters (m, b, and k). (Hint: Use physical units to check your answers.)*

1.7 Free Response

The free response for our model is the solution to the second-order model (1.10) with initial conditions, but no forcing function (also known as the homogeneous equation)

$$\ddot{y}(t) + 2\zeta\omega_n\dot{y}(t) + \omega_n^2 y(t) = 0 \tag{1.12}$$

where the initial conditions at $t = 0$ are

- $y(0) = y_o$ and
- $\dot{y}(0) = \dot{y}_o$.

Hopefully you've been convinced that this type of differential equation is straightforward to solve (in theory), and we can present (without derivation) that the solution to this equation is the function

$$y(t) = C\left(e^{-\zeta\omega_n t}\right)\left(\cos(\omega_d t - \phi)\right). \tag{1.13}$$

where

$$\begin{aligned} \omega_d &= \omega_n\sqrt{1-\zeta^2}, & (1.14)\\ C &= \sqrt{y_o^2 + \left(\frac{\dot{y}_o + \zeta\omega_n y_o}{\omega_d}\right)^2} \qquad \text{and} & (1.15)\\ \tan(\phi) &= \frac{\dot{y}_o + \zeta\omega_n y_o}{\omega_d y_o}. & (1.16) \end{aligned}$$

The free response (1.13) is the key part of all this. There are many systems that behave in this general way with a decaying oscillatory response. To illustrate this we can annotate the equation to highlight the three parts of the solution:

$$y(t) = \underbrace{C}_{\text{constant}}\ \underbrace{\left(e^{-\zeta\omega_n t}\right)}_{\text{exponential decay}}\ \underbrace{\left(\cos(\omega_d t - \phi)\right)}_{\text{oscillation}}. \tag{1.17}$$

One important detail is that there are two slightly different "natural frequencies" to keep track of. The undamped natural frequency (ω_n) is the parameter we saw in the system model and which is included in the exponential decay component of (1.17). The damped natural frequency (ω_d) is the frequency of oscillation, the angular frequency inside the cos term of (1.17). The relation between the two values is given in (1.14). For small values of ζ, so called "lightly damped systems", there is little difference between the two values.

Another way to "see" this solution is to graph the time response. The short MATLAB script in Listing 1.3 generates Figure 1.6. This figure illustrates the key characteristics of a second order free response:

1. the system oscillates at a constant frequency (the $\cos(\omega_d t)$ term) and
2. the amplitude of the oscillations decays exponentially over time (the $e^{=\zeta\omega_n t}$ term).

Listing 1.3: Script for plotting the free response of a second-order model. (Filename=second_order_free.m, `http://web.eng.hawaii.edu/~bsb/me402/book/code/second_order_free.m`)

```
% Illustration of Second-Order LTI Free Response

% Model Parameters
wn = 1*2*pi;  % undamped nat'l freq. [rad/s]
zeta = 0.1;   % damping ratio [n/a]cd

% Initial condition
y0 = 1;  %  y(t=0)

% Describe a vector of times for the solution
dt = 0.001;   % Time step [s]
Tend = 10;  % End of time horizon for simulation [s]
tt = 0:dt:Tend;

% Solution to DE
wd = wn*sqrt(1-zeta^2);
Yeqn = y0*exp(-zeta*wn*tt).*cos(wd*tt);

% Plot the Results
figure(1); clf()
plot(tt,Yeqn,'b-')
xlabel('Time [sec]')
ylabel('y(t) [n/a]')
title('Free Response of Second-Order Model')
```

Exercise 1.9 *There is a mistake in Listing 1.3 which attempted to display the solution for the case where there is only a "displacement" initial condition: i.e.,* $y(0) = y_o$ *and* $\dot{y}(0) = 0$. *The mistake is that the solution in the code uses the values* $C = y_o$ *and* $\phi = 0$ *which are not correct (but they are close).*

- *Derive expressions for the constants* C *and* ϕ *based on the initial conditions* $y(0) = y_o$ *and* $\dot{y}(0) = 0$.
- *Copy the script in Listing 1.3 and create the graph shown in Figure 1.6.*
- *Correct the script in Listing 1.3 and plot a second curve, on the same axes, with the corrected solution for* $y_o = 1$.
  - *Include your MATLAB script with your solution.*
  - *Include your graph as a properly formatted figure.*

The error is simply in the calculation of C *and* ϕ*; the structure of the script in Listing 1.3 should work just fine.*

Exercise 1.10 *Consider the effect of the damping ratio* (ζ) *on the shape of free response. Using Listing 1.3 as a starting point, use MATLAB to create a single graph for the free response with the initial conditions* $y(0) = 1$ *and* $\dot{y}(0) = 0$. *On a single set of axes, plot the response for systems with the following values for the damping ratio:* $\zeta = \{0.0, 0.01, 0.1, 0.5, 0.9\}$. *Use a legend to declare which curve is associated with each value of* ζ. *Submit this single graph as an appropriately*

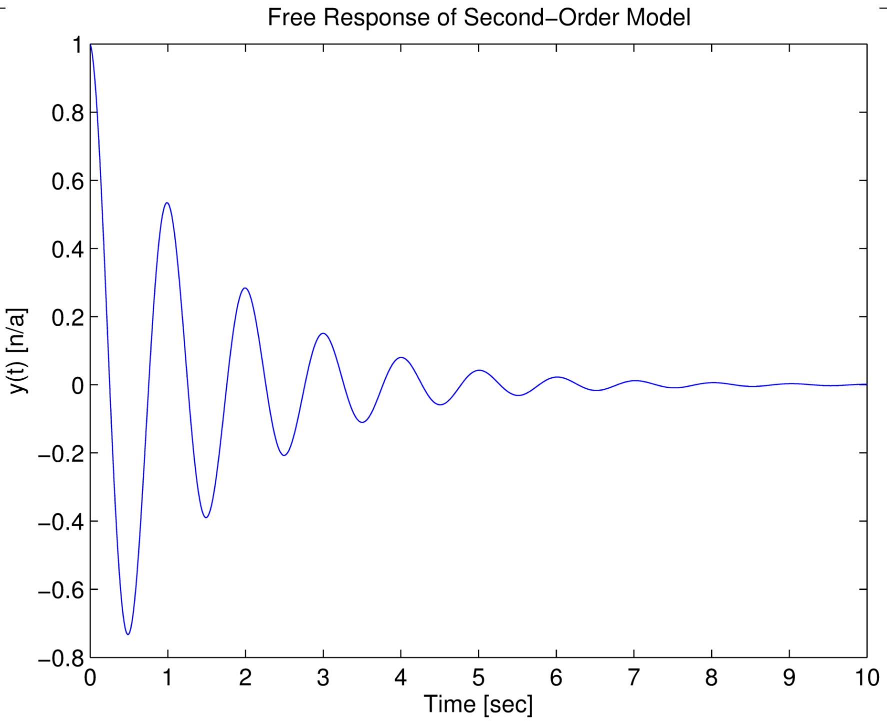

Figure 1.6: Graph of the free response (1.5) with $\omega_n = 1\,\mathrm{Hz}$, $\zeta = 0.1$ (10%) and $y(t = 0) = 1$.

formatted figure with a short description of what you observe about the effect of changing ζ.

Exercise 1.11 *Consider the model's free response to velocity-only initial conditions—$y(0) = 0$ and $\dot{y}(0) = v_o$.*

- *Derive expressions for the constants C and ϕ based on these initial conditions.*
- *Substitute your expressions into the response (1.13) and write a simplified equation for the response. (Hint: $\cos\left(u - \frac{\pi}{2}\right) = \sin(u)$.)*
- *Using MATLAB, create a figure the graphs the response ($y(t)$) to this initial condition.*

Exercise 1.12 *Given a damping ratio of $\zeta = 0.01$ (1% damping), write and expression for the damped natural frequency as a function of the undamped natural frequency.*
Repeat the exercise for a 10% damping.

1.8 Example: Cantilevered Beam

Another physical system that has a response that can be approximated by our second-order model is a cantilevered beam. Imagine a yardstick fixed on one end. When the free end is subject to a displacement (the initial condition) and then released, the tip will oscillate. A cantilevered beam, unlike our lumped mass system in Figure 1.5, is a continuous system, but for now we'll approximate it by examining the first natural frequency.

1.8.1 First-Mode of Cantilevered Beam

The first natural frequency of a uniform cantilevered beam, Figure 1.7, can be estimated by using classical beam theory. The resulting expression for the first natural frequency, in rad/s, is

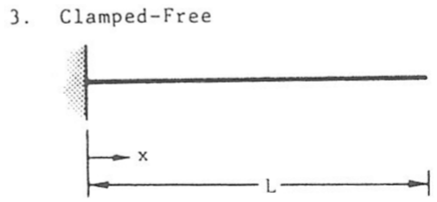

Figure 1.7: Image of a cantilevered beam model. From "Formulas for Natural Frequency and Mode Shape" by R. D. Blevins.

$$\omega_n = \frac{(1.875)^2}{L^2}\sqrt{\left(\frac{EI}{\rho}\right)} \tag{1.18}$$

where ρ is the mass per unit length of the beam (the product of the density and the cross-section area), E is the modulus of elasticity of the beam material, I is the second moment of area of the beam and L is the length of the beam.

1.8.2 First-Mode of Cantilevered Beam with Added Mass

Similarly, it is possible to predict the natural frequency of a uniform cantilevered beam with a point-mass at the free end of the beam as illustrated in Figure 1.8. The first natural frequency for

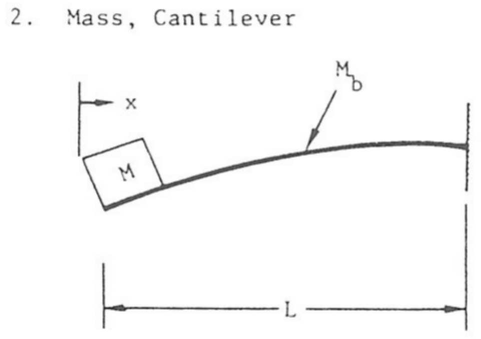

Figure 1.8: Image of a cantilevered beam with point mass model from "Formulas for Natural Frequency and Mode Shape" by R. D. Blevins.

this model is

$$\omega_n = \sqrt{\frac{3EI}{L^3(M + 0.24M_b)}} \tag{1.19}$$

where M is the point mass at the free end of the beam and M_b is the total mass of the beam.

Notice that both predicted natural frequencies (1.18) and (1.19) have the general from of $\omega_n = \sqrt{K/M}$ where K is the "stiffness" of the system and M is the "mass" of the system.

Exercise 1.13 *Given a uniform cantilevered beam with the following properties:*

- *Length = 0.5 m*
- *Width = 2.54 cm*
- *Thickness = 1.59 mm*
- *Modulus of Elasticity (E) = 68.9 GPa*
- *Density = 2.70 g/cm³*

Predict the undamped natural frequency. Express the answer in both of the following units: rad/s and Hz.

Exercise 1.14 *Given the uniform cantilevered beam described in the previous example, and a damping ratio of $\zeta = 0.02$ (2% damping)...*

- *Write an expression for a mathematical model of the form in (1.10) to describe the system. (You should just need to substitute in numerical values for the system parameters ζ and ω_n.)*
- *Using MATLAB, produce a graph that predicts the response of the tip of the cantilevered beam to an initial displacement of 1.0 cm and zero initial velocity: $y(0) = 1.0\,\text{cm}$ and $\dot{y}(0) = 0\,\text{cm/s}$.*

1.9 Numerical Solutions

One reason that the first and second order models are a good place to start our discussion is that both of these differential equations have analytical (or closed-form) solutions, i.e., a solution that satisfies the differential equation can be expressed as a function. This provides us instant gratification; we can (hopefully) clearly see how the parameters of the model (the mass, damping, etc.) affect the solution (how fast it changes, whether it oscillates, etc.). Often the model we are interested in using doesn't have a tractable analytical solution. Instead we need to rely on numerical approximations to predict the response of our model using computational tools.

1.9.1 Solution for first-order ODE

Numerical solutions to the differential equations based models common in robotics can be implemented using numerical integration. A simple, but sometimes brittle, algorithm is Euler integration. Consider a first-order differential equation $\dot{y}(t) = f(t, y(t))$. If we integrate this expression for $\dot{y}(t)$ will will generate a solution model—$y(t)$. To get things started we also need to no the initial condition of the function, where to start, which can be expressed as $y(t = 0) = y_0$. Euler integration is now a set of discrete, repeated steps to approximate the solution. We can start at our initial time $t = 0$ were we already know the answer

$$y(t = 0) = y_0.$$

Now we step forward in time by some small increment in time dt

$$\begin{aligned} y(dt) &= y(0) + (\dot{y}(0))\,(dt) && (1.20)\\ &= y_0 + (f(t=0, y(t=0)))(dt) && (1.21)\\ &= y_0 + (f(0, y_0))(dt). && (1.22)\end{aligned}$$

Then we can step forward again by the same time increment

$$\begin{aligned} y(2(dt)) &= y(dt) + (\dot{y}(dt))\,(dt) && (1.23)\\ &= (y_0 + (f(0, y_0))(dt)) + (f(dt, (y_0 + (f(0, y_0))(dt))))(dt). && (1.24)\end{aligned}$$

As we continue to step forward we follow the same step: take the old value of $y(t)$, evaluate our expression for the derivative $\dot{y}$ at the previous time step and sum the previous value of $y(t)$ with the product of the derivative and step size $\dot{y}(t)(dt)$. To express this efficiently we can break up continuous time into a set of discrete steps $t_k = t_0 + k(dt)$. The Euler method for solving this first-order ODE gives us an approximation for the solution at each of these discrete steps. This discrete values will be noted as $y[k] = y(t_k = t_0 + k(dt)$ so we can now express the solution at discrete times as

$$y[k+1] = y[k] + \dot{y}[k](dt) = y[k] + f(t_k, y[k])(dt).$$

Now we have an algorithm that is well-suited for implementing with a computer program. An illustrative example is given in Listing 1.4.

Listing 1.4: Numerical solution for a first-order ODE using Euler integration: firstorder_euler.m

```
% Illustration of Euler method for solving a 1st order ODE
% Given the ODE \dot{y} + (1/tau) y = F and initial conditions
% Find a solution to the ODE - y(t)
clear

% Define the model parameters as constants
tau = 10;  % time constant in seconds
F = 1.0;   % forcing function is a constant - same units as y(t)

% Initial conditions
y0 = 0;    % y(t=0)

% Numeric soution setup
dt = 0.1;  % define the time step in seconds
N = 1000;  % number of time steps

y(1) = y0;  % the init. cond. as the first element in a vector
t(1) = 0;   % start a vector to record the time in seconds
for k = 1:N-1
    ydot = -1/tau*y(k)+F;   % the expresion for the derivative
    y(k+1) = y(k) + ydot*dt;  % append a new element to the vector
    t(k+1) = k*dt;            % record the time value
end

% Plot the results
figure(1);
```

```
clf();
plot(t,y,'.-')
xlabel('Time [s]')
ylabel('y(t)')
title('Numerical Solution (Euler Method) for First-Order ODE')
grid on
```

Exercise 1.15 *Listing 1.2 illustrates how to graph the analytical solution to a first-order ODE model of a vehicle moving in one dimension. Adapt Listing 1.4 to solve the same problem solved analytically in Listing 1.2.*

- *Create a graph that includes both the analytical solution and the numerical solution on the same axes. You should use different line types (and/or symbols) along with a legend annotate the graph.*
- *Create a second graph that displays the error between the two solutions ($y_{\text{analytical}} - y_{\text{numerical}}$) as a function of time.*

Your graphs should include labels on both axes, a title, a caption and a legend where appropriate.

Exercise 1.16 *Start with the program in Listing 1.4 and increase the value of the timestep (dt).*

- *What is the maximum value of dt that successfully reproduces the first-order response that we expect? Report the maximum value of dt and the ratio of dt/τ.*
- *Choose a value for dt that is 1.8 times the size of the time constant. Plot the numerical solution ($y(t)$) as a function of time. Write a short explanation (a few sentences) to describe what you observe.*
- *Repeat the above exercise with a value of dt that is 2.5 times the size of the time constant.*

Your graphs should include labels on both axes, a title, a caption and a legend where appropriate.

1.9.2 Solution for higher-order ODEs

We can extend this same approach to higher order models. The key step in this extension is to transform the higher order ODE into a set of first-order ODEs. (When we discuss linear algebra later in this text we'll see how this can be done efficiently.) To start with an example we'll revisit our canonical second order model from (1.10) repeated here:

$$\ddot{y}(t) + 2\zeta\omega_n(\dot{y}(t)) + \omega_n^2(y(t)) = f(t). \tag{1.25}$$

To transform this second order ODE into two coupled first order ODEs we need to introduce two auxiliary variables which we might call the *states* of the system:

$$\begin{aligned} x_1(t) &= y(t) \\ x_2(t) &= \dot{y}(t). \end{aligned} \tag{1.26}$$

Now we express the first derivative of these states using the second order model which yields

$$\begin{aligned} \dot{x}_1(t) &= \dot{y}(t) = x_2(t) \\ \dot{x}_2(t) &= \ddot{y}(t) = -2\zeta\omega_n x_2(t) - \omega_n^2 x_1(t) + f(t) \end{aligned} \tag{1.27}$$

Next we use the same Euler integration approach above to step forward in time, starting from the initial conditions: $y(t=0) = x_1(t=0) = y_0$ and $\dot{y}(t=0) = x_2(t=0) = \dot{y}_0$. This results in computational algorithm

$$\begin{aligned} x_1[k+1] &= x_1[k] + x_2[k](dt) \\ x_2[k+1] &= x_2[k] + \left(-2\zeta\omega_n(x_2[k]) - \omega_n^2(x_1[k]) + f[k]\right)(dt) \end{aligned} \tag{1.28}$$

where $f[k]$ is the forcing function (the input) at time $t = k(dt)$.

Exercise 1.17 *The analytical solution in (1.17 is the closed-form solution to our second order model. Listing 1.3 attempts to graph this solution given values for the model parameters and initial conditions.*

- *Develop a program to numerically approximate the solution of the model under these conditions and graph the solution $y(t)$ with respect to time.*
- *To develop this program you will have to have chosen a time increment size, a timestep. Starting from a working simulation, increase the size of this timestep until the solution is no longer stable. Report the maximum allowable timestep value as the value of dt and the ratio of dt/ω.*

1.10 Glossary

Notation	Description	Symbol	Page List
damped natural frequency	The actual frequency of oscillation (in rad/s of the response of a second-order model with damping (loss) included: $\omega_d = \omega_n\sqrt{1-\zeta^2}$	ω_d	10
damping ratio	"The damping ratio is a dimensionless measure describing how oscillations in a system decay after a disturbance"—from Wikipedia!	ζ	8
first-order model	A first-order, linear, time-invariant ordinary differential equation.		2
first-order model	For a first-order model with a step input, the time duration from the initiation of the step to the time when the model output is 63.2% of the way to the steady-state value.	τ	5
second-order model	A linear, ordinary, time-invariant, second-order differential equation. For our purposes we assume the model is underdamped or undamped.		8
steady-state response	The output of a mathematical model or physical system when in equilibrium. The steady-state response is often contrasted with the transient response.		5
step input	A mathematical function that is zero for all time less than $t = 0$ and unity for all time greater than or equal to $t = 1$.	$\mu(t)$	3
step response	The output of a mathematical function, as a function of time, when a step input is given as the input or forcing function.		3
superposition	The property of linear mathematical models that states that the total response of the system caused by two (or more) inputs is the sum of the responses to each input considered independently		8
undamped natural frequency	The theoretical frequency of oscillation (in rad/s) of the response of a second-order model if there was no damping (loss) in the system.	ω_n	8, 10

Chapter 2

Feedback

2.1 Uses of Feedback

- disturbance rejection
- parameter sensitivity
- command tracking

2.2 Anatomy of Feedback Control

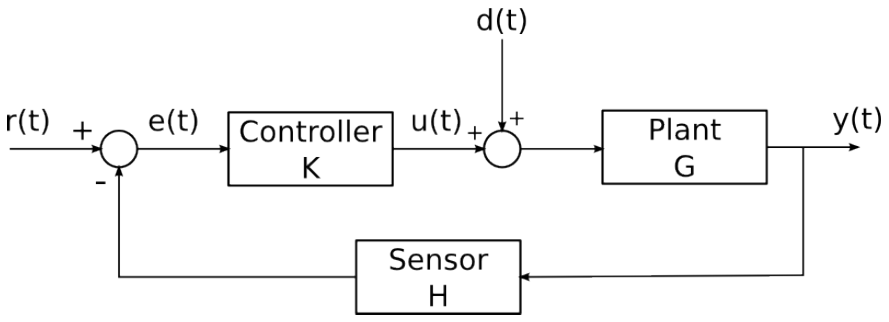

Figure 2.1: Canonical feedback control system block diagram.

- process
- disturbance
- actuator
- plant
- controller
- sensor

2.3 PID Control

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\mathrm{d}\tau + K_d \frac{\mathrm{d}}{\mathrm{d}t} e(t) \tag{2.1}$$

2.4 Assessing Stability

Conceptually, a dynamic system is considered stable if the outputs of the system remain finite in response to finite inputs. A slightly more restrictive definition is known as bounded-input bounded-output (BIBO) stability. A system is BIBO stable if the output is less than some value, the bound, in response to an input that is bounded. In either case, the output of an unstable system will grow to very large values—certainly something to avoid when designing control systems for robots.

As an example consider our first-order model of a car (1.2) repeated here as

$$m\left(\dot{v}(t)\right) + b(v(t)) = f(t)$$

and the solution for the step response with zero initial velocity

$$v(t) = F/b\left(1 - e^{-t/(m/b)}\right).$$

Consider the same model, but the value for the damping parameter b is negative. Let's say $b = -1$, but really any negative number will do. Now the solution to the differential equations is of the form

$$v(t) = F\left(1 - e^{t/(m)}\right).$$

The velocity grows exponentially with time towards infinity!

Exercise 2.1 *Show that a first order system with* **positive** *feedback can produce an unstable response.*

2.5 Assessing Performance

One way of quantifying the performance of a closed-loop design is to assess particular aspects of the step response of the system, i.e., the time response of the output of the system to a step change in input. These quantities can also be sued to specify the requirements of a design. We'll concentrate on the time-domain specifications illustrated in Figure 2.2:

Rise Time (t_r) the time it takes the system output to go from 10% to 90% of the steady-state value,

Settling Time (t_s) the time it takes the system output transients to decay to within 1% of the steady-state value,

Overshoot (OS) the maximum amount the system exceeds the steady-state value after achieving the value the first time, expressed as a percentage or as an absolute value with the same units as the output,

Stead-State Error (e_{ss}) the absolute difference between the steady-state output and the command[1], expressed as a percentage or with the same units as the output.

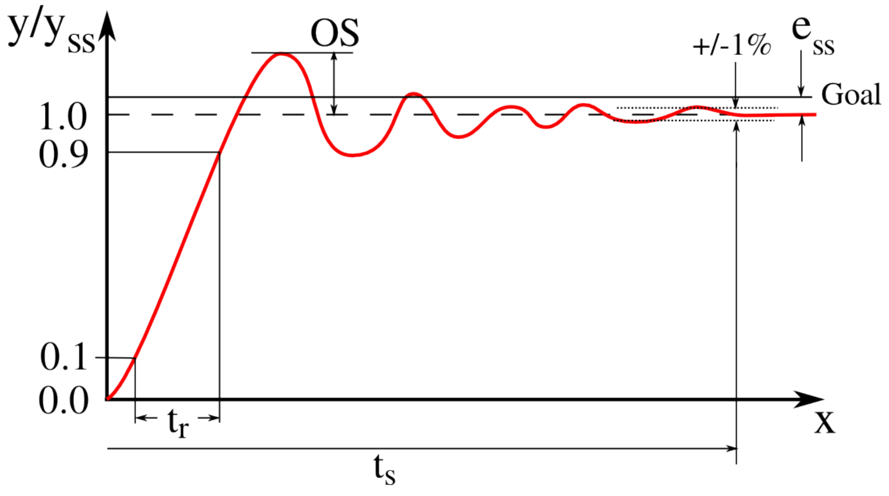

Figure 2.2: Illustration of time-domain performance assessment metrics based on the step response of a closed-loop system.

2.6 Exercises

Exercise 2.2 *Using our first order car model with the parameters given in Exercise 1.4 design a closed-loop feedback controller, a "cruise control", that will regulate the car's velocity. Here are the requirements for your controller:*

- *Minimize the rise time, overshoot and stead-state error*
- *The control force (car input) cannot exceed 15,000 N.*

Our test scenario is a step change in the speed command ($r(t)$) from 0 to 60 mph. Produce a graph of the speed of the car starting from stationary with a constant reference command of 60 mph. Also produce a graph of the control force for the same test.

Submit a short description of your control algorithm that documents the algorithm in equations and full sentences. Report the rise time, percent overshoot and steady-state error for your design in the test scenario (0 to 60 mph

Exercise 2.3 *In the previous exercise you generated a control algorithm to regulate the speed of "car". In this exercise you will generate a control algorithm to regulate the position of your "car".*

First you will need a model. Write a continuous time second-order differential equation to describe the position ($x(t)$) of the car. Write a program to solve this model using the Euler method.

Once you have a working model, design a control algorithm to regulate the position of the car. Here are the requirements for your controller:

- *Minimize the rise time, overshoot and stead-state error*

[1]The terms *command*, *goal*, *reference* and *setpoint* are all interchangeable in this context.

- *The control force (car input) cannot exceed 15,000 N.*

Our test scenario is a step change in the position command ($r(t)$) from 0 to 100 m. Produce a graph of the speed of the car starting from stationary with a constant reference command of 100 m. Also produce a graph of the control force for the same test.

Submit a short description of your control algorithm that documents the algorithm in equations and full sentences. Report the rise time, percent overshoot and steady-state error for your design in the test scenario (0 to 100 m

Chapter 3

Mobile Robot Models

In this chapter specifies some robot models useful for prototyping control designs. These are simple models that capture some of the key dynamics of the way mobile robots move.

3.1 Direct Drive Horizontal Model

One useful model mathematically moves on a two-dimensional plane. Figure 3.1 illustrates the motion of this robot moving within a fixed (inertial) coordinate frame shown as $x - y$.

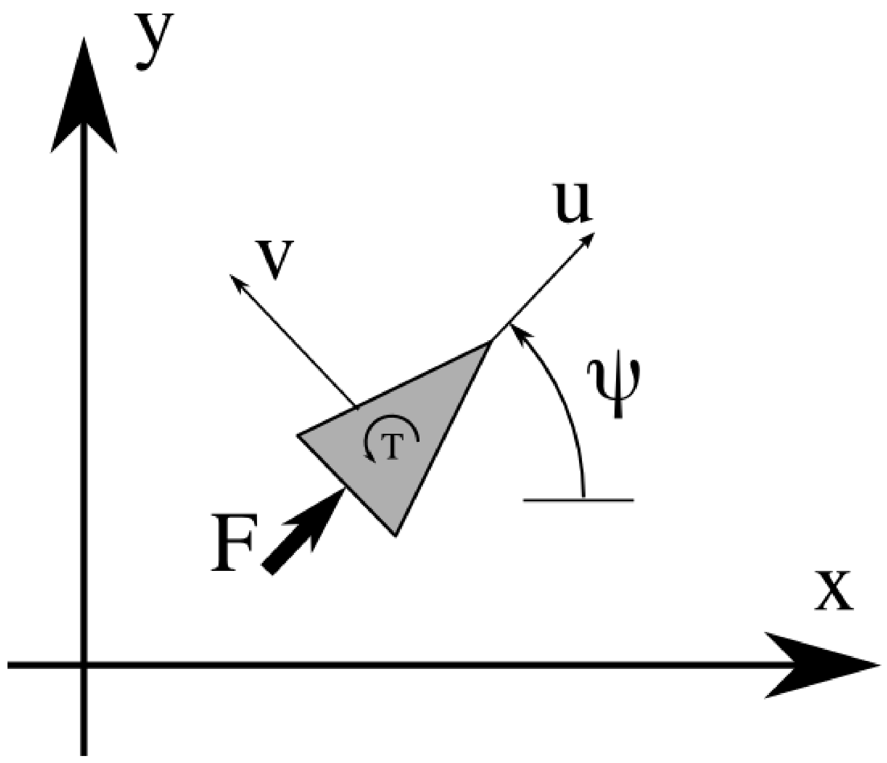

Figure 3.1: Illustration of the direct drive horizontal robot model.

This model has three degrees of freedom; it can translate forward (the u direction) and laterally (the v direction) while rotating about the middle (the ψ direction). To describe this model we will use the following six states of the model, two inputs and five physical parameters.

- Model States
  - x and y: Position of the vehicle with respect to a fixed frame of reference.
  - ψ: Angular position of the vehicle relative to the x axis.
  - u: Forward velocity of the vehicle with respect to the body-frame.
  - v: Lateral velocity of the vehicle with respect to the body-frame.
  - r: Angular velocity of the vehicle.
- Inputs
  - F: Force input in the forward direction with respect to the body-frame.
  - T: Torque input in the counter-clockwise direction.
- Model Parameters
  - M: Translational inertia (mass) of the vehicle
  - I: Rotational inertia (moment-of-inertia) of the vehicle
  - d_u: Translational damping/drag on the vehicle in the forward (u) direction
  - d_v: Translational damping/drag on the vehicle in the lateral (v) direction
  - d_r: Rotational damping/drag on the vehicle in the rotational (ψ) direction.

The equations of motion can be written as a set of six, coupled, first-order differential equations.

Kinematics:

$$\begin{aligned}\dot{x} &= \cos(\psi)u - \sin(\psi)v \\ \dot{y} &= \sin(\psi)u + \cos(\psi)v \\ \dot{\psi} &= r \qquad (3.1)\end{aligned}$$

Kinetics:

$$\begin{aligned}\dot{u} &= -\frac{d_u}{M}u + \frac{1}{M}F \\ \dot{v} &= -\frac{d_v}{M}v \\ \dot{r} &= -\frac{d_r}{I}r + \frac{1}{I}T \qquad (3.2) \\ & \qquad (3.3)\end{aligned}$$

If the damping factors (d_u, d_v and d_r) are constants, this model represents linear drag behavior.

Exercise 3.1 *Derive the mathematical model in (3.2) based on the physics of a rigid body moving on a plane. A good place to start is with a free-body-diagram.*

Exercise 3.2 *Write a computational simulation of the motion described in (3.1) and (3.2) using the Euler numerical integration method described in Section 1.9. Start by writing on paper the discrete-time formulation of the model and then write a program/script that instantiates the dynamics. Use the following model parameters:*

- $M = 10\,\text{kg}$
- $I = 10\,\text{kgm}^2$

- $d_u = 5\,\mathrm{N/m/s}$
- $d_v = 25\,\mathrm{N/m/s}$
- $d_r = 4\,\mathrm{N/rad/s}$

The initial conditions are $x = y = 0$ and $\psi = \pi/4\,\mathrm{rad}$.

Test your simulation by creating graphs of the position (y versus x) as well as the angular position (ψ versus time) for the following test inputs:

1. $F = 10\,\mathrm{N}$ *and* $T = 0\,\mathrm{Nm}$
2. $F = 0\,\mathrm{N}$ *and* $T = 1\,\mathrm{Nm}$
3. $F = 10\,\mathrm{N}$ *and* $T = 1\,\mathrm{Nm}$

Note, you should be able to analytically predict the maximum (steady-state) tranlational and rotational velocities for this sytem. This prediction will help you debug your simulation; if you know the analytical solution, you can determine if your simulation is 'working'. You should also be able to predict the time constants associated with these first order equations. This should give you some guidance about how small you will need to make dt in your numerical integration.

Submit graphs that illustrate your numerical solution for the three test cases above. These graphs should be discussed with brief text to clearly illustrate what you are trying to show. Remember to use labels with units!

Exercise 3.3 *Re-write the simulation described in the previous exercise, but write the numerical integration step as a function. The function should have the following inputs:*

- *The current state of the robot as a 6 element vector at time k.*
- *The current value for F*
- *The current value for T*
- *The size of the time-step dt*

The function should return a 6 element vector for the new state at time $k + 1$.

Now you should have a function for the Euler integration step (going from time k to time $k + 1$ AND a program that simulates the dynamics by moving through a set of discrete time steps. Re-run the test cases in the previous exercise to validate your results.

Exercise 3.4 *A very similar standard model captures the dynamics of the same robot with two different inputs. Figure 3.2 illustrates the same scenario as Figure 3.1, but, instead of a forward thrust and torque inputs, this model considers two, independent thrust inputs F_L and F_R separated by a distance of $2r$.*

Re-write the equations (3.1 and (3.2) for this new model.

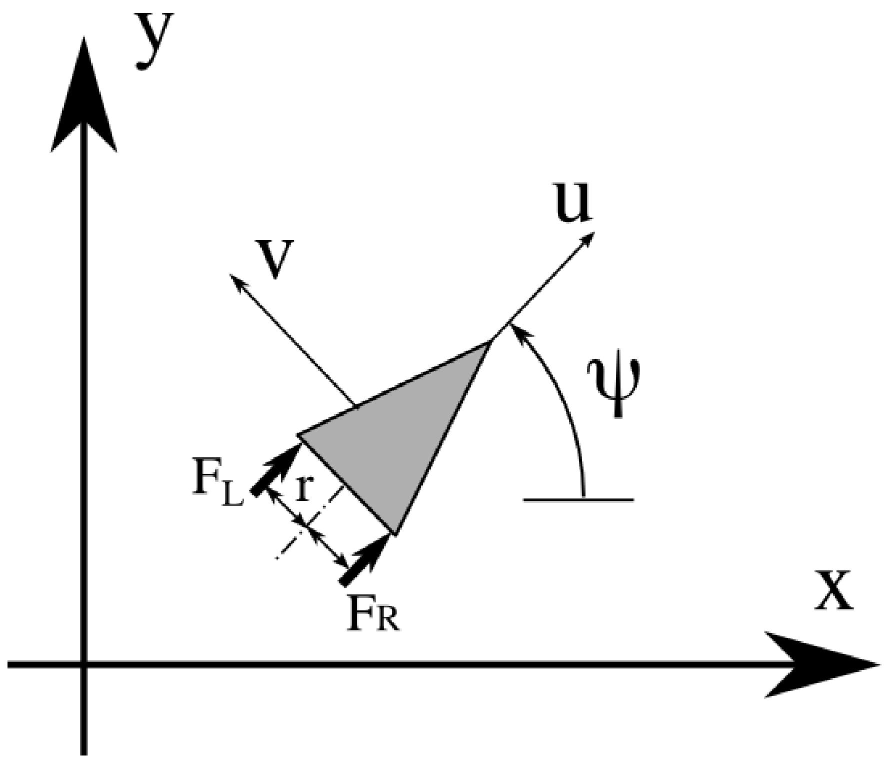

Figure 3.2: Illustration of the differential drive variant of the horizontal robot model shown in Figure 3.1.

Chapter 4

State Space

4.1 Introduction

State-space is representation of dynamic models based on presenting the governing equations as a set of first-order differential equations. This representation makes heavy use of linear algebra to compactly capture the set of differential equations. By using this representation for the ODE-based models we can bring considerable mathematical tools to bear on the problem, giving us new methods to analyze the system, predict stability and performance and synthesize control algorithms.

4.2 Canonical State-Space Form

As the name implies, the state space representation is based on the state of the system, represented by a vector with n elements. (The system is n^{th} order.) The representative form of a state space mathematical expression for a model with linear, time-invariant dynamics is a set of two matrix equations

$$\begin{aligned} \dot{\mathbf{x}}(t) &= \mathbf{A}\cdot\mathbf{x}(t)+\mathbf{B}\cdot\mathbf{u}(t) \\ \mathbf{y}(t) &= \mathbf{C}\cdot\mathbf{x}(t)+\mathbf{D}\cdot\mathbf{u}(t) \end{aligned} \tag{4.1}$$

where

- $\mathbf{x}$(t) is the $n\times 1$ state vector,
- $\mathbf{A}$ is the $n\times n$ system matrix,
- $\mathbf{B}$ is the $n\times m$ input matrix,
- $\mathbf{u}$(t) is the $m\times 1$ input vector,
- $\mathbf{y}$(t) is the $p\times 1$ output vector,
- $\mathbf{C}$ is the $p\times n$ output matrix, and
- $\mathbf{D}$[1] is the $p\times m$ direct transmission matrix[2].

[1]Often the direct transmission term can be omitted because the input does not directly influence the output in the governing equations.

[2] Notation: We will try to be consistent with the following notation. A vector variable is represented by a bold, lower-case variable, and when the elements of the vector shown curly brackets ({}) are used. A matrix variable is

The first equation in (4.1) captures the dynamics of the system—how the state changes with time and in response to a set of inputs $\mathbf{u}$. The second equation in (4.1) describes how the outputs of the system $\mathbf{y}$ (typically the quantities we can observe or measure) are related to the internal states of the system and perhaps the input.

The continuous-time dynamics of the mathematical model are described in (4.1). A discrete time approximation can be written in a similar form

$$\begin{aligned}\mathbf{x}[k+1] &= \mathbf{F}\mathbf{x}[k] + \mathbf{G}\mathbf{u}[k] \\ \mathbf{y}[k] &= \mathbf{H}\mathbf{x}[k] + \mathbf{J}\mathbf{u}[k]\end{aligned} \tag{4.2}$$

where

- $\mathbf{x}[k]$ is the $n \times 1$ state vector at time $t = k(dt)$,
- $\mathbf{G}$ is the $n \times n$ system matrix,
- $\mathbf{F}$ is the $n \times m$ input matrix,
- $\mathbf{u}[k]$ is the $m \times 1$ input vector at time $t = k(dt)$,
- $\mathbf{y}[k]$ is the $p \times 1$ output vector at time $t = k(dt)$,
- $\mathbf{H}$ is the $p \times n$ output matrix, and
- $\mathbf{J}$ is the $p \times m$ direct transmission matrix.

4.2.1 Example: Second-Order System

Recall in Section 1.9.2 that we transformed our second-order model

$$\ddot{y}(t) + 2\zeta\omega_n(\dot{y}(t)) + \omega_n^2(y(t)) = f(t) \tag{4.3}$$

into two coupled first-order ODEs by defining two states of the system

$$\begin{aligned}x_1(t) &= y(t) \\ x_2(t) &= \dot{y}(t).\end{aligned} \tag{4.4}$$

We can now write this model in our canonical state space form with the state vector

$$\mathbf{x}(t) = \left\{ \begin{array}{c} x_1(t) \\ x_2(t) \end{array} \right\}$$

as

$$\begin{aligned}\dot{\mathbf{x}}(t) &= \begin{bmatrix} 0 & 1 \\ -\omega_n^2 & -2\zeta\omega_n \end{bmatrix} \mathbf{x}(t) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} f(t) \\ y(t) &= \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x}(t)\end{aligned} \tag{4.5}$$

Exercise 4.1 *Write a new state space description of the model (4.3) with the state definition*

$$\mathbf{x}(t) = \left\{ \begin{array}{c} \dot{y}(t) \\ y(t) \end{array} \right\}.$$

As shown in the example, the input to the system is the scalar $f(t)$ and the output of the system is the scalar $y(t)$.

represented by a bold, upper-case variable, and when the elements of the matrix are shown square brackets ([]) are used. In (4.1) the matrix products are highlighted by the dot-operator (e.g, $\mathbf{A} \cdot \mathbf{x}(t)$); for brevity this operator will not be included as we move forward.

Exercise 4.2 *Write the discrete approximation of our second-order ODE given in (1.28 in the discrete state-space form (4.2. Note that* **F** *and* **G** *will be functions of the time-step dt.*

4.3 Numerical Approximation

Once we have out model dynamics expressed in state space form (4.1) there are a variety of ways of approximating the solution to this set of ODEs using numerical integration. Perhaps the simplest (but not terribly accurate) method is to apply the forward Euler method discussed in Section 1.9, expanding the approach from scalar-valued ODEs to vector-valued ODEs. Using this technique we can approximate the state at time $k+1$ using the previous state, the rate-of-change of the state given by (4.1) and the constant scalar value for the time-step ($\mathrm{d}t$) as

$$\begin{aligned}
\mathbf{x}[k+1] &= \mathbf{x}[k] + \dot{\mathbf{x}}[k](\mathrm{d}t) \\
&= \mathbf{x}[k] + \left[\mathbf{A}\mathbf{x}[k] + \mathbf{B}\mathbf{u}[k]\right](\mathrm{d}t) \\
&= \left[\mathbf{I} + \mathbf{A}(\mathrm{d}t)\right]\mathbf{x}[k] + \left[\mathbf{B}(\mathrm{d}t)\right]\mathbf{u}[k]. \\
\mathbf{y}[k+1] &= \mathbf{C}\mathbf{x}[k+1] + \mathbf{D}\mathbf{u}[k] \qquad (4.6)
\end{aligned}$$

So using Euler integration our numerical approximation (4.2) of the continuous time state space model is

$$\begin{aligned}
\mathbf{F} &= \left[\mathbf{I} + \mathbf{A}(\mathrm{d}t)\right] \\
\mathbf{G} &= \left[\mathbf{B}(\mathrm{d}t)\right] \\
\mathbf{H} &= \mathbf{C} \\
\mathbf{J} &= \mathbf{D} \qquad (4.7)
\end{aligned}$$

4.4 Exercises

Exercise 4.3 *Write the equations of motion for our direct drive horizontal model ((3.1) and (3.2)) in the canonical state space form (4.1). This system is non-linear so the matrices will be non-linear functions of some of the state variables.*

Exercise 4.4 *Create a computational simulation of the model (i.e, a numerical approximation of the solution to the system of ODEs) using the parameters given in Exercise 3.2. Formulate your numerical approximation using the continuous-time dynamics (in state space form) and the forward Euler approximation (4.6). This computational solution should make use of matrix operations (multiplication).*

Illustrate that your simulation behaves the same as your scalar implementation from Exercise 3.2.

4.5 Control Exercises

Exercise 4.5 *In this exercise you will develop a cruise control algorithm for the direct drive robot model in (3.1) and (3.2). Use the parameters in given in Exercise 3.2 for the model. In addition* *our robot has limited actuation authority; it can supply a maximum of 10 N of thrust and 10 N·m of torque. Your controllers should attempt to minimize rise-time, overshoot and steady-state error while observing these maximums.*

Start by working on a heading controller. Develop a control algorithm that takes a heading setpoint and calculates a torque to achieve the setpoint. You can test the response of this algorithm using step inputs. A reasonable size of the step input is $\pi/2$ *radians. To deal with any quadrant errors it is recommended that you verify your heading controller works for the following scenarios:*

- *Initial Heading: 0 rad, Heading Goal: pi/2 rad*
- *Initial Heading: 0 rad, Heading Goal: -pi/2 rad*
- *Initial Heading:* $3\pi/4$ *rad, Heading Goal:* $5pi/4$ *rad*
- *Initial Heading:* $5\pi/4$ *rad, Heading Goal:* $3pi/4$ *rad*

This assumes all other states have zero initial conditions.

Now put that aside and work on a velocity controller (with no heading control). Develop a control algorithm that takes a velocity setpoint and calculates a forward thrust to achieve the setpoint. Test your controller with the following scenarios:

- *Initial Heading: 0 rad, Initial Forward Velocity: 0 m/s, Velocity Goal: 1.0 m/s*
- *Initial Heading: 0 rad, Initial Forward Velocity: 1.0 m/s, Velocity Goal: 0 m/s*

Now we are ready to put these two independent controllers together into a hybrid *controller to regulate both heading and velocity. Demonstrate the behavior of your controller with the following sequential commands:*

1. *Begin with zero initial conditions on all states*
2. *Provide your controller with a heading goal of* $\pi/4$ *rad and a velocity goal of 0.5 m/s for 5 s.*
3. *Provide your controller with a heading goal of* $3\pi/4$ *rad and a velocity goal of 1.0 m/s for 5 s.*

Submit the following:

- *A short description of your heading and velocity controllers - what are your controllers doing?*
- *A graph of the x-y position of reported by your model for the final scenario.*
- *Graphs of the states of your model as a function of time for the final scenario.*
- *A discussion of what is pertinent in the graphs - what do you observe that makes you think the simulation and controllers are acting appropriately?*

Chapter 5

Behaviors

In the last chapter we developed a controller for simultaneously regulating the speed and heading of our direct-drive robot model. Figure 5.1 illustrates how we might visualize this type of controller. The reasons this approach works so well is that dynamics of our system are such that the

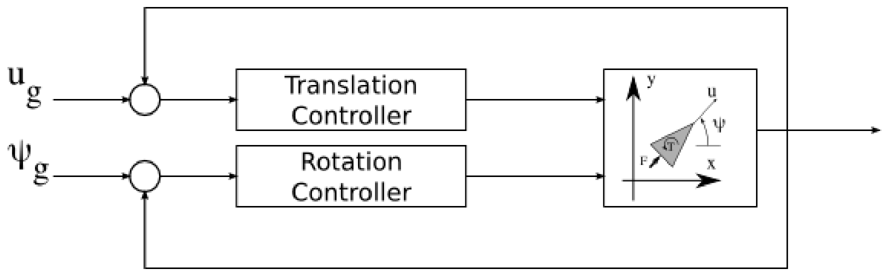

Figure 5.1: Simultaneous, independent control for translation and rotation.

system states we are controlling ($u(t)$ and $\psi(t)$) uncoupled, i.e., the forward speed and heading are independent, and the inputs to the system ($F(t)$ and $T(t)$) affect these states independently. With this kind of scenario we can be fairly confident that sequential loop closure will result a reasonable stability and performance.

In this chapter we'll develop two examples of behavior controllers where we want to achieve something (going to a waypoint or following a line) that requires that we coordinate our inputs, i.e., our goal state is dependent on both control inputs in coupled manner. One way to approach this kind of problem is illustrated in Figure 5.2 were we still have our independent translation and rotation controllers, but we've added a higher-level controller that provides continually varying setpoints to these low-level controllers.

5.1 Waypoint Control

Often we would like to control the position of a mobile robot, giving it commands that consist of a sequential list of waypoints to achieve. Figure 5.3 illustrates this scenario with our horizontal, direct-drive robot model. Our challenge is to design a control algorithm for the force (F) and

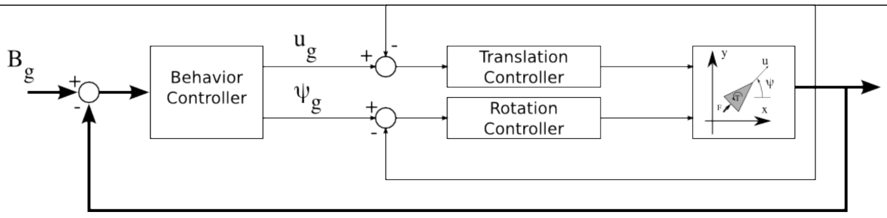

Figure 5.2: Behavior control approach. The high-level Behavior Controller supplies continually varying setpoints to the two low-level controllers for translation and rotation.

torque (T) inputs that will minimize the error between the robot position ((x, y)) and the waypoint position ((x_1, y_1)).

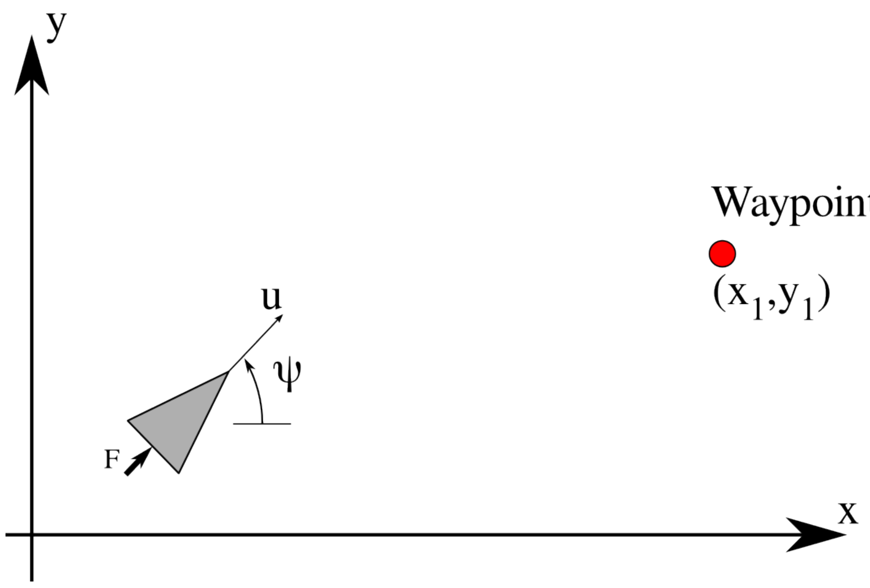

Figure 5.3: Waypoint

There are many ways to solve this problem. Some of these solutions might minimize the time taken to get to the waypoint, minimize the total energy used, etc. Here we present a simple, but useful waypoint control approach that makes use of our previous work on controlling the heading and velocity. Keep in mind this is just one of many solutions.

- Calculate the angle from the current robot position to the waypoint: ψ_{goal}
- Regulate the heading to achieve ψ_{goal}, i.e., ψ_{goal} is the setpoint for a heading controller that determines the torque applied to the robot model.
- Calculate the distance from the current robot position the waypoint: d_{goal}
- If we are far from the waypoint ($d_{\text{goal}} > R$)
  - Regulate the speed to achieve u_{goal}
- If we are close the waypoint ($d_{\text{goal}} \leq R$)
  - Regulate the distance to minimize the distance from the waypoint
- If we are very close to the waypoint ($d_{\text{goal}} \leq S$)
  - Declare the waypoint 'achieved' and move on to the next waypoint.

$$d_{\text{goal}} = \sqrt{(x_1 - x)^2 + (y_1 - y)^2}$$

$$\psi_{\text{goal}} = \arctan \frac{y_1 - y}{x_1 - x}$$

A common mistake in calculating ψ_{goal} is to not properly consider all for possible quadrants. Most computational platforms have a function `atan2(Y,X)` that returns an angle in the appropriate quadrant.

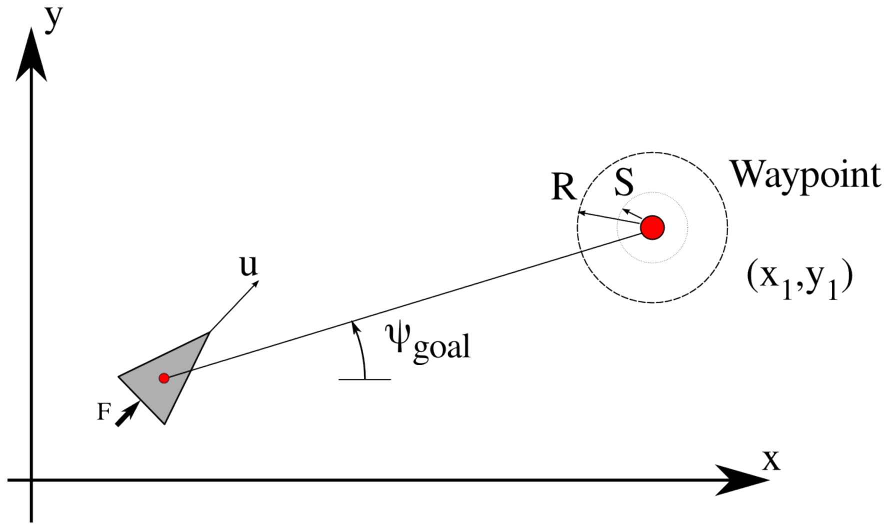

Figure 5.4: Waypoint

5.2 Line Following

A waypoint controller attempts to achieve a final state—the waypoint position—without specifying how we get there. A line following controller attempts to achieve a final state—the end of the line—but also specifies the path to take to get to that point. (Line following is a specific example of a *path following* controller.) Figure 5.5 illustrates this scenario where we'd like our direct-drive robot model to follow the line described by the two endpoints, ending up at the final point $((x_2, y_2))$.

5.2.1 Geometry

Slope of the line

$$m = \frac{y_2 - y_1}{x_2 - x_1}$$

Intersection point

$$\begin{aligned} x_i &= \frac{m}{m^2+1}\left(y - y_1 + m * x_1 + \frac{x}{m}\right) \\ y_i &= y_1 + m(x_i - x_1) \end{aligned} \tag{5.1}$$

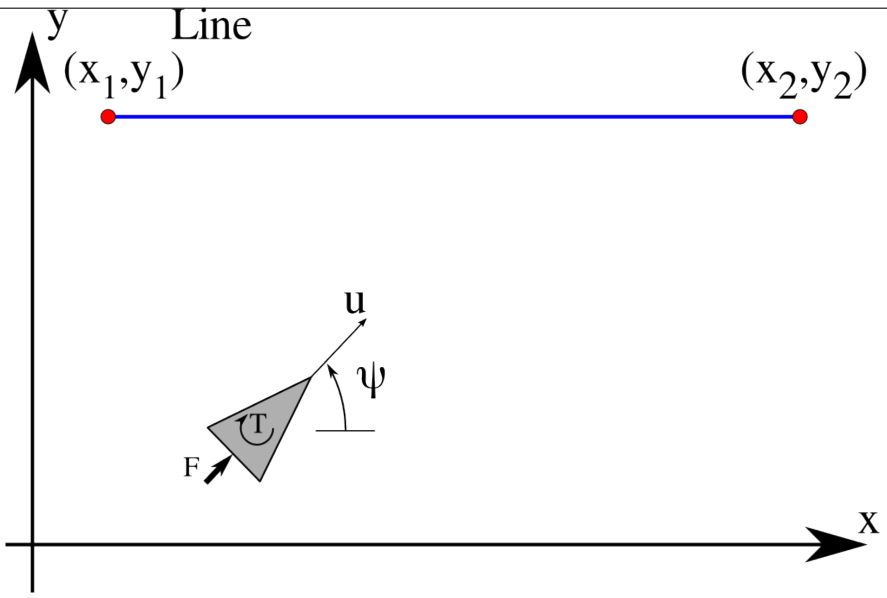

Figure 5.5: Line following

Exercise 5.1 *Derive the expressions in (5.1) given the current position (x, y) and the endpoints of the line (x_1, y_1) and (x_2, y_2). You might recall that two perpendicular lines have slopes that are negative reciprocals of each other.*

Exercise 5.2 *The expression for the intersection point (5.1 does not hold if the slope (m) is zero or infinity.*

- *If $m = 0$ (horizontal line), provide expressions for x_i and y_i given the current position (x, y) and the endpoints of the line (x_1, y_1) and (x_2, y_2).*
- *If $m = \infty$ (vertical line), provide expressions for x_i and y_i given the current position (x, y) and the endpoints of the line (x_1, y_1) and (x_2, y_2).*