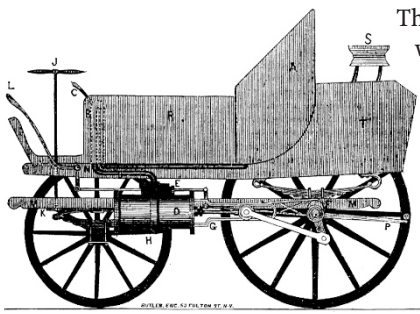
Chapter

4 Mobile Robot Vehicles



This chapter discusses how a robot platform moves, that is, how its pose changes with time as a function of its control inputs. There are many different types of robot platform as shown on pages 95–97 but in this chapter we will consider only four important exemplars. Section 4.1 covers three different types of wheeled vehicle that operate in a 2-dimensional world. They can be propelled forwards or backwards and their heading direction controlled by some steering mechanism. Section 4.2 describes a quadrotor, a flying vehicle, which is an example of a robot that moves in 3-dimensional space. Quadrotors are becoming increasing popular as a robot platform since they are low cost and can be easily modeled and controlled.

Section 4.3 revisits the concept of configuration space and dives more deeply into important issues of under-actuation and nonholonomy.

4.1 Wheeled Mobile Robots

Wheeled locomotion is one of humanity's great innovations. The wheel was invented around 3000 BCE and the two-wheeled cart around 2000 BCE. Today four-wheeled vehicles are ubiquitous and the total automobile population of the planet is over one billion. The effectiveness of cars, and our familiarity with them, makes them a natural choice for robot platforms that move across the ground.

We know from our everyday experience with cars that there are limitations on how they move. It is not possible to drive sideways, but with some practice we can learn to follow a path that results in the vehicle being to one side of its initial position – this is parallel parking. Neither can a car rotate on the spot, but we can follow a path that results in the vehicle being at the same position but rotated by 180° – a three-point turn. The necessity to perform such maneuvers is the hall mark of a system that is nonholonomic – an important concept which is discussed further in Sect. 4.3. Despite these minor limitations the car is the simplest and most effective means of moving in a planar world that we have yet found. The car's motion model and the challenges it raises for control will be discussed in Sect. 4.1.1.

In Sect. 4.1.2 we will introduce differentially-steered vehicles which are mechanically simpler than cars and do not have steered wheels. This is a common configuration for small mobile robots and also for larger machines like bulldozers. Section 4.1.3 introduces novel types of wheels that *are* capable of omnidirectional motion and then models a vehicle based on these wheels.

4.1.1 Car-Like Mobile Robots

Cars with steerable wheels are a very effective class of vehicle and the archetype for most ground robots such as those shown in Fig. II.4a–c. In this section we will create a model for a car-like vehicle and develop controllers that can drive the car to a point, along a line, follow an arbitrary trajectory, and finally, drive to a specific pose.

A commonly used model for the low-speed behavior of a four-wheeled car-like vehicle is the kinematic bicycle model shown in Fig. 4.1. The bicycle has a rear wheel fixed to the body and the plane of the front wheel rotates about the vertical axis to steer the vehicle. We assume that the velocity of each wheel is in the plane of the wheel, and that the wheel rolls without slipping sideways

Often incorrectly called the Ackermann model.

$$^{B}\boldsymbol{v} = (v, 0)$$

The pose of the vehicle is represented by its body coordinate frame $\{B\}$ shown in Fig. 4.1, with its x-axis in the vehicle's forward direction and its origin at the center of the rear axle. The *configuration* of the vehicle is represented by the generalized coordinates $\boldsymbol{q} = (x, y, \theta) \in \mathcal{C}$ where $\mathcal{C} \subset \mathbb{R}^2 \times \mathbb{S}^1$.

The dashed lines show the direction along which the wheels cannot move, the lines of no motion, and these intersect at a point known as the Instantaneous Center of Rotation (ICR). The reference point of the vehicle thus follows a circular path and its angular velocity is

$$\dot{\theta} = \frac{v}{R_B} \tag{4.1}$$

and by simple geometry the turning radius is $R_B = L / \tan\gamma$ where L is the length of the vehicle or *wheel base*. As we would expect the turning circle increases with vehicle length. The steering angle γ is typically limited mechanically and its maximum value dictates the minimum value of R_B.

Vehicle coordinate system. The coordinate system that we will use, and a common one for vehicles of all sorts is that the x-axis is forward (longitudinal motion), the y-axis is to the left side (lateral motion) which implies that the z-axis is upward. For aerospace and underwater applications the z-axis is often downward and the x-axis is forward.

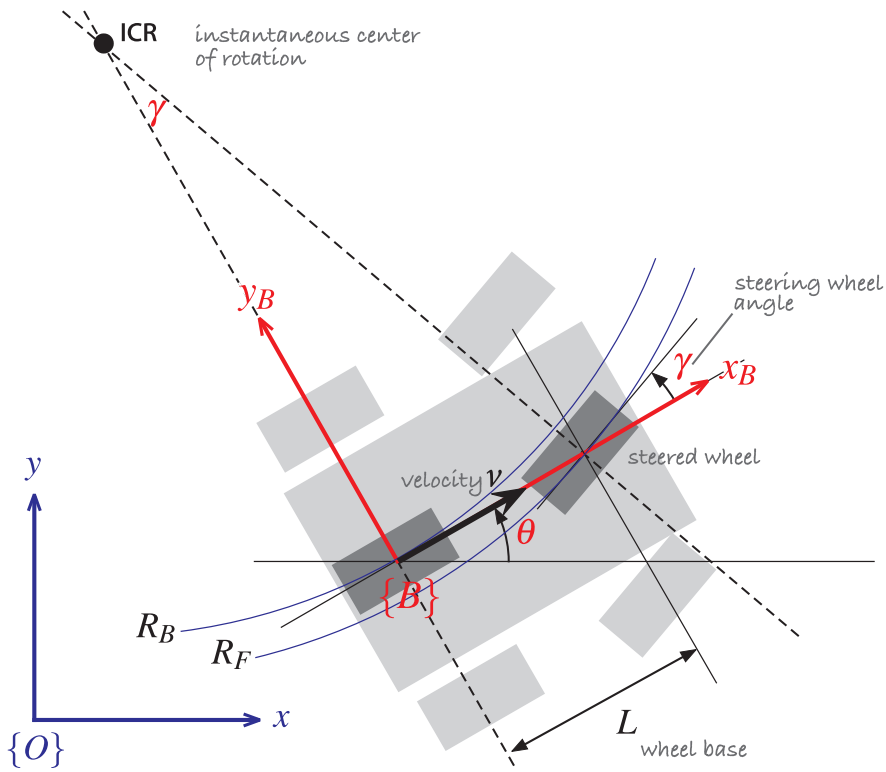
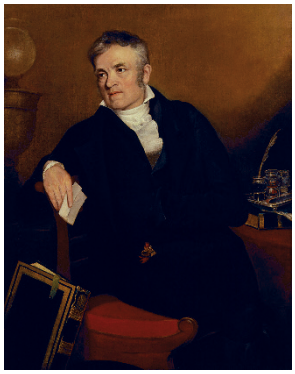


Fig. 4.1.
Bicycle model of a car. The car is shown in *light grey*, and the bicycle approximation is *dark grey*. The vehicle's body frame is shown in *red*, and the world coordinate frame in *blue*. The steering wheel angle is γ and the velocity of the back wheel, in the x-direction, is v. The two wheel axes are extended as dashed lines and intersect at the Instantaneous Center of Rotation (ICR) and the distance from the ICR to the back and front wheels is R_B and R_F respectively



Rudolph Ackermann (1764–1834) was a German inventor born at Schneeberg, in Saxony. For financial reasons he was unable to attend university and became a saddler like his father. For a time he worked as a saddler and coach-builder and in 1795 established a print-shop and drawing-school in London. He published a popular magazine "The Repository of Arts, Literature, Commerce, Manufactures, Fashion and Politics" that included an eclectic mix of articles on water pumps, gas-lighting, and lithographic presses, along with fashion plates and furniture designs. He manufactured paper for landscape and miniature painters, patented a method for waterproofing cloth and paper and built a factory in Chelsea to produce it. He is buried in Kensal Green Cemetery, London.

In 1818 Ackermann took out British patent 4212 on behalf of the German inventor George Lankensperger for a steering mechanism which ensures that the steered wheels move on circles with a common center. The same scheme was proposed and tested by Erasmus Darwin (grandfather of Charles) in the 1760s. Subsequent refinement by the Frenchman Charles Jeantaud led to the mechanism used in cars to this day which is known as Ackermann steering.

Arcs with smoothly varying radius. Dubbins and Reeds-Shepp paths comprises constant radius circular arcs and straight line segments.

For a fixed steering wheel angle the car moves along a circular arc. For this reason curves on roads are circular arcs or clothoids◂ which makes life easier for the driver since constant or smoothly varying steering wheel angle allow the car to follow the road. Note that $R_F > R_B$ which means the front wheel must follow a longer path and therefore rotate more quickly than the back wheel. When a four-wheeled vehicle goes around a corner the two steered wheels follow circular paths of different radii and therefore the angles of the steered wheels γ_L and γ_R should be very slightly different. This is achieved by the commonly used Ackermann steering mechanism which results in lower wear and tear on the tyres. The driven wheels must rotate at different speeds on corners which is why a differential gearbox is required between the motor and the driven wheels.

Fig. 124.

From Sharp 1896

The velocity of the robot in the world frame is $(v\cos\theta, v\sin\theta)$ and combined with Eq. 4.1 we write the equations of motion as

$$\begin{aligned}\dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \frac{v}{L}\tan\gamma\end{aligned} \tag{4.2}$$

This model is referred to as a kinematic model since it describes the velocities of the vehicle but not the forces or torques that cause the velocity. The rate of change of heading $\dot{\theta}$ is referred to as turn rate, heading rate or yaw rate and can be measured by a gyroscope. It can also be deduced from the angular velocity of the nondriven wheels on the left- and right-hand sides of the vehicle which follow arcs of different radius, and therefore rotate at different speeds.

Equation 4.2 captures some other important characteristics of a car-like vehicle. When $v = 0$ then $\dot{\theta} = 0$; that is, it is not possible to change the vehicle's orientation when it is not moving. As we know from driving, we must be moving in order to turn. When the steering angle $\gamma = \frac{\pi}{2}$ the front wheel is orthogonal to the back wheel, the vehicle cannot move forward and the model enters an undefined region.

In the world coordinate frame we can write an expression for velocity in the vehicle's y-direction

$$\dot{y}\cos\theta - \dot{x}\sin\theta \equiv 0 \tag{4.3}$$

which is the called a nonholonomic constraint and will be discussed further in Sect. 4.3.1. This equation cannot be integrated to form a relationship between x, y and θ.

The Simulink® system

```
>> sl_lanechange
```

The model also includes a maximum velocity limit, a velocity rate limiter to model finite acceleration, and a limiter on the steering angle to model the finite range of the steered wheel. These can be accessed by double clicking the Bicycle block in Simulink.

shown in Fig. 4.2 uses the Toolbox `Bicycle` block which implements Eq. 4.2◂. The velocity input is a constant, and the steering wheel angle is a finite positive pulse followed by a negative pulse. Running the model simulates the motion of the vehicle and adds a new variable `out` to the workspace

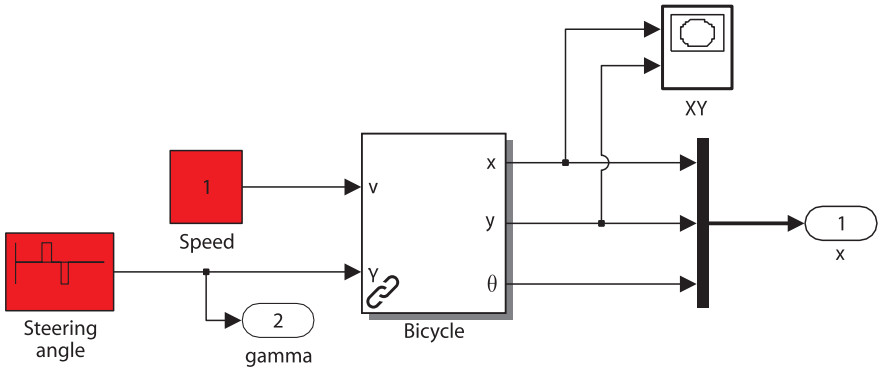


Fig. 4.2. Simulink model `sl_lanechange` that results in a lane changing maneuver. The pulse generator drives the steering angle left then right. The vehicle has a default wheelbase $L = 1$

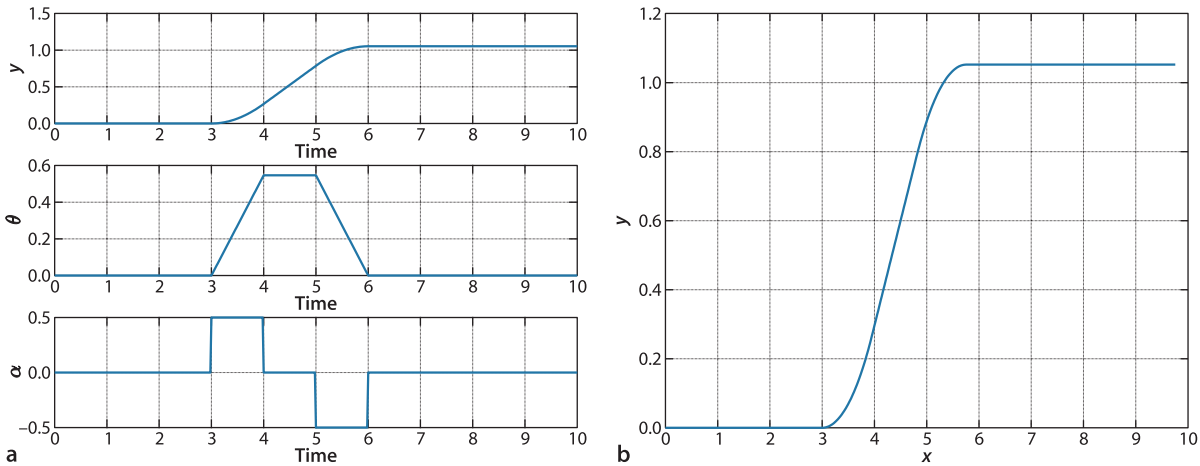


Fig. 4.3. Simple lane changing maneuver. **a** Vehicle response as a function of time, **b** motion in the xy-plane, the vehicle moves in the positive x-direction

```
>> out
Simulink.SimulationOutput:
    t: [504x1 double]
    y: [504x4 double]
```

from which we can retrieve the simulation time and other variables

```
>> t = out.get('t'); q = out.get('y');
```

Configuration is plotted against time

```
>> mplot(t, q)
```

in Fig. 4.3a and the result in the xy-plane

```
>> plot(q(:,1), q(:,2))
```

shown in Fig. 4.3b demonstrates a simple *lane-changing* trajectory.

4.1.1.1 Moving to a Point

Consider the problem of moving toward a goal point (x^*, y^*) in the plane. We will control the robot's velocity to be proportional to its distance from the goal

$$v^* = K_v \sqrt{\left(x^* - x\right)^2 + \left(y^* - y\right)^2}$$

and to steer toward the goal which is at the vehicle-relative angle▸ in the world frame of

$$\theta^* = \tan^{-1} \frac{y^* - y}{x^* - x}$$

▸ This angle can be anywhere in the interval $[-\pi, \pi)$ and is computed using the `atan2` function.

using a proportional controller

$$\gamma = K_h\left(\theta^* \ominus \theta\right), \quad K_h > 0$$

which turns the steering wheel toward the target. Note the use of the operator $\ominus$ since θ^* and θ are angles $\in \mathbb{S}^1$ not real numbers◀. A Simulink model

```
>> sl_drivepoint
```

is shown in Fig. 4.4. We specify a goal coordinate

```
>> xg = [5 5];
```

and an initial pose

```
>> x0 = [8 5 pi/2];
```

and then simulate the motion

```
>> r = sim('sl_drivepoint');
```

The variable `r` is an object that contains the simulation results from which we extract the configuration as a function of time

```
>> q = r.find('y');
```

The vehicle's path in the plane is

```
>> plot(q(:,1), q(:,2));
```

The Toolbox function `angdiff` computes the difference between two angles and returns a difference in the interval $[-\pi, \pi)$. This is also the shortest distance around the circle, as discussed in Sect. 3.3.4.1. Also available in the Toolbox Simulink blockset `roblocks`.

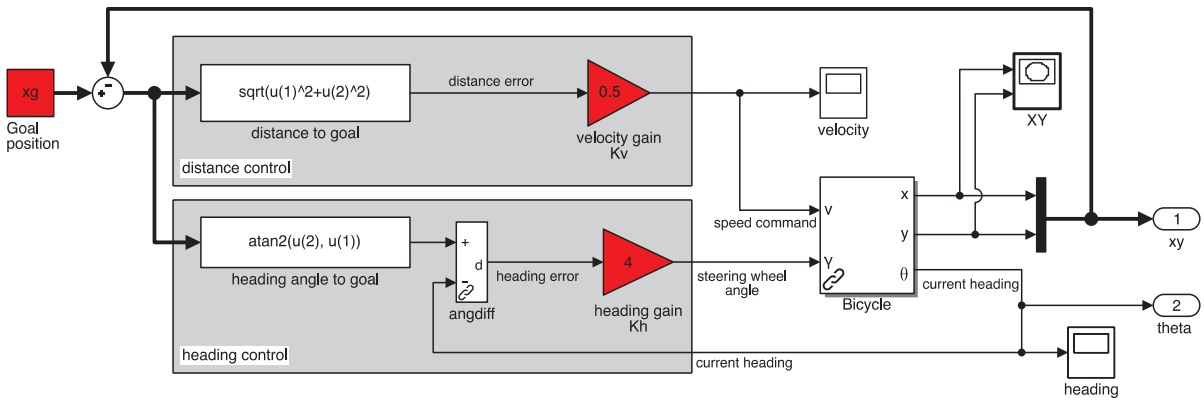


Fig. 4.4. `sl_drivepoint`, the Simulink model that drives the vehicle to a point. Red blocks have parameters that you can adjust to investigate the effect on performance

To run the Simulink model called `model` we first load it

```
>> model
```

and a new window is popped up that displays the model in block-diagram form. The simulation can be started by pressing the play button on the toolbar of the model's window. The model can also be run directly from the MATLAB command line

```
>> sim('model')
```

Many Toolbox models create additional figures to display robot animations or graphs as they run.

All models in this chapter have the simulation data export option set to create a MATLAB `SimulationOutput` object. All the unconnected output signals are concatenated, in port number order, to form a row vector and these are stacked to form a matrix `y` with one row per timestep. The corresponding time values form a vector `t`. These variables are packaged in a `SimulationOutput` object which is written to the workspace variable `out` or returned if the simulation is invoked from MATLAB

```
>> r = sim('model')
```

Displaying `r` or `out` lists the variables that it contains and their value is obtained using the `find` method, for example

```
>> t = r.find('t');
```

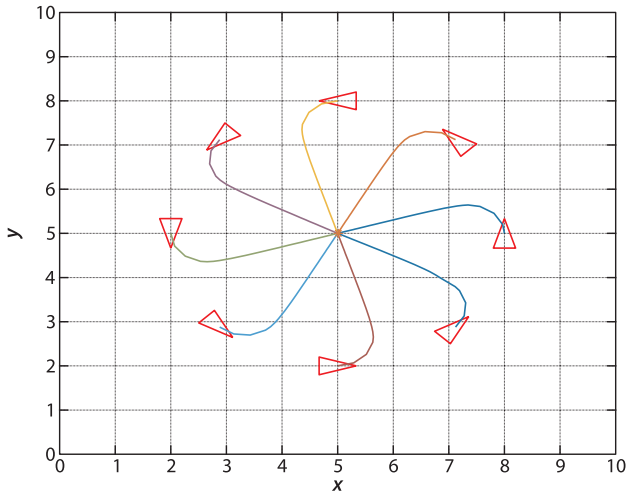


Fig. 4.5.
Simulation results for `sl_drivepoint` for different initial poses. The goal is (5, 5)

which is shown in Fig. 4.5 for a number of starting poses. In each case the vehicle has moved forward and turned onto a path toward the goal point. The final part of each path is a straight line and the final orientation therefore depends on the starting point.

4.1.1.2 Following a Line

Another useful task for a mobile robot is to follow a line on the plane▸ defined by $ax + by + c = 0$. This requires two controllers to adjust steering. One controller

2-dimensional lines in homogeneous form are discussed in Sect. C.2.1.

$$\alpha_d = -K_d d,\ K_d > 0$$

turns the robot toward the line to minimize the robot's normal distance from the line

$$d = \frac{(a, b, c) \cdot (x, y, 1)}{\sqrt{a^2 + b^2}}$$

The second controller adjusts the heading angle, or orientation, of the vehicle to be parallel to the line

$$\theta^* = \tan^{-1} \frac{-a}{b}$$

using the proportional controller

$$\alpha_h = K_h \left(\theta^* \ominus \theta \right),\ K_h > 0$$

The combined control law

$$\gamma = -K_d d + K_h \left(\theta^* \ominus \theta \right)$$

turns the steering wheel so as to drive the robot toward the line and move along it.

The Simulink model

```
>> sl_driveline
```

is shown in Fig. 4.6. We specify the target line as a 3-vector (a, b, c)

```
>> L = [1 -2 4];
```

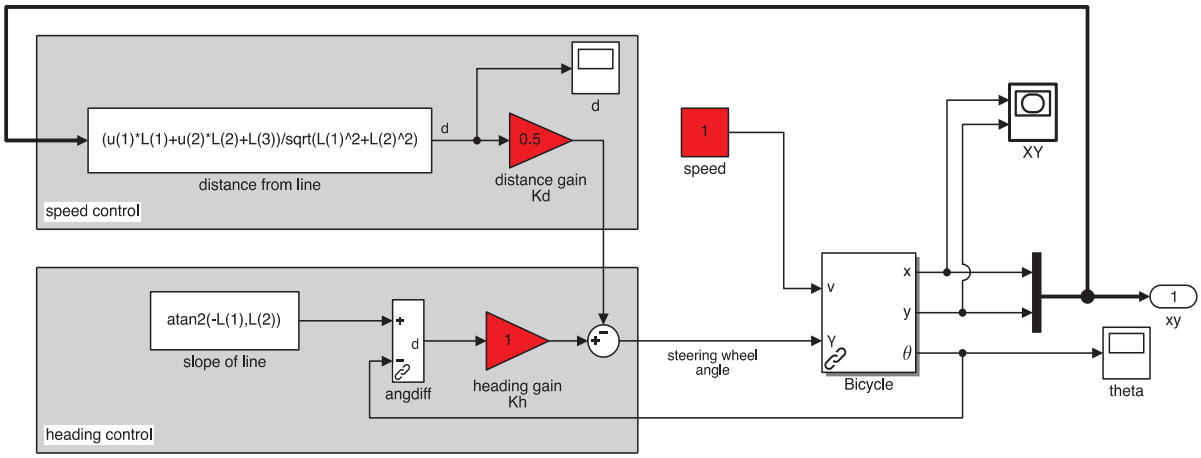


Fig. 4.6. The Simulink model `sl_driveline` drives the vehicle along a line. The line parameters (a, b, c) are set in the workspace variable `L`. Red blocks have parameters that you can adjust to investigate the effect on performance

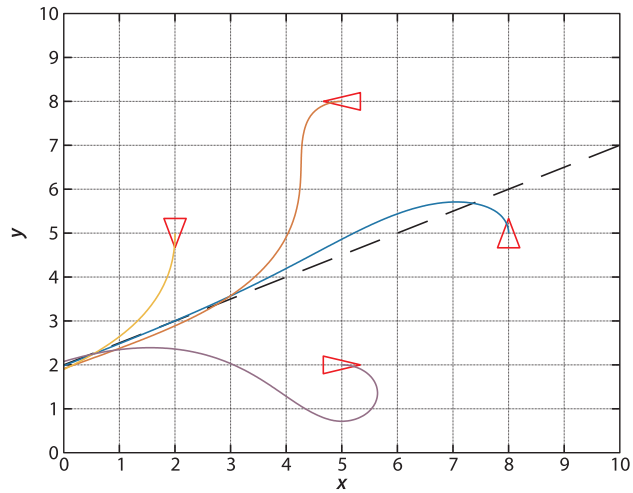


Fig. 4.7. Simulation results from different initial poses for the line $(1, -2, 4)$

and an initial pose

```
>> x0 = [8 5 pi/2];
```

and then simulate the motion

```
>> r = sim('sl_driveline');
```

The vehicle's path for a number of different starting poses is shown in Fig. 4.7.

4.1.1.3 Following a Trajectory

Instead of a straight line we might wish to follow a trajectory that is a timed sequence of points on the xy-plane. This might come from a motion planner, such as discussed in Sect. 3.3 or 5.2, or in real-time based on the robot's sensors.

A simple and effective algorithm for trajectory following is pure pursuit in which the goal point $(x^*(t), y^*(t))$ moves along the trajectory, in its simplest form at constant speed. The vehicle always heads toward the goal – think carrot and donkey.

This problem is very similar to the control problem we tackled in Sect. 4.1.1.1, moving to a point, except this time the point is moving. The robot maintains a distance d^* behind the pursuit point and we formulate an error

$$e = \sqrt{\left(x^* - x\right)^2 + \left(y^* - y\right)^2} - d^*$$

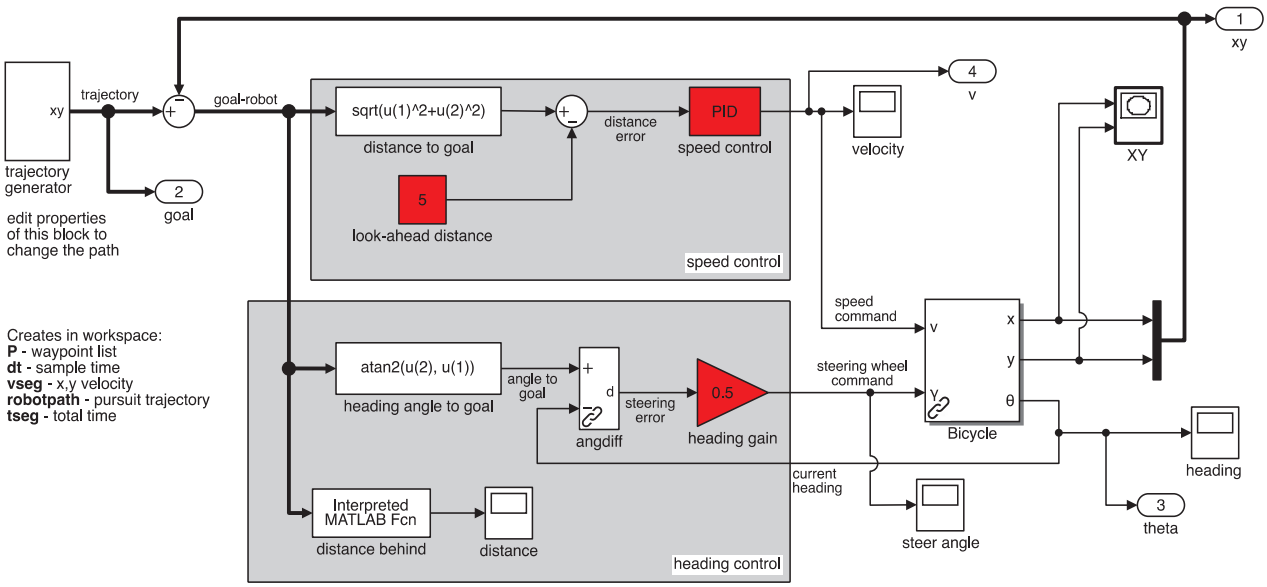


Fig. 4.8. The Simulink model `sl_pursuit` drives the vehicle along a piecewise linear trajectory. Red blocks have parameters that you can adjust to investigate the effect on performance

that we regulate to zero by controlling the robot's velocity using a proportional-integral (PI) controller

$$v^* = K_v e + K_i \int e \, dt$$

The integral term is required to provide a nonzero velocity demand v^* when the following error is zero. The second controller steers the robot toward the target which is at the relative angle

$$\theta^* = \tan^{-1} \frac{y^* - y}{x^* - x}$$

and a simple proportional controller

$$\gamma = K_h \left(\theta^* \ominus \theta \right), \quad K_h > 0$$

turns the steering wheel so as to drive the robot toward the target.

The Simulink model

```
>> sl_pursuit
```

shown in Fig. 4.8 includes a target that moves at constant velocity along a piecewise linear path defined by a number of waypoints. It can be simulated

```
>> r = sim('sl_pursuit')
```

and the results are shown in Fig. 4.9a. The robot starts at the origin but catches up to, and follows, the moving goal. Figure 4.9b shows how the speed converges on a steady state value when following at the desired distance. Note the slow down at the end of each segment as the robot *short cuts* across the corner.

4.1.1.4 Moving to a Pose

The final control problem we discuss is driving to a specific pose (x^*, y^*, θ^*). The controller of Fig. 4.4 could drive the robot to a goal position but the final orientation depended on the starting position.

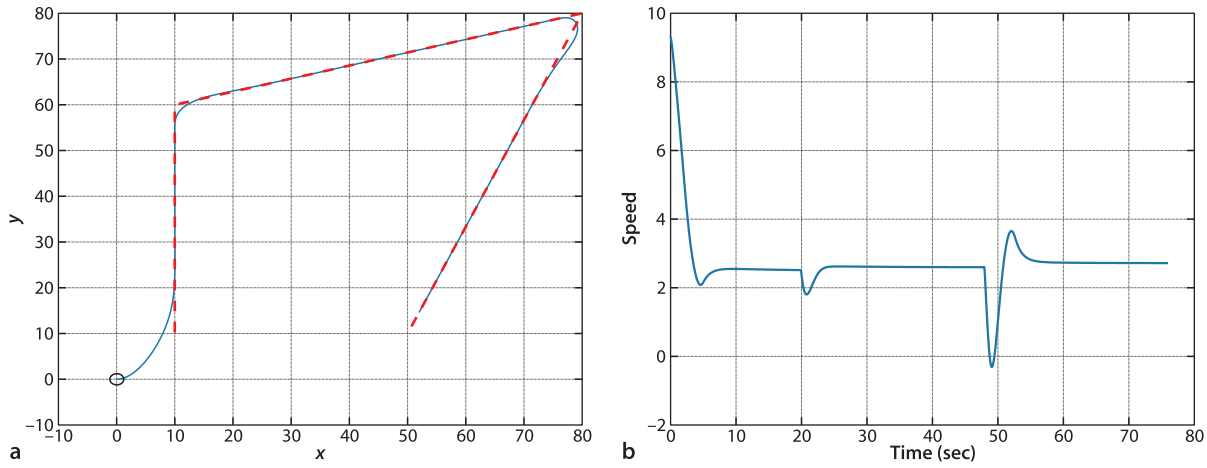


Fig. 4.9. Simulation results from pure pursuit. **a** Path of the robot in the xy-plane. The red dashed line is the path to be followed and the blue line in the path followed by the robot, which starts at the origin. **b** The speed of the robot versus time

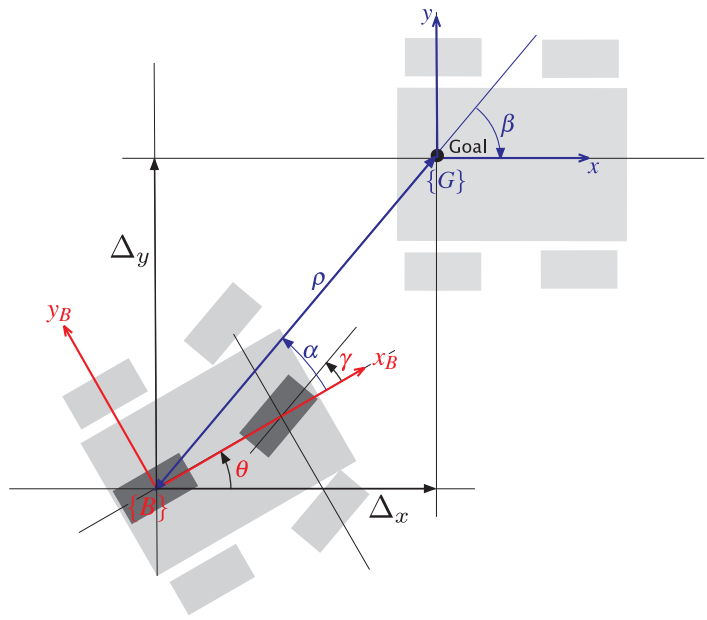


Fig. 4.10. Polar coordinate notation for the bicycle model vehicle moving toward a goal pose: ρ is the distance to the goal, β is the angle of the goal vector with respect to the world frame, and α is the angle of the goal vector with respect to the vehicle frame

In order to control the final orientation we first rewrite Eq. 4.2 in matrix form

$$\begin{pmatrix} \dot{x} \\ \dot{\omega} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} \cos\theta & 0 \\ \sin\theta & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

where the inputs to the vehicle model are the speed v and the turning rate ω which can be achieved by applying the steering angle◂

We have effectively converted the Bicycle kinematic model to a Unicycle model which we discuss in Sect. 4.1.2.

$$\gamma = \tan^{-1}\frac{\omega L}{v}$$

We then transform the equations into polar coordinate form using the notation shown in Fig. 4.10 and apply a change of variables

$$\rho = \sqrt{\Delta_x^2 + \Delta_y^2}$$

$$\alpha = \tan^{-1}\frac{\Delta_y}{\Delta_x} - \theta$$

$$\beta = -\theta - \alpha$$

which results in

$$\begin{pmatrix}\dot\rho\\ \dot\alpha\\ \dot\beta\end{pmatrix} = \begin{pmatrix}-\cos\alpha & 0\\ \frac{\sin\alpha}{\rho} & -1\\ -\frac{\sin\alpha}{\rho} & 0\end{pmatrix}\begin{pmatrix}v\\ \omega\end{pmatrix},\ \text{if } \alpha \in \left(-\frac{\pi}{2}, \frac{\pi}{2}\right]$$

and assumes the goal frame $\{G\}$ is in front of the vehicle. The linear control law

$$v = k_\rho \rho$$
$$\omega = k_\alpha \alpha + k_\beta \beta$$

drives the robot to a unique equilibrium at $(\rho, \alpha, \beta) = (0, 0, 0)$. The intuition behind this controller is that the terms $k_\rho\rho$ and $k_\alpha\alpha$ drive the robot along a line toward $\{G\}$ while the term $k_\beta\beta$ rotates the line so that $\beta \to 0$. The closed-loop system

The control law introduces a discontinuity at $\rho = 0$ which satisfies Brockett's theorem.

$$\begin{pmatrix}\dot\rho\\ \dot\alpha\\ \dot\beta\end{pmatrix} = \begin{pmatrix}-k_\rho\cos\alpha\\ k_\rho\sin\alpha - k_\alpha\alpha - k_\beta\beta\\ -k_\rho\sin\alpha\end{pmatrix}$$

is stable so long as

$$k_\rho > 0,\ k_\beta < 0,\ k_\alpha - k_\rho > 0$$

The distance and bearing to the goal (ρ, α) could be measured by a camera or laser range finder, and the angle β could be derived from α and vehicle orientation θ as measured by a compass.

For the case where the goal is behind the robot, that is $\alpha \notin (-\frac{\pi}{2}, \frac{\pi}{2}]$, we reverse the vehicle by negating v and γ in the control law. The velocity v always has a constant sign which depends on the initial value of α.

So far we have described a *regulator* that drives the vehicle to the pose (0, 0, 0). To move the robot to an arbitrary pose (x^*, y^*, θ^*) we perform a change of coordinates

$$x' = x - x^*,\ y' = y - y^*,\ \theta' = \theta,\ \beta = \beta' + \theta^*$$

This pose controller is implemented by the Simulink model

```
>> sl_drivepose
```

shown in Fig. 4.11 and the transformation from Bicycle to Unicycle kinematics is clearly shown, mapping angular velocity ω to steering wheel angle γ. We specify a goal pose

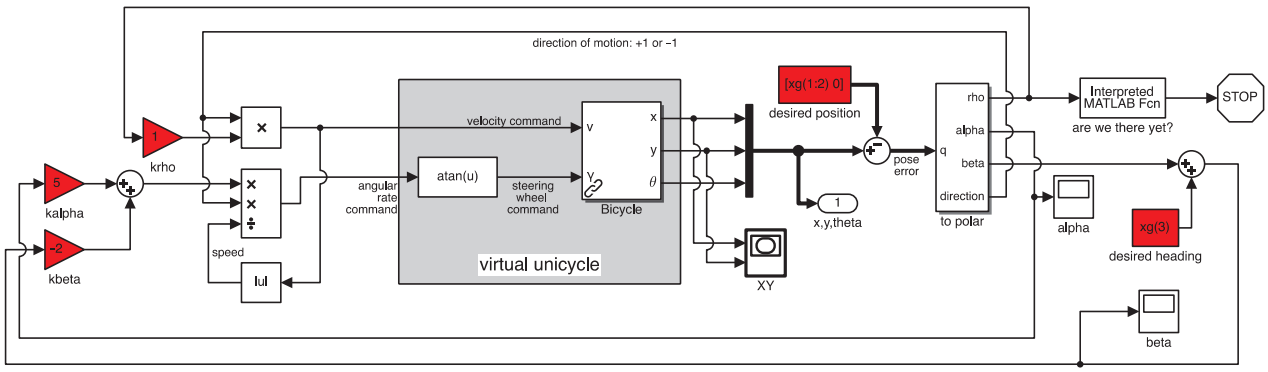


Fig. 4.11. The Simulink model `sl_drivepose` drives the vehicle to a pose. The initial and final poses are set by the workspace variable `x0` and `xf` respectively. Red blocks have parameters that you can adjust to investigate the effect on performance

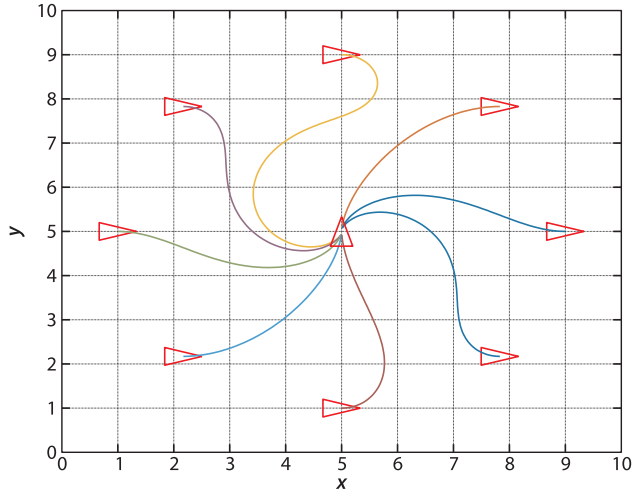


Fig. 4.12. Simulation results from different initial poses to the final pose $(5, 5, \frac{\pi}{2})$. Note that in some cases the robot has *backed into* the final pose

```
>> xg = [5 5 pi/2];
```

and an initial pose

```
>> x0 = [9 5 0];
```

and then simulate the motion

```
>> r = sim('sl_drivepose');
```

As before, the simulation results are stored in `r` and can be plotted

```
>> q = r.find('y');
>> plot(q(:,1), q(:,2));
```

to show the vehicle's path in the plane. The vehicle's path for a number of starting poses is shown in Fig. 4.12. The vehicle moves forwards or backward and takes a smooth path to the goal pose.◀

The controller is based on the bicycle model but the Simulink model `Bicycle` has additional hard nonlinearities including steering angle limits and velocity rate limiting. If those limits are violated the pose controller may fail.

4.1.2 Differentially-Steered Vehicle

Having steerable wheels as in a car-like vehicle is mechanically complex. Differential steering does away with this and steers by independently controlling the speed of the wheels on each side of the vehicle – if the speeds are not equal the vehicle will turn. Very simple differential steer robots have two driven wheels and a front and back castor to provide stability. Larger differential steer vehicles such as the one shown in Fig. 4.13 employ a pair of wheels on each side, with each pair sharing a drive motor via some mechanical transmission. Very large differential steer vehicles such as bulldozers and tanks sometimes employ caterpillar tracks instead of wheels. The vehicle's velocity is by definition v in the vehicle's x-direction, and zero in the y-direction since the wheels cannot slip sideways. In the vehicle frame $\{B\}$ this is

$$^{B}\boldsymbol{v} = (v, 0)$$

The pose of the vehicle is represented by the body coordinate frame $\{B\}$ shown in Fig. 4.14, with its x-axis in the vehicle's forward direction and its origin at the centroid of the four wheels. The configuration of the vehicle is represented by the generalized coordinates $\boldsymbol{q} = (x, y, \theta) \in \mathcal{C}$ where $\mathcal{C} \subset \mathbb{R}^2 \times \mathbb{S}^1$.

The vehicle follows a curved path centered on the Instantaneous Center of Rotation (ICR). The left-hand wheels move at a speed of v_L along an arc with a radius of R_L