

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 increasingly 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  $180^\circ$  – 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

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

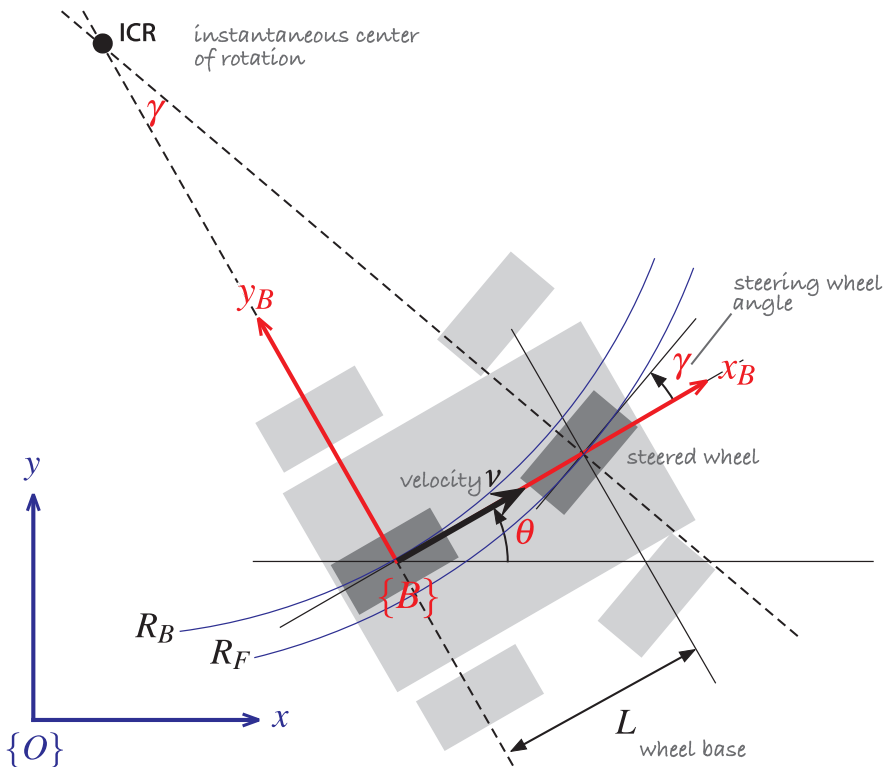
The pose of the vehicle is represented by its body coordinate frame  $\{B\}$  shown in Fig. 4.1, with its  $x$ -axis in the vehicle's forward direction and its origin at the center of the rear axle. The configuration of the vehicle is represented by the generalized coordinates  $\mathbf{q} = (x, y, \theta) \in \mathcal{C}$  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  $\gamma$  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.



Often incorrectly called the Ackermann model.

**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  $\gamma$  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.



FIG. 124.

From Sharp 1896

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  $\gamma_L$  and  $\gamma_R$  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.

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

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

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

$$\dot{y} \cos \theta - \dot{x} \sin \theta \equiv 0 \quad (4.3)$$

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  $\theta$ .

The Simulink® system

```
>> sl_lanechange
```

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

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.

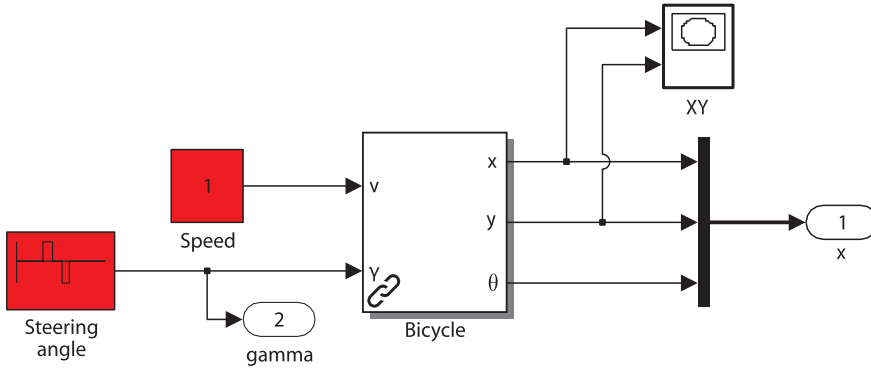


Fig. 4.2. Simulink model `s1_lanechange` that results in a lane changing maneuver. The pulse generator drives the steering angle left then right. The vehicle has a default wheelbase  $L = 1$

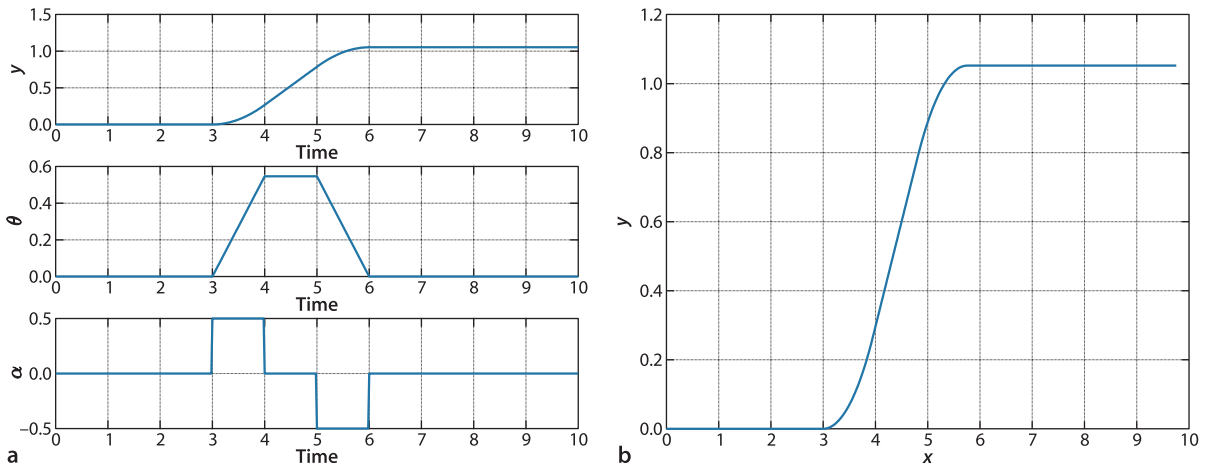


Fig. 4.3. Simple lane changing maneuver. **a** Vehicle response as a function of time, **b** motion in the  $xy$ -plane, the vehicle moves in the positive  $x$ -direction

```
>> 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

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

and to steer toward the goal which is at the vehicle-relative angle  $\theta^*$  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

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

which turns the steering wheel toward the target. Note the use of the operator  $\ominus$  since  $\theta^*$  and  $\theta$  are angles  $\in \mathbb{S}^1$  not real numbers<sup>4</sup>. 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

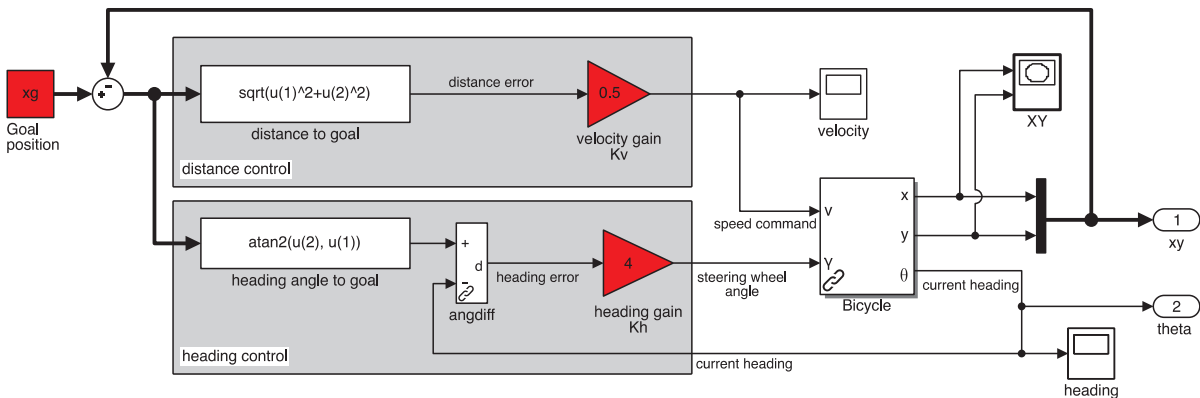
```
>> r = sim('sl_drivepoint');
```

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));
```



**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')
```

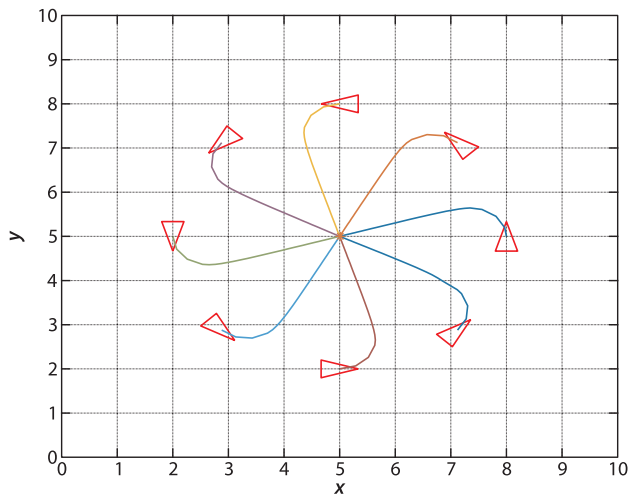
Many Toolbox models create additional figures to display robot animations or graphs as they run.

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

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

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

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

The combined control law

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

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

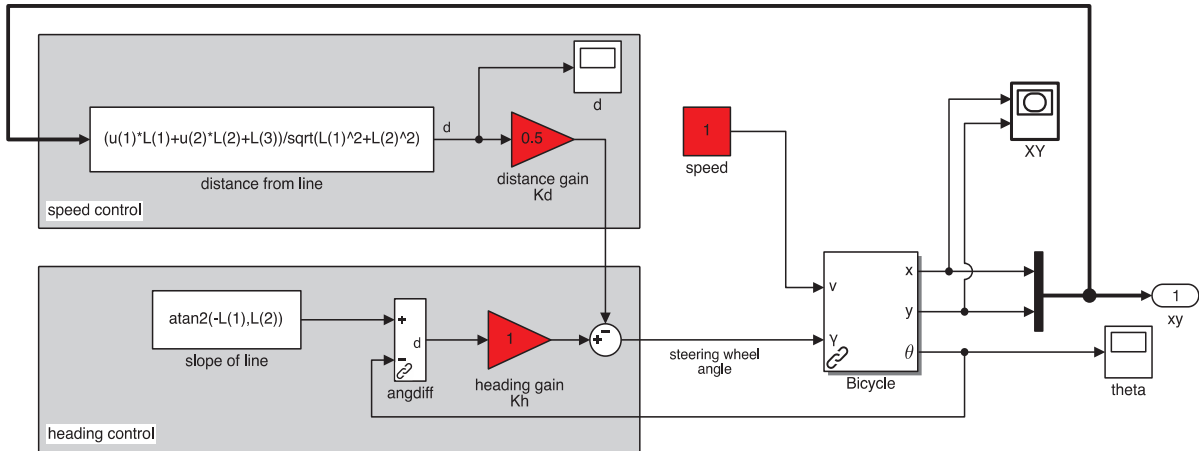
The Simulink model

```
>> sl_driveline
```

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

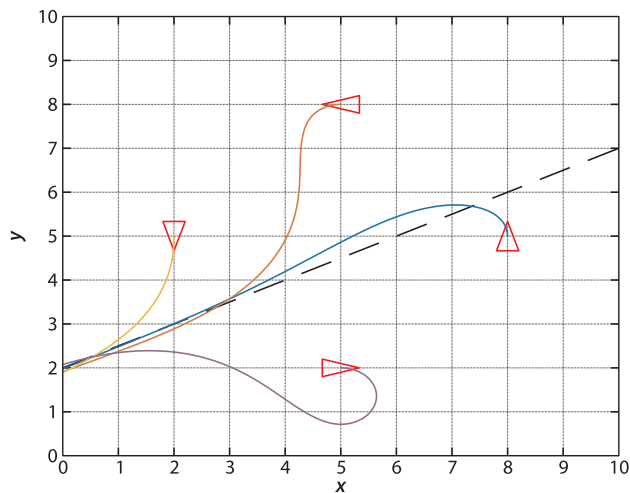
```
>> L = [1 -2 4];
```

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



▲ **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

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

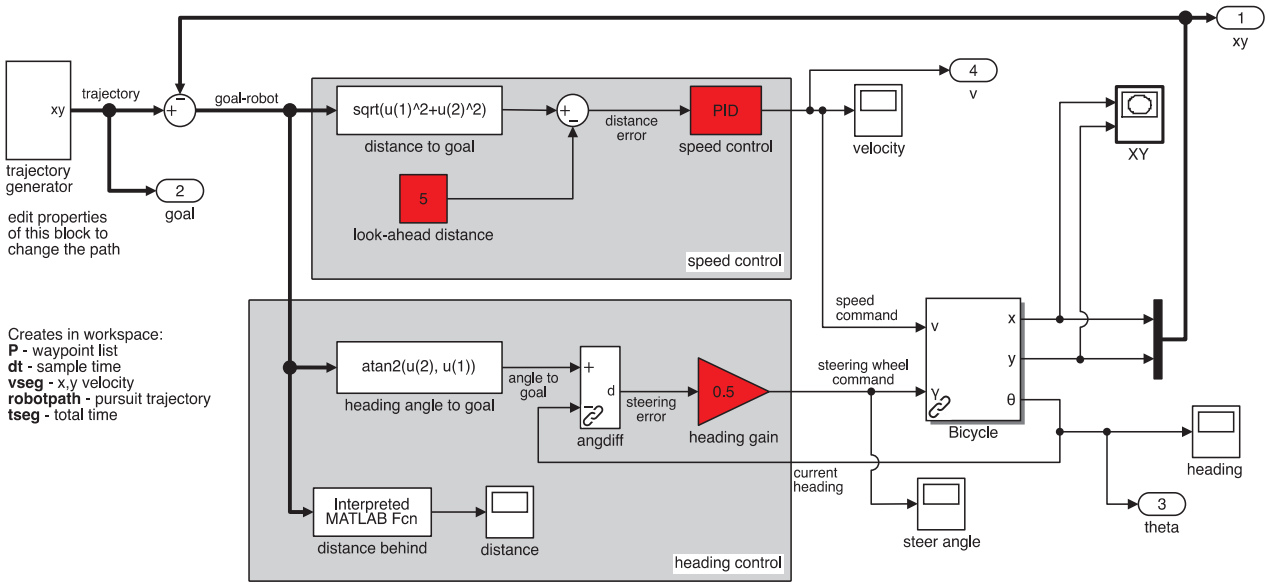


Fig. 4.8. The Simulink model `sl_pursuit` drives the vehicle along a piecewise linear trajectory. Red blocks have parameters that you can adjust to investigate the effect on performance

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

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

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

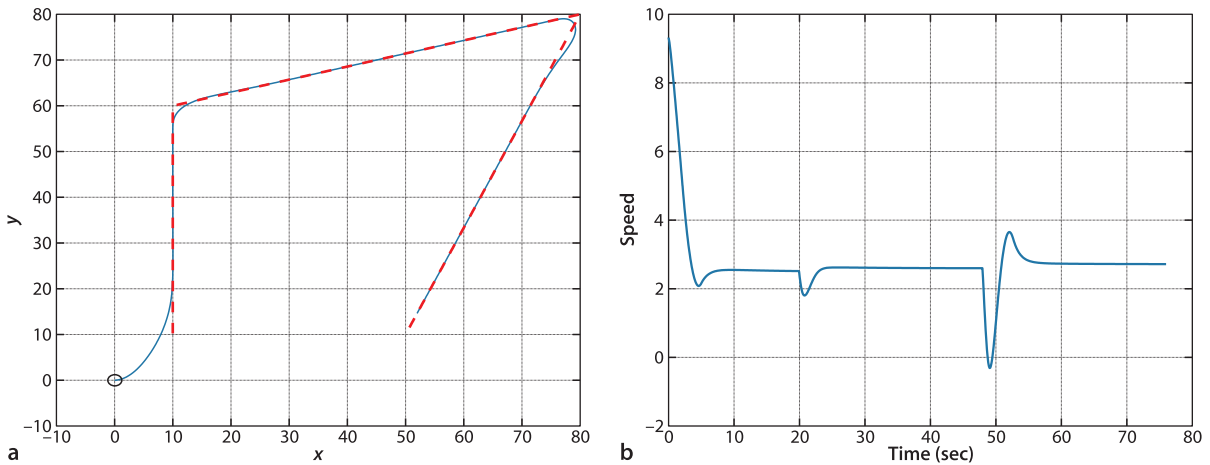
```
>> r = sim('sl_pursuit')
```

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.

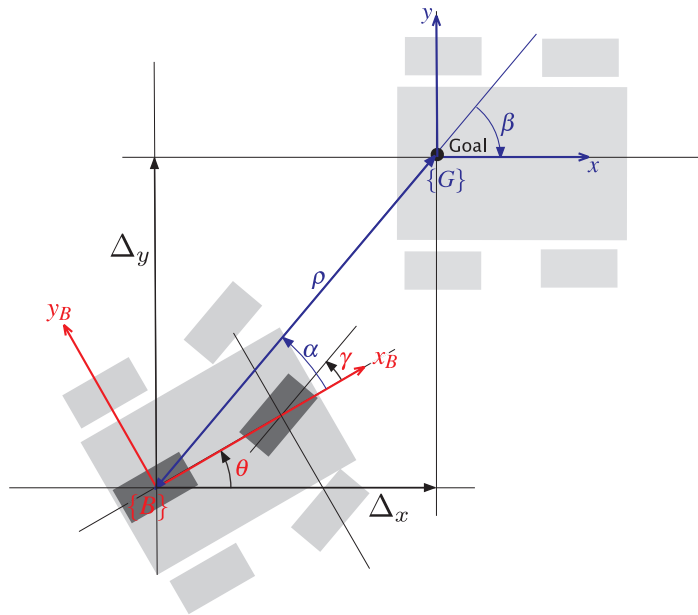
#### 4.1.1.4 Moving to a Pose

The final control problem we discuss is driving to a specific pose  $(x^*, y^*, \theta^*)$ . 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:  $\rho$  is the distance to the goal,  $\beta$  is the angle of the goal vector with respect to the world frame, and  $\alpha$  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

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

where the inputs to the vehicle model are the speed  $v$  and the turning rate  $\omega$  which can be achieved by applying the steering angle  $\gamma$

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

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

$$\begin{aligned} \rho &= \sqrt{\Delta x^2 + \Delta y^2} \\ \alpha &= \tan^{-1} \frac{\Delta y}{\Delta x} - \theta \\ \beta &= -\theta - \alpha \end{aligned}$$

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

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

$$\begin{aligned} v &= k_\rho \rho \\ \omega &= k_\alpha \alpha + k_\beta \beta \end{aligned}$$

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  $\beta \rightarrow 0$ . The closed-loop system

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

is stable so long as

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

The distance and bearing to the goal  $(\rho, \alpha)$  could be measured by a camera or laser range finder, and the angle  $\beta$  could be derived from  $\alpha$  and vehicle orientation  $\theta$  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  $\gamma$  in the control law. The velocity  $v$  always has a constant sign which depends on the initial value of  $\alpha$ .

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  $(x^*, y^*, \theta^*)$  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  $\omega$  to steering wheel angle  $\gamma$ . We specify a goal pose

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

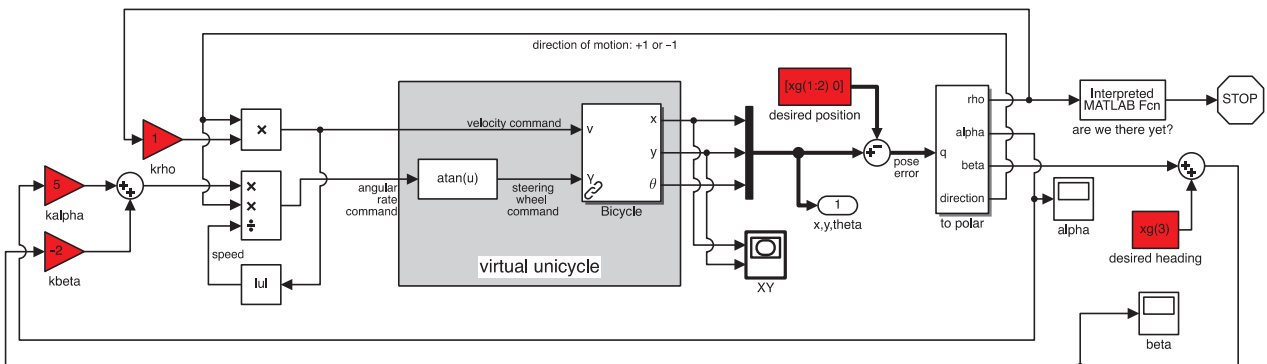


Fig. 4.11. The Simulink model `sl_drivepose` drives the vehicle to a pose. The initial and final poses are set by the workspace variable `x0` and `xf` respectively. Red blocks have parameters that you can adjust to investigate the effect on performance

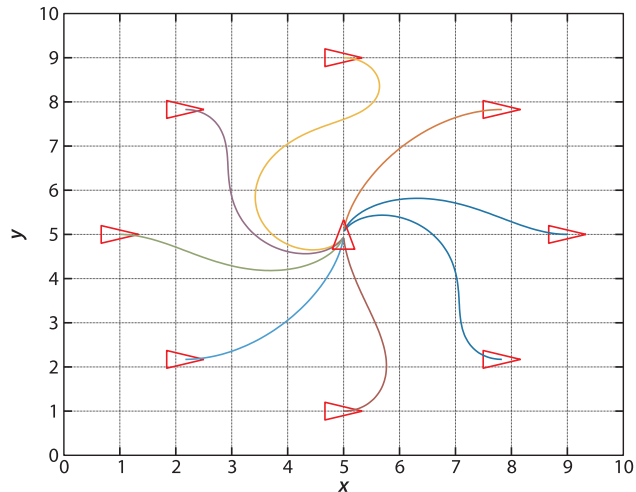


Fig. 4.12.

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

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

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  $\mathbf{q} = (x, y, \theta) \in \mathcal{C}$  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  $R_L$



Fig. 4.13. Clearpath Husky robot with differential drive steering (photo by Tim Barfoot)

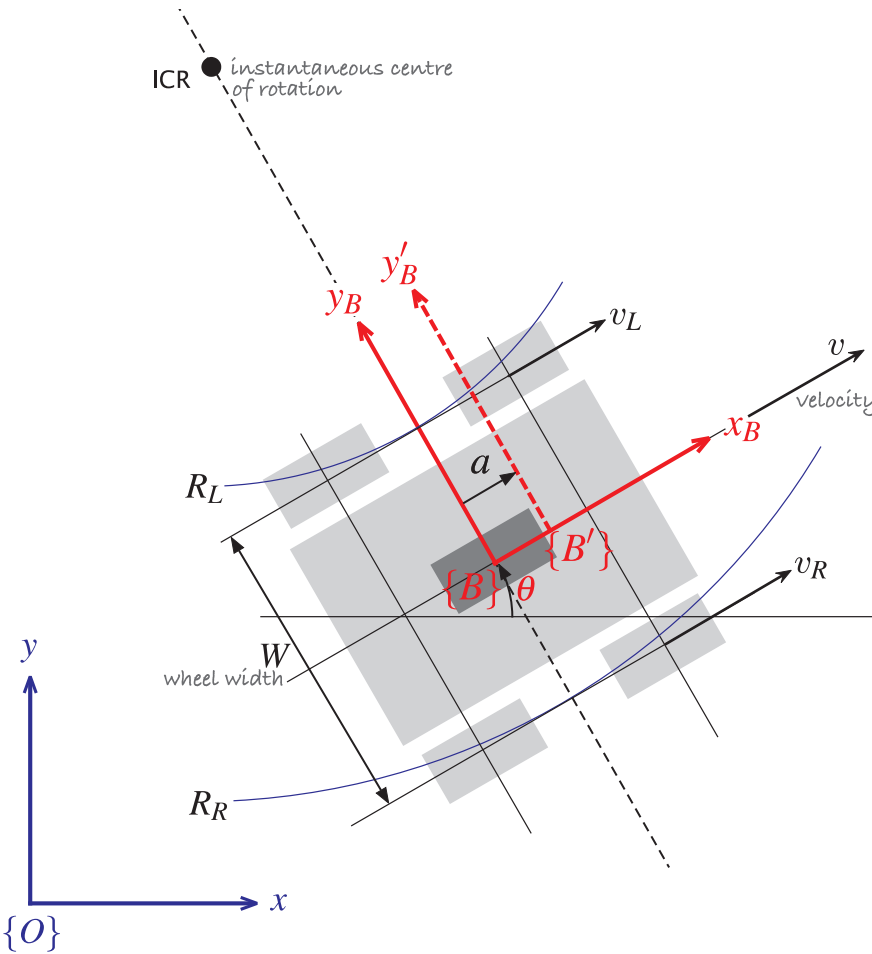


Fig. 4.14. Differential drive robot is shown in light grey, and the unicycle approximation is dark grey. The vehicle's body coordinate frame is shown in red, and the world coordinate frame in blue. The vehicle follows a path around the Instantaneous Center of Rotation (ICR) and the distance from the ICR to the left and right wheels is  $R_L$  and  $R_R$  respectively. We can use the alternative body frame  $\{B'\}$  for trajectory tracking control

while the right-hand wheels move at a speed of  $v_R$  along an arc with a radius of  $R_R$ . The angular velocity of  $\{B\}$  is

$$\dot{\theta} = \frac{v_L}{R_L} = \frac{v_R}{R_R}$$

and since  $R_R = R_L + W$  we can write the turn rate

$$\dot{\theta} = \frac{v_R - v_L}{W} \quad (4.4)$$

in terms of the differential velocity and wheel separation  $W$ . The equations of motion are therefore

$$\begin{aligned} \dot{x} &= v \cos \theta \\ \dot{y} &= v \sin \theta \\ \dot{\theta} &= \frac{v_{\Delta}}{W} \end{aligned} \quad (4.5)$$

where  $v = \frac{1}{2}(v_R + v_L)$  and  $v_{\Delta} = v_R - v_L$  are the average and differential velocities respectively. For a desired speed  $v$  and turn rate  $\dot{\theta}$  we can solve for  $v_R$  and  $v_L$ . This kinematic model is often called the *unicycle model*.

There are similarities and differences to the bicycle model of Eq. 4.2. The turn rate for this vehicle is directly proportional to  $v_{\Delta}$  and is independent of speed – the vehicle can turn even when not moving forward. For the 4-wheel case shown in Fig. 4.14 the axes of the wheels do not intersect the ICR, so when the vehicle is turning the wheel velocity vectors  $v_L$  and  $v_R$  are not tangential to the path – there is a component in the lateral direction which violates the no-slip constraint. This causes skidding or scuffing ◀ which is extreme when the vehicle is turning on the spot – hence differential steering is also called skid steering. Similar to the car-like vehicle we can write an expression for velocity in the vehicle's  $y$ -direction expressed in the world coordinate frame

$$\dot{y} \cos \theta - \dot{x} \sin \theta \equiv 0 \quad (4.6)$$

which is the nonholonomic constraint. It is important to note that the ability to turn on the spot does not make the vehicle holonomic and is fundamentally different to the ability to move in an arbitrary direction which we will discuss next.

If we move the vehicle's reference frame to  $\{B'\}$  and ignore orientation we can re-write Eq. 4.5 in matrix form as

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} \cos \theta & -a \sin \theta \\ \sin \theta & a \cos \theta \end{pmatrix} \begin{pmatrix} v \\ \omega \end{pmatrix}$$

and if  $a \neq 0$  this can be inverted

$$\begin{pmatrix} v \\ \omega \end{pmatrix} = \begin{pmatrix} \cos \theta & \sin \theta \\ -\frac{1}{a} \sin \theta & \frac{1}{a} \cos \theta \end{pmatrix} \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} \quad (4.7)$$

to give the required forward speed and turn rate to achieve an arbitrary velocity  $(\dot{x}, \dot{y})$  for the origin of frame  $\{B'\}$ .

The Toolbox Simulink block library `robblocks` contains a block called `Unicycle` to implement this model and the coordinate frame shift  $a$  is one of its parameters. It has the same outputs as the `Bicycle` model used in the last section. Equation 4.7 is implemented in the block called `Tracking Controller`.

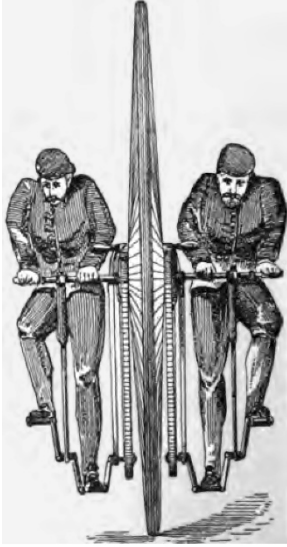
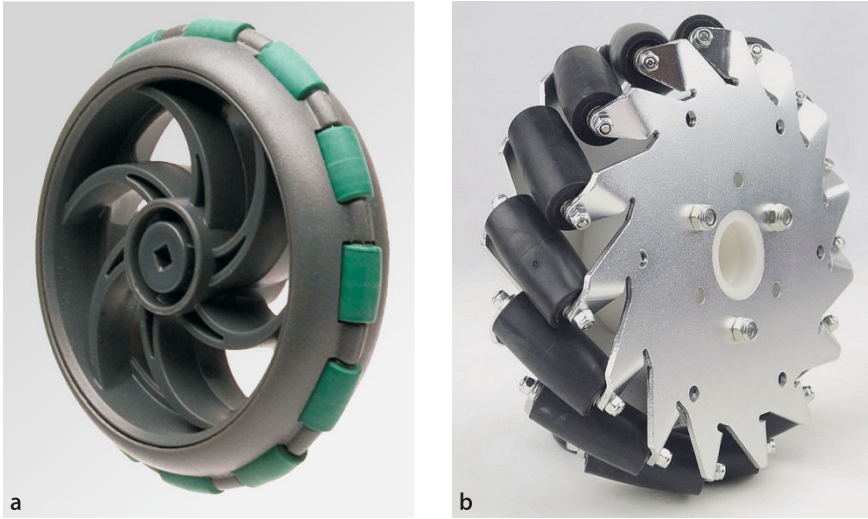


FIG. 171.

From Sharp 1896

For indoor applications this can destroy carpet.





**Fig. 4.15.** Two types of omnidirectional wheel, note the different roller orientation. **a** Allows the wheel to *roll* sideways (courtesy Vex Robotics); **b** allows the wheel to *drive* sideways (courtesy of Nexus Robotics)

### 4.1.3 Omnidirectional Vehicle

The vehicles we have discussed so far have a constraint on lateral motion, the non-holonomic constraint, which necessitates complex maneuvers in order to achieve some goal poses. Alternative wheel designs such as shown in Fig. 4.15 remove this constraint and allow omnidirectional motion. Even more radical is the spherical wheel shown in Fig. 4.16.

In this section we will discuss the mecanum or “Swedish” wheel shown in Fig. 4.15b and schematically in Fig. 4.17. It comprises a number of rollers set around the circumference of the wheel with their axes at an angle of  $\alpha$  relative to the axle of the wheel. The dark roller is the one on the bottom of the wheel and currently in contact with the ground. The rollers have a barrel shape so only one point on the roller is in contact with the ground at any time.

As shown in Fig. 4.17 we establish a wheel coordinate frame  $\{W\}$  with its  $x$ -axis pointing in the direction of wheel motion. Rotation of the wheel will cause forward velocity of  $R\varpi\hat{x}_w$  where  $R$  is the wheel radius and  $\varpi$  is the wheel rotational rate. However because the roller is free to roll in the direction indicated by the green line, normal to the roller’s axis, there is potentially arbitrary velocity in that direction. A desired velocity  $v$  can be resolved into two components, one parallel to the direction of wheel motion  $\hat{x}_w$  and one parallel to the rolling direction

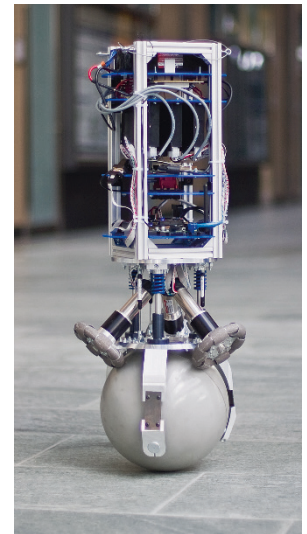
$$\begin{aligned} v &= \underbrace{v_w \hat{x}_w}_{\text{driven}} + \underbrace{v_r (\cos \alpha \hat{x}_w + \sin \alpha \hat{y}_w)}_{\text{rolling}} \\ &= (v_w + v_r \cos \alpha) \hat{x}_w + v_r \sin \alpha \hat{y}_w \end{aligned} \quad (4.8)$$

where  $v_w$  is the speed due to wheel rotation and  $v_r$  is the rolling speed. Expressing  $v = v_x \hat{x}_w + v_y \hat{y}_w$  in component form allows us to solve for the rolling speed  $v_r = v_y / \sin \alpha$  and substituting this into the first term we can solve for the required wheel velocity

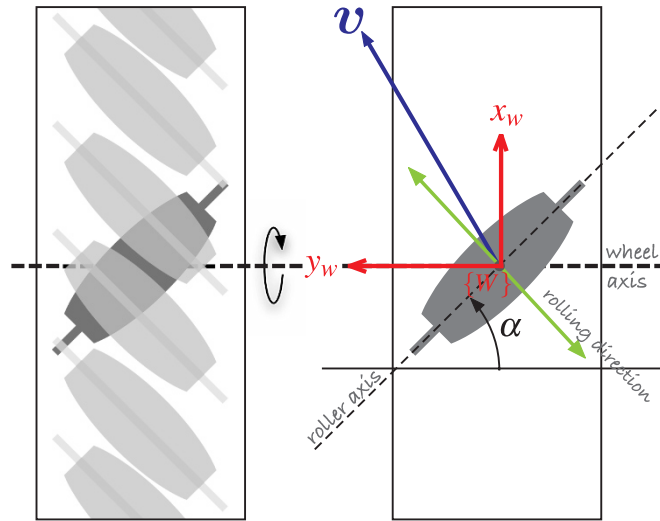
$$v_w = v_x - v_y \cot \alpha \quad (4.9)$$

The required wheel rotation rate is then  $\varpi = v_w / R$ . If  $\alpha = 0$  then  $v_w$  is undefined since the roller axes are parallel to the wheel axis and the wheel can provide no traction. If  $\alpha = \frac{\pi}{2}$  as in Fig. 4.15a, the wheel allows sideways rolling but not sideways driving since there is zero coupling from  $v_w$  to  $v_y$ .

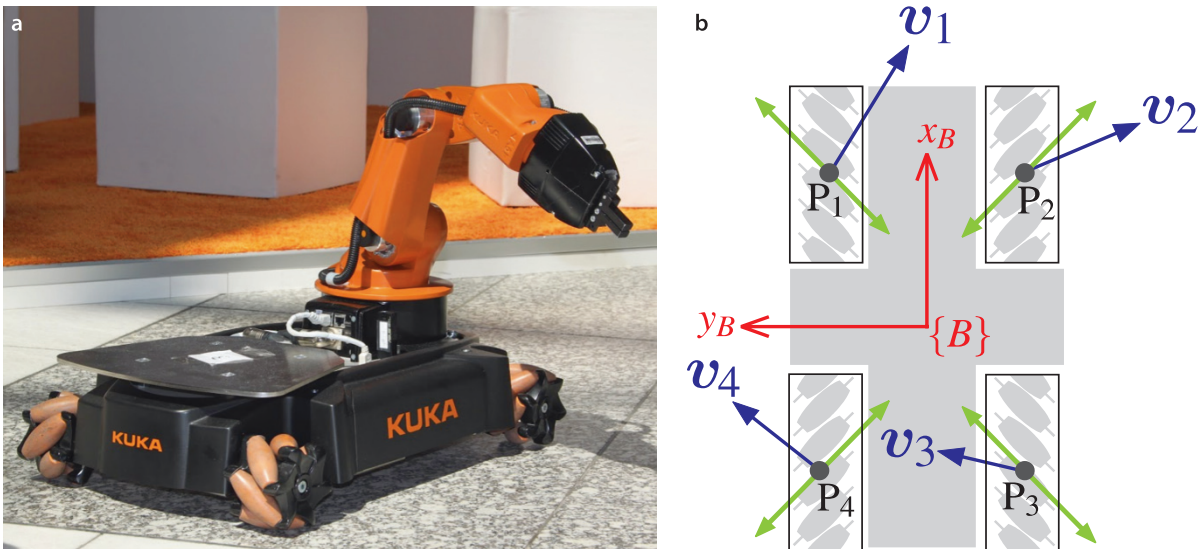
Mecanum was a Swedish company where the wheel was invented by Bengt Ilon in 1973. It is described in US patent 3876255.



**Fig. 4.16.** The Rezero ballbot developed at ETH Zurich (photo by Péter Fankhauser)



**Fig. 4.17.** Schematic of a mecanum wheel in plan view. The *light rollers* are on top of the wheel, the *dark roller* is in contact with the ground. The *green arrow* indicates the rolling direction



**Fig. 4.18.** **a** Kuka youBot, which has four mecanum wheels (image courtesy youBot Store); **b** schematic of a vehicle with four mecanum wheels in the youBot configuration

A single mecanum wheel does not allow any control in the rolling direction but for three or more mecanum wheels, suitably arranged, the motion in the rolling direction of any one wheel will be driven by the other wheels. A vehicle with four mecanum wheels is shown in Fig. 4.18. Its pose is represented by the body frame  $\{B\}$  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  $\mathbf{q} = (x, y, \theta) \in \mathcal{C}$  where  $\mathcal{C} \subset \mathbb{R}^2 \times \mathbb{S}^1$ . The rolling axes of the wheels are orthogonal which means that when the wheels are not rotating the vehicle cannot roll in any direction or rotate.

The four wheel contact points indicated by *grey dots* have coordinate vectors  ${}^B\mathbf{p}_i$ . For a desired body velocity  ${}^B\mathbf{v}_B$  and angular rate  ${}^B\omega$  the velocity at each wheel contact point is

$${}^B\mathbf{v}_i = {}^B\mathbf{v}_B + {}^B\omega \hat{\mathbf{z}}_B \times {}^B\mathbf{p}_i$$

and we then apply Eq. 4.8 and 4.9 to determine wheel rotational rates  $\varpi_i$ , while noting that  $\alpha$  has the opposite sign for wheels 2 and 4 in Eq. 4.8.

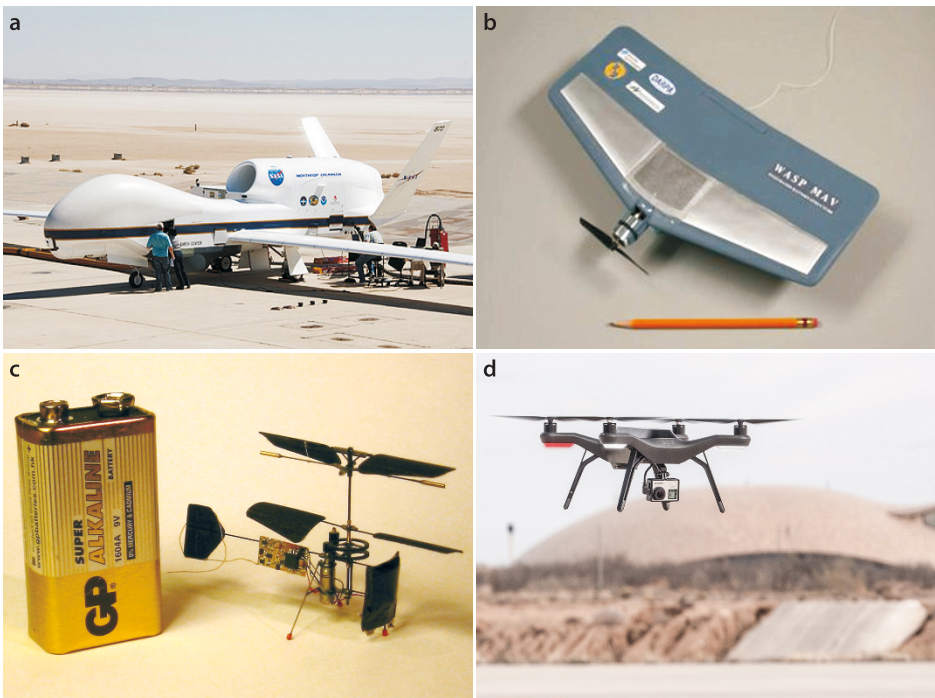
## 4.2 Flying Robots

*In order to fly, all one must do is simply miss the ground.*  
Douglas Adams

Flying robots or unmanned aerial vehicles (UAV) are becoming increasingly common and span a huge range of size and shape as shown in shown in Fig. 4.19. Applications include military operations, surveillance, meteorological observation, robotics research, commercial photography and increasingly hobbyist and personal use. A growing class of flying machines are known as micro air vehicles or MAVs which are smaller than 15 cm in all dimensions. Fixed wing UAVs are similar in principle to passenger aircraft with wings to provide lift, a propeller or jet to provide forward thrust and control surface for maneuvering. Rotorcraft UAVs have a variety of configurations that include conventional *helicopter* design with a main and tail rotor, a *coax* with counter-rotating coaxial rotors and *quadrotors*. Rotorcraft UAVs have the advantage of being able to take off vertically and to hover.

Flying robots differ from ground robots in some important ways. Firstly they have 6 degrees of freedom and their configuration  $\mathbf{q} \in \mathcal{C}$  where  $\mathcal{C} \subset \mathbb{R}^3 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1$ . Secondly they are actuated by forces; that is their motion model is expressed in terms of forces, torques and accelerations rather than velocities as was the case for the ground vehicle models – we use a dynamic rather than a kinematic model. Underwater robots have many similarities to flying robots and can be considered as vehicles that *fly through water* and there are underwater equivalents to fixed wing aircraft and rotorcraft. The principal differences underwater are an upward buoyancy force, drag forces that are much more significant than in air, and added mass.

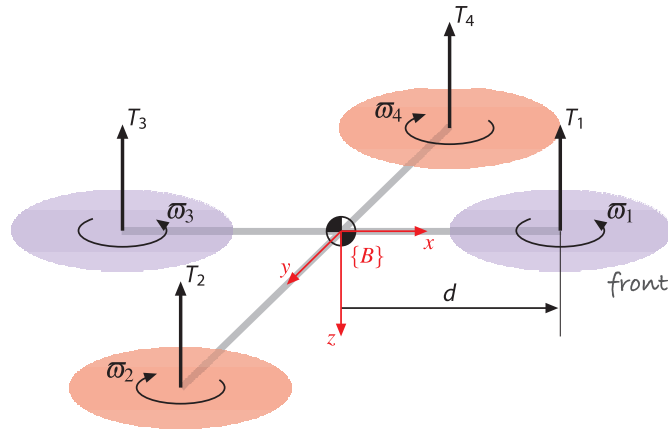
In this section we will create a model for a quadrotor flying vehicle such as shown in Fig. 4.19d. Quadrotors are now widely available, both as commercial products and as open-source projects. Compared to fixed wing aircraft they are highly maneuverable and can be flown safely indoors which makes them well suited for laboratory or hobbyist use. Compared to conventional helicopters, with a large main rotor and tail rotor, the quadrotor is easier to fly, does not have the complex swash plate mechanism and is easier to model and control.



**Fig. 4.19.** Flying robots. **a** Global Hawk unmanned aerial vehicle (UAV) (photo courtesy of NASA), **b** a micro air vehicle (MAV) (photo courtesy of AeroVironment, Inc.), **c** a 1 gram co-axial helicopter with 70 mm rotor diameter (photo courtesy of Petter Muren and Proxflyer AS), **d** a quadrotor which has four rotors and a block of sensing and control electronics in the middle (photo courtesy of 3DRobotics)



**Fig. 4.20.** Quadrotor notation showing the four rotors, their thrust vectors and directions of rotation. The body frame  $\{B\}$  is attached to the vehicle and has its origin at the vehicle's center of mass. Rotors 1 and 3 (blue) rotate counter-clockwise (viewed from above) while rotors 2 and 4 (red) rotate clockwise



The notation for the quadrotor model is shown in Fig. 4.20. The body coordinate frame  $\{B\}$  has its  $z$ -axis downward following the aerospace convention. The quadrotor has four rotors, labeled 1 to 4, mounted at the end of each cross arm. Hex- and octo-rotors are also popular, with the extra rotors providing greater payload lift capability. The approach described here can be generalized to  $N$  rotors, where  $N$  is even.

The rotors are driven by electric motors powered by electronic speed controllers. Some low-cost quadrotors use small motors and reduction gearing to achieve sufficient torque. The rotor speed is  $\varpi_i$  and the thrust is an upward vector

$$T_i = b\varpi_i^2, \quad i = 1, 2, 3, 4 \quad (4.10)$$

in the vehicle's negative  $z$ -direction, where  $b > 0$  is the lift constant that depends on the air density, the cube of the rotor blade radius, the number of blades, and the chord length of the blade. ◀

The translational dynamics of the vehicle in world coordinates is given by Newton's second law

$$m\dot{\mathbf{v}} = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} - {}^0\mathbf{R}_B \begin{pmatrix} 0 \\ 0 \\ T \end{pmatrix} - B\mathbf{v} \quad (4.11)$$

where  $\mathbf{v}$  is the velocity of the vehicle's center of mass in the world frame,  $g$  is gravitational acceleration,  $m$  is the total mass of the vehicle,  $B$  is aerodynamic friction and  $T = \sum T_i$  is the total upward thrust. The first term is the force of gravity which acts downward in the world frame, the second term is the total thrust in the vehicle frame rotated into the world coordinate frame and the third term is aerodynamic drag.

Pairwise differences in rotor thrusts cause the vehicle to rotate. The torque about the vehicle's  $x$ -axis, the *rolling* torque, is generated by the moments

$$\tau_x = dT_4 - dT_2$$

The propeller blades on a rotor craft have fascinating dynamics. When flying into the wind the blade tip coming forward experiences greater lift while the receding blade has less lift. This is equivalent to a torque about an axis pointing into the wind and the rotor blades behave like a gyroscope (see Sect. 3.4.1.1) so the net effect is that the rotor blade plane pitches up by an amount proportional to the apparent or net wind speed, countered by the blade's bending stiffness and the change in lift as a function of blade bending. The pitched blade plane causes a component of the thrust vector to retard the vehicle's forward motion and this velocity dependent force acts like a friction force. This is known as blade flapping and is an important characteristic of blades on all types of rotorcraft.

Close to the ground, height  $< 2d$ , the vehicle experiences increased lift due to a cushion of air beneath it – this is ground effect.

where  $d$  is the distance from the rotor axis to the center of mass. We can write this in terms of rotor speeds by substituting Eq. 4.10

$$\tau_x = db(\varpi_4^2 - \varpi_2^2) \quad (4.12)$$

and similarly for the  $y$ -axis, the *pitching* torque is

$$\tau_y = db(\varpi_1^2 - \varpi_3^2) \quad (4.13)$$

The torque applied to each propeller by the motor is opposed by aerodynamic drag

$$Q_i = k\varpi_i^2$$

where  $k$  depends on the same factors as  $b$ . This torque exerts a reaction torque on the airframe which acts to rotate the airframe about the propeller shaft in the opposite direction to its rotation. The total reaction torque about the  $z$ -axis is

$$\begin{aligned} \tau_z &= Q_1 - Q_2 + Q_3 - Q_4 \\ &= k(\varpi_1^2 + \varpi_3^2 - \varpi_2^2 - \varpi_4^2) \end{aligned} \quad (4.14)$$

where the different signs are due to the different rotation directions of the rotors. A yaw torque can be created simply by appropriate coordinated control of all four rotor speeds.

The total torque applied to the airframe according to Eq. 4.12 to 4.14 is  $\tau = (\tau_x, \tau_y, \tau_z)^T$  and the rotational acceleration is given by Euler's equation of motion from Eq. 3.10

$$J\dot{\omega} = -\omega \times J\omega + \tau \quad (4.15)$$

where  $J$  is the  $3 \times 3$  inertia matrix of the vehicle and  $\omega$  is the angular velocity vector.

The motion of the quadrotor is obtained by integrating the forward dynamics equations Eq. 4.11 and Eq. 4.15 where the forces and moments on the airframe

$$\begin{pmatrix} T \\ \tau \end{pmatrix} = \begin{pmatrix} -b & -b & -b & -b \\ 0 & -db & 0 & db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{pmatrix} \begin{pmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{pmatrix} = A \begin{pmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{pmatrix} \quad (4.16)$$

are functions of the rotor speeds. The matrix  $A$  is constant, and full rank if  $b, k, d > 0$  and can be inverted

$$\begin{pmatrix} \varpi_1^2 \\ \varpi_2^2 \\ \varpi_3^2 \\ \varpi_4^2 \end{pmatrix} = A^{-1} \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} \quad (4.17)$$

to solve for the rotor speeds<sup>►</sup> required to apply a specified thrust  $T$  and moment  $\tau$  to the airframe.

To control the vehicle we will employ a nested control structure which we describe for pitch and  $x$ -translational motion. The innermost loop uses a proportional and derivative controller<sup>►</sup> to compute the required pitching torque on the airframe

$$\tau_y^* = K_{\tau,p}(\theta_p^* - \theta_p^\#) + K_{\tau,d}(\dot{\theta}_p^* - \dot{\theta}_p^\#) \quad (4.18)$$

based on the error between desired and actual pitch angle.<sup>►</sup> The gains  $K_{\tau,p}$  and  $K_{\tau,d}$  are determined by classical control design approaches based on an approximate dy-

The direction of rotation is as shown in Fig. 4.20. Control of motor velocity is discussed in Sect. 9.1.6.

The rotational dynamics has a second-order transfer function of  $\Theta_y(s) / \tau_y(s) = 1 / (Js^2 + Bs)$  where  $J$  is rotational inertia and  $B$  is aerodynamic damping which is generally quite small. To regulate a second-order system requires a proportional-derivative controller.

The term  $\dot{\theta}_p^*$  is commonly ignored.

dynamic model and then tuned to achieve good performance. The actual vehicle pitch angle  $\theta_p^\#$  would be estimated by an inertial navigation system as discussed in Sect. 3.4 and  $\dot{\theta}_p^\#$  would be derived from gyroscopic sensors. The required rotor speeds are then determined using Eq. 4.17.

Consider a coordinate frame  $\{B'\}$  attached to the vehicle and with the same origin as  $\{B\}$  but with its  $x$ - and  $y$ -axes in the horizontal plane and parallel to the ground. The thrust vector is parallel to the  $z$ -axis of frame  $\{B\}$  and pitching the nose down, rotating about the  $y$ -axis by  $\theta_p$ , generates a force

$${}^{B'}\mathbf{f} = \mathcal{R}_y(\theta_p) \cdot \begin{pmatrix} 0 \\ 0 \\ T \end{pmatrix} = \begin{pmatrix} T \sin \theta_p \\ 0 \\ T \cos \theta_p \end{pmatrix}$$

which has a component

$${}^{B'}\mathbf{f}_x = T \sin \theta_p \approx T \theta_p$$

that accelerates the vehicle in the  ${}^{B'}x$ -direction, and we have assumed that  $\theta_p$  is small. We can control the velocity in this direction with a proportional control law

$${}^{B'}\mathbf{f}_x^* = m K_f ({}^{B'}v_x^* - {}^{B'}v_x^\#)$$

where  $K_f > 0$  is a gain. Combining these two equations we obtain the desired pitch angle

$$\theta_p^* \approx \frac{m}{T} K_f ({}^{B'}v_x^* - {}^{B'}v_x^\#) \quad (4.19)$$

required to achieve the desired forward velocity. Using Eq. 4.18 we compute the required pitching torque, and then using Eq. 4.17 the required rotor speeds. For a vehicle in vertical equilibrium the total thrust equals the weight force so  $m/T \approx 1/g$ .

The actual vehicle velocity  ${}^{B'}v_x$  would be estimated by an inertial navigation system as discussed in Sect. 3.4 or a GPS receiver. If the position of the vehicle in the  $xy$ -plane of the world frame is  $\mathbf{p} \in \mathbb{R}^2$  then the desired velocity is given by the proportional control law


$${}^0\mathbf{v}^* = K_p ({}^0\mathbf{p}^* - {}^0\mathbf{p}^\#) \quad (4.20)$$

based on the error between the desired and actual position. The desired velocity in the  $xy$ -plane of frame  $\{B'\}$  is

$${}^{B'}\mathbf{v} = \ominus^0 \mathcal{R}_{B'}(\theta_y) \cdot {}^0\mathbf{v}, \quad \mathcal{R} \in \mathbf{SO}(2)$$

which is a function of the yaw angle  $\theta_y$

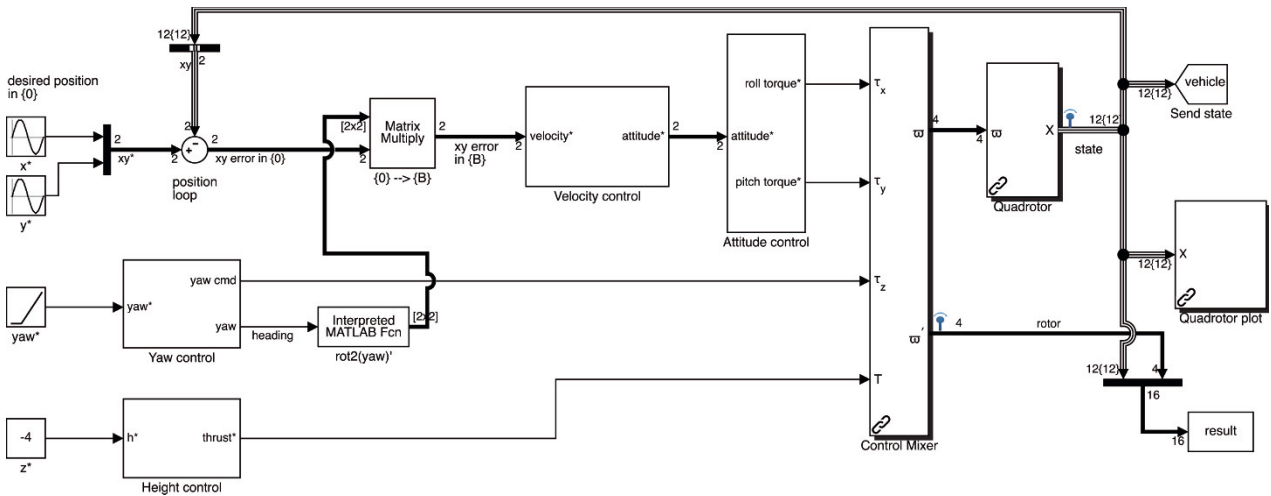
$$\begin{pmatrix} {}^{B'}v_x \\ {}^{B'}v_y \end{pmatrix} = \begin{pmatrix} \cos \theta_y & -\sin \theta_y \\ \sin \theta_y & \cos \theta_y \end{pmatrix}^T \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

Figure 4.21 shows a Simulink model of the complete control system for a quadrotor  which can be loaded and displayed by

```
>> sl_quadrotor
```

Working our way left to right and starting at the top we have the desired position of the quadrotor in world coordinates. The position error is rotated from the world frame to the body frame and becomes the desired velocity. The velocity controller implements Eq. 4.19 and its equivalent for the roll axis and outputs the desired pitch and roll angles of the quadrotor. The attitude controller is a proportional-derivative controller that determines the appropriate pitch and roll torques to achieve these

This model is hierarchical and organized in terms of subsystems. Click the down arrow on a subsystem (can be seen on-screen but not in the figure) to reveal the detail. Double-click on the subsystem box to modify its parameters.



**Fig. 4.21.** The Simulink® model `sl_quadrotor` which is a closed-loop simulation of the quadrotor. The vehicle takes off and flies in a circle at constant altitude. A Simulink bus is used for the 12-element state vector  $X$  output by the `Quadrotor` block. To reduce the number of lines in the diagram we have used `Goto` and `From` blocks to transmit and receive the state vector

angles based on feedback of current attitude and attitude rate. The yaw control block determines the error in heading angle and implements a proportional-derivative controller to compute the required yaw torque which is achieved by speeding up one pair of rotors and slowing the other pair.

Altitude is controlled by a proportional-derivative controller

$$T = K_p(z^* - z^\#) + K_d(\dot{z}^* - \dot{z}^\#) + T_0$$

which determines the average rotor speed.  $T_0 = mg$  is the weight of the vehicle and this is an example of feedforward control – used here to counter the effect of gravity which otherwise is a constant disturbance to the altitude control loop. The alternatives to feedforward control would be to have very high gain for the altitude loop which often leads to actuator saturation and instability, or a proportional-integral (PI) controller which might require a long time for the integral term to increase to a useful value and then lead to overshoot. We will revisit gravity compensation in Chap. 9 applied to arm-type robots.

The control mixer block combines the three torque demands and the vertical thrust demand and implements Eq. 4.17 to determine the appropriate rotor speeds. Rotor speed limits are applied here. These are input to the quadrotor block which implements the forward dynamics integrating Eq. 4.16 to give the position, velocity, orientation and orientation rate. The output of this block is the state vector  $x = ({}^0p, {}^0T, {}^B\dot{p}, {}^B\dot{T}) \in \mathbb{R}^{12}$ . As is common in aerospace applications we represent orientation  $T$  and orientation rate  $\dot{T}$  in terms of roll-pitch-yaw angles. Note that position and attitude are in the world frame while the rates are expressed in the body frame.

The parameters of a specific quadrotor can be loaded

```
>> mdl_quadrotor
```

which creates a structure called `quadrotor` in the workspace, and its elements are the various dynamic properties of the quadrotor. The simulation can be run using the Simulink menu or from the MATLAB command line

```
>> sim('sl_quadrotor');
```

and it displays an animation in a separate window. The vehicle lifts off and flies around a circle while spinning slowly about its own  $z$ -axis. A snapshot is shown in Fig. 4.22. The simulation writes the results from each timestep into a matrix in the workspace

```
>> about result
result [double] : 2412x16 (308.7 kB)
```

Note that according to the coordinate conventions shown in Fig. 4.20  $x$ -direction motion requires a negative rotation about the  $y$ -axis (pitch angle) and  $y$ -direction motion requires a positive rotation about the  $x$ -axis (roll angle) so the gains have different signs for the roll and pitch loops.

The Simulink library `roblocks` also includes a block for an  $N$ -rotor vehicle.

Loading and displaying the model using `>> sl_quadrotor` automatically loads the default quadrotor model. This is done by the `PreLoadFcn` callback set from model's properties File+Model Properties+Model Properties+Call-backs+PreLoadFcn.

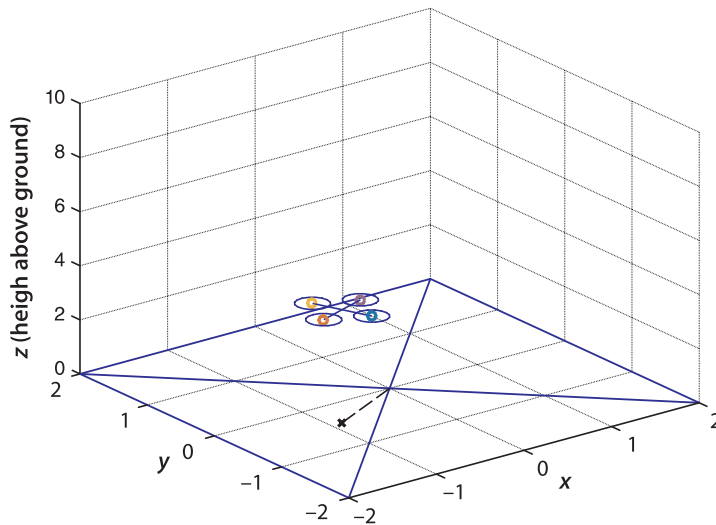


Fig. 4.22.

One frame from the quadrotor simulation. The marker on the ground plane is a projection of the vehicle's centroid

which has one row per timestep, and each row contains the time followed by the state vector (elements 2–13) and the commanded rotor speeds  $\omega_i$  (elements 14–17). To plot  $x$  and  $y$  versus time is

```
>> plot(result(:,1), result(:,2:3));
```

To recap on control of the quadrotor. A position error results in a required translational velocity. To achieve this requires appropriate pitch and roll angles so that a component of the vehicle's thrust acts in the horizontal plane and generates a force to accelerate the vehicle. ◀ As it approaches its goal the airframe must be rotated in the opposite direction so that a component of thrust decelerates the motion. To achieve the pitch and roll angles requires differential propeller thrust to create a moment that rotationally accelerates the airframe.

This indirection from translational motion to rotational motion is a consequence of the vehicle being under-actuated – we have just four rotor speeds to adjust but the vehicle's configuration space is 6-dimensional. In the configuration space we cannot move in the  $x$ - or  $y$ -direction, but we can move in the pitch- or roll-direction which results in motion in the  $x$ - or  $y$ -direction. The cost of under actuation is once again a maneuver. The pitch and roll angles are a means to achieve translation control and cannot be independently set.

The total thrust must be increased so that the vertical thrust component still balances gravity.

## 4.3 Advanced Topics

### 4.3.1 Nonholonomic and Under-Actuated Systems

We introduced the notion of configuration space in Sect. 2.3.5 and it is useful to revisit it now that we have discussed several different types of mobile robot platform. Common vehicles – as diverse as cars, hovercrafts, ships and aircraft – are all able to move forward effectively but are unable to instantaneously move sideways. This is a very sensible tradeoff that simplifies design and caters to the motion we most commonly require of the vehicle. Sideways motion for occasional tasks such as parking a car, docking a ship or landing an aircraft are possible, albeit with some complex maneuvering but humans can learn this skill.

Consider a hovercraft which moves over a planar surface. To fully describe all its constituent particles we need to specify three generalized coordinates: its position in the  $xy$ -plane and its rotation angle. It has three degrees of freedom and its configuration space is  $\mathcal{C} \subset \mathbb{R}^2 \times \mathbb{S}^1$ . This hovercraft has two propellers whose axes are parallel but not

collinear. The sum of their thrusts provide a forward force and the difference in thrusts generates a yawing torque for steering. The number of actuators, two, is less than its degrees of freedom  $\dim \mathcal{C} = 3$  and we call this an under-actuated system. This imposes significant limitations on the way in which it can move. At any point in time we can control the forward (parallel to the thrust vectors) acceleration and the rotational acceleration of the hovercraft but there is zero sideways (or lateral) acceleration since it cannot generate any lateral thrust. Nevertheless with some clever maneuvering, like with a car, the hovercraft can follow a path that will take it to a place to one side of where it started. In the hovercraft's 3-dimensional configuration space this means that at any point there are certain directions in which *acceleration* is not possible. We can reach points in those direction but not directly, only by following some circuitous path.

All flying and underwater vehicles have a configuration that is completely described by six generalized coordinates – their position and orientation in 3D space.  $\mathcal{C} \subset \mathbb{R}^3 \times \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1$  where the orientation is expressed in some three-angle representation – since  $\dim \mathcal{C} = 6$  the vehicles have six degrees of freedom. A quadrotor has four actuators, four thrust-generating propellers, and this is fewer than its degrees of freedom making it under-actuated. Controlling the four propellers causes motion in the up/down, roll, pitch and yaw directions of the configuration space but not in the forward/backward or left/right directions. To access those degrees of freedom it is necessary to perform a *maneuver*: pitch down so that the thrust vector provides a horizontal force component, accelerate forward, pitch up so that the thrust vector provides a horizontal force component to decelerate, and then level out.

For a helicopter only four of the six degrees of freedom are practically useful: up/down, forward/backward, left/right and yaw. Therefore a helicopter requires a minimum of four actuators: the main rotor generates a thrust vector whose magnitude is controlled by the collective pitch and whose direction is controlled by the lateral and longitudinal cyclic pitch. The tail rotor provides a yawing moment. This leaves two degrees of freedom unactuated, roll and pitch angles, but clever design ensures that gravity actuates them and keeps them close to zero – without gravity a helicopter cannot work. A fixed-wing aircraft moves forward very efficiently and also has four actuators: engine thrust provides acceleration in the forward direction and the ailerons, elevator and rudder exert respectively roll, pitch and yaw moments on the aircraft. ▶ To access the missing degrees of freedom such as up/down and left/right translation, the aircraft must pitch or yaw while moving forward.

The advantage of under-actuation is having fewer actuators. In practice this means real savings in terms of cost, complexity and weight. The consequence is that at any point in its configuration space there are certain directions in which the vehicle cannot move. Full actuation is possible but not common, for example the DEPTHX underwater robot shown on page 96 has six degrees of freedom and six actuators. These can exert an arbitrary force and torque on the vehicle, allowing it to accelerate in any direction or about any axis.

A 4-wheeled car has many similarities to the hovercraft discussed above. It moves over a planar surface and its configuration can be fully described by its generalized coordinates: its position in the  $xy$ -plane and a rotation angle. It has three degrees of freedom and its configuration space is  $\mathcal{C} \subset \mathbb{R}^2 \times \mathbb{S}^1$ . A car has two actuators, one to move forwards or backwards and one to change the heading direction. A car, like a hovercraft, is under-actuated.

We know from our experience with cars that we cannot move directly in certain directions and sometimes needs to perform a maneuver to reach our goal. A differential- or skid-steered vehicle, such as a tank, is also under-actuated – it has only two actuators, one for each track. While this type of vehicle can turn on the spot it cannot move sideways. To do that it has to turn, proceed, stop then turn – this need to maneuver is the clear signature of an under-actuated system.

We might often wish for an ability to drive our car sideways but the standard wheel provides real benefit when cornering – lateral friction between the wheels and the

Some low-cost hobby aircraft have no rudder and rely only on ailerons to bank and turn the aircraft. Even cheaper hobby aircraft have no elevator and rely on engine speed to control height.

**Table 4.1.**  
Summary of configuration space characteristics for various robots. A nonholonomic system is under-actuated and/or has a rolling constraint

	dim $\mathcal{C}$	Degrees of freedom	Number of actuators	Actuation	Rolling constraints	Holonomic
Train	1	1	1	full		✓
2-joint robot arm	2	2	2	full		✓
6-joint robot arm	6	6	6	full		✓
10-joint robot arm	10	10	10	over		✓
Hovercraft	3	3	2	under		
Car	3	2	2	under	✓	
Helicopter	6	6	4	under		
Fixed wing aircraft	6	6	4	under		
DEPTHX AUV	6	6	6	full		✓

road provides, for free, the centripetal force which would otherwise require an extra actuator to provide. The hovercraft has many similarities to a car but we can push a hovercraft sideways – we cannot do that with a car. This lateral friction is a distinguishing feature of the car.

The hovercraft, aerial and underwater vehicles are controlled by forces so in this case the constraints are on vehicle acceleration in configuration space not velocity.

The inability to slip sideways is a constraint, the *rolling* constraint, on the velocity of the vehicle just as under-actuation is. A vehicle with one or more velocity constraints, due to under-actuation or a rolling constraint, is referred to as a nonholonomic system. A key characteristic of these systems is that they cannot move *directly* from one configuration to another – they must perform a maneuver or sequence of motions. A car has a velocity constraint due to its wheels and is also under-actuated.

For example fixing the end of the 10-joint robot arm introduces six holonomic constraints (position and orientation) so the arm would have only 4 degrees of freedom.

A holonomic constraint restricts the possible configurations that the system can achieve – it can be expressed as an equation written in terms of the configuration variables. A nonholonomic constraint such as Eq. 4.3 and 4.6 is one that restricts the *velocity* (or acceleration) of a system in configuration space – it can only be expressed in terms of the derivatives of the configuration variables. The nonholonomic constraint does not restrict the possible configurations the system can achieve but it does preclude instantaneous velocity or acceleration in certain directions.

The constraint cannot be integrated to a constraint in terms of configuration variables, so such systems are also known as nonintegrable systems.

In control theoretic terms Brockett’s theorem (Brockett 1983) states that nonholonomic systems are controllable but they cannot be stabilized to a desired state using a differentiable, or even continuous, pure state-feedback controller. A time-varying or nonlinear control strategy is required which means that the robot follows some generally nonlinear path. One exception is an under-actuated system moving in 3-dimensional space within a force field, for example a gravity field – gravity acts like an additional actuator and makes the system linearly controllable (but not holonomic), as we showed for the quadrotor example in Sect. 4.2.

Mobility parameters for the various robots that we have discussed here, and earlier in Sect. 2.3.5, are tabulated in Table 4.1. We will discuss under- and over-actuation in the context of arm robots in Chap. 8.

## 4.4 Wrapping Up

In this chapter we have created and discussed models and controllers for a number of common, but quite different, robot platforms. We first discussed wheeled robots. For car-like vehicles we developed a kinematic model which we used to develop a number of different controllers in order that the platform could perform useful tasks such as driving to a point, driving along a line, following a trajectory or driving to a pose. We then discussed differentially steered vehicles on which many robots are based, and omnidirectional robots based on novel wheel types. Then we we discussed a simple but common



flying vehicle, the quadrotor, and developed a dynamic model and a hierarchical control system that allowed the quadrotor to fly a circuit. This hierarchical or nested control approach is described in more detail in Sect. 9.1.7 in the context of robot arms.

We also extended our earlier discussion about configuration space to include the velocity constraints due to under actuation and rolling constraints from wheels.

The next chapters in this Part will discuss how to plan paths for robots through complex environments that contain obstacles and then how to determine the location of a robot.

---

### Further Reading

Comprehensive modeling of mobile ground robots is provided in the book by Siegwart et al. (2011). In addition to the models covered here, it presents in-depth discussion of a variety of wheel configurations with different combinations of driven wheels, steered wheels and passive castors. The book by Kelly (2013) also covers vehicle modeling and control. Both books also provide a good introduction to perception, localization and navigation which we will discuss in the coming chapters.

The paper by Martins et al. (2008) discusses kinematics, dynamics and control of differential steer robots. The Handbook of Robotics (Siciliano and Khatib 2016, part E) covers modeling and control of various vehicle types including aerial and underwater. The theory of helicopters with an emphasis on robotics is provided by Mettler (2003) but the definitive reference for helicopter dynamics is the very large book by Prouty (2002). The book by Antonelli (2014) provides comprehensive coverage of modeling and control of underwater robots.

Some of the earliest papers on quadrotor modeling and control are by Pounds, Mahony and colleagues (Hamel et al. 2002; Pounds et al. 2004, 2006). The thesis by Pounds (2007) presents comprehensive aerodynamic modeling of a quadrotor with a particular focus on blade flapping, a phenomenon well known in conventional helicopters but largely ignored for quadrotors. A tutorial introduction to the control of multi-rotor flying robots is given by Mahony, Kumar, and Corke (2012). Quadrotors are now commercially available from many vendors at quite low cost. There are also a number of hardware kits and open-source software projects such as ArduCopter and Mikrokopter.

Mobile ground robots are now a mature technology for transporting parts around manufacturing plants. The research frontier is now for vehicles that operate autonomously in outdoor environments (Siciliano and Khatib 2016, part F). Research into the automation of passenger cars has been ongoing since the 1980s and a number of automative manufacturers are talking about commercial autonomous cars by 2020.

**Historical and interesting.** The Navlab project at Carnegie-Mellon University started in 1984 and a series of autonomous vehicles, Navlabs, were built and a large body of research has resulted. All vehicles made strong use of computer vision for navigation. In 1995 the supervised autonomous Navlab 5 made a 3 000-mile journey, dubbed “No Hands Across America” (Pomerleau and Jochem 1995, 1996). The vehicle steered itself 98% of the time largely by visual sensing of the white lines at the edge of the road.

In Europe, Ernst Dickmanns and his team at Universität der Bundeswehr München demonstrated autonomous control of vehicles. In 1988 the VaMoRs system, a 5 tonne Mercedes-Benz van, could drive itself at speeds over  $90 \text{ km h}^{-1}$  (Dickmanns and Graefe 1988b; Dickmanns and Zapp 1987; Dickmanns 2007). The European Prometheus Project ran from 1987–1995 and in 1994 the robot vehicles VaMP and VITA-2 drove more than 1 000 km on a Paris multi-lane highway in standard heavy traffic at speeds up to  $130 \text{ km h}^{-1}$ . They demonstrated autonomous driving in free lanes, convoy driving, automatic tracking of other vehicles, and lane changes with autonomous passing



of other cars. In 1995 an autonomous S-Class Mercedes-Benz made a 1 600 km trip from Munich to Copenhagen and back. On the German Autobahn speeds exceeded  $175 \text{ km h}^{-1}$  and the vehicle executed traffic maneuvers such as overtaking. The mean time between human interventions was 9 km and it drove up to 158 km without any human intervention. The UK part of the project demonstrated autonomous driving of an XJ6 Jaguar with vision (Matthews et al. 1995) and radar-based sensing for lane keeping and collision avoidance. More recently, in the USA a series of Grand Challenges were run for autonomous cars. The 2005 desert and 2007 urban challenges are comprehensively described in compilations of papers from the various teams in Buehler et al. (2007, 2010). More recent demonstrations of self-driving vehicles are a journey along the fabled silk road described by Bertozzi et al. (2011) and a classic road trip through Germany by Ziegler et al. (2014).

Ackermann's magazine can be found online at <http://smithandgosling.wordpress.com/2009/12/02/ackermanns-repository-of-arts> and the carriage steering mechanism is published in the March and April issues of 1818. King-Hele (2002) provides a comprehensive discussion about the prior work on steering geometry and Darwin's earlier invention.

---

### Toolbox and MATLAB Notes

In addition to the Simulink `Bicycle` model used in this chapter the Toolbox also provides a MATLAB class which implements these kinematic equations and which we will use in Chap. 6. For example we can create a vehicle model with steer angle and speed limits

```
>> veh = Bicycle('speedmax', 1, 'steermax', 30*pi/180);
```

and evaluate Eq. 4.2 for a particular state and set of control inputs ( $v, \gamma$ )

```
>> veh.deriv([], [0 0 0], [0.3, 0.2])
ans =
    0.3000         0    0.0608
```

The `Unicycle` class is used for a differentially-steered robot and has equivalent methods.

The Robotics System Toolbox™ from The MathWorks has support for differentially-steered mobile robots which can be created using the function `ExampleHelperRobotSimulator`. It also includes a class `robotics.PurePursuit` that implements pure pursuit for a differential steer robot. An example is given in the Toolbox RST folder.

---

### Exercises

- For a 4-wheel vehicle with  $L = 2 \text{ m}$  and width between wheel centers of  $1.5 \text{ m}$ 
  - What steering wheel angle is needed for a turn rate of  $10 \text{ deg s}^{-1}$  at a forward speed of  $20 \text{ km h}^{-1}$ ?
  - compute the difference in wheel steer angle for Ackermann steering around curves of radius 10, 50 and 100 m.
  - If the vehicle is moving at  $80 \text{ km h}^{-1}$  compute the difference in back wheel rotation rates for curves of radius 10, 50 and 100 m.
- Write an expression for turn rate in terms of the angular rotation rate of the two back wheels. Investigate the effect of errors in wheel radius and vehicle width.
- Consider a car and bus with  $L = 4$  and  $12 \text{ m}$  respectively. To follow a curve with radius of 10, 20 and 50 m determine the respective steered wheel angles.
- For a number of steered wheel angles in the range  $-45$  to  $45^\circ$  and a velocity of  $2 \text{ m s}^{-1}$  overlay plots of the vehicle's trajectory in the  $xy$ -plane.

5. Implement the  $\ominus$  operator used in Sect. 4.1.1.1 and check against the code for `angdiff`.
6. Moving to a point (page 103) plot  $x$ ,  $y$  and  $\theta$  against time.
7. Pure pursuit example (page 106)
  - a) Investigate what happens if you vary the look-ahead distance, heading gain or proportional gain in the speed controller.
  - b) Investigate what happens when the integral gain in the speed controller is zero.
  - c) With integral set to zero, add a constant to the output of the controller. What should the value of the constant be?
  - d) Add a velocity feedforward term.
  - e) Modify the pure pursuit example so the robot follows a slalom course.
  - f) Modify the pure pursuit example to follow a target moving back and forth along a line.
8. Moving to a pose (page 107)
  - a) Repeat the example with a different initial orientation.
  - b) Implement a parallel parking maneuver. Is the resulting path practical?
  - c) Experiment with different control parameters.
9. Use the MATLAB GUI interface to make a simple steering wheel and velocity control, and use this to create a very simple driving simulator. Alternatively interface a gaming steering wheel and pedal to MATLAB.
10. Adapt the various controllers in Sect. 4.1.1 to the differentially steered robot.
11. Derive Eq. 4.4 from the preceding equation.
12. For constant forward velocity, plot  $v_L$  and  $v_R$  as a function of ICR position along the  $y$ -axis. Under what conditions do  $v_L$  and  $v_R$  have a different sign?
13. Using Simulink implement a controller using Eq. 4.7 that moves the robot in its  $y$ -direction. How does the robot's orientation change as it moves?
14. Sketch the design for a robot with three mecanum wheels. Ensure that it cannot roll freely and that it can drive in any direction. Write code to convert from desired vehicle translational and rotational velocity to wheel rotation rates.
15. For the 4-wheel omnidirectional robot of Sect. 4.1.3 write an algorithm that will allow it to move in a circle of radius 0.5 m around a point with its nose always pointed toward the center of the circle.
16. Quadrotor (page 115)
  - a) At equilibrium, compute the speed of all the propellers.
  - b) Experiment with different control gains. What happens if you reduce the damping gains to zero?
  - c) Remove the gravity feedforward and experiment with large altitude gain or a PI controller.
  - d) When the vehicle has nonzero roll and pitch angles, the magnitude of the vertical thrust is reduced and the vehicle will slowly descend. Add compensation to the vertical thrust to correct this.
  - e) Simulate the quadrotor flying inverted, that is, its  $z$ -axis is pointing upwards.
  - f) Program a ballistic motion. Have the quadrotor take off at 45 deg to horizontal then remove all thrust.
  - g) Program a smooth landing.
  - h) Program a barrel roll maneuver. Have the quadrotor fly horizontally in its  $x$ -direction and then increase the roll angle from 0 to  $2\pi$ .
  - i) Program a flip maneuver. Have the quadrotor fly horizontally in its  $x$ -direction and then increase the pitch angle from 0 to  $2\pi$ .
  - j) Add another four rotors.
  - k) Use the function `mstraj` to create a trajectory through ten via points  $(X_p, Y_p, Z_p, \theta_y)$  and modify the controller of Fig. 4.21 for smooth pursuit of this trajectory.
  - l) Use the MATLAB GUI interface to make a simple joystick control, and use this to create a very simple flying simulator. Alternatively interface a gaming joystick to MATLAB.