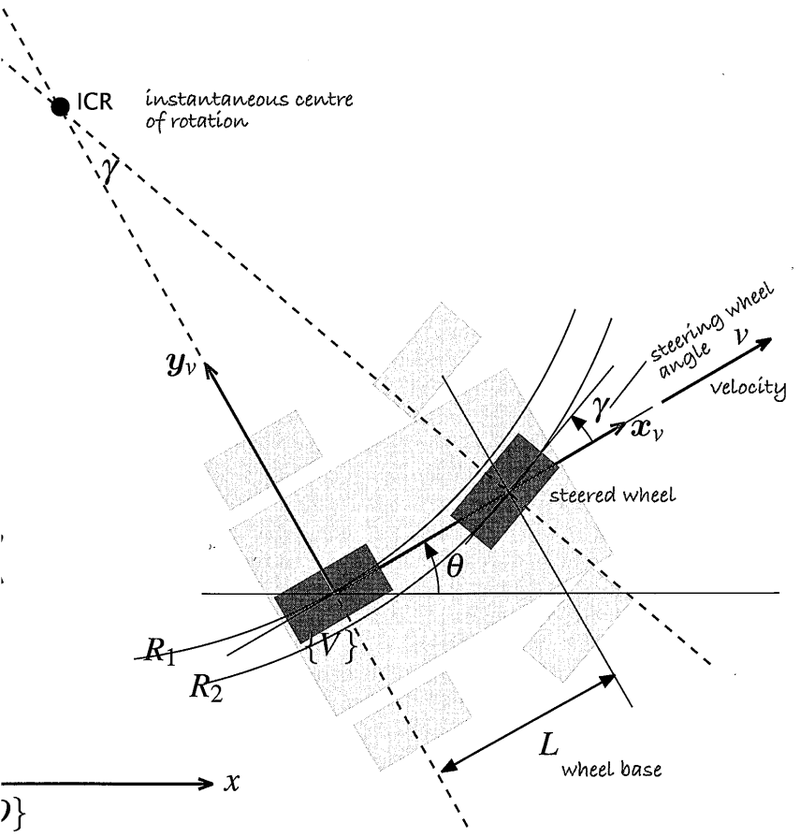


Fig. 4.2.
Bicycle model of a car. The car is shown in light grey, and the bicycle approximation is dark grey. The vehicle's coordinate frame is shown in red, and the world coordinate frame in blue. The steering wheel angle is γ and the velocity of the back wheel, in the x-direction, is v. The two wheel axes are extended as dashed lines and intersect at the Instantaneous Centre of Rotation (ICR) and the distance from the ICR to the back and front wheels is R_1 and R_2 respectively

A commonly used model for a four-wheeled car-like vehicle is the bicycle model▶ wn in Fig. 15.1. The bicycle has a rear wheel fixed to the body and the plane of the nt wheel rotates about the vertical axis to steer the vehicle.

Often incorrectly called the Ackerman model.

The pose of the vehicle is represented by the coordinate frame $\{V\}$ shown in Fig. 4.2, h its x-axis in the vehicle's forward direction and its origin at the centre of the rear e. The configuration of the vehicle is represented by the generalized coordinates $(x, y, \theta) \in \mathcal{C}$ where $\mathcal{C} \subset SE(2)$. The vehicle's velocity▶ is by definition v in the vehicle's irection, and zero in the y-direction since the wheels cannot slip sideways. In the icle frame $\{V\}$ this is

Other well known models include the Reeds-Shepp model which has only three speeds: forward, backward and stopped, and the Dubbins car which has only two speeds: forward and stopped.

$$^V\dot{x} = v, \quad ^V\dot{y} = 0$$

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

$$\dot{\theta} = \frac{v}{R_1} \tag{4.1}$$

l by simple geometry the turning radius is $R_1 = L / \tan\gamma$ where L is the length of vehicle or *wheel base*. As we would expect the turning circle increases with vehicle gth. The steering angle γ is limited mechanically and its maximum value dictates minimum value of R_1.

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.

Paths that arcs with smoothly varying radius.

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 $R_2 > R_1$ 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 radius 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 Ackerman 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

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

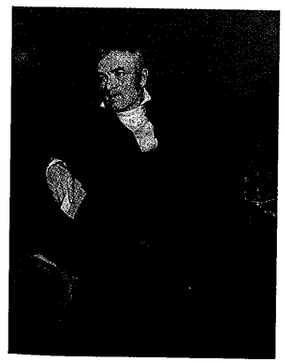
This model is referred to as a kinematic model since it describes the velocities of the vehicle but not the forces or torques that cause the velocity. The rate of change of heading $\dot{\theta}$ is referred to as turn rate, heading rate or yaw rate and can be measured by a gyroscope. It can also be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

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

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

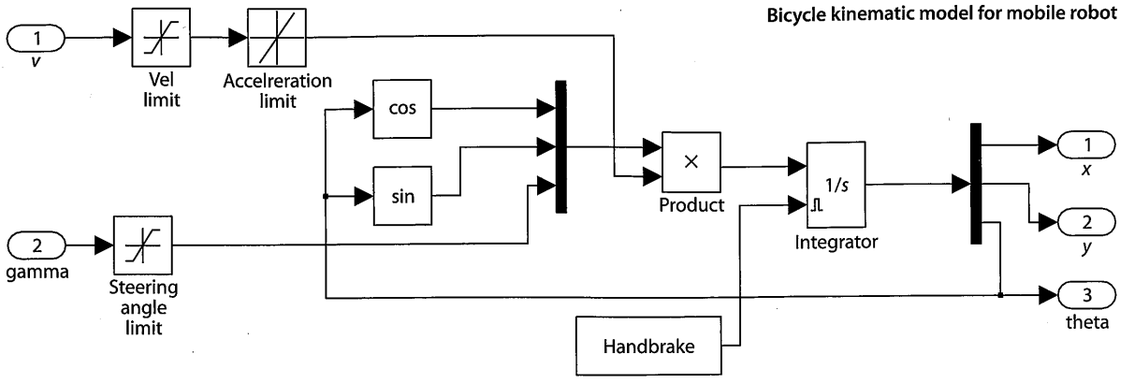
which is the non-holonomic constraint. This equation cannot be integrated to form a relationship between x, y and θ.

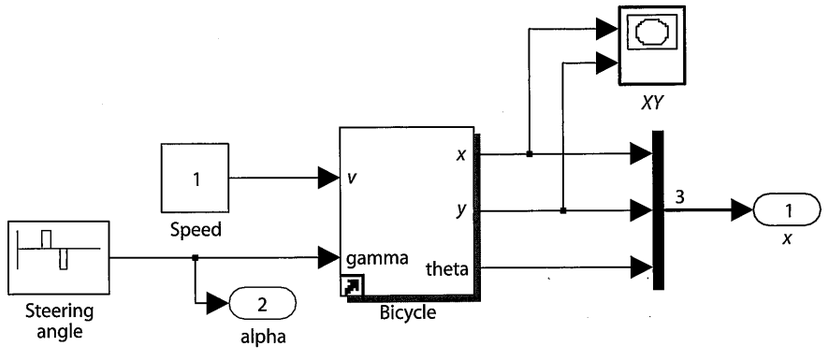
Equation 4.2 captures some other important characteristics of a wheeled vehicle. When $v = 0$ then $\dot{\theta} = 0$, that is, it is not possible to change the vehicle's orientation when it is not moving. As we know from driving we must be moving in order to turn. If the steering angle is $\frac{\pi}{2}$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



Rudolph Ackerman (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.

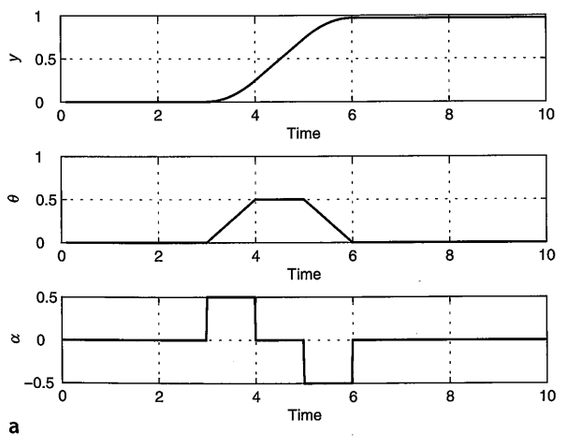
In 1818 Ackermann took out British patent 4212 on behalf of the German inventor George Lankensperger for a steering mechanism which ensures that the steered wheels move on circles with a common centre. The same scheme was proposed and tested by Erasmus Darwin (grandfather of Charles) in the 1760s. Subsequent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





▲ **Fig. 4.3.** Toolbox `Bicycle` block implements the bicycle kinematic model. The velocity input has a rate limiter to model finite acceleration, and limiters on velocity and steering wheel angle. The block labelled handbrake is a constant, from the block's parameter dialog, that resets the integrator and enforces the initial condition

◀ **Fig. 4.4.** Simulink® model `sl_lanechange` that results in a lane changing manoeuvre. The pulse generator drives the steering angle left then right



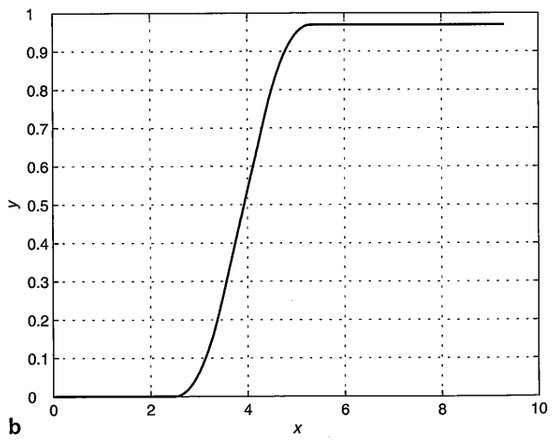


Fig. 4.5. Simple lane changing maneuver. **a** Vehicle response as a function of time, **b** motion in the *xy*-plane, the vehicle moves in the positive *x*-direction

Figure 4.3 shows a Simulink® implementation of the bicycle model. It implements Eq. 4.2 and 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. The Simulink® system

```
>> sl_lanechange
```

shown in Fig. 4.4 uses the `Bicycle` block in a system with a constant velocity demand. The steering input is a positive then negative pulse on the steering angle and the configuration is plotted against time in Fig. 4.5a. The result, in the *xy*-plane, is shown in Fig. 4.5b and shows a simple *lane-changing* trajectory.

4.2.1 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

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

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 $\ominus$ operator since $\theta^*, \theta \in \mathbb{S}$ we require the angular difference◀ to also lie within $\mathbb{S}$. A Simulink® model

The Toolbox function `angdiff` computes the difference between two angles and returns a difference in the interval $[-\pi, \pi)$. Also available in the Toolbox Simulink® blockset `roblocks`.

```
>> sl_drivepoint
```

is shown in Fig. 4.6. We specify a goal coordinate

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

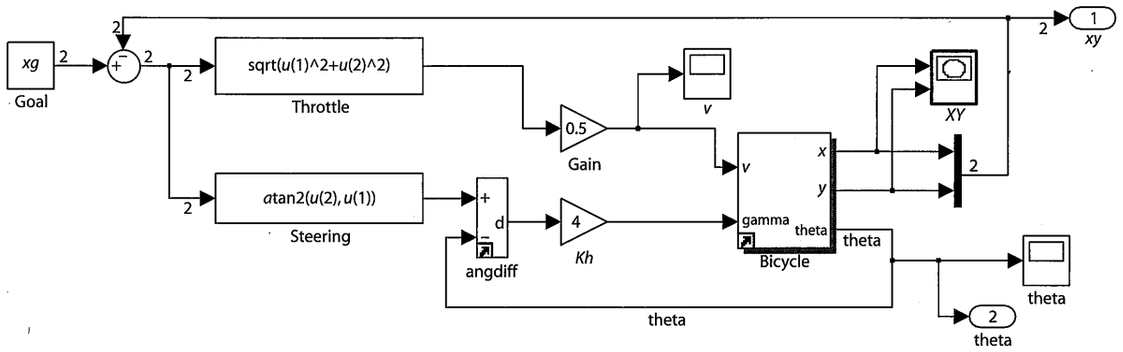


Fig. 4.6. `sl_drivepoint`, the Simulink® model that drives the vehicle to a point

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 typing control-T or by using `Simulation+Start` option from the toolbar on 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.

Some models write data to the MATLAB® workspace for subsequent analysis. Some models simply have unconnected output ports. All models in this chapter have the simulation data export option set to `Format=Array`, so the signals are concatenated, in port number order, to form a row vector and these are stacked to form a matrix `yout` with one row per timestep. The corresponding time values form a vector `tout`. These variables can be returned from the simulation

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

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

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

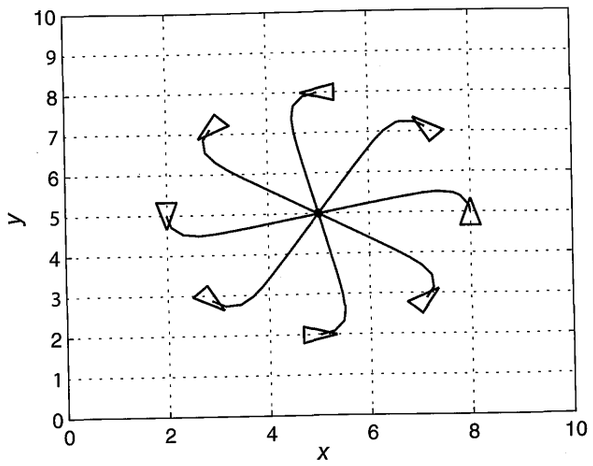


Fig. 4.7.
Simulation results for `sl_drivepoint` for different initial poses. The goal is (5, 5)

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 the configuration as a function of time

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

The vehicle's path in the plane is

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

which is shown in Fig. 4.7 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.

4.2.2 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 steers the robot to minimize the robot's normal distance from the line which according to Eq. I.1 is

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

The proportional controller

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

turns the robot toward the line. 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. 4.8. We specify the target line as a 3-vector (a, b, c)

```
>> L = [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. 4.9.

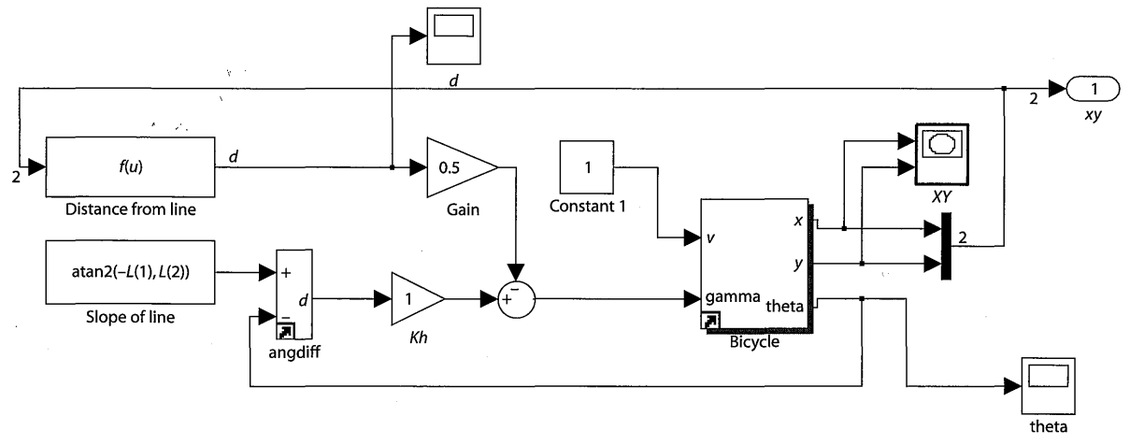


Fig. 4.8. The Simulink® model `sl_driveline` drives the vehicle along a line. The line parameters (a, b, c) are set in the workspace variable `L`

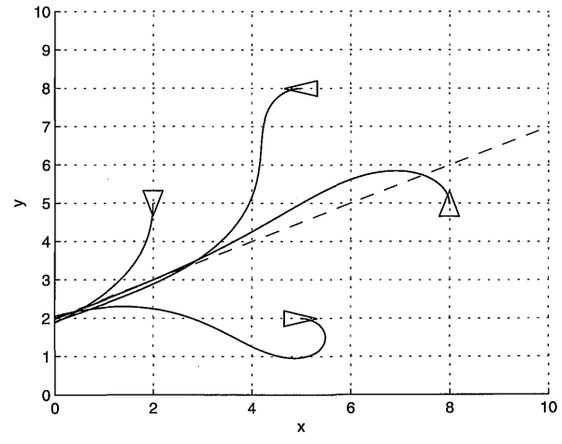


Fig. 4.9.
Simulation results from different initial poses for the line (1, −2, 4)

4.2.3 Following a Path

Instead of a straight line we might wish to follow a path that is defined more generally as some locus on the xy-plane. The path might come from a sequence of coordinates generated by a motion planner, such as discussed in Sect. 5.2, or in real-time based on the robot's sensors.

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

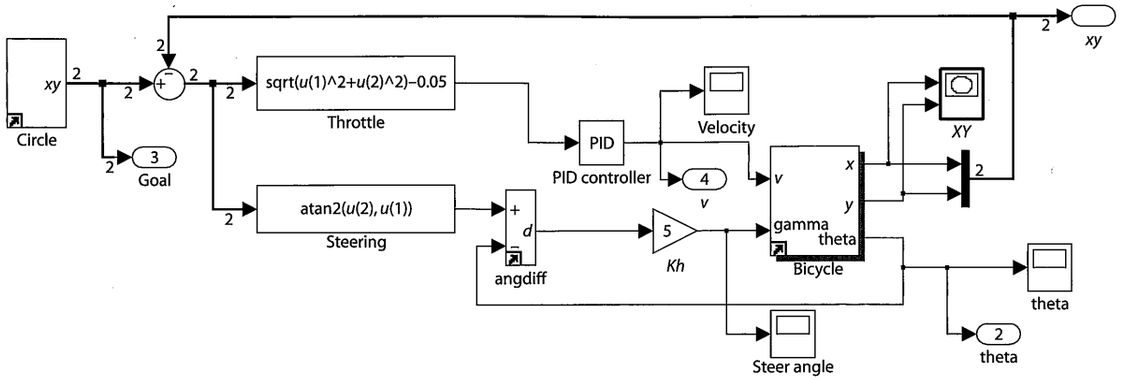


Fig. 4.10. The Simulink® model `sl_pursuit` drives the vehicle to follow an arbitrary moving target using pure pursuit. In this example the vehicle follows a point moving around a unit circle with a frequency of 0.1 Hz

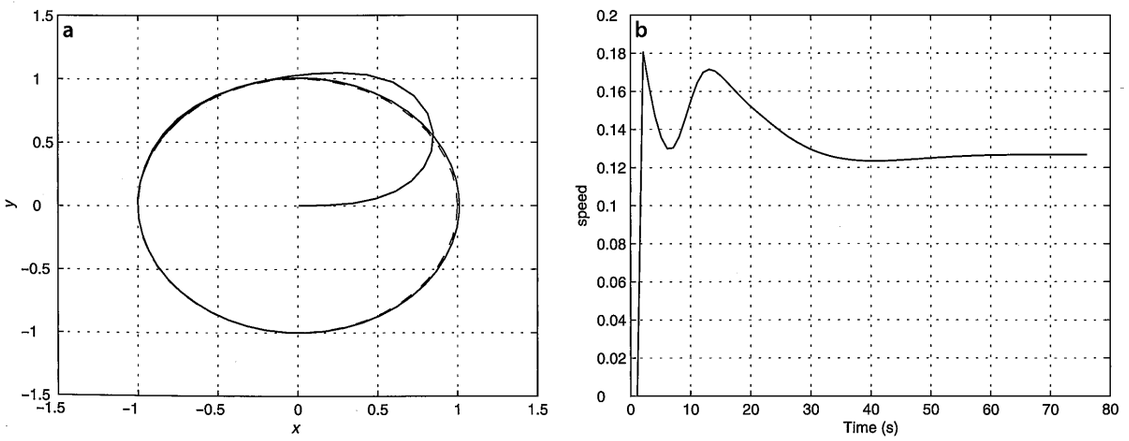


Fig. 4.11. Simulation results from pure pursuit. **a** Path of the robot in the xy-plane. The black dashed line is the circle to be tracked and the blue line in the path followed by the robot. **b** The speed of the vehicle versus time

This problem is very similar to the control problem we tackled in Sect. 4.2.1, 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

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

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 finite 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

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

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

The Simulink® model

```
>> sl_pursuit
```

shown in Fig. 4.10 includes a target that moves around a unit circle. It can be simulated

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

and the results are shown in Fig. 4.11a. The robot starts at the origin but catches up to, and follows, the moving goal. Figure 4.11b shows how the speed demand picks up smoothly and converges to a steady state value at the desired following distance.

4.2.4 Moving to a Pose

The final control problem we discuss is driving to a specific pose (x^*, y^*, θ^*). The controller of Fig. 4.6 could drive the robot to a goal position but the final orientation depended on the starting position.

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

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

and then transform the equations into polar coordinate form using the notation shown in Fig. 4.12. We apply a change of variables

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

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

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

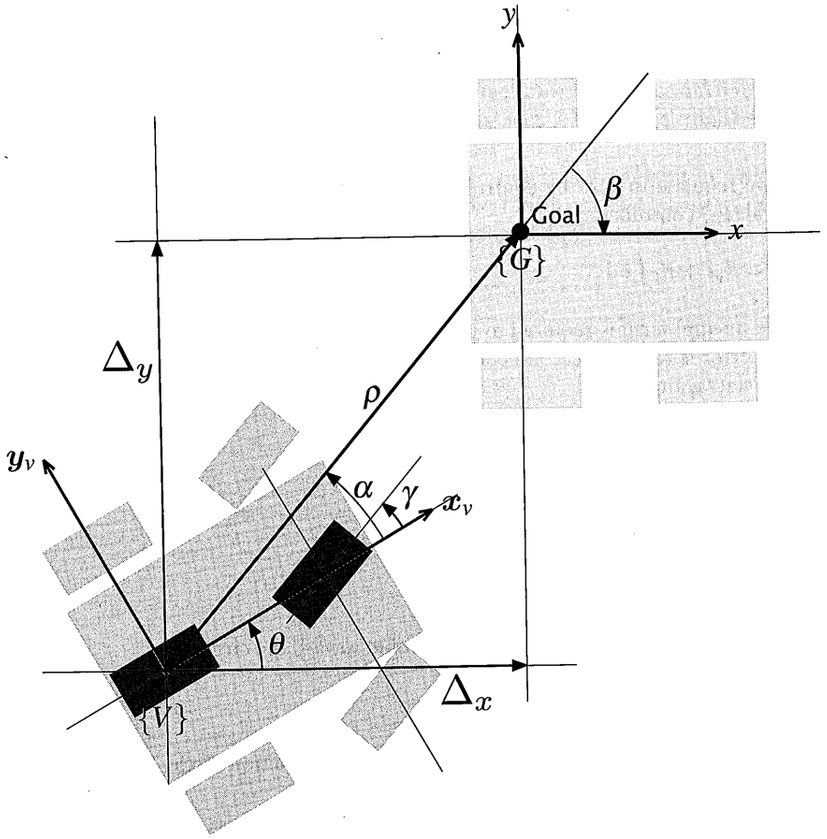


Fig. 4.12.
Polar coordinate notation for the bicycle model vehicle moving toward a goal pose: ρ is the distance to the goal, β is the angle of the goal vector with respect to the world frame, and α is the angle of the goal vector with respect to the vehicle frame

which results in

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

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

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

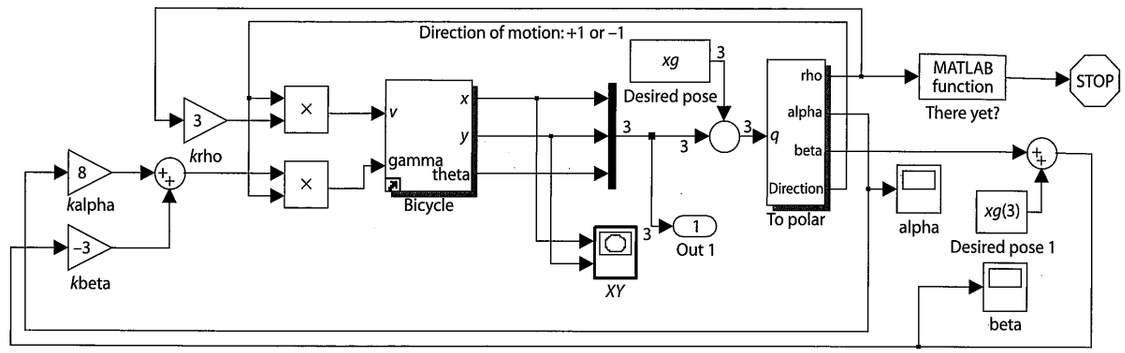
drives the robot to a unique equilibrium▶ at $(\rho, \alpha, \beta) = (0, 0, 0)$. The intuition behind this controller is that the terms $k_\rho\rho$ and $k_\alpha\alpha$ drive the robot along a line toward $\{G\}$ while the term $k_\beta\beta$ rotates the line so that $\beta\to 0$. The closed-loop system

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

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



▲ Fig. 4.13. 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

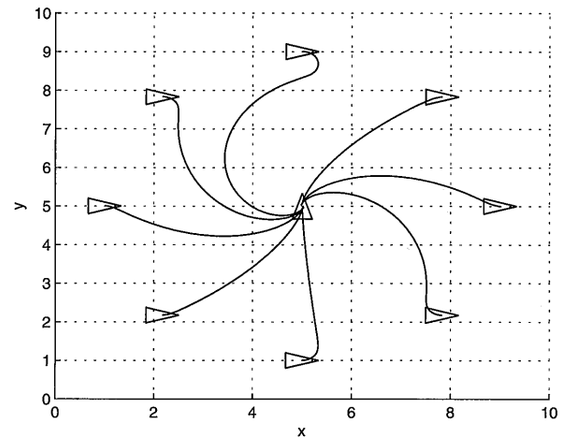


Fig. 4.14.
Simulation results from different initial poses to the final pose $(5, 5, \frac{\pi}{2})$. Note that in some cases the robot has *backed into* the final pose

The distance and bearing to the goal (ρ, α) could be measured by a camera or laser range finder, and the angle β 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

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

The pose controller is implemented by the Simulink® model

```
>> sl_drivepose
```

shown in Fig. 4.13. We specify a goal pose

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

and an initial pose

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

and then simulate the motion

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

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

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

to show the vehicle's path in the plane. The vehicle's path for a number of starting poses is shown in Fig. 4.14. The vehicle moves forwards or backward and takes a smooth path to the goal pose.▶

The controller is based on the linear bicycle model but the Simulink® model `Bicycle` has hard non-linearities including steering angle limits and velocity rate limiting.

4.3 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. 4.15. Applications include military operations, surveillance, meteorological investigations and robotics research. Fixed wing UAVs are similar in principle to passenger aircraft with wings to provide lift, a propellor or jet to provide forward thrust and control surface for manoeuvring. 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 quadcopters. Rotorcraft UAVs are used for inspection and research and have the advantage of being able to take off vertically.

Flying robots differ from ground robots in some important ways. Firstly they have 6 degrees of freedom and their configuration $\boldsymbol{q} \in SE(3)$. Secondly they are actuated by forces, that is their motion model is expressed in terms of forces and torques rather than velocities as was the case for the bicycle model – 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 principle 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 quadcopter flying vehicle such as shown in Fig. 4.15d. Quadcopters are now widely available, both as commercial products and

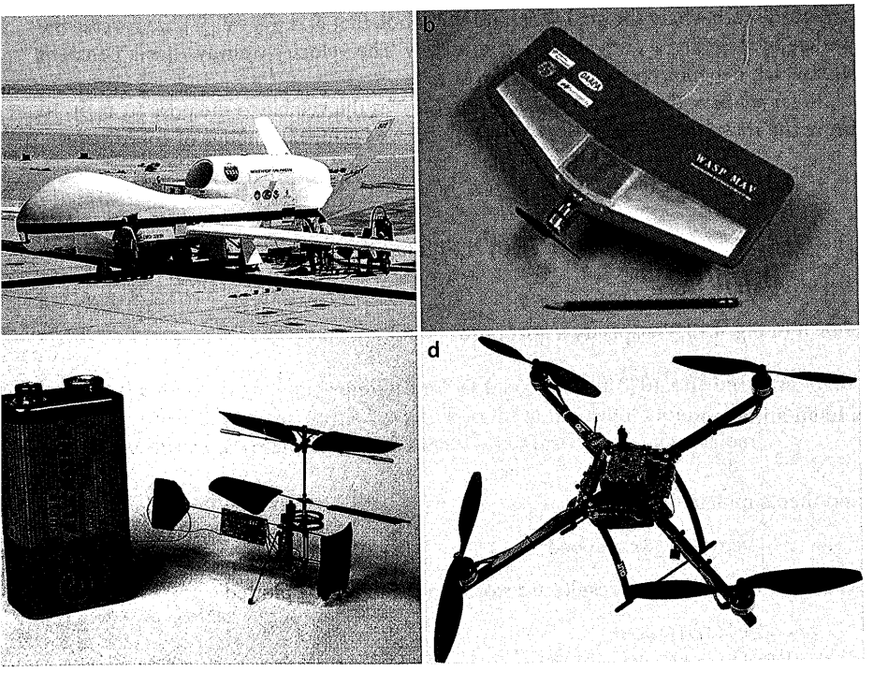


Fig. 4.15.
Flying robots. **a** Global Hawk unmanned aerial vehicle (UAV) (photo: courtesy of NASA), **b** a micro air vehicle (MAV) (photo: courtesy of AeroVironment, Inc.), **c** a 1 gram co-axial helicopter with 70 mm rotor diameter (photo courtesy of Petter Muren and Proxflyer AS), **d** a quad-copter, also known as an X4, which has four rotors and a block of sensing and control electronics are in the middle (photo: courtesy of Inkyu Sa)

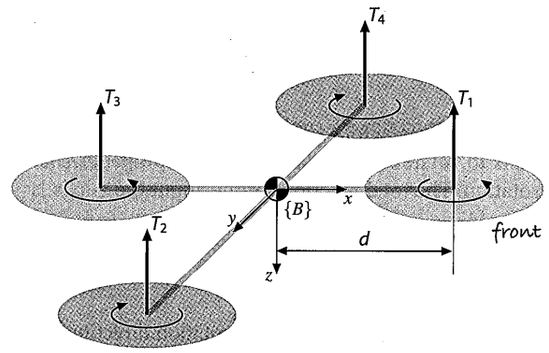


Fig. 4.16.
Quad-rotor notation showing the four rotors, their thrust vectors and directions of rotation. The body-fixed frame $\{B\}$ is attached to the vehicle and has its origin at the vehicle's centre of mass. Rotors 1 and 3 rotate counter-clockwise (viewed from above) while rotors 2 and 4 rotate clockwise

as open-source projects. Compared to fixed wing aircraft they are highly manoeuvrable and can be flown safely indoors which makes them well suited for laboratory or hobbyist use. Compared to conventional helicopters, with the large main rotor and tail rotor, the quadcopter is easier to fly, does not have the complex swash plate mechanism and is easier to model and control.

The notation for the quadcopter model is shown in Fig. 4.16. The body-fixed coordinate frame $\{B\}$ has its z-axis downward following the aerospace convention. The quadcopter has four rotors, labelled 1 to 4, mounted at the end of each cross arm. The rotors are driven by electric motors powered by electronic speed controllers. Some low-cost quadcopters use small motors and reduction gearing to achieve sufficient torque. The rotor speed is ω_i and the thrust is an upward vector

$$T_i = b\omega_i^2, \quad i = 1, 2, 3, 4 \tag{4.3}$$

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

$$m\dot{\boldsymbol{v}} = \begin{pmatrix} 0 \\ 0 \\ mg \end{pmatrix} - {}^0\boldsymbol{R}_B \begin{pmatrix} 0 \\ 0 \\ T \end{pmatrix} \tag{4.4}$$

where $\boldsymbol{v}$ is the velocity of the vehicle in the world frame, g is gravitational acceleration, m is the total mass of the vehicle and $T = \Sigma T_i$ is the total upward thrust. The first term is the force of gravity which acts downward in the world frame and the second term is the total thrust in the vehicle frame rotated into the world coordinate frame.

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

$$\tau_x = dT_4 - dT_2$$

where d is the distance from the motor to the centre of mass. We can write this in terms of rotor speeds by substituting Eq. 4.3

$$\tau_x = db\left(\omega_4^2 - \omega_2^2\right) \tag{4.5}$$

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

$$\tau_y = db\left(\omega_1^2 - \omega_3^2\right) \tag{4.6}$$

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

$$Q_i = k\omega_i^2$$

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

$$\begin{aligned}\tau_z &= Q_1 - Q_2 + Q_3 - Q_4 \\ &= k\left(\omega_1^2 + \omega_3^2 - \omega_2^2 - \omega_4^2\right)\end{aligned} \tag{4.7}$$

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 rotational acceleration of the airframe is given by Euler's equation of motion

$$J\dot{\omega} = -\omega \times J\omega + \boldsymbol{\Gamma} \tag{4.8}$$

where J is the 3×3 inertia matrix of the vehicle, ω is the angular velocity vector and $\boldsymbol{\Gamma} = (\tau_x, \tau_y, \tau_z)^T$ is the torque applied to the airframe according to Eq. 4.5 to 4.7.

The motion of the quadcopter is obtained by integrating the forward dynamics equations Eq. 4.4 and Eq. 4.8 where the forces and moments on the airframe

$$\begin{pmatrix} T \\ \boldsymbol{\Gamma} \end{pmatrix} = \begin{pmatrix} -b & -b & -b & -b \\ 0 & -db & 0 & db \\ db & 0 & -db & 0 \\ k & -k & k & -k \end{pmatrix} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \boldsymbol{A} \begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} \tag{4.9}$$

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

$$\begin{pmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \end{pmatrix} = \boldsymbol{A}^{-1} \begin{pmatrix} T \\ \tau_x \\ \tau_y \\ \tau_z \end{pmatrix} \tag{4.10}$$

to give the rotor speeds required to apply a specified thrust T and moment $\boldsymbol{\Gamma}$ to the airframe.

To control the vehicle we will employ a nested control structure which we illustrate for pitch and x-translational motion. The innermost loop uses a proportional and derivative controller▶ to compute the required pitching torque on the airframe

$$\tau_y = K_p(\theta_p^* - \theta_p) + K_d(\dot{\theta}_p^* - \dot{\theta}_p)$$

based on the error between desired and actual pitch angle.▶ The gains $K_{\tau,p}$ and $K_{\tau,d}$ are determined by classical control design approaches based on an approximate dynamic model and then tuned to achieve good performance. The actual vehicle pitch angle θ_p would be estimated by an inertial navigation system. The required rotor speeds are then determined using Eq. 4.10.

Consider a coordinate frame $\{V\}$ attached to the vehicle and with the same origin as $\{B\}$ but with its x- and y-axes parallel to the ground. To move the vehicle in the ^{V}x-direction we pitch the nose down which generates a force

$$\boldsymbol{f} = \boldsymbol{R}_y(\theta_p) \begin{pmatrix} 0 \\ 0 \\ T \end{pmatrix} = \begin{pmatrix} T\sin\theta_p \\ 0 \\ T\cos\theta_p \end{pmatrix}$$

The rotational dynamics has a second-order transfer function of $\Theta_y(s)/\tau_y(s) = 1/(Js^2 + Bs)$ where 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.

which has a component

$$f_x = T\sin\theta_p \approx T\theta_p$$

that accelerates the vehicle in the ${}^V x$-direction. We can control the velocity in this direction with a proportional control law

$$f_x^* = mK_f\left({}^V v_x^* - {}^V v_x\right)$$

Combine these two equations we obtain the pitch angle

$$\theta_p^* = \frac{m}{T} K_f\left({}^V v_x^* - {}^V v_x\right) \tag{4.11}$$

required to achieve the desired forward velocity. The actual vehicle velocity ${}^V v_x$ would be estimated by an inertial navigation system or GPS receiver. For a vehicle in vertical equilibrium the total thrust equals the weight force so $m/T \approx 1/g$.

If the position of the vehicle in the xy-plane of the world frame is $\boldsymbol{p} \in \mathbb{R}^2$ then the desired velocity is given by the proportional control law

$$\boldsymbol{v}^* = K_p(\boldsymbol{p}^* - \boldsymbol{p}) \tag{4.12}$$

based on the error between the desired and actual position. The desired velocity in frame $\{V\}$ is

$${}^V\boldsymbol{v} = {}^V\boldsymbol{R}_0(\theta_y)\boldsymbol{v} = {}^0\boldsymbol{R}_V^T(\theta_y)\boldsymbol{v}$$

which is a function of the yaw angle θ_y

$$\begin{pmatrix} {}^V v_x \\ {}^V v_y \end{pmatrix} = {}^0R_V^T(\theta_y) \begin{pmatrix} v_x \\ v_y \end{pmatrix}$$

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

To reach a desired position we can compute the appropriate velocity and from that the appropriate pitch angle which generates a force to move the vehicle. This indirection is a consequence of the vehicle being underactuated – we have just four rotor speeds to adjust but the vehicle's configuration space is 6-dimensional. To move forward the quadcopter airframe must first pitch down so that the thrust vector has a horizontal component to accelerate it.◀ As it approaches its goal the airframe must be rotated in the opposite direction, pitching up, so that the backward component of thrust decelerates the forward motion. Finally the airframe rotates to the horizontal with the thrust vector vertical. The cost of under-actuation is once again a manoeuvre. The pitch angle cannot be arbitrarily set, it is a means to achieve translation control.

The rotational inertia of a body that moves in $SE(3)$ is represented by the 3×3 symmetric matrix

$$J = \begin{pmatrix} J_{xx} & J_{xy} & J_{xz} \\ J_{xy} & J_{yy} & J_{yz} \\ J_{xz} & J_{yz} & J_{zz} \end{pmatrix}$$

The diagonal elements are the moments of inertia, and the off-diagonal elements are products of inertia. Only six of these nine elements are unique: three moments and three products of inertia. The products of inertia are zero if the object's mass distribution is symmetrical with respect to the coordinate frame.

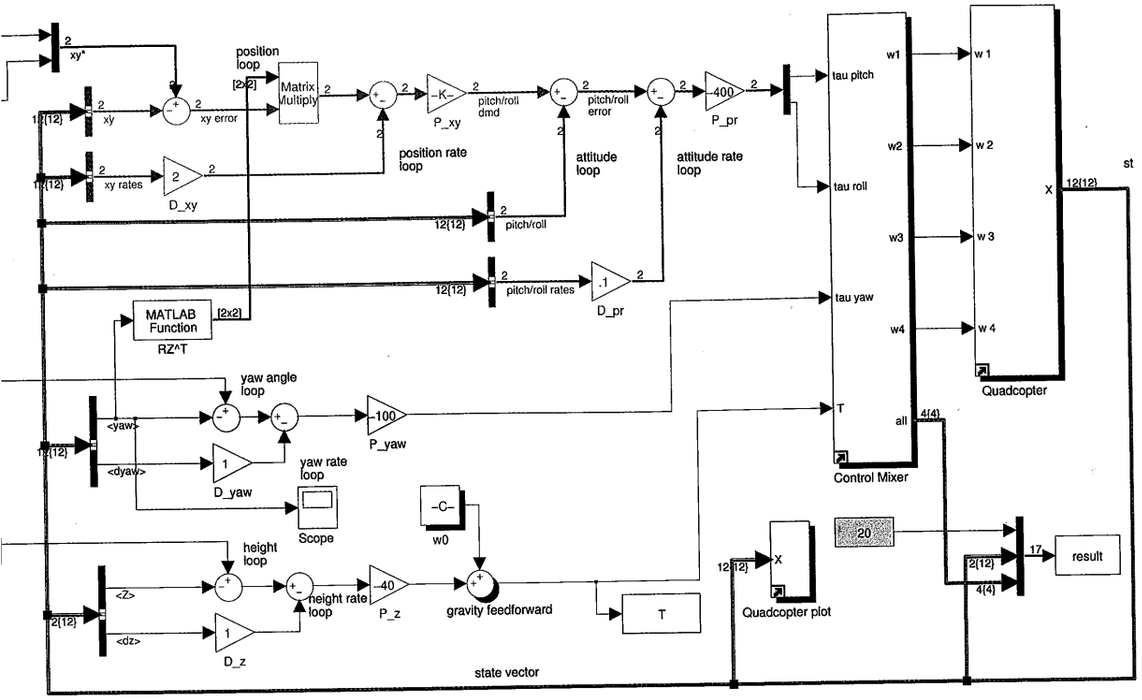


Figure 4.17 shows a Simulink® model of the quadcopter in a closed-loop control structure. The inputs to the quadcopter block are the speeds of the four rotors and from Eq. 4.9 the torques and forces on the quadcopter are computed and it integrates Eq. 4.4, Eq. 4.8 and Eq. 4.9 to give the position, velocity, orientation and orientation rate. The output of this block is the state vector $\boldsymbol{x} = (x, y, z, \theta_r, \theta_p, \theta_y, \dot{x}, \dot{y}, \dot{z}, \dot{\theta}_r, \dot{\theta}_p, \dot{\theta}_y)$. As is common in aerospace applications we represent orientation and orientation rate in terms of roll-pitch-yaw angles. The control part of the block diagram involves multiple nested control loops that compute the required thrust and torques so that the vehicle moves to the setpoint.

The vehicle's position control loops, as just discussed, are shown in the top left of the diagram. The innermost loop, shown in blue, controls the attitude of the vehicle and its inputs are the actual and desired roll and pitch angles, as well as the roll and pitch angular rates to provide damping. The outer loop, shown in orange, controls the xy-position of the flyer by requesting changes in roll and pitch angle so as to provide a component of thrust in the direction of desired xy-plane motion. In the diagram Eq. 4.11 and Eq. 4.12 have been combined into the form

$$\theta_p^* = K_1\left({}^V p_x^* - {}^V p_x - K_2\, {}^V v_x\right)$$

The xy-position error is computed in the world frame and rotated by ${}^0\boldsymbol{R}_V^T(\theta_y)$ into frame $\{V\}$. Note that according to the coordinate conventions shown in Fig. 4.16 ${}^V x$-direction motion requires a negative rotation about the y-axis (pitch angle) and ${}^V y$-direction motion requires a positive rotation about the x-axis (roll angle) so the gains have different signs for the roll and pitch loops.

Yaw is controlled by a proportional-derivative controller

$$\tau_z = K_p\left(\theta_y^* - \theta_y\right) + K_d\left(\dot{\theta}_y^* - \dot{\theta}_y\right)$$

shown in black and $\dot{\theta}_y^*$ is ignored since it is typically small.

Fig. 4.17. The Simulink® model `sl_quadcopter` which is a closed-loop simulation of the quadcopter. The flyer takes off and flies in a circle at constant altitude. The dynamics block implements Eq. 4.9, and the mixer block implements its inverse while also enforcing limits on rotor speed. A Simulink® bus is used for the 12-element state vector `X` output by the `Quadcopter` block

Altitude is controlled by a proportional-derivative controller

$$T = K_p(z^* - z) + K_d(\dot{z}^* - \dot{z}) + \omega_0$$

shown in red which which determines the average rotor speed. The additive term

$$\omega_0 = \sqrt{\frac{mg}{4b}} \tag{4.13}$$

is the rotor speed necessary to generate a thrust equal to the weight of the vehicle. 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 parameters of a specific quadcopter can be loaded

```
>> mdl_quadcopter
```

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

```
>> sim('sl_quadcopter');
```

The simulation loads the default quadcopter model before it starts, through the `PreLoadFcn` callback set from model's properties File+Model Properties+Callbacks+PreLoadFcn.

and it displays an animation in a separate window.◀ The vehicles lifts off and flies in a circle while spinning slowly about its own z-axis. A snapshot is shown in Fig. 4.18. The simulation writes the results from each timestep into a matrix in the workspace

```
>> about(result)
result [double] : 419x17 (56984 bytes)
```

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 ω_i (elements 14–17). To plot x and y versus time is

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

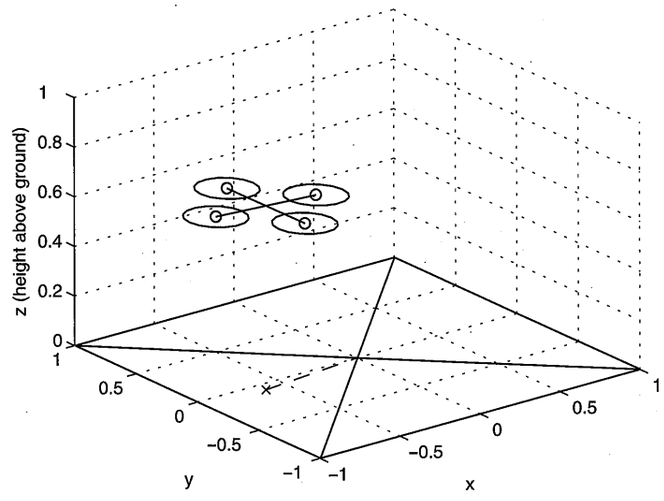


Fig. 4.18. One frame from the quadcopter simulation. The marker on the ground plane is a projection of the vehicle's centroid