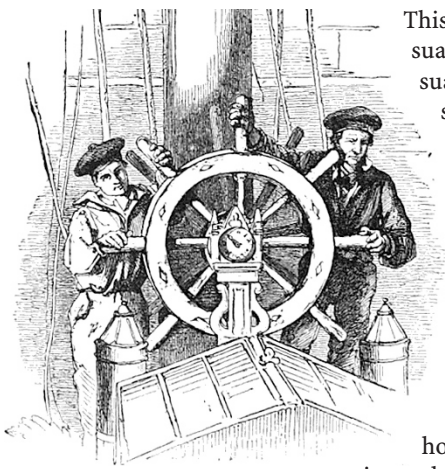
Chapter

16 Advanced Visual Servoing



This chapter builds on the previous one and introduces some advanced visual servo techniques and applications. Section 16.1 introduces a hybrid visual servo method that avoids some of the limitations of the IBVS and PBVS schemes described previously.

Wide-angle cameras such as fisheye lenses and catadioptric cameras have significant advantages for visual servoing. Section 16.2 shows how IBVS can be reformulated for polar rather than Cartesian image-plane coordinates. This is directly relevant to fisheye lenses but also gives improved rotational control when using a perspective camera. The unified imaging model from Sect. 11.4 allows most cameras (perspective, fisheye and panoramic) to be represented by a spherical projection model, and Sect. 16.3 shows how IBVS can reformulated for spherical cameras.

Section 16.4 presents a number of application examples. These illustrate how visual servoing can be used with different types of cameras (perspective and spherical) and different types of robots (arm-type robots, mobile ground robots and flying robots). Examples include a 6 degree of freedom robot arm manipulating a camera; a mobile robot moving to a specific pose which could be used for navigating through a doorway or docking; and a quadrotor moving to, and hovering at, a fixed pose with respect to a goal on the ground.

16.1 XY/Z-Partitioned IBVS

In the last chapter, in Sect. 15.2.4, we encountered the problem of camera retreat in an IBVS system. This phenomenon can be explained intuitively by the fact that our IBVS control law causes feature points to move in straight lines on the image plane, but for a rotating camera the points will naturally move along circular arcs. The linear IBVS controller dynamically changes the overall image scale so that motion along an arc appears as motion along a straight line. The scale change is achieved by z-axis translation.

Partitioned methods eliminate camera retreat by using IBVS to control some degrees of freedom while using a different controller for the remaining degrees of freedom. The XY/Z hybrid schemes consider the x- and y-axes as one group, and the z-axes as another group. The approach is based on a couple of insights. Firstly, and intuitively, the camera retreat problem is a z-axis phenomenon: z-axis rotation leads to unwanted z-axis translation. Secondly, from Fig. 15.7, the image-plane motion due to x- and y-axis translational and rotation motion are quite similar, whereas the optical flow due to z-axis rotation and translation are radically different.

We partition the point feature optical flow of Eq. 15.7 so that

$$\dot{\boldsymbol{p}} = \boldsymbol{J}_{xy}\boldsymbol{\nu}_{xy} + \boldsymbol{J}_z\boldsymbol{\nu}_z \tag{16.1}$$

where $\boldsymbol{\nu}_{xy} = (v_x, v_y, \omega_x, \omega_y)$, $\boldsymbol{\nu}_z = (v_z, \omega_z)$, and $\boldsymbol{J}_{xy}$ and $\boldsymbol{J}_z$ are respectively columns $\{1, 2, 4, 5\}$ and $\{3, 6\}$ of $\boldsymbol{J}_p$. Since $\boldsymbol{\nu}_z$ will be computed by a different controller we can write Eq. 16.1 as

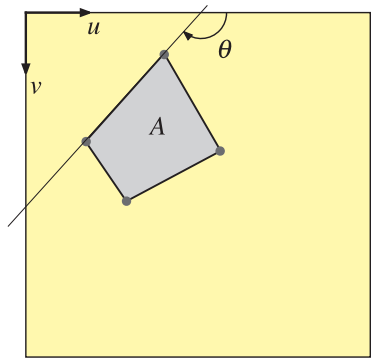


Fig. 16.1.
Image features for XY/Z partitioned IBVS control. As well as the coordinates of the four points (blue dots), we use the polygon area A and the angle of the longest line segment θ

$$\boldsymbol{\nu}_{xy} = \lambda \boldsymbol{J}_{xy}^{+} \left(\dot{\boldsymbol{p}}^{*} - \boldsymbol{J}_{z} \boldsymbol{\nu}_{z} \right) \tag{16.2}$$

where $\dot{\boldsymbol{p}}^{*}$ is the desired feature point velocity as in the traditional IBVS scheme Eq. 15.10.

The z-axis velocities v_z and ω_z are computed directly from two additional image features A and θ shown in Fig. 16.1. The first image feature $\theta \in [0, \pi)$, is the angle between the u-axis and the directed line segment joining feature points i and j. For numerical conditioning it is advantageous to select the longest line segment that can be constructed from the feature points, and allowing that this may change during the motion as the feature point configuration changes. The desired rotational rate is obtained using a simple proportional control law

$$\omega_{z}^{*} = \lambda_{\omega_z} \left(\theta^{*} \ominus \theta \right)$$

where the operator $\ominus$ indicates modulo-2π subtraction which is implemented by the Toolbox function `angdiff`. As always with motion on a circle there are two directions to move to achieve the goal. If the rotation is limited, for instance by a mechanical stop, then the sign of ω_z should be chosen so as to avoid motion through that stop.

The second image feature that we use is a function of the area $A \in \mathbb{R}$ of the regular polygon whose vertices are the image feature points. The advantages of this measure are: it is a scalar; it is rotation invariant▸ thus decoupling camera rotation from z-axis translation; and it can be cheaply computed. The area of the polygon is just the zeroth-order moment, m_{00} which can be computed from the vertices using the Toolbox function `mpq_poly(p, 0, 0)`. The feature for control is the square root of area

Rotationally invariant to rotation about the z-axis, not the x- and y-axes.

$$\sigma = \sqrt{m_{00}}$$

which has units of length, in pixels. The desired camera z-axis translation rate is obtained using a simple proportional control law

$$v_{z}^{*} = \lambda_{v_z} \left(\sigma^{*} - \sigma \right) \tag{16.3}$$

The features discussed above for z-axis translation and rotation control are simple and inexpensive to compute, but work best when the goal's normal is within $\pm 40°$ of the camera's optical axis. When the goal plane is not orthogonal to the optical axis its area will appear diminished, due to perspective, which causes the camera to initially approach the goal. Perspective will also change the perceived angle of the line segment which can cause small, but unnecessary, z-axis rotational motion.

The Simulink® model

```
>> sl_partitioned
```

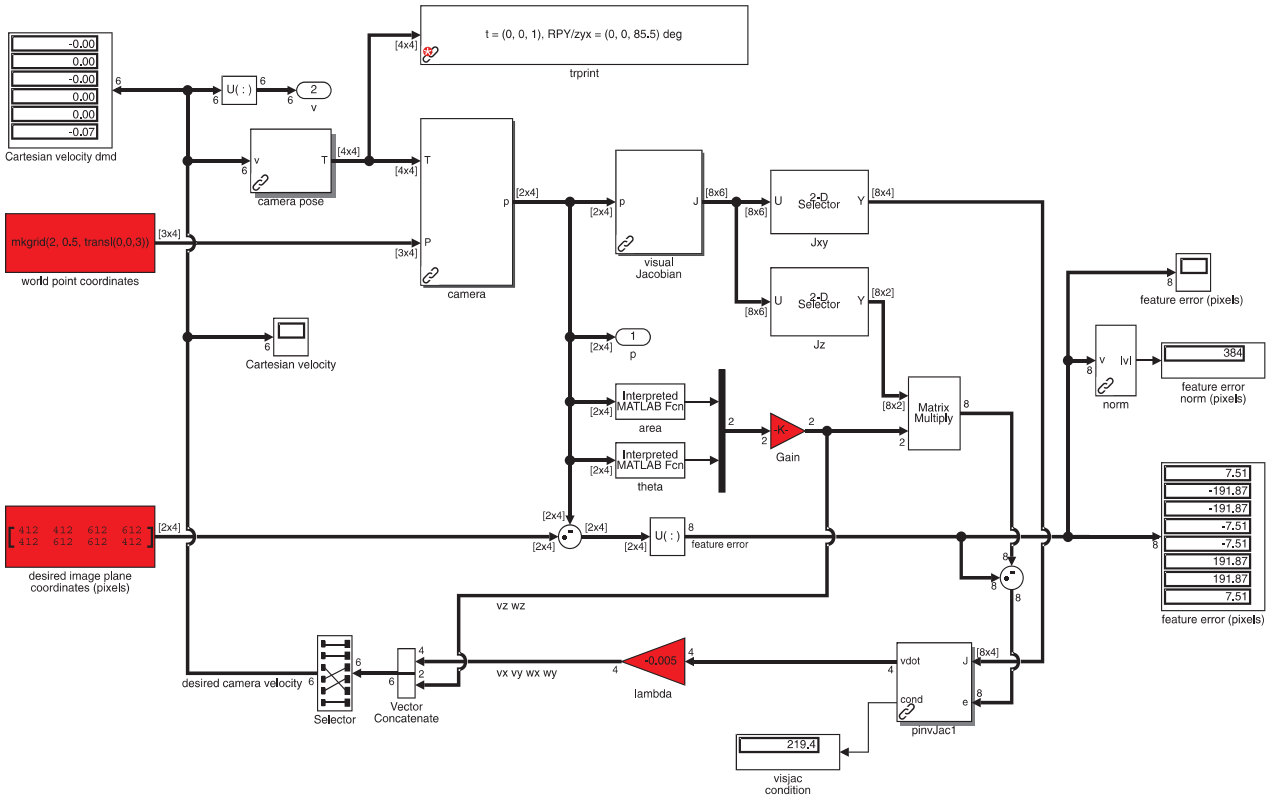


Fig. 16.2. The Simulink model `sl_partitioned` is an XY/Z-partitioned visual servo scheme, an extension of the IBVS system shown in Fig. 15.9. The initial camera pose is set in the `camera pose` block and the desired image-plane points $\boldsymbol{p}^*$ are set in the lower left red block

is shown in Fig. 16.2. The initial pose of the camera is set by a parameter of the `pose` block. The simulation is run by

```
>> sim('sl_partitioned')
```

and the camera pose, image-plane feature error and camera velocity are animated. Scope blocks also plot the camera velocity and feature error against time.

If points are moving toward the edge of the field of view the simplest way to keep them in view is to move the camera away from the scene. We define a repulsive force that acts on the camera, pushing it away as a point approaches the boundary of the image plane

$$F_z(\boldsymbol{p}) = \begin{cases} \left(\frac{1}{d(\boldsymbol{p})} - \frac{1}{d_0}\right)\frac{1}{d^2(\boldsymbol{p})} & \text{if } d(\boldsymbol{p}) \leq d_0 \\ 0 & \text{if } d(\boldsymbol{p}) > d_0 \end{cases}$$

where $d(\boldsymbol{p})$ is the shortest distance to the edge of the image plane from the image point coordinate $\boldsymbol{p}$, and d_0 is the width of the image zone in which the repulsive force acts. For a $W \times H$ image

$$d(\boldsymbol{p}) = \min\{u, v, W - u, H - v\} \tag{16.4}$$

Such a repulsion force could be incorporated into the z-axis translation controller

$$v_z^* = \lambda_{v_z}\left(\sigma^* - \sigma\right) - \eta\sum_{i=1}^{N} F_z\left(\boldsymbol{p}_i\right)$$

where η is a gain constant with units of damping. The repulsion force is discontinuous and may lead to chattering where the feature points oscillate in and out of the repulsive force – this can be remedied by introducing smoothing filters and velocity limiters.

16.2 IBVS Using Polar Coordinates

In Sect. 15.3 we showed image feature Jacobians for nonpoint features, but here we will show the point feature Jacobian expressed in terms of a different coordinate system. In polar coordinates the image point is written $\mathbf{p} = (r, \phi)$ where r is the distance of the point from the principal point

$$r = \sqrt{\overline{u}^2 + \overline{v}^2} \tag{16.5}$$

where we recall that $\overline{u}$ and $\overline{v}$ are the image coordinates with respect to the principal point rather than the image origin. The angle from the u-axis to a line joining the principal point to the image point is

$$\phi = \tan^{-1} \frac{\overline{v}}{\overline{u}} \tag{16.6}$$

The two coordinate representations are related by

$$\overline{u} = r\cos\phi, \; \overline{v} = r\sin\phi \tag{16.7}$$

and taking the derivatives with respect to time

$$\begin{pmatrix} \dot{\overline{u}} \\ \dot{\overline{v}} \end{pmatrix} = \begin{pmatrix} \cos\phi & -r\sin\phi \\ \sin\phi & r\cos\phi \end{pmatrix} \begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix}$$

and inverting

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \begin{pmatrix} \cos\phi & \sin\phi \\ -\frac{1}{r}\sin\phi & \frac{1}{r}\cos\phi \end{pmatrix} \begin{pmatrix} \dot{\overline{u}} \\ \dot{\overline{v}} \end{pmatrix}$$

which we substitute into Eq. 15.6 along with Eq. 16.7 to write

$$\begin{pmatrix} \dot{r} \\ \dot{\phi} \end{pmatrix} = \mathbf{J}_{p,p} \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{16.8}$$

where the feature Jacobian is

$$\mathbf{J}_{p,p} = \begin{pmatrix} -\frac{f}{Z}\cos\phi & -\frac{f}{Z}\sin\phi & \frac{r}{Z} & \frac{f^2+r^2}{f}\sin\phi & -\frac{f^2+r^2}{f}\cos\phi & 0 \\ \frac{f}{rZ}\sin\phi & -\frac{f}{rZ}\cos\phi & 0 & \frac{f}{r}\cos\phi & \frac{f}{r}\sin\phi & -1 \end{pmatrix} \tag{16.9}$$

This Jacobian is unusual in that it has three constant elements. In the first row the zero indicates that radius r is invariant to rotation about the z-axis. In the second row the zero indicates that polar angle is invariant to translation along the optical axis (points move along radial lines), and the negative one indicates that the angle of a feature (with respect to the u-axis) decreases with positive camera rotation. As for the Cartesian point features, the translational part of the Jacobian (the first 3 columns) are proportional to $1/Z$. Note also that the Jacobian is undefined for $r = 0$, that is for a point on the optical axis. The interaction matrix is computed by the `visjac_p_polar` method of the `CentralCamera` class.

The desired feature velocity is a function of feature error

$$\dot{\boldsymbol{p}}^* = \lambda \begin{pmatrix} r^* - r \\ \phi^* \ominus \phi \end{pmatrix}$$

where $\ominus$ is modulo-2π subtraction for the angular component subtraction for the angular component. The choice of units (pixels and radians) means that $|r| \gg |\phi|$ and radius should be normalized

$$r = \frac{\sqrt{\overline{u}^2 + \overline{v}^2}}{\sqrt{W^2 + H^2}}$$

so that r and ϕ are of approximately the same order.

An example of IBVS using polar coordinates is implemented by the class `IBVS_polar`. We first create a canonic camera, that has normalized image coordinates

```
>> cam = CentralCamera('default')
>> T_C0 = SE3(-0.3, 0.2, -2)*SE3.Rz(pi/2);
>> vs = IBVS_polar(cam, 'T0', T_C0, 'verbose')
```

and we run run a simulation

```
>> vs.run()
```

The animation shows the feature motion in the image, and the camera and world points in a world view. The camera motion is quite different compared to the Cartesian IBVS scheme introduced in the previous chapter. For the previously problematic case of large optical-axis rotation the camera has simply moved toward the goal and rotated. The features have followed straight line paths on the $r\phi$-plane. The performance of polar IBVS is the complement of Cartesian IBVS – it generates good camera motion for the case of large rotation, but poorer motion for the case of large translation.

The methods `plot_error`, `plot_vel` and `plot_camera` can be used to show data recorded during the simulation. An additional method

```
>> vs.plot_features()
```

displays the path of the features in ϕr-space and this is shown in Fig. 16.3 along with the camera motion which shows no sign of camera retreat.

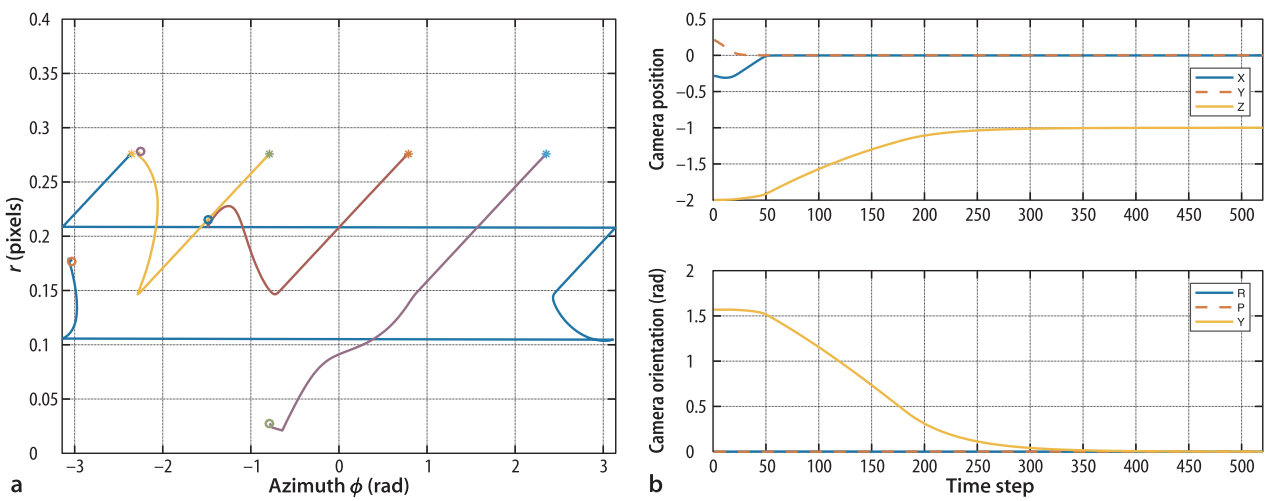


Fig. 16.3. IBVS using polar coordinates. **a** Feature motion in polar ϕr-space; **b** camera motion in Cartesian space

16.3 IBVS for a Spherical Camera

In Sect. 11.3 we looked at nonperspective cameras such as the fisheye lens camera and the catadioptric camera. Given the particular projection equations for any camera we can derive an image feature Jacobian from first principles. However the many different lens and mirror shapes leads to many different projection models and image Jacobians. In Sect. 11.4 we showed that feature points from any type of camera can be projected to a sphere, so we need to derive an image Jacobian for visual servo control on the sphere.

The image Jacobian for the sphere is derived in a manner similar to the perspective camera in Sect. 15.2.1. Referring to Fig. 11.21, the world point **P** is represented by the vector $\boldsymbol{P} = (X, Y, Z)$ in the camera frame, and is projected onto the surface of the sphere at the point $\mathbf{p} = (x, y, z)$ by a ray passing through the center of the sphere

$$x = \frac{X}{R},\ y = \frac{Y}{R},\ \text{and}\ \ z = \frac{Z}{R} \tag{16.10}$$

where $R = \sqrt{(X^2 + Y^2 + Z^2)}$ is the distance from the camera origin to the world point.

The spherical surface constraint $x^2 + y^2 + z^2 = 1$ means that one of the Cartesian coordinates is redundant so we will use a minimal spherical coordinate system comprising the angle of colatitude

Colatitude is zero at the north pole and increases as we move southwards.

$$\theta = \sin^{-1} r,\ \ \theta \in [0,\ \pi] \tag{16.11}$$

where $r = \sqrt{x^2 + y^2}$, and the azimuth angle (or longitude)

$$\phi = \tan^{-1} \frac{y}{x},\ \ \phi \in [-\pi,\ \pi) \tag{16.12}$$

which yields the point feature vector $\boldsymbol{p} = (\theta, \phi)$.

Taking the derivatives of Eq. 16.11 and Eq. 16.12 with respect to time and substituting Eq. 15.2 as well as

$$X = R\sin\theta\cos\phi,\ Y = R\sin\theta\sin\phi,\ Z = R\cos\theta \tag{16.13}$$

we obtain, in matrix form, the spherical optical flow equation

$$\begin{pmatrix} \dot{\theta} \\ \dot{\phi} \end{pmatrix} = \boldsymbol{J}_{p,s}(\theta, \phi, R) \begin{pmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{pmatrix} \tag{16.14}$$

where the image feature Jacobian is

$$\boldsymbol{J}_{p,s} = \begin{pmatrix} -\frac{\cos\phi\cos\theta}{R} & -\frac{\sin\phi\cos\theta}{R} & \frac{\sin\theta}{R} & \sin\phi & -\cos\phi & 0 \\ \frac{\sin\phi}{R\sin\theta} & -\frac{\cos\phi}{R\sin\theta} & 0 & \frac{\cos\phi\cos\theta}{\sin\theta} & \frac{\sin\phi\cos\theta}{\sin\theta} & -1 \end{pmatrix} \tag{16.15}$$

There are similarities to the Jacobian derived for polar coordinates in the previous section. Firstly, the constant elements fall at the same place, indicating that colatitude is invariant to rotation about the optical axis, and that azimuth angle is invariant to translation along the optical axis but equal and opposite to camera rotation about the optical axis. As for all image Jacobians the translational submatrix (the first three columns) is a function of point depth $1 / R$.

The Jacobian is not defined at the north and south poles where $\sin\theta = 0$ and azimuth also has no meaning at these points. This is a singularity, and as we remarked in Sect. 2.2.1.3, in the context of Euler angle representation of orientation, this is a consequence of using a minimal representation. However, in general the benefits outweigh the costs for this application.

For control purposes we follow the normal procedure of computing one 2×6 Jacobian, Eq. 16.15, for each of N feature points and stacking them to form a $2N \times 6$ matrix

$$\begin{pmatrix} \dot{\theta}_1 \\ \dot{\phi}_1 \\ \vdots \\ \dot{\theta}_N \\ \dot{\phi}_N \end{pmatrix} = \begin{pmatrix} \boldsymbol{J}_1 \\ \vdots \\ \boldsymbol{J}_N \end{pmatrix} \boldsymbol{\nu} \tag{16.16}$$

The control law is

$$\boldsymbol{\nu} = \boldsymbol{J}^{+} \dot{\boldsymbol{p}}^{*} \tag{16.17}$$

where $\dot{\boldsymbol{p}}^{*}$ is the desired velocity of the features in $\phi\theta$-space. Typically we choose this to be proportional to feature error

$$\dot{\boldsymbol{p}}^{*} = \lambda\left(\boldsymbol{p}^{*} \ominus \boldsymbol{p}\right) \tag{16.18}$$

where λ is a positive gain, $\boldsymbol{p}$ is the current point in $\phi\theta$-coordinates, and $\boldsymbol{p}^{*}$ the desired value. This results in locally linear motion of features within the feature space. ◂ $\ominus$ denotes modulo subtraction and returns the smallest angular distance given that $\theta \in [0, \pi]$ and $\phi = [-\pi, \pi)$.

Note that motion on this plane is in general not a great circle on the sphere – only motion along lines of longitude and the equator are great circles.

An example of IBVS using spherical coordinates (Fig. 16.4) is implemented by the class `IBVS_sph`. We first create a spherical camera

```
>> cam = SphericalCamera()
```

and then a spherical IBVS object

```
>> T_C0 = SE3(0.3, 0.3, -2)*SE3.Rz(0.4);
>> vs = IBVS_sph(cam, 'T0', T_C0, 'verbose')
```

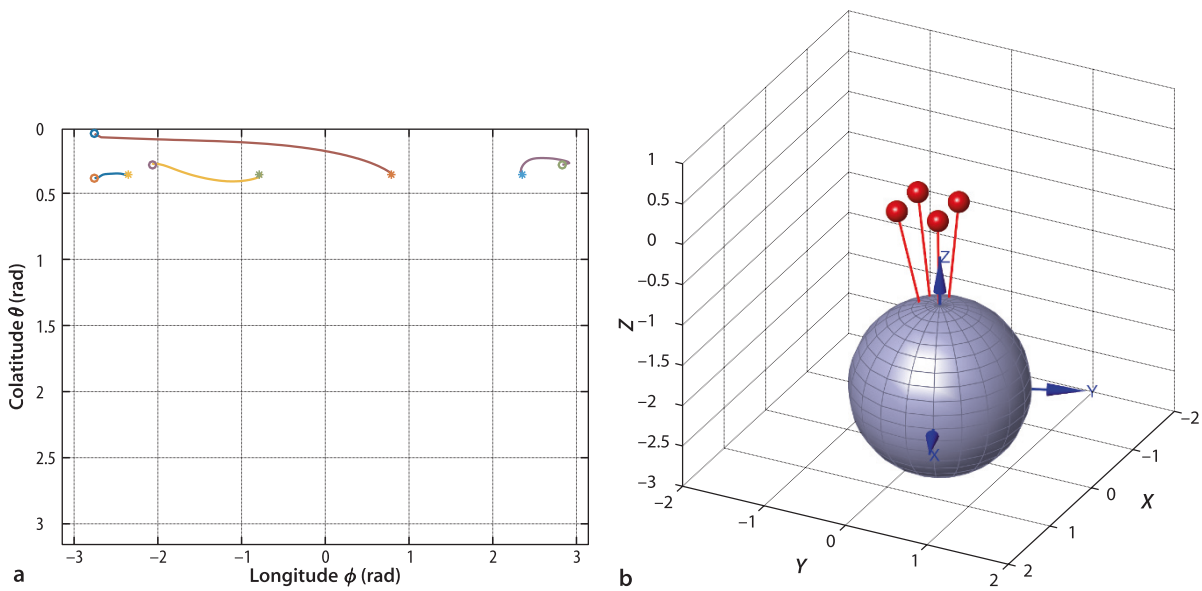


Fig. 16.4. IBVS using spherical camera and coordinates. **a** Feature motion in $\theta - \phi$ space; **b** four goal points projected onto the sphere in its initial pose

and we run run a simulation

```
>> vs.run()
```

The animation shows the feature motion on the $\phi\theta$-plane and the camera and world points in a world view. Spherical imaging has many advantages for visual servoing. Firstly, a spherical camera eliminates the need to explicitly keep features in the field of view which is a problem with both position-based visual servoing and some hybrid schemes. Secondly, we previously observed an ambiguity between the optical flow fields for R_x and $-T_y$ motion (and R_y and T_x motion) for a small field of view. For IBVS with a long focal length this can lead to slow convergence and/or sensitivity to noise in feature coordinates. For a spherical camera, with the largest possible field of view, this ambiguity is reduced.

Provided that the world points are well distributed around the sphere.

Spherical cameras do not yet exist but we can can project features from one or more cameras of any type onto the spherical image plane, and compute the control law in terms of spherical coordinates.

The camera of Fig. 11.27b (page 349) comes close with 90% of a spherical field of view.

16.4 Applications

16.4.1 Arm-Type Robot

In this example the camera is carried by a 6-axis robot which can control all six degrees of camera motion. We will assume that the robot's joints are ideal velocity sources, that is, they move at precisely the velocity that was commanded. A modern robot is very close to this ideal, typically having high performance joint controllers using velocity and position feedback from encoders on the joints.

The *nested* control structure for a robot joint was discussed in Sect. 9.1.7. The inner velocity loop uses joint velocity feedback to ensure that the joint moves at the desired speed. The outer position loop uses joint position feedback to determine the joint speed required to follow the trajectory. In this visual servo system the position loop function is provided by the vision system. Vision sensors have a low sample rate compared to an encoder, typically 25 or 30 Hz, and often with a high latency of one or two sample times.

The Simulink model of this eye-in-hand system

```
>> sl_arm_ibvs
```

is shown in Fig. 16.5. This is a complex example that simulates not only the camera and IBVS control but also the robot, in this case the ubiquitous Puma 560 from Part III of

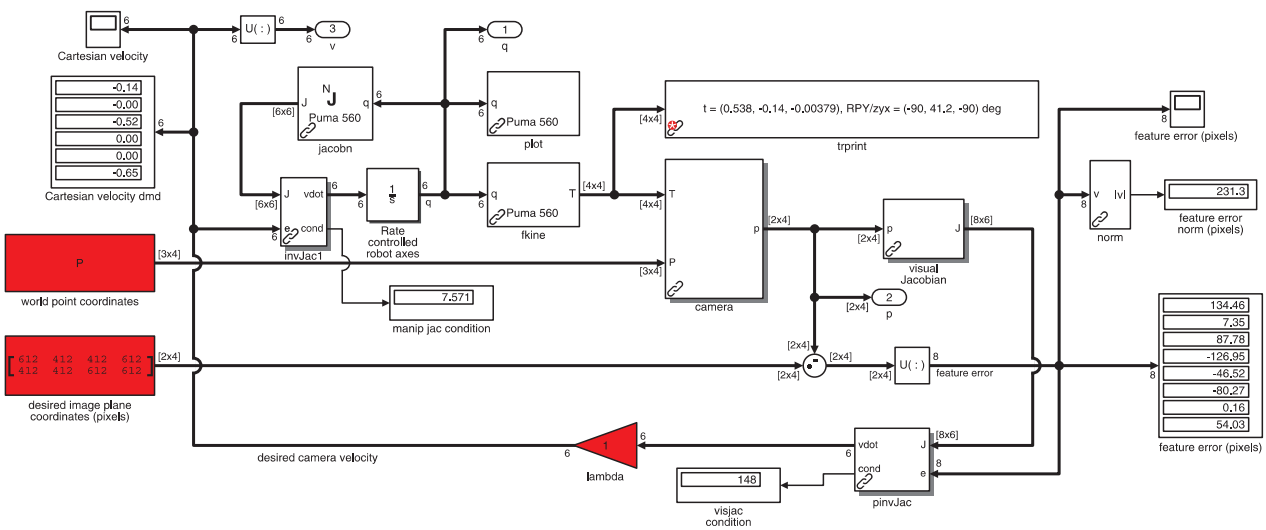


Fig. 16.5. The Simulink model `sl_arm_ibvs` uses IBVS to drive a Puma robot arm that is holding a camera

this book. The joint angles are the outputs of an integrator which represents the robot's velocity loops. These angles are input to a forward kinematics block which outputs the end-effector pose. A perspective camera with default parameters is mounted on the robot's end-effector and its axes are aligned with the end-effector coordinate frame. The camera block has one parameter which is a `CentralCamera` object, and its inputs are the camera pose world and the coordinates of the goal points which are the corners of a square in the *yz*-plane. The output image features are used to compute a Jacobian with an assumed *Z* value for every point, and also to determine the feature error in image space. The image Jacobian is inverted and a gain applied to determine the spatial velocity of the camera. The inverse manipulator Jacobian maps this to joint rates which are integrated to determine joint angles. This closed loop system drives the robot to the desired pose with respect to a square goal object.

We run this model

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

which displays the robot moving and the image plane of a virtual camera. This model, and the others in this chapter, use the `InitFcn` callback to create variables required by the Simulation in the MATLAB® workspace◀.

Simulink menu File+Model Properties +Callbacks+PreLoadFcn. These commands are executed once when a model is loaded.

The signals at the various output blocks are stored in the simulation results object `r` and the joint angles at each time step, output port one, are

```
>> q = squeeze(r.find('yout').signals(1).values)';
>> about(q)
q [double] : 60x6 (2880 bytes)
```

with one row per time step. Note that this model does not include any dynamics of the robot arm or the vision system. The joints are modeled as perfect velocity control devices, and the vision system is modeled as having no delay. This model could form the basis of more realistic system models that incorporate these real-world effects.

16.4.2 Mobile Robot

In this section we consider a camera mounted on a mobile robot moving in a planar environment. We will first consider a holonomic robot, that is one that has an omnidirectional base and can move in any direction, and then extend the solution to a nonholonomic car-like base which touches on some of the issues discussed in Chap. 4. The camera observes two or more point landmarks that have known 3-dimensional coordinates, that is, they can be placed above the plane on which the robot operates. The visual servo controller will drive the robot until its view of the landmarks matches the desired view.

16.4.2.1 Holonomic Mobile Robot

For this problem we assume a central perspective camera fixed to the robot and a number of landmarks with known locations that are continuously visible to the camera as the robot moves along the path. The vehicle's coordinate frame is such that the *x*-axis is forward and the *z*-axis is upward.

We define a perspective camera

```
>> cam = CentralCamera('default', 'focal', 0.002);
```

with a wide field of view so that it can keep the landmarks in view as it moves. The camera is mounted on the vehicle with a relative pose ${}^{B}\xi_{C}$

```
>> V_T_C = SE3(0.2, 0.1, 0.3)*SE3.Rx(-pi/4);
```

relative to the vehicle coordinate frame. This is to the front left of the vehicle, 30 cm above ground level, with its optical axis forward but pitched upward at 45°, and its x-axis pointing to the right of the vehicle. The two landmarks are 2 m above the ground and situated at $x = 0$ and $y = \pm 1$ m

```
>> P = [0 0; 1 -1; 2 2]
```

The desired vehicle position is with the center of the rear axle at $(-2, 0, 0)$.

Since the robot operates in the xy-plane and can rotate only about the z-axis we can remove the columns from Eq. 15.6 that correspond to nonpermissible motion and write

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} -\frac{f}{\rho_u Z} & 0 & \overline{v} \\ 0 & -\frac{f}{\rho_v Z} & -\overline{u} \end{pmatrix} \begin{pmatrix} v_x \\ v_y \\ \omega_z \end{pmatrix} \tag{16.19}$$

As for standard IBVS case we stack these Jacobians, one per landmark, and then invert the equation to solve for the vehicle velocity. Since there are only three unknown components of velocity, and each landmark contributes two equations, we need two or more feature points in order to solve for velocity.

The Simulink model

```
>> sl_omni_vs
```

is shown in Fig. 16.6 and is similar in principle to earlier models such as Fig. 16.5 and 15.9. The model is simulated by

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

and displays an animation of the vehicle's path in the xy-plane and the camera view. Results are stored in the simulation results object `r` and can be displayed as for previous examples. The parameters and camera are defined in the properties of the model's various blocks.

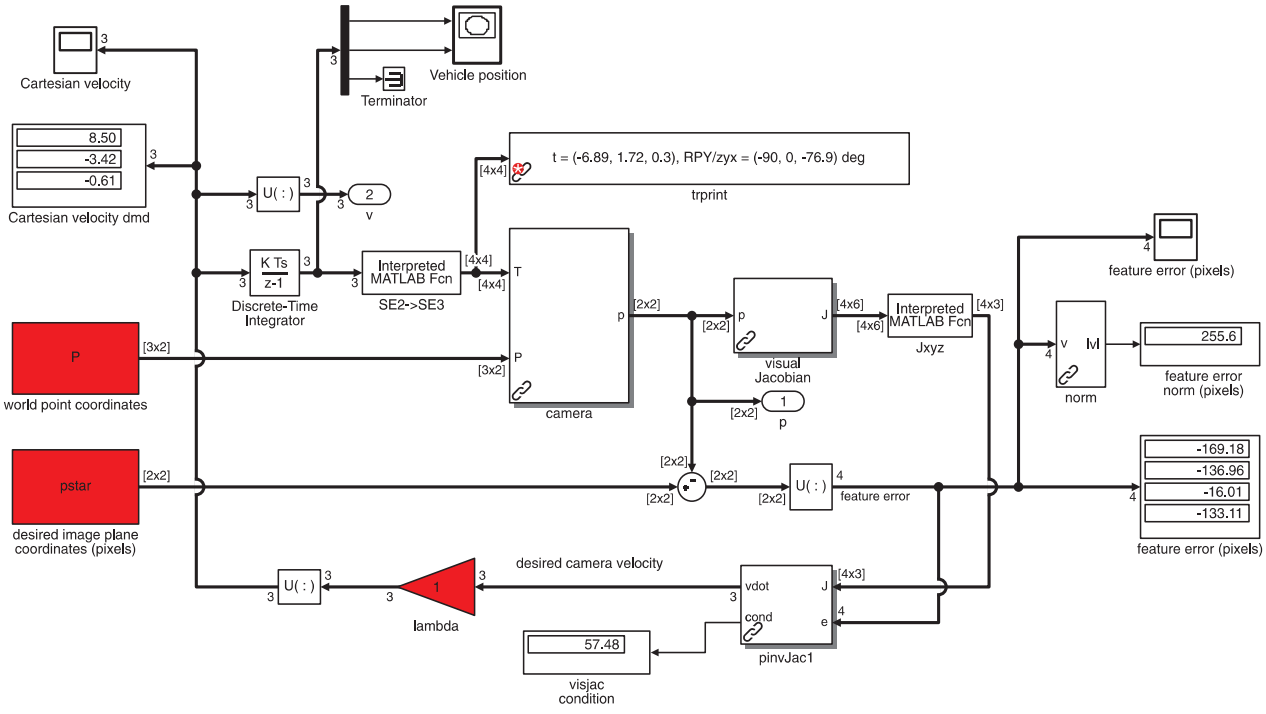


Fig. 16.6. The Simulink model `sl_mobile_vs` drives a holonomic mobile robot to a pose using IBVS control

16.4.2.2 Nonholonomic Mobile Robot

The difficulties of servoing a nonholonomic mobile robot to a pose were discussed earlier and a nonlinear pose controller was introduced in Sect. 4.1.1.4. The notation for our problem is shown in Fig. 16.7 and once again we use a controller based on the polar coordinates ρ, α and β. For this control example we will use PBVS techniques to estimate the variables needed for control. We assume a central perspective camera that is fixed to the robot body frame with a relative pose ${}^{B}\xi_{C}$, a number of landmarks with known locations that are continuously visible to the camera as it moves along the path, and that the vehicle's orientation θ is also known, perhaps using a compass or some other sensor.

The Simulink model

```
>> sl_drivepose_vs
```

is shown in Fig. 16.8. The initial pose of the camera is set by a parameter of the `Bicycle` block. The view of the landmarks is simulated by the camera block and its output, the projected points, are input to a pose estimation block and the known locations of the landmarks are set as parameters. As discussed in Sect. 11.2.3 at least three landmarks are needed and in this example four landmarks are used. The output ${}^{C}\hat{\xi}_{L}$ is the estimated pose of the landmarks with respect to the camera. The vehicle pose in the world frame is obtained by a chain of simple transform operations $\hat{\xi}_{B} = \ominus\, {}^{C}\hat{\xi}_{0} \ominus {}^{B}\xi_{C}$. The x- and y-components of this transform are combined with estimated heading angle◂ to yield an estimate of the vehicle's configuration $(\hat{x}, \hat{y}, \hat{\theta})$ which is input to the pose controller. The remainder of the system is essentially the same as the example from Fig. 4.11.

In a real system heading angle would come from a compass, in this simulation we "cheat" and simply use the true heading angle.

The simulation is run by

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

and the camera pose, image-plane feature error and camera velocity are animated. Scope blocks also plot the camera velocity and feature error against time. Results are stored in the simulation results object `r` and can be displayed as for previous examples.

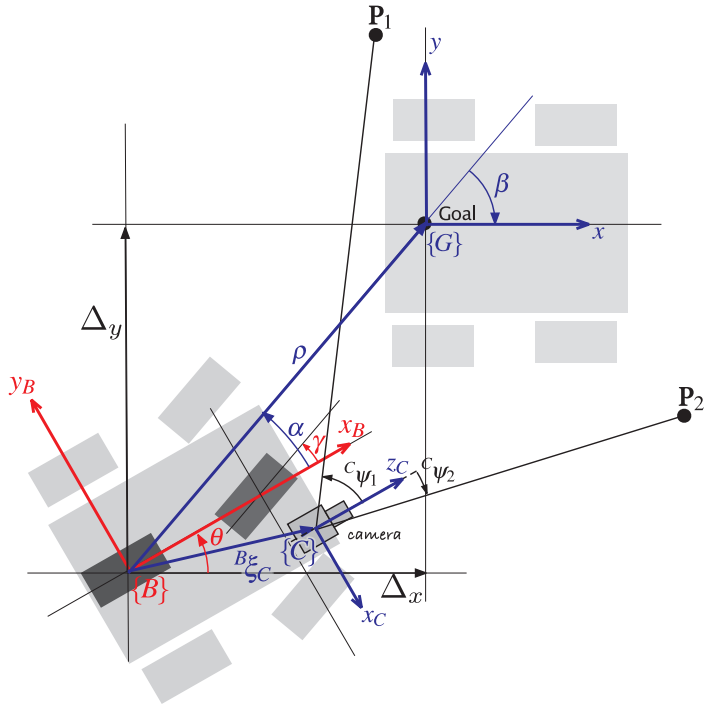


Fig. 16.7. PBVS for nonholonomic vehicle (bicycle model) vehicle moving toward a goal pose: ρ is the distance to the goal, β is the angle of the goal vector with respect to the world frame, and α is the angle of the goal vector with respect to the vehicle frame. $\mathbf{P}_1$ and $\mathbf{P}_2$ are landmarks which are at bearing angles of ψ_1 and ψ_2 with respect to the camera

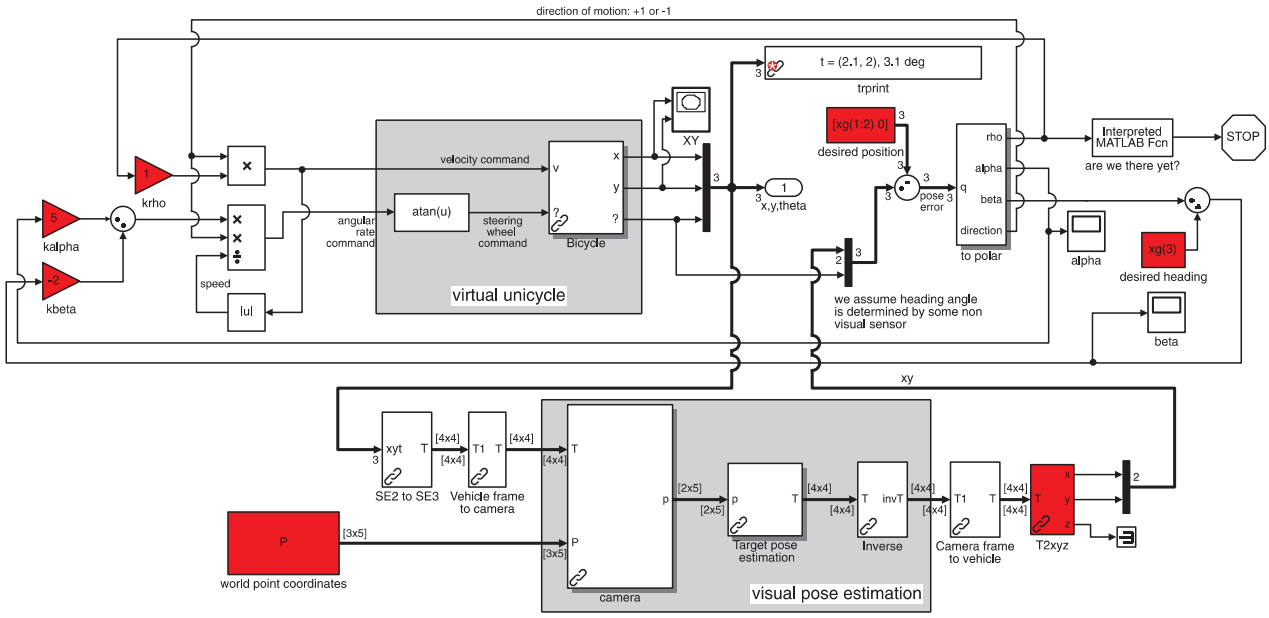


Fig. 16.8. The Simulink model `sl_drivepose_vs` drives a non-holonomic mobile robot to a pose (derived from Fig. 4.11)

16.4.3 Aerial Robot

A spherical camera is particularly suitable for platforms that move in $\mathbf{SE}(3)$ such as aerial and underwater robots. In this example we consider a spherical camera attached to a quadrotor and we will use IBVS to servo the quadrotor to a particular pose with respect to four goal points on the ground.

As we discussed in Sect. 4.3 the quadrotor is under-actuated and we cannot independently control all 6 degrees of freedom in task space. We can control position (X, Y, Z) and also yaw angle. Roll and pitch angle are manipulated to achieve translation in the horizontal plane and must be zero when the vehicle is in equilibrium. The Simulink model

```
>> sl_quadrotor_vs
```

is shown in Fig. 16.9. This controller attempts to keep the quadrotor at a constant relative pose with respect to the goal points on the ground. If the goal moves so too will the quadrotor – we could imagine a scheme like this being used to land a quadrotor on a car.

The model is a hybrid of the quadrotor controller from Fig. 4.21 and the under-actuated IBVS system of Fig. 16.6. There are however a number of key differences. Firstly, in the quadrotor control of Fig. 4.21 we used a rotation matrix to map xy-error in the world frame to the pitch and roll demand of the vehicle. This is not needed for the visual servo case since the xy-error is given in the camera, or vehicle, frame rather than the world frame. Secondly, like the mobile robot case the vehicle is under-actuated, and here the Jacobian comprises only the columns corresponding to $(v_x, v_y, v_z, \omega_z)$. Thirdly, we are using a spherical camera, so a `SphericalCamera` object is passed to the camera and visual Jacobian blocks.

Fourthly, there is coupling between the roll and pitch motion of the quadrotor and the image-plane feature coordinates. We recall how the quadrotor cannot translate without first tilting into the direction it wishes to translate, and this will cause the features to move in the image and increase the image feature error. For small amounts of roll and pitch this can be ignored but for aggressive maneuvers it must be taken into account. We can use the image Jacobian to approximate▸ the displacements in θ and ϕ as a function of displacements in camera roll and pitch angle which are rotations about the x- and y-axes respectively

▸ This is a first-order approximation to the feature motion.

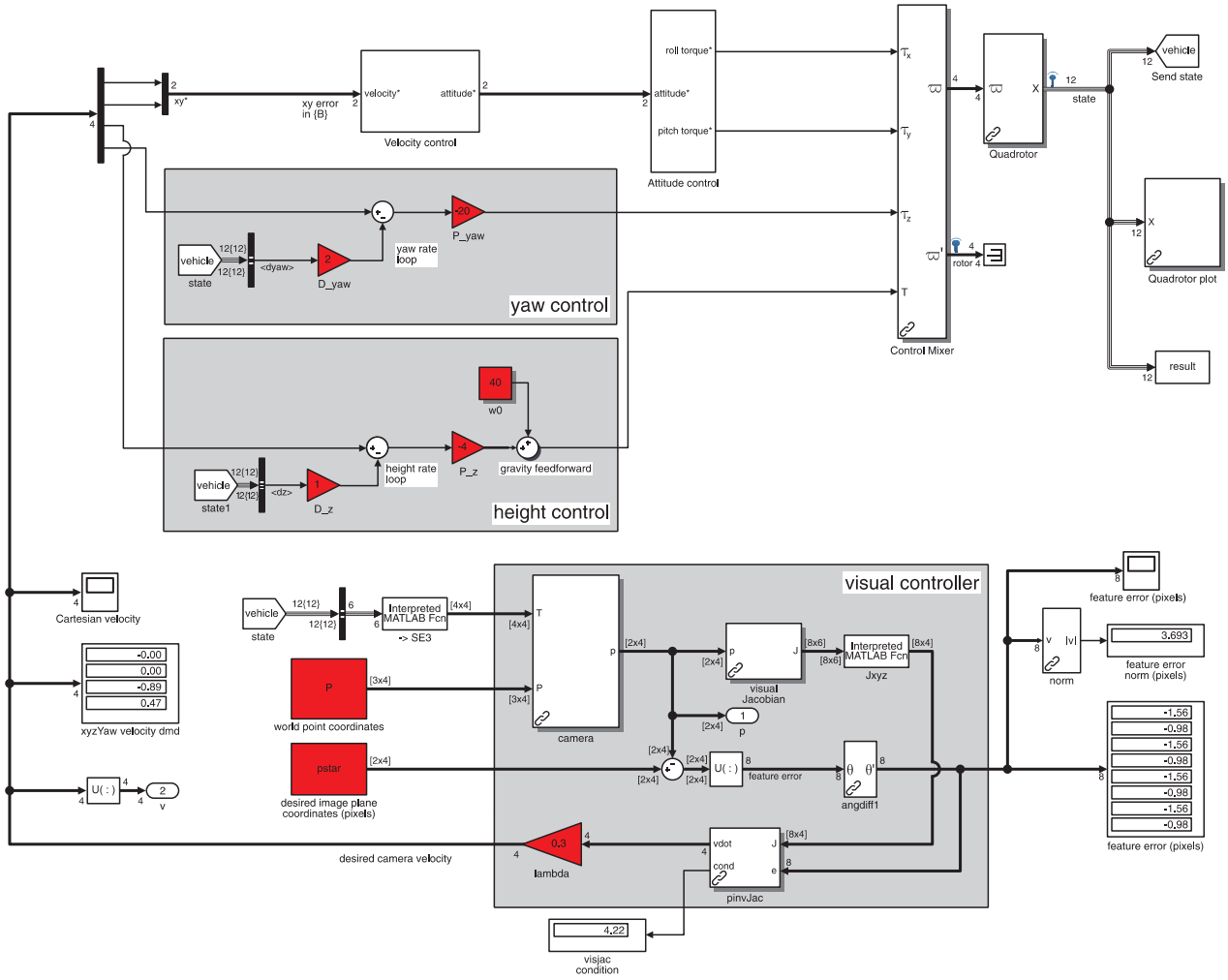


Fig. 16.9. The Simulink model `sl_quadrotor_vs`. IBVS with a spherical camera for hovering over a goal. Compared to the previous models this one has an `angdiff` block after the feature error summing junction to allow for proper handling of angles on the sphere

$$\begin{pmatrix}\Delta\theta\\ \Delta\phi\end{pmatrix}=\begin{pmatrix}\frac{\cos\phi\cos\theta}{\sin\theta} & \frac{\sin\phi\cos\theta}{\sin\theta}\\ \sin\phi & -\cos\phi\end{pmatrix}\begin{pmatrix}\Delta\theta_R\\ \Delta\theta_P\end{pmatrix}$$

and these are subtracted from the features observed by the camera to give the features that would be observed by a camera in the vehicle's body frame {*B*}. This scheme is sometimes referred to as feature derotation since it mimics in software the effect of a nonrotating or gimbal-stabilized camera.

Comparing Fig. 16.9 to Fig. 4.21 we see the visual controller performs the function of the outermost *position loops* for *x*- and *y*-position, altitude and yaw and generates the required velocities for the velocity loops of these degrees of freedom directly. Note that rate information is still required as input to the velocity loops and in a real robot this would be derived from an inertial measurement unit.

The simulation is run by

```
>> sim('sl_quadrotor_vs')
```

Simulink menu File+Model Properties +Callbacks+InitFcn. These commands are always executed prior to the beginning of a simulation.

and the camera pose, image-plane feature error and camera velocity are animated. Scope blocks also plot the camera velocity and feature error against time. The simulation results can be obtained from the simulation output object `out`. The initial pose of the camera is set in the model's properties◀.

16.5 Wrapping Up

Further Reading

A good introduction to advanced visual servo techniques is the tutorial article by Chaumette and Hutchinson (2007) and also the visual servoing chapter in Siciliano and Khatib (2016, § 34). Much of the interest in so-called hybrid techniques was sparked by Chaumette's paper (Chaumette 1998) which introduced the specific example that drives the camera of a point-based IBVS system to infinity for the case of goal rotation by π about the optical axis. One of the first methods to address this problem was 2.5D visual servoing, proposed by Malis et al. (1999), which augments the image-based point features with a minimal Cartesian feature. Other notable early hybrid methods were proposed by Morel et al. (2000) and Deguchi (1998) which partitioned the image Jacobian into a translational and rotational part. An homography is computed between the initial and final view (so the goal points must be planar) and then decomposed to determine a rotation and translation. Morel et al. combine this rotational information with translational control based on IBVS of the point features. Conversely, Deguchi et al. combine this translational information with rotational control based on IBVS. Since translation is only determined up to an unknown scale factor some additional means of determining scale is required.

Corke and Hutchinson (2001) presented an intuitive geometric explanation for the problem of the camera moving away from the goal during servoing, and proposed a partitioning scheme split by axes: x- and y-translation and rotation in one group, and z-translation and rotation in the other. Another approach to hybrid visual servoing is to switch rapidly between IBVS and PBVS approaches (Gans et al. 2003). The performance of several partitioned schemes is compared by Gans et al. (2003).

The polar form of the image Jacobian for point features (Iwatsuki and Okiyama 2002a; Chaumette and Hutchinson 2007) handles the IBVS failure case nicely, but results in somewhat suboptimal camera translational motion (Corke et al. 2009) – the converse of what happens for the Euclidean formulation.

The Jacobian for a spherical camera is similar to the polar form. The two angle parameterization was first described in Corke (2010) and was used for control and structure-from-motion estimation. There has been relatively little work on spherical visual servoing. Fomena and Chaumette (2007) consider the case for a single spherical object from which they extract features derived from the projection to the spherical imaging plane such as the center of the circle and its apparent radius. Tahri et al. (2009) consider spherical image features such as lines and moments. Hamel and Mahony (2002) describe kino-dynamic control of an under-actuated aerial robot using point features.

The robot manipulator dynamics Eq. 9.8 and the perspective projection Eq. 11.2 are highly nonlinear and a function of the state of the manipulator and the goal. Almost all visual servo systems consider that the robot is velocity controlled, and that the underlying dynamics are suppressed and linearized by tight control loops. As we learned in Sect. 9.1 this is the case for arm-type robots and in the quadrotor example we used a similar nested control structure. This approach is necessitated by the short time constants of the underlying mechanism and the slow sample rate and latency of any visual control loop. Modern computers and high-speed cameras make it theoretically possible to do away with axis-level velocity loops but it is far simpler to use them.

Visual servoing of nonholonomic robots is nontrivial since Brockett's theorem (1983) shows that no linear time-invariant controller can control it. The approach used in this chapter was position based which is a minor extension of the pose controller introduced in Sect. 4.1.1.4. IBVS approaches have been proposed (Tsakiris et al. 1998; Masutani et al. 1994) but require that the camera is attached to the base by a robot with a small number of degrees of freedom. Mariottini et al. (1994, 2007) describe a two-step servoing approach where the camera is rigidly attached to the base and the epipoles of the geometry defined by the current and desired camera views are explicitly servoed. Usher

(Usher et al. 2003; Usher 2005) describes a switching control law that takes the robot onto a line that passes through the desired pose, and then along the line to the pose – experimental results on an outdoor vehicle are presented. The similarity between mobile robot navigation and visual servoing problem is discussed in Corke (2001).

Resources

The controllers demonstrated in this chapter have all worked with simulated robotic systems, and have executed much slower than real time. In order to put visual control into practice we need to have fast image processing and feature extraction algorithms, as well as means of communicating with the robot hardware. Fortunately there are lots of tools and technologies to help with this: the Robot Operating System (aka, ROS www.ros.org) is a comprehensive robot software framework for creating robots, OpenCV for image processing (www.opencv.org), ViSP for creating visual trackers and controllers (www.irisa.fr/lagadic/visp). Simulink supports real-time vision through the Computer Vision System Toolbox, and the automatic synthesis of controllers that can run on your computer, can be exported to real-time hardware, or be exported as source code of a complete ROS node.

Exercises

1. XY/Z-partitioned IBVS (page 567)
   a) Investigate the generated motion for different combinations of initial camera translation and rotation, and compare to the classical IBVS scheme of the last chapter.
   b) Create a scenario where the features leave the image.
   c) Add a repulsion field to ensure that the features remain within the image.
   d) Investigate variations of Eq. 16.3. Instead of driving the difference of area to zero, try driving the ratio of current and desired area to one, or the logarithm of this ratio to zero.
2. Investigate the performance of polar and spherical IBVS for different combinations of initial camera translation and rotation, and compare to the classical IBVS scheme of the last chapter.
3. Arm-robot IBVS example (page 572)
   a) Add an offset (rotation and/or translation) between the end-effector and the camera. Your controller will need to incorporate an additional Jacobian (see Sect. 3.1.2) to account for this.
   b) Add a discrete time sampler and delay after the camera block to model the camera's frame rate and image processing time. Investigate the response as the delay is increased, and the tradeoff between gain and delay. You might like to plot a discrete-time root locus diagram for this dynamic system.
   c) Model a moving goal. Hint use the `Camera2` block from the `roblocks` library. Show the tracking error, that is, the distance between the camera and the goal.
   d) Investigate feedforward techniques to improve the control (Corke 1996b). Hint, instead of basing the control on where the goal was seen by the camera, base it on where it will be some short time into the future. How far into the future? What is a good model for this estimation? Check out the Toolbox class `AlphaBeta` for a simple to use tracking filter (challenging).
   e) An eye-in-hand camera for a docking task might have problems as the camera gets really close to the goal. How might you configure the goal points and camera to avoid this?
4. Mobile robot visual servo (page 574)
   a) For the holonomic and nonholonomic cases replace the perspective camera with a catadioptric camera.

   b) For the holonomic case with a catadioptric camera, move the robot through a series of via points, each defined in terms of a set of desired feature coordinates.
   c) For the nonholonomic case implement the pure pursuit and line following controllers from Chap. 4 but in this case using visual features. For pure pursuit consider the object being pursued carries one or two point features. For the line following case consider using one or two line features.
5. Display the feature flow fields, like Fig. 15.7, for the polar $r - \phi$ and spherical $\theta - \phi$ projections (Sect. 16.2 and 16.3). For the spherical case can you plot the flow vectors on the surface of a sphere?
6. Quadrotor
   a) Replace the spherical camera with a perspective camera.
   b) Create a controller to follow a series of point features rather than hover over a single point (challenging).
   c) Create a controller to follow a series of point features rather than hover over a single point (challenging).
   d) Add image feature derotation to minimize the effect of vehicle roll and pitch on the visual control.
7. Implement the 2.5D visual servo scheme by Malis (1999) (challenging).