CSE 276A:
Introduction to Robotics

Lecturer: Henrik Christensen

Notes by Owen Jow

1 Mobile Robots 10/08/20

1.1 Kinematic Models

1.1.1 Differential Drive

In a differential drive system, there is a left wheel and a right wheel (both typically facing forward) which can be controlled independently. If you drive one wheel in one direction and the other wheel in the other direction, you can make the robot rotate in place. The wheels need not actually yaw!

If the angular velocities of the left and right wheels are equal, the robot will move forward/backward. If the angular velocities of the left and right wheels are opposite, the robot will rotate in place.

1.1.2 Bicycle Model

In the kinematic bicycle model, the rear wheels control the velocity of the robot and the front wheels control the steering angle. You can approximate this type of setup as a bicycle (drive with the rear, steer with the front). This is the more typical model for a car.

2 Visual Servoing 11/03/20

Use visual information to control the motion of a robot.

2.1 Position-Based Visual Servoing (PBVS)

Estimate the 3D pose of landmark object(s) with respect to the camera. Then run controller.

2.2 Image-Based Visual Servoing (IBVS)

Don't estimate the 3D pose of the object or the camera – just go by the image. Move the camera[1] so that certain feature points (in the image) go to the places (in the image) we want them to go.

[1] Assumption: "eye-in-hand" camera placement, where the camera is attached to the moving robot.

The relationship between feature point motion and camera motion is

$$\begin{bmatrix}\dot{u}\\ \dot{v}\end{bmatrix} = \begin{bmatrix}-\frac{f}{Z} & 0 & \frac{u}{Z} & \frac{uv}{f} & -\frac{f^2+u^2}{f} & v\\ 0 & -\frac{f}{Z} & \frac{v}{Z} & \frac{f^2+v^2}{f} & -\frac{uv}{f} & -u\end{bmatrix}\begin{bmatrix}v_x\\ v_y\\ v_z\\ \omega_x\\ \omega_y\\ \omega_z\end{bmatrix}$$

$$\dot{\mathbf{p}} = \mathbf{J}(u, v, Z)\dot{\mathbf{q}}$$

where u and v are the pixel coordinates (*relative to the principal point*) of a feature point, $\dot{u}$ and $\dot{v}$ represent the change in pixel coordinates, f is the focal length, Z is the depth of the 3D point corresponding to (u, v), and $(\mathbf{v}, \omega)$ are the translational and angular velocities of the camera.

$\mathbf{J}$ is the image Jacobian. It maps camera velocity to pixel velocity.

With n feature points,

$$\begin{bmatrix}\dot{\mathbf{p}}_1\\ \vdots\\ \dot{\mathbf{p}}_n\end{bmatrix} = \begin{bmatrix}\mathbf{J}(u_1, v_1, Z_1)\\ \vdots\\ \mathbf{J}(u_n, v_n, Z_n)\end{bmatrix}\dot{\mathbf{q}}$$

Thus we can figure out how to move the camera as

$$\dot{\mathbf{q}} = \lambda \begin{bmatrix}\mathbf{J}(u_1, v_1, Z_1)\\ \vdots\\ \mathbf{J}(u_n, v_n, Z_n)\end{bmatrix}^{\dagger}\begin{bmatrix}\mathbf{p}_1^* - \mathbf{p}_1\\ \vdots\\ \mathbf{p}_n^* - \mathbf{p}_n\end{bmatrix}$$

where λ is an arbitrary scaling factor and $\mathbf{p}_i^*$ is the target location for each feature point $\mathbf{p}_i$.

Note: to solve for $\dot{\mathbf{q}}$, you need to know the focal length f, the principal point, and the depth Z_i for each feature point $\mathbf{p}_i$. You can get the focal length and the principal point from camera calibration, and the depth from an arbitrary estimation technique without worrying too much.[2]

3 SLAM

11/05/20 - 11/10/20

3.1 Kalman Filter

Estimate (in real time) the state of a dynamic system given noisy observations.[3]

Imagine an HMM graph where, at each time step, a state emits an observation and feeds into the state at the next time step. Under the assumptions of a *linear dynamical system* and *Gaussian random variables*, and given (1) the previous state estimate, (2) the control input, and (3) the current observation, a Kalman filter can estimate the current state in a predictive-corrective fashion.

[2] Apparently, IBVS is quite tolerant to error in Z.

[3] "*Filter out the noise in the information to get the underlying true state.*"

3.1.1 Linear Dynamical System

Let

- s_t denote the state at time t. $[n \times 1]$
- o_t denote the observation at time t. $[m \times 1]$
- u_t denote the control input at time t. $[k \times 1]$

The Kalman filter assumes a linear mapping from s_{t-1} to s_t and from s_t to o_t:

$$s_t = \mathbf{F}s_{t-1} + \mathbf{G}u_{t-1} + w_t$$
$$o_t = \mathbf{H}s_t + v_t$$

where

- $\mathbf{F}$ is an $n \times n$ matrix mapping s_{t-1} to s_t in the absence of external influence or noise.
- $\mathbf{G}$ is an $n \times k$ matrix mapping the control input u_{t-1} to its effect on s_t.
- $\mathbf{H}$ is an $m \times n$ matrix mapping s_t to o_t in the absence of noise.
- $w_t \sim \mathcal{N}(0, \mathbf{Q})$ is an $n \times 1$ Gaussian random variable representing process noise.
  - $\mathbf{Q}$ is the $n \times n$ process noise covariance matrix.
- $v_t \sim \mathcal{N}(0, \mathbf{R})$ is an $m \times 1$ Gaussian random variable representing observation noise.
  - $\mathbf{R}$ is the $m \times m$ observation noise covariance matrix.

The $\mathbf{F}$, $\mathbf{G}$, $\mathbf{H}$, $\mathbf{Q}$, and $\mathbf{R}$ matrices are assumed to be time-independent.

3.1.2 Kalman Filter

At each time step, the Kalman filter predicts the current state based on previous information ("*how did we expect the system to evolve*?"), and then corrects that prediction given the current observation ("*how closely does the actual observation agree with the observation we expected*?")

It maintains an estimate s_t of the state, and the covariance matrix Σ_t associated with that estimate. Initialize s_0 and Σ_0, and then perform the following update at every time step...

3.1.3 Prediction Step

$$\hat{s}_t = \mathbf{F}s_{t-1} + \mathbf{G}u_{t-1}$$
$$\hat{\Sigma}_t = \mathbf{F}\Sigma_{t-1}\mathbf{F}^\top + \mathbf{Q}$$

3.1.4 Correction Step

$$\mathbf{K}_t = \hat{\Sigma}_t\mathbf{H}^\top(\mathbf{H}\hat{\Sigma}_t\mathbf{H}^\top + \mathbf{R})^{-1}$$
$$s_t = \hat{s}_t + \mathbf{K}_t(o_t - \mathbf{H}\hat{s}_t)$$
$$\Sigma_t = (\mathbf{I} - \mathbf{K}_t\mathbf{H})\hat{\Sigma}_t$$

Note: $\mathbf{I}$ is the $n \times n$ identity matrix, $\mathbf{K}_t$ is the "Kalman gain," and $(o_t - \mathbf{H}\hat{s}_t)$ is the error signal.

3.1.5 Extended Kalman Filter

The **extended Kalman filter** is able to perform state estimation for nonlinear dynamical systems.

3.1.6 Kalman Filter Versus Particle Filter

The Kalman filter is a unimodal framework (*one* state estimate and associated uncertainty); the particle filter can maintain many different hypotheses (one for each particle). Also, the Kalman filter assumes Gaussianity, whereas the particle filter can approximate any probability distribution.

3.2 Localization

Localization: given a model/map of the environment, a kinematic/dynamic model for the robot, a set of sensors to detect features, and a strategy to associate features with the environment model, *estimate the robot pose (position and orientation).* "Where is the robot in a map of the world?"

3.2.1 Kalman-Filter-Based Localization

Let s_t denote the pose at discrete time step t. Approximate $P(s_t)$ by a multivariate Gaussian with covariance matrix (uncertainty estimate) Σ_t. Update estimates of s_t and Σ_t using a Kalman filter.

3.3 Mapping

Mapping: given a kinematic/dynamic model for the robot, a set of sensors to detect features, a strategy for propagating uncertainty over time, a data association[4] strategy, and knowledge of the trajectory traversed by the robot, *estimate the positions of features in the environment.*

3.4 Simultaneous Localization and Mapping (SLAM)

SLAM: given a kinematic/dynamic model for the robot, a set of sensors to detect features, a strategy for propagating uncertainty over time, and a data association strategy, *estimate the robot pose and the positions of map features while traversing an environment.*

3.4.1 Kalman-Filter-Based SLAM

You can perform SLAM by using a Kalman filter to continually estimate a state vector consisting of the *robot pose* and the *coordinates of landmarks/features in the environment.*

4 Path Planning 11/17/20

Given the robot pose and a map, you can generate a representation of those things that will make it more efficient to plan a path.[5] Then you can use that representation to search for the best path.

[4] When you see something new, associate it with something known. "*Align features you see to features you expect.*"

[5] e.g. a **Voronoi diagram**, which can maximize the space between (a) obstacles and (b) the robot as it moves

References

1. Henrik Christensen, Fall 2020 CSE 276A lectures.
2. *Robotics, Vision and Control* by Peter Corke.
3. "Image-Based Visual Servoing" by Peter Corke.
4. "How a Kalman filter works, in pictures" by Tim Babb.
5. "An Introduction to the Kalman Filter" by Greg Welch and Gary Bishop.
6. "Simultaneous localization and mapping with the extended Kalman filter" by Joan Solà.