1

Embedded Technology for Mobile Robotics

Mobile robot applications are popular in various domains such as security, surveillance, automation, agriculture, and space missions. While the autonomy and capability to perform complex tasks are being possible, it is always required to control the robot operations including low-level control of actuators and processing sensors. It is increasing demand of accommodating all the controls and processing on an embedded system to optimize power consumption, size, and/or cost. The performance of a robotic system not only depends on the attached sensors but also its controller design. Further, the implementation of controller and processing sensors largely contribute to the desired behavior of the robot. Hence, a great deal of importance is to be given for implementation aspects of the controller.

The embedded system solution provides a customized solution for a controller. This customization may target optimization for cost, power, size, speed, or combination of these. Furthermore, the controller on an embedded platform may be designed to achieve real-time requirements. A single platform may be used for multiple controllers working in synchronization. The synchronization or parallel triggering is easily achievable if the architecture for the controllers is designed on a single chip using either Application-Specific Integrated Circuit (ASIC) or Field Programmable Gate Array (FPGA) technology. However, one needs to learn designing architecture to best utilize these technologies.

There exists availability of cross-platforms like *system generator* for high-level programming platforms such as MatLab that makes the job easier for developing embedded controllers using high-level codes. But, unless the control engineer knows about the architectural features of an embedded platform, the advanced properties are difficult to choose. Learning to develop an elementary architecture details would enable a control engineer to optimize the implementation. Hence, the study of Embedded Control Systems (ECSs) involves not only designs of control methodologies but also concepts related to elementary knowledge of embedded requirements. This chapter covers *ECSs*, *Mobile Robots* as a system to be controlled, and *Embedded Technologies* used in existing mobile robots.

Embedded Control for Mobile Robotic Applications, First Edition.
Leena Vachhani, Pranjal Vyas, and Arunkumar G. K.
© 2022 The Institute of Electrical and Electronics Engineers, Inc. Published 2022 by John Wiley & Sons, Inc.
Companion website: www.wiley.com/go/vachhani/embeddedcontrolforroboticapp

COPYRIGHTED MATERIAL

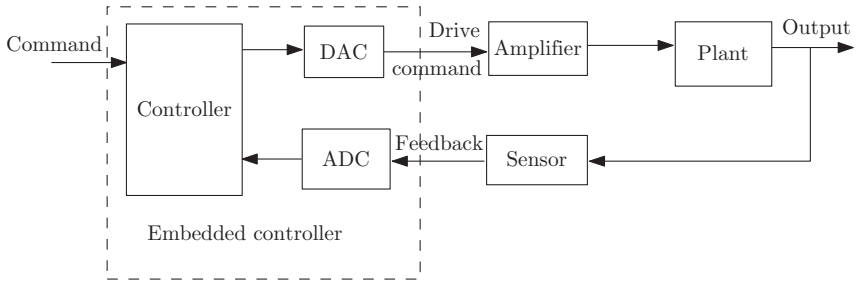


Figure 1.1 Block diagram of an embedded control system

1.1 Embedded Control System

The embedded controller referred here is the controller implemented on an embedded platform. Figure 1.1 illustrates a block diagram of an ECS where the controller is implemented on an embedded platform. The embedded controller is controlling a plant or a system which in our case is a mobile robot. In practice, the plant characteristic is often analog in nature.

The physics of the plant or system is typically modeled using transfer function if the system can be very well described by a linear system. The transfer function of the system describes the input–output relationship in the frequency domain. In particular, a transfer function is the ratio of Laplace transform of output to input of the system. The modern control theory represents a linear system as a state-space model. For representing a system, let the state vector of the system be $X \in \mathbb{R}^n$, input vector to the system be $U \in \mathbb{R}^m$, and output vector be $Y \in \mathbb{R}^p$. The state space representation of the linear system is now given by

$$\dot{X} = AX + BU; \quad Y = CX \tag{1.1}$$

where A is system matrix, B is input matrix, and C is output matrix with appropriate dimensions. For a nonlinear system, the state-space representation is given by

$$\dot{X} = f(X, U); \quad Y = h(X) \tag{1.2}$$

where $f(\cdot)$ and $h(\cdot)$ are nonlinear functions. In an interesting and popular representation of state-space in mobile robotics, the input to system is the derivative of a variable. For example, input to the system is acceleration which is derivative of velocity. The double integrator model is then represented for state variables $X = [X_1 \quad X_2]$ and state-space representation of the system is given by

$$\dot{X}_1 = f(X_1, X_2); \quad \dot{X}_2 = U; \quad y = h(X) \tag{1.3}$$

Similarly, chained form of system description is also very popular and many controllers have been designed for the system in chained form. The input-to-state

relation in the chained form of representing the three dimensional system for state vector $X = [x_1 \quad x_2 \quad x_3]^T$ and input vector $U = [u_1 \quad u_2]^T$ is described by

$$\dot{x}_1 = u_1, \quad \dot{x}_2 = u_2, \quad \dot{x}_3 = x_2 u_1 \tag{1.4}$$

In many cases, mobile robot models are represented in double integrator and chained form. Appropriate transformations are used for representing the mobile robot models in these standard forms. An appropriate transformation for control commands accordingly needs to be implemented on the embedded platform. Moreover, the controller design development depends on the application objectives and input–output relationship of the mobile robot represented as a system. While application-specific objectives for embedded implementations are discussed in Chapter 3, the system representation of various popular mobile robots is discussed next.

1.2 Mobile Robotics

There exists variety of mobile robotic platforms that facilitate autonomous and remotely controlled robotic applications. These can be categorized under wheeled robots, aerial vehicles, and underwater vehicles. These vehicles can further have generic categorization of their movements in 2D (planar) and 3D. While these robotic platforms are customized with various payloads like camera, manipulator arm, or application-specific sensors, the common functionality movement can be modeled by describing their kinematic model. The kinematic model provides the relationship between the vehicle velocities and commanded velocities. In this context, the kinematic models describing 2D and 3D motions are as follows.

1.2.1 Robot Model for 2D Motion

To develop robot model, let us consider a point mass moving in a planar environment. The plane is represented in X–Y coordinate frame with a fixed origin O as shown in Figure 1.2. Let the current position of the point representing the robot be

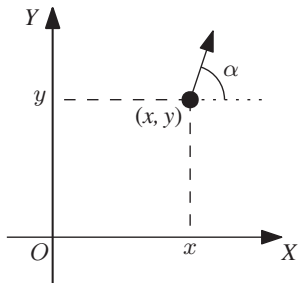


Figure 1.2 A point moving in planar environment

described by its coordinates (x, y) in the fixed X–Y frame. Since the point robot is a moving point, it further needs a description to show its current orientation. Let the current orientation of the point robot be α with respect to the X-axis as shown in Figure 1.2. Now, 2D motion of the point robot is described by $[\dot{x} \quad \dot{y} \quad \dot{\alpha}]^T$. In order to define the input–output relationship for controller design, the output is the 2D motion of the point robot while the input depends on the command given to the point robot. Accordingly the robot model is developed. Popular robot models and their descriptions are as follows:

1.2.1.1 Generic Model

Let the 2D motion of the robot be given by velocities in the body frame of reference X_b–Y_b, where X_b–Y_b frame is oriented in the forward direction of the robot (at an orientation α) with respect to the inertial frame of reference X–Y. In particular, the velocities are surge-forward v_x and sway-left v_y as shown in Figure 1.3.

Knowing that the rotation by angle α results in transforming a 2D point by matrix R_α, which is given by

$$R_\alpha = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix}$$

the kinematic model of generic 2D mobile robot is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = R_\alpha \begin{bmatrix} v_x \\ v_y \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} v_x \\ v_y \end{bmatrix} \tag{1.5}$$

While the generic model describes the commanded velocities in body frame of reference, the commanded velocities in 2D unicycle model are linear and angular velocities as described next.

1.2.1.2 Unicycle Model

Given that the commanded velocities are linear velocity v and angular velocity ω, the model is obtained by relating the angular velocity with change in orientation

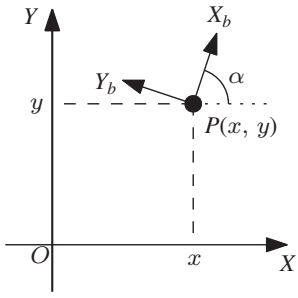


Figure 1.3 Generic 2D robot model

projecting linear velocity v on X and Y axes. These X- and Y-projections define the velocities $\dot{x}$ and $\dot{y}$, respectively. Therefore,

$$\begin{aligned} \dot{x} &= v\cos\alpha \\ \dot{y} &= v\sin\alpha \\ \dot{\alpha} &= \omega \end{aligned} \tag{1.6}$$

The models of many mobile robots are derived from the unicycle model given by (1.6). Some examples are as follows:

- Differential-Drive Mobile Robot (DDMR).
- Bicycle.
- Car-like Mobile Robot (CMR).

1.2.1.3 Differential-Drive Mobile Robot or DDMR

In order to give the linear and angular motions to the mobile robot, two wheels of the robot in DDMR configuration are independently driven. These independently driven wheels are fixed to the robot's body in the same orientation on a common axis with a center-point M as shown in Figure 1.4. Typically, there is a third caster wheel for balancing the robot which is not driven. Let v_L and v_R be the left and right wheel velocities, respectively. The wheels can move in forward as well as backward directions and, therefore, differential velocity renders the angular motion to the robot. For example, if the left and right wheels are driven by the same speed but in opposite direction, the DDMR rotates at the point M ideally without any linear velocity. Figure 1.5 explains the rotation of the robot in anticlockwise and clockwise directions. The axis of rotation is the center of common axis of two wheels (point M). For the anticlockwise rotation of the DDMR at point M (zero linear velocity), the left wheel is driven in forward direction while right wheel is driven in the reverse direction but with the same speed as that of left wheel as shown in Figure 1.5a. Similarly, Figure 1.5b illustrates the clockwise rotation of the DDMR at point M when the left and right wheels are driven with same speed V in reverse and forward directions, respectively.

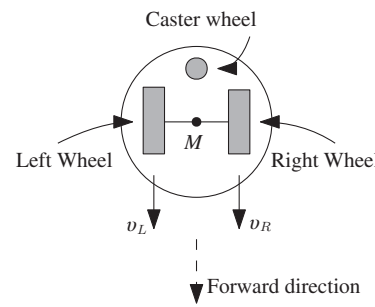
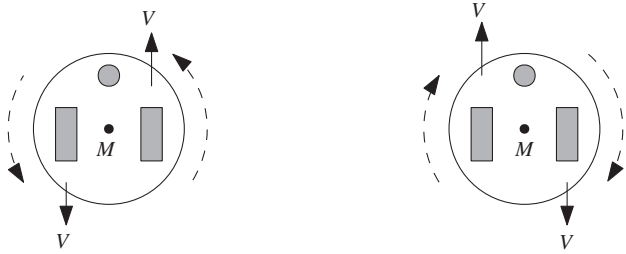


Figure 1.4 Differential drive mobile robot schematic illustrating wheel arrangement as seen from top



(a) Anticlockwise rotation ($v_L = V$ and $v_R = -V$)

(b) Clockwise rotation ($v_L = -V$ and $v_R = V$)

Figure 1.5 DDMR rotation when wheel velocities are same but in opposite direction

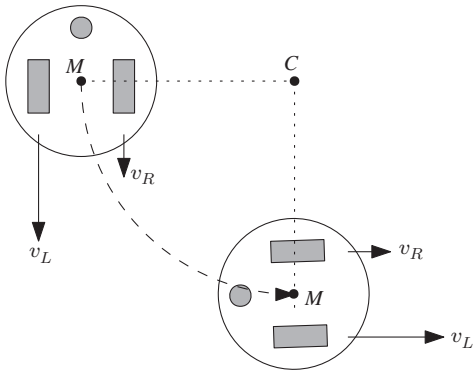


Figure 1.6 Circular motion of the DDMR

In order to further understand the motion of DDMR, let us consider giving differential speeds to two wheels. For example, the left and right wheel velocities are such that $|v_L| > |v_R|$, but in the same direction as shown in Figure 1.6. The DDMR moves in the circular path, center of this circle is point M which is not same as the center of axle joining the wheels.

Now, to formulate the movement of robot's body (linear and angular velocities) in terms of the left and right wheel velocities, we use conversion of angular velocity to linear velocity on the wheels. Let the distance between two wheels be D and the angular rotation of body be with reference to the point M as shown in Figure 1.7. The angular velocity of the robot's body, ω when projected on the left wheel, is $\omega D/2$ as the distance between point M and left wheel is $D/2$. This velocity component is in the forward direction of left wheel, while the direction is reverse when projected on the right wheel. Therefore, total left and right wheel velocities are

$$\begin{aligned} v_L &= v + \frac{\omega D}{2} \\ v_R &= v - \frac{\omega D}{2} \end{aligned} \tag{1.7}$$

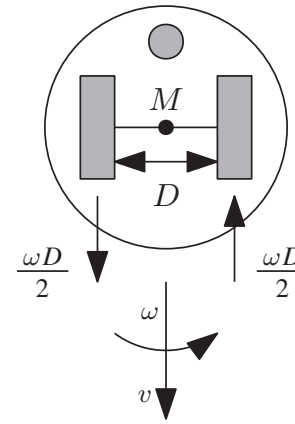


Figure 1.7 Illustration for deriving kinematic model of DDMR

Using (1.7), the linear and angular velocities of DDMR in terms of left and right wheel velocities are given by

$$\begin{aligned} v &= \frac{1}{2}(v_L + v_R) \\ \omega &= \frac{1}{D}(v_L - v_R) \end{aligned} \tag{1.8}$$

Now, linear and angular velocities obtained in (1.8) and using (1.6), the kinematic model of DDMR is given by

$$\begin{aligned} \dot{x} &= \frac{1}{2}(v_L + v_R)\cos\alpha \\ \dot{y} &= \frac{1}{2}(v_L + v_R)\sin\alpha \\ \dot{\alpha} &= \frac{1}{D}(v_L - v_R) \end{aligned} \tag{1.9}$$

The kinematic model as described by (1.9) renders the relationship between the robot's body motion $(\dot{x}, \dot{y}, \dot{\alpha})$ and the commanded velocities (v_L, v_R).

While providing linear motion through wheels of the robot is straightforward and intuitive specially when the wheel orientations are same with respect to the robot body. The differential drive mechanism provides angular motion by independently driving the two wheels. We next discuss the working of driving the robot by front wheel steering.

1.2.1.4 Front Wheel Steering Robot or FWSR

The Front Wheel Steering Robot or FWSR as the name suggests has a wheel or a pair of wheels in front that steers the robot left or right and provides angular motion to the robot. Either the front or rear wheels are driven to provide forward motion. Popular wheel configurations are shown in Figure 1.8. In case of car-like

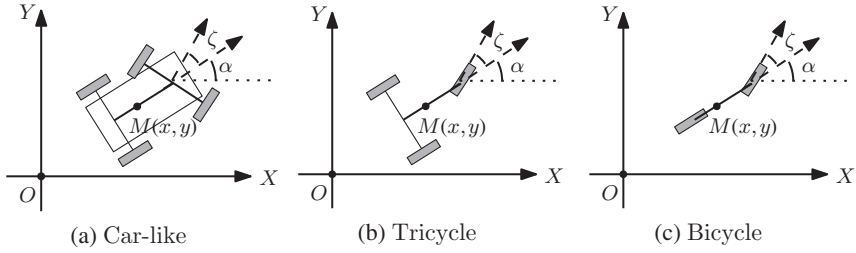


Figure 1.8 Schematics for front wheel steering with various wheel arrangements

(refer Figure 1.8a) robot, the number of wheels is 4 and the front pair of wheels are connected using axle that is steered. Similarly rear pair of wheels are connected using axle so that they are commanded with the same velocity. Similarly, in the Tricycle and Bicycle robots the front wheel is steered. The rear wheels in case of Tricycles again get same actuation. The pair of wheels provides balancing and higher payload capabilities. For simplicity, consider the reference point on the robot's body as any arbitrary point $M(x, y)$ on the line joining mid-points of front and rear wheel axles as shown in Figure 1.9.

Knowing that the pair of wheels either in front or rear are driven in by a same actuation, the kinematics is same for a virtual wheel in the middle of axle of these wheels representing the corresponding pair of wheels. Therefore, the kinematic model is derived using the bicycle configuration with two wheels, one in front and the other in rear. Let the steering angle of front wheel with respect to the body frame of reference be ζ. Similar to the approach for finding kinematic model of the DDMR, the relationship depends on the point of rotation. Figure 1.9 illustrates two cases based on driving wheel. In the first case (refer Figure 1.9a), the driving wheel is the front one, while in the second case (refer Figure 1.9b), the rear wheel is the

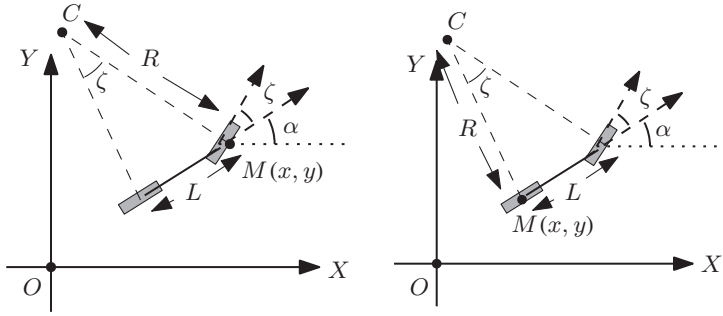


Figure 1.9 Illustrations for working principle of FWSR

driving one. Let the center of curvature be point C and radius of curvature be R. The reference point $M(x, y)$ is accordingly at the center of front or second wheel. It is clear that the angle made at the point C with the front and rear wheels is ζ.

For the case when the reference point $M(x, y)$ is at the front, we have

$$\sin\zeta = \frac{L}{R}$$

Let the front wheel velocity be v_w. Therefore, angular velocity of the FWSR is

$$\omega = \frac{v_w}{R} = \frac{v_w}{L}\sin\zeta \tag{1.10}$$

Similar to the unicycle model kinematic model, the X and Y projections of velocity vector are now given by

$$\begin{aligned}\dot{x} &= v_w\cos(\alpha+\zeta)\\ \dot{y} &= v_w\sin(\alpha+\zeta)\end{aligned} \tag{1.11}$$

From (1.11) and (1.10), the kinematic model of FWSR with front wheel driven velocity v_w is given by

$$\begin{aligned}\dot{x} &= v_w\cos(\alpha+\zeta)\\ \dot{y} &= v_w\sin(\alpha+\zeta)\\ \dot{\alpha} &= \omega = \frac{v_w}{L}\sin\zeta\end{aligned} \tag{1.12}$$

If the mechanism drives the steering velocity $\dot{\eta}$ as against steering angle η, the kinematic model will include the dynamics of steering angle as well. If the steering wheel velocity is ω_s, it is given by

$$\begin{aligned}\dot{x} &= v_w\cos(\alpha+\zeta)\\ \dot{y} &= v_w\sin(\alpha+\zeta)\\ \dot{\alpha} &= \omega = \frac{v_w}{L}\sin\zeta\\ \dot{\zeta} &= \omega_s\end{aligned} \tag{1.13}$$

Now, let us consider the case of reference point $M(x, y)$ at the rear as shown in Figure 1.9b. In this case, we have

$$\tan\zeta = \frac{L}{R}$$

and therefore, angular velocity of FWSR

$$\omega = \frac{v_w}{R} = \frac{v_w}{L}\tan\zeta \tag{1.14}$$

It is worth noting that the wheel velocity v_w now refers to the rear wheel velocity instead of front wheel velocity in (1.10). Referring to Figure 1.9b and (1.14), we get

the kinematic model of rear wheel driven FWSR

$$\begin{aligned}\dot{x} &= v_w \cos\alpha \\ \dot{y} &= v_w \sin\alpha \\ \dot{\theta} &= \frac{v_w}{L}\tan\zeta\end{aligned} \tag{1.15}$$

A specific case of $\zeta = \pi/2$ needs attention. We can see that $\dot{\alpha} = \infty$ for $\zeta = \pi/2$. This clearly shows that $\zeta = \pi/2$ case is to be avoided which is precisely the reason for fixing the maximum steering angle less than $\pi/2$ for four wheel automobiles. It is also useful to represent the kinematic models in the form of classical system representations and one of this representation is covered next.

1.2.1.5 Chained form of Unicycle

For the unicycle model described by (1.6), the state vector is $[x \quad y \quad \alpha]^T$ and input vector is $[v \quad \omega]^T$. Therefore, a conversion from unicycle model (1.6) to chained form (1.4) is obtained by a transformation developed in Murray and Sastry (1993) as in

$$\begin{aligned}&x_1 = x,\ x_2 = \tan\alpha,\ x_3 = y \\ &u_1 = v\cos\alpha,\ u_2 = \omega\sec^2\alpha\end{aligned}$$

Now, taking derivative of each state variable and using (1.6), we get

$$\dot{x}_1 = \dot{x} = v\cos\alpha = u_1$$

$$\dot{x}_2 = \sec^2\alpha\ \dot{\alpha}$$

Since $\dot{\alpha} = \omega$ from (1.6), we get

$$\dot{x}_2 = \omega\sec^2\alpha = u_2$$

Similarly,

$$\dot{x}_3 = \dot{y} = v\sin\alpha = v\cos\alpha\tan\alpha$$

But, $x_2 = \tan\alpha$ and $u_1 = v\cos\alpha$, we get

$$\dot{x}_3 = x_2 u_1$$

Similar transformations are applied to convert into other classical forms of system representations. However, it is clear that these transformations require trigonometric functions and sometimes exponential and logarithmic functions to be implemented in the embedded platform.

1.2.1.6 Single Integrator Model of Unicycle

Another common approach in the mobile robot literature (Glotfelter and Egerstedt, 2018) is to use a single integrator model for developing the complex

algorithms for single or multi-agent scenarios. This is due to the simple dynamics of the system which allows complex formulation of algorithms and their analysis. Let us understand the aspects behind representing a unicycle model as a single integrator. A single integrator system with states $x(t)$ and input $u(t)$ is given by

$$\dot{x}(t) = u(t) \tag{1.16}$$

Our aim is to explore the relation between the 2D unicycle position variables and the velocity inputs with a pair of single integrator and their corresponding inputs. Let us consider a point $\mathbf{x_d} = (x_d, y_d)$, d distance ahead ($d > 0$) of the position of the unicycle robot at (x, y) on the line along the heading direction. The point can be represented in terms of the state variables of unicycle robot as follows:

$$\begin{bmatrix} x_d \\ y_d \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d\cos\alpha \\ d\sin\alpha \end{bmatrix} \tag{1.17}$$

The dynamics of the point $\mathbf{x_d}$ corresponding to the dynamics of the robot is now given by

$$\begin{aligned} \begin{bmatrix} \dot{x}_d \\ \dot{y}_d \end{bmatrix} &= \begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} + \begin{bmatrix} -d\sin\alpha\dot{\alpha} \\ d\cos\alpha\dot{\alpha} \end{bmatrix} \\ &= \begin{bmatrix} v\cos\alpha \\ v\sin\alpha \end{bmatrix} + \begin{bmatrix} -d\omega\sin\alpha \\ d\omega\cos\alpha \end{bmatrix} \end{aligned} \tag{1.18}$$

Rewriting (1.18) as system model renders

$$\dot{\mathbf{x}}_\mathbf{d} = \mathbf{u_d} \tag{1.19}$$

where $\mathbf{u_d} = [u_{xd} \quad u_{yd}]$ and

$$u_{xd} = v\cos\alpha - d\omega\sin\alpha \tag{1.20}$$

$$u_{yd} = v\sin\alpha + d\omega\cos\alpha \tag{1.21}$$

The representation obtained in (1.19) is the single integrator form (refer (1.16)). Now, the inputs to the unicycle robot (v, ω) are transformed using $\mathbf{u_d}$ as follows:

$$\begin{bmatrix} v \\ \omega \end{bmatrix} = \begin{bmatrix} \cos\alpha - d\sin\alpha \\ \sin\alpha d\cos\alpha \end{bmatrix}^{-1} \begin{bmatrix} u_{xd} \\ u_{yd} \end{bmatrix} \tag{1.22}$$

The existence of inverse of the matrix used in (1.22) is given in Olfati-Saber (2002).

1.2.1.7 Discrete-time Unicycle Model

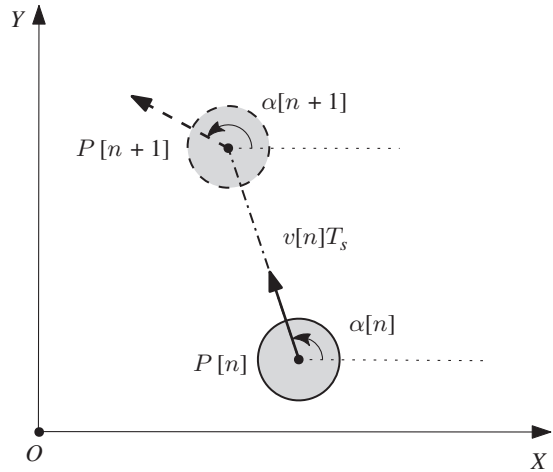
The implementation of the controllers on the models described above requires continuous monitoring of the system states and an uninterrupted update of the control command. But in practice, the monitoring and control of the system is done in a discrete manner. The system states are measured every sample time interval and control command is computed and updated in these sampling

instances in an ECS as shown in Figure 1.1. Thus, the design of the controllers for a practical system demands a discrete counterpart for the unicycle kinematics presented in (1.6). Let us formalize this by describing the robot states and control commands in discrete time steps. The states and input are assumed to be sampled and calculated every time interval, T_s. Let the states of the robot at nth time instance be $(x(nT_s), y(nT_s), \alpha(nT_s))$. The states at discrete instances are $P[n] = (x[n], y[n], \alpha[n])$. Similarly, let the input to the unicycle robot at nth instant be $(v[n], \omega[n])$. The states of the robot at the successive time instant are denoted by $P[n+1] = (x[n+1], y[n+1], \alpha[n+1])$. A direct approach for discretizing is to use the first-order approximation of Taylor series of the model given in (1.6) or forward difference method. The resultant equations will be as follows:

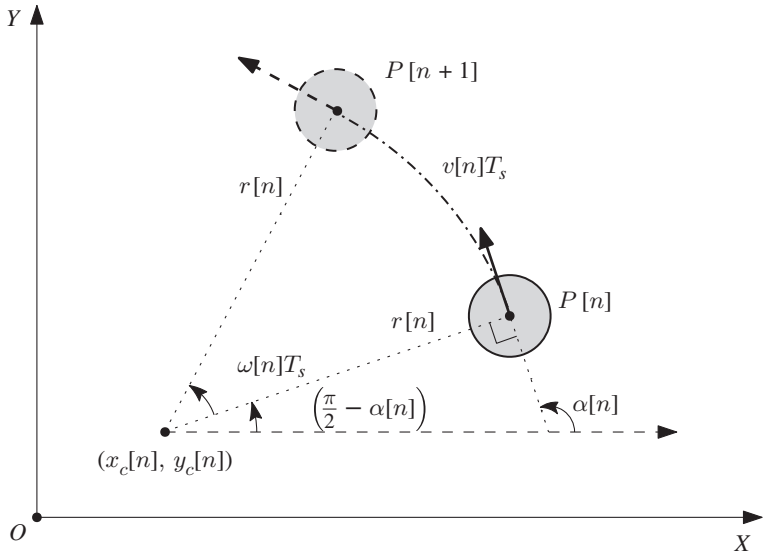
$$\begin{aligned} x[n+1] &= x[n] + T_s v[n] \cos(\alpha[n]) \\ y[n+1] &= y[n] + T_s v[n] \sin(\alpha[n]) \\ \alpha[n-1] &= \alpha[n] + T_s \omega[n] \end{aligned} \tag{1.23}$$

Now let us analyze this model further to understand the accuracy of this method of discretization. The inputs to the system are assumed to be constant during the time interval $(nT_s, (n+1)T_s]$. Figure 1.10a shows the evolution of the robot for two successive time instances, n and $n+1$, according to the equations given in (1.23). It can be observed that the robot is moving in a straight line for a distance of $v[n]T_s$ in the direction of $\alpha[n]$ and reaches $(x[n+1], y[n+1])$. Also, an instantaneous correction of the orientation is done at $(n+1)$th instant to modify $\alpha[n+1]$. This discretization does not consider the effect of rotation while moving from the location $(x[n], y[n])$ to $(x[n+1], y[n+1])$ and assumes $\omega[n] = 0$. Thus there is an error which accumulates in the difference equation model and may affect the performance of the implementation. Let us consider modifying the difference equation by taking into account the rotation of the unicycle robot during the time interval, $(nT_s, (n+1)T_s]$. The robot will trace an arc when the input to the system, $(v[n], \omega[n])$ is kept constant during the time interval $(nT_s, (n+1)T_s]$. The updated position in the successive time instant is to be calculated considering the circular motion of the robot. The radius of the circle which contains the path of the robot is given by $r[n] = v[n]/\omega[n]$. Referring to Figure 1.10b, the initial position of the robot at nth instant is marked $P[n]$. The circular path taken by the robot during the interval is shown with dash-dotted lines and the final position is marked with $P[n+1]$. The coordinates of the center of the circle $\left(x_c[n], y_c[n]\right)$ corresponding to the circular path with constant radius $r[n]$ is obtained for nth sample time instance as follows:

$$\begin{aligned} x_c[n] &= x[n] - r[n] \sin(\alpha[n]) \\ y_c[n] &= y[n] - r[n] \cos(\alpha[n]) \end{aligned} \tag{1.24}$$



(a) Movement of robot according to forward difference method



(b) Movement of robot in exact discretization

Figure 1.10 Geometric interpretation of different discretization methods

Further, the coordinates of the robot position at $(n+1)$th instant are now given by

$$\begin{aligned} x[n+1] &= x_c[n] + r[n]\sin\left(\alpha[n] + \omega[n]T_s\right) \\ y[n+1] &= y_c[n] - r[n]\cos\left(\alpha[n] + \omega[n]T_s\right) \end{aligned} \tag{1.25}$$

On substituting the equations for the coordinates $\left(x_c[n], y_c[n]\right)$ given by (1.24) in (1.25), the discrete equations for unicycle robots are obtained as follows:

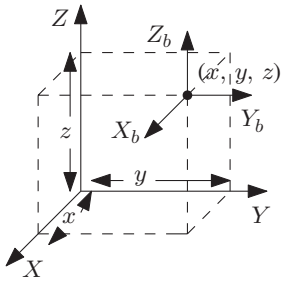
$$\begin{aligned} x[n+1] &= x[n] - r[n]\sin\left(\alpha[n]\right) + r[n]\sin\left(\alpha[n] + \omega[n]T_s\right) \\ y[n+1] &= y[n] - r[n]\cos\left(\alpha[n]\right) - r[n]\cos\left(\alpha[n] + \omega[n]T_s\right) \\ \alpha[n-1] &= \alpha[n] + T_s\omega[n] \end{aligned} \tag{1.26}$$

It is worth noting that the equations in (1.26) will reduce to the equations given in (1.23) in the limiting case where $\omega[n] = 0$. These sets of equations are very useful while designing controllers or while using the model for estimating the states as given in Thrun et al. (2005).

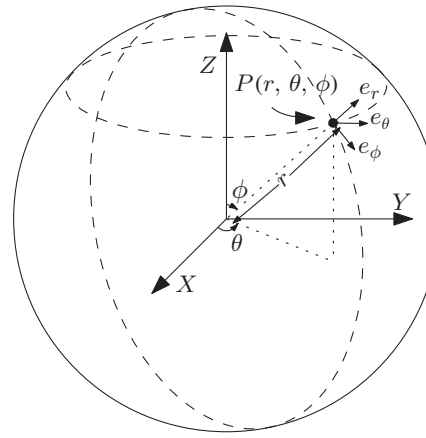
1.2.2 Robot Model for 3D Motion

The extension from 2D to 3D is not straightforward due to increase in state variables. While 2D motion can be described by two position coordinates and an orientation, the 3D motion in its most general form is described by three position coordinates and three orientations. The position coordinates can be represented in cartesian (x, y, z), spherical (R, θ, ϕ), or cylindrical (ρ, ϕ, z) as shown in Figure 1.11. Suitable transformations between the coordinate systems are well-known. They are used when the system dynamics require change in representation. In particular, sometimes the problem being addressed can be well represented in spherical coordinates as compared to cartesian or vice versa. Accordingly, the system description is obtained using transformations to convert the position representations. Figure 1.11a also shows body frame of reference X_b–Y_b–Z_b. This body frame of reference is in the direction of inertial frame of reference (X–Y–Z). In particular, the vehicle (or 3D rigid body) located at a point $P(x, y, z)$ in Figure 1.11a has orientation same as that of inertial. The unit directions in the corresponding inertial frame of direction for spherical and cylindrical coordinate system are shown as e_r–e_θ–e_ϕ and e_ρ–e_θ–e_z, respectively.

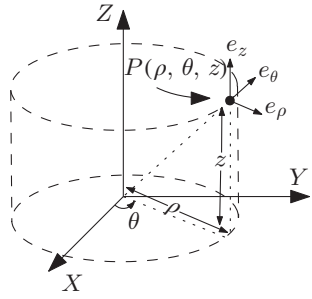
The orientation of the robot in 3D (or the orientation of the body frame) with respect to an inertial frame can be represented using three composed rotations. The angles of rotations are called Euler angles. There exist many representations for orientation in 3D using Euler angles depending on the choice of the axis of rotation such as XYZ, ZYX, ZYZ, and ZXZ. The axis of rotations can be of inertial frame or of the body frame. For example, $Z_bY_bX_b$ Euler angles also known as Yaw–Pitch–Roll rotations or Tait–Bryan angles are most popular in 3D vehicle descriptions. Henceforth we follow Yaw–Pitch–Roll convention of angles to



(a) Cartesian coordinates



(b) Spherical coordinates

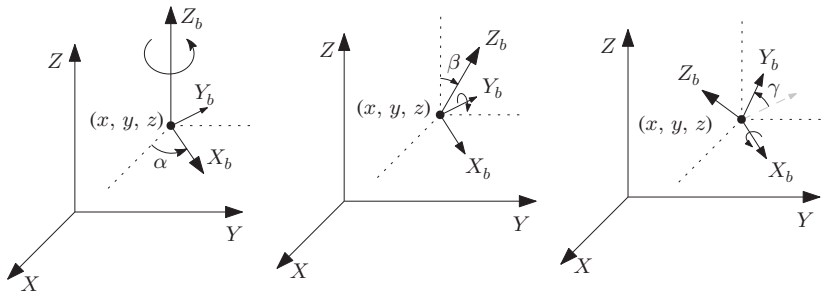


(c) Cylindrical coordinates

Figure 1.11 Typical 3D position representations

represent the orientation. A generic rotation of body frame in 3D that can be represented using Euler angles which are yaw, pitch, and roll is shown in Figure 1.12. Refer to the rotation of body frame of reference in Figure 1.12. The rotation about Z_b-axis is the yaw (α), while rotations about Y_b-axis and X_b-axis are pitch (β) and roll (γ) respectively. Also, note that the order of the rotations is important and not interchangeable. The transformation matrix for each of the rotations from inertial frame to body frame is defined by elementary rotations along each of the axes. Following are the transformation matrices corresponding to yaw, pitch, and roll with respect to Z_b, Y_b, and X_b axes, respectively, by yaw angle α, pitch angle β, and roll angle γ.

$$\mathcal{R}_{Z_b,\alpha} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{1.27}$$



(a) Rotate along Z-axis, Yaw angle α

(b) Rotate along Y-axis, Pitch angle β

(c) Rotate along X-axis, Roll angle γ

Figure 1.12 Illustration of 3D orientations

$$\mathcal{R}_{Y_b,\beta} = \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \tag{1.28}$$

$$\mathcal{R}_{X_b,\gamma} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix} \tag{1.29}$$

Since these transformations are rotation matrices, they are orthogonal and satisfy the following:

$$\mathcal{R}\mathcal{R}^T = \mathcal{R}^T\mathcal{R} = I; \quad det\ \mathcal{R} = 1 \implies \mathcal{R}^{-1} = \mathcal{R}^T \tag{1.30}$$

Now, if the surge, sway, and roll velocities in body frame are v_x, v_y, and v_z, respectively, then

$$\begin{aligned} \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \end{bmatrix} &= \left(\mathcal{R}_{X_b,\gamma}\mathcal{R}_{Y_b,\beta}\mathcal{R}_{Z_b,\alpha}\right)^{-1} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \\ &= \mathcal{R}^{-1}_{Z_b,\alpha}\mathcal{R}^{-1}_{Y_b,\beta}\mathcal{R}^{-1}_{X_b,\gamma} \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \end{aligned} \tag{1.31}$$

The transformation matrix $\mathcal{R}^{-1}_{Z_b,\alpha}\mathcal{R}^{-1}_{Y_b,\beta}\mathcal{R}^{-1}_{X_b,\gamma}$ is calculated as follows:

$$\begin{aligned} \mathcal{R}^{-1}_{Z_b,\alpha}\mathcal{R}^{-1}_{Y_b,\beta}\mathcal{R}^{-1}_{X_b,\gamma} &= \begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\beta & 0 & \sin\beta \\ 0 & 1 & 0 \\ -\sin\beta & 0 & \cos\beta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma & \cos\gamma \end{bmatrix} \\ &= \begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma \\ s_\alpha c_\beta & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma \end{bmatrix} \end{aligned} \tag{1.32}$$

where s_* and c_* represent $\sin(*)$ and $\cos(*)$, respectively. In order to map the 3D angular velocities of the vehicle ω_x, ω_y, and ω_z with the time derivatives of yaw–pitch–roll $\dot{\alpha}$, $\dot{\beta}$, and $\dot{\gamma}$, we consider a map $E(\alpha, \beta, \gamma)$ that maps 3D angular velocities to time derivatives of roll-pitch–yaw. In particular, we consider

$$\begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = E(\alpha, \beta, \gamma) \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix}$$

The Euler angle rates are related to the 3D angular velocities of the vehicle in the body frame under the following transformation.

$$\begin{aligned} \begin{bmatrix} \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} &= \begin{bmatrix} \dot{\gamma} \\ 0 \\ 0 \end{bmatrix} + T_{X_b,\gamma} \begin{bmatrix} 0 \\ \dot{\beta} \\ 0 \end{bmatrix} + T_{X_b,\gamma} T_{Y_b,\beta} \begin{bmatrix} 0 \\ 0 \\ \dot{\alpha} \end{bmatrix} \\ &= \begin{bmatrix} 1 & 0 & -\sin\beta \\ 0 & \cos\gamma & \cos\beta\sin\gamma \\ 0 & -\sin\gamma & \cos\beta\cos\gamma \end{bmatrix} \begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} \\ &= E^{-1}(\alpha, \beta, \gamma) \begin{bmatrix} \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} \end{aligned} \tag{1.33}$$

Further E is computed from (1.33) as follows.

$$E(\alpha, \beta, \gamma) = \begin{bmatrix} 1 & \sin\gamma\tan\beta & \cos\gamma\tan\beta \\ 0 & \cos\gamma & -\sin\gamma \\ 0 & \sin\gamma/\cos\beta & \cos\gamma/\cos\beta \end{bmatrix} \tag{1.34}$$

In summary, the kinematic model of the 3D vehicular motion is given by

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\gamma} \\ \dot{\beta} \\ \dot{\alpha} \end{bmatrix} = \begin{bmatrix} c_\alpha c_\beta & c_\alpha s_\beta s_\gamma - s_\alpha c_\gamma & c_\alpha s_\beta c_\gamma + s_\alpha s_\gamma & 0 & 0 & 0 \\ s_\alpha c_\beta & s_\alpha s_\beta s_\gamma + c_\alpha c_\gamma & s_\alpha s_\beta c_\gamma - c_\alpha s_\gamma & 0 & 0 & 0 \\ -s_\beta & c_\beta s_\gamma & c_\beta c_\gamma & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & s_\gamma t_\beta & c_\gamma t_\beta \\ 0 & 0 & 0 & 0 & c_\gamma & -s_\gamma \\ 0 & 0 & 0 & 0 & s_\gamma/c_\beta & c_\gamma/c_\beta \end{bmatrix} \begin{bmatrix} v_x \\ v_y \\ v_z \\ \omega_x \\ \omega_y \\ \omega_z \end{bmatrix} \tag{1.35}$$

Now, to generate the linear and angular velocities of a 3D vehicle, there exist various configurations. Two of these configurations are discussed next. The first configuration covers an aerial vehicle design, while the second configuration covers an underwater vehicle.

1.2.2.1 Quadcopter – An Aerial Vehicle

The quadcopter as the name suggests has four propellers mounted on the corners of a virtual square. The diagonals of this virtual square are connected as a base. A rough schematic of a typical quadcopter is shown in Figure 1.13. Four propellers are placed in front, right, rear, and left. Let the angular speeds of front, right, rear, and left propellers be Ω_1, Ω_2, Ω_3, and Ω_4, respectively. The thrust generated through a propeller is proportional to square of the corresponding angular speed. Hence, the upward thrust (in Z_b direction) of the quadcopter τ_z is given by

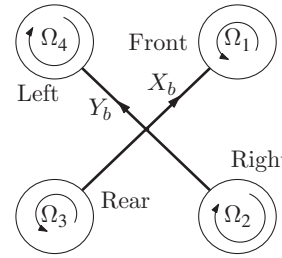


Figure 1.13 Schematic of a quadcopter

$$\tau_z = k_1(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{1.36}$$

where k_1 is a proportionality constant with appropriate dimensions. Now, the thrust in yaw (angular orientation around Z_b axis) is due to the differential thrust generated by left and right propellers, or the thrust in yaw direction is given by

$$\tau_\alpha = k_2(-\Omega_2^2 + \Omega_4^2) \tag{1.37}$$

Similarly, thrust generated in pitch and roll directions is given by

$$\tau_\beta = k_3(-\Omega_1^2 + \Omega_3^2) \tag{1.38}$$

$$\tau_\gamma = k_4(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \tag{1.39}$$

Note that the proportionality constants k_2, k_3, and k_4 have appropriate dimensions.

Now, if the state vector of a 3D vehicle is given by $X = [x \;\; y \;\; z \;\; \alpha \;\; \beta \;\; \gamma]^T$. It is clear from (1.36)–(1.39), the actuation (input vector) to the quadcopter is $[\tau_z \;\; \tau_\alpha \;\; \tau_\beta \;\; \tau_\gamma]^T$. We know that the thrust is proportional to the acceleration. Moreover, the vertical thrust τ_z is responsible for providing accelerations in X_b–Y_b–Z_b directions depending on the instantaneous pitch β and roll γ. Therefore, the relation of thrusts with corresponding accelerations and appropriate proportionality constants is described by

$$\begin{bmatrix}\tau_z & \tau_\alpha & \tau_\beta & \tau_\gamma\end{bmatrix}^T = \begin{bmatrix}k_1' g(\dot{v}_x, \dot{v}_y, \dot{v}_z) & k_2'\dot{\omega}_\alpha & k_3'\dot{\omega}_\beta & k_4'\dot{\omega}_\gamma\end{bmatrix}^T$$

This way the quadcopter model is represented as a double integrator model as described by (1.3).

1.2.2.2 Six-Thrusters Configuration – An Underwater Vehicle

As the name suggests, the vehicle has six thrusters to provide control on surge (X_b), sway (Y_b), heave (Z_b), yaw (α), and pitch (β). The underwater world is upside down; hence, the body frame of reference (X_b–Y_b–Z_b) shows Z_b in the downward direction. The configuration in which they are placed, provides the capability for

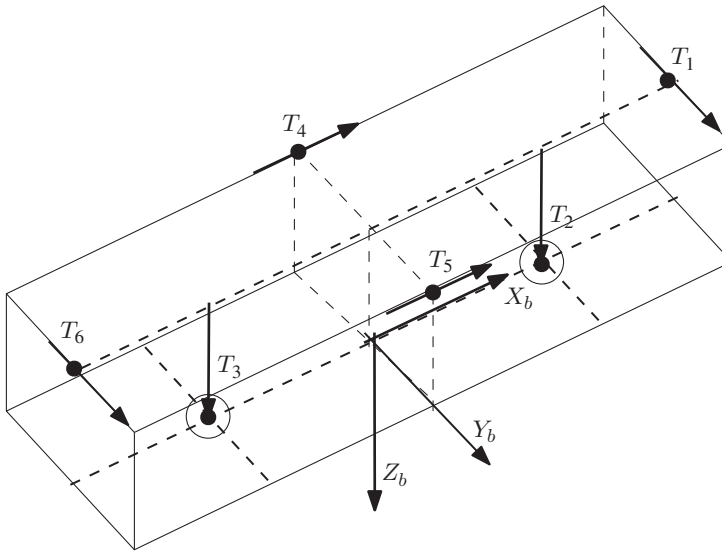


Figure 1.14 Schematic of six-thrusters configuration; $T_1 - T_6$ show typical placement and directions of six thrusters

such a control. A schematic of six-thrusters configuration is shown in Figure 1.14. The six thrusters shown using T_1 to T_6 present a typical positioning and direction of corresponding thrusters. As in differential drive vehicle, two actuators (motors) provide the forward and yaw motion, the thrusters T_4 and T_5 provide surge and yaw motions to the 3D underwater vehicle. Likewise, T_1 and T_6 provide sway and yaw motions, while T_2 and T_3 provide heave and pitch motions. Therefore, if the corresponding actuation velocities of thrusters are $\Omega_1 - \Omega_6$, the thrusts $\tau_x, \tau_y, \tau_z, \tau_\alpha$, and τ_β in X_b, Y_b, Z_b, α, and β directions are given by

$$\begin{aligned}
\tau_x &= k_1(\Omega_4^2 + \Omega_5^2) \\
\tau_y &= k_2(\Omega_1^2 + \Omega_6^2) \\
\tau_z &= k_3(\Omega_2^2 + \Omega_3^2) \\
\tau_\alpha &= k_4(\Omega_4^2 - \Omega_5^2) + (\Omega_1^2 - \Omega_6^2) \\
\tau_\beta &= k_5(\Omega_3^2 - \Omega_2^2)
\end{aligned} \tag{1.40}$$

The proportionality constants k_1–k_5 are appropriately dimensioned and used here to show the relationship between the thrusters' actuation and thrusts generated in various directions.

With this background of various models for 2D and 3D mobile robots, it is important to understand the perspective of emerging embedded technology. The next section presents various aspects of latest embedded technology.

1.3 Embedded Technology

An embedded controller computes a dedicated function, with real-time computing constraints. The controller output can also be generated using general-purpose computing machines which may overcome real-time computing constraints as well; however, the undesirable architecture features and resources consume considerable power. This power consumption is at least 20 times more than the power required for computing dedicated function. The embedded platforms provide a cost-effective solution with low-power consumption and size. It also facilitates easy installation. Typically, an embedded controller is designed to provide a customized solution for performing a specific task and its objective is to consume just sufficient embedded resources according to its needs. In particular, while designing the embedded controllers, the objective is not to use extra peripheral/resource other than the ones available on the selected embedded platform. A holistic approach of application-oriented controller implementation has a major challenge in designing the embedded controller maintaining the trade-off between the limitations of embedded computing and combined objectives to satisfy size, power, cost, cooling, and weight requirements. The selection of embedded platform decides the constraints on developing controller. The controller may also need to operate in extreme operating conditions such as vibration, shock, extreme heat and cold, or radiation. These operating conditions need to be considered while selecting an embedded platform.

The challenge in designing embedded controller is to optimize the design metrics. Following are a few common design parameters:

- *NRE* (*Non-Recurrent Engineering*) *cost*: A one-time design cost involves design iterations. The design may undergo various iterations where embedded controller design and its interface may require redesigning hardware as well as software.
- *Unit cost*: This refers to the per unit cost after designing.
- *Size*: The spatial area occupied by the controller.
- *Performance*: This is mainly weighed in terms of the response time (latency) and throughput of the controller. The latency is defined as the time required for sensing, executing the controller functionality, and actuating. Throughput is defined as the number of controller outputs generated per unit time. Therefore, latency and throughput are reciprocal of each other if a single processor without pipelining is being used. For example, if the latency of a controller is 0.1 second, then the controller output is generated 10 times per second or throughput of the controller is 10 control outputs/s.
- *Power consumption*: The battery life and the cooling requirements are dictated by the total power consumption of the design.

- *Flexibility*: This refers to the ability to change functionality without incurring heavy NRE cost.
- *Time-to-market*: This includes time to design, test, and manufacture.
- *Safety*: The design must be able to operate in the specified operating conditions.
- *Correctness*: This reflects confidence in the design.

In a typical scenario, a trade-off between power, size, performance, and NRE cost is required to be achieved. The effort in reducing the power consumption usually increases the size, performance, and subsequently, NRE cost. An optimal design for embedded controller is achieved by understanding different technologies available in embedded platforms. An in-depth understanding is not required, but a fair idea of these technologies helps in selecting an appropriate embedded platform. A comprehensive idea of these technologies is covered from the perspectives of processor and Integrated Circuit (IC) technologies. These technologies cover commercially available embedded platforms used in mobile robots like Microcontroller, FPGA, Digital Signal Processor (DSP), etc.

1.3.1 Processor Technology

In this technology, architecture of the selected processor dictates the functionality of the controller. This can be classified in following three categories:

1. *General-purpose processor*: The controller functionality is implemented through software. The evaluation of design parameters is as follows:
   - The NRE cost is low (desirable feature)
   - The design is flexible (desirable feature)
   - Power consumption is high (undesirable feature)
   - Occupies a larger area (undesirable feature).
2. *Single-purpose processor*: The processor is designed to generate a controller output and results in a system-on-chip solution. The evaluation of design parameters is as follows:
   - It achieves low latency or high throughput (desirable feature)
   - It occupies smaller area and consumes low power (desirable feature)
   - It typically achieves low unit cost (desirable feature)
   - The design takes longer time to optimize; therefore, NRE cost is usually high (undesirable feature)
   - The design is not flexible (undesirable feature).
3. *Application-specific processor*: These processors are designed for solving a class of problems, e.g.; embedded control, image processing, and telecommunication. Microcontroller technology is most appropriate for designing embedded controllers. The evaluation of design parameters is as follows:
   - It achieves considerably low latency or high throughput

- It occupies smaller area and consumes low power
- It typically achieves low unit cost
- The design does not take longer time to optimize; therefore, NRE cost is moderate
- The design is not completely flexible, but has a flexibility to modify the controller parameters and functionality by reprogramming the processor.

1.3.2 IC Technology

The ICs technology is independent of the processor technology. This technology refers to layers created over semiconductor and designing appropriate masks for creating these layers. These sets of masks define the layout of the design. Figure 1.15 shows different layers. The bottom-most is the transistor layer. A set of transistors connected in a predefined manner provides the functionality of different logic components. This layer is referred to as *logic component* layer. The interconnections of logic components form the *wire* layer. The design is complete once the *wire* layer is designed. Classification of the embedded system is also based on these layer formations. The classification is as follows:

1. *Full custom/VLSI* (*Very Large-Scale Integration*): This technology optimizes all the layers. In particular, the system is designed at the transistor level. The evaluation of design parameters is as follows:
   - It requires longer designing time resulting in high NRE cost
   - It results in small size and low power requirements.

   This technology is usually preferred in extremely performance-critical conditions with high production requirement.
2. *Semiconductor ASIC (Application-Specific Integrated Circuit)*: The system is designed at middle layer of logic components. The transistor layer is already built and fixed. The evaluation of design metric is as follows:
   - It achieves considerably good performance
   - It results in small size and lesser NRE cost as compared to VLSI technology.
3. *Programmable Logic Devices* (*PLDs*): The transistor and logic connector level layers are fixed in this technology. It creates or destroys connections between the wires that connect gates by programming the PLD. The popular PLDs in

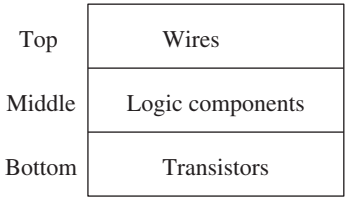


Figure 1.15 Various layers of IC technology

market are PLA (Programmable Logic Array), FPGA, and CPLD(Complex Programmable Logic Array). The evaluation of design parameters is as follows:

- It results in low NRE cost, but high unit cost
- It results in bigger size and higher power consumption than that achieved by the ASIC technology
- It achieves reasonable performance.

1.4 Commercially Available Embedded Processors

There are many popular commercially available embedded processors based on the processor and IC technologies that are being used in the embedded control designs. A brief idea on these commercial embedded platforms is needed prior to designing the embedded controllers. Learning the features of the architecture would provide understanding of the capabilities and limitations that is useful in selecting the embedded platform for the controller design and implementation. The categorization of processor with their commercial names is presented next.

1.4.1 Microprocessor

The general-purpose technology used for microprocessor includes basic architecture elements like arithmetic and Logic Unit (ALU), internal registers and memory and I/O (Input/Output) port interfaces. The processor is selected based on the computation power and interface capabilities with peripherals in terms of its data and address lines. For example, a 16-bit microprocessor like 8086 has 16-bit internal registers, 16-bit ALU, and 16-bit external data bus, but 20-bit address lines. The latest processors like Intel i7 also has multiple cores for creating parallel threads, and complex computations for optimal and nonlinear controls can then be possible with real-time performance. However, the general-purpose processor lacks customization capabilities and typically is not very suitable for embedded control designs. Many popular processors have been used in control designs. Popular choices in this category are Intel Core i7, AMD Athlon X7, Intel 8085, Intel 8086, ARM NEON, Zilog Z80, etc.

The latest category of microprocessors provide customized solutions through embedded Linux. The Linux kernel can be customized for the controller applications and thus can be light-weight. Available processors in the market under this category are ARM Cortex series, XScale PXA series, MIPS MSP series, AT91 by ATmel, PowerPC, Freescale MPC5200, etc. This choice of embedded platform facilitates easier interfacing and real-time performance, but requires specialized coding skills.

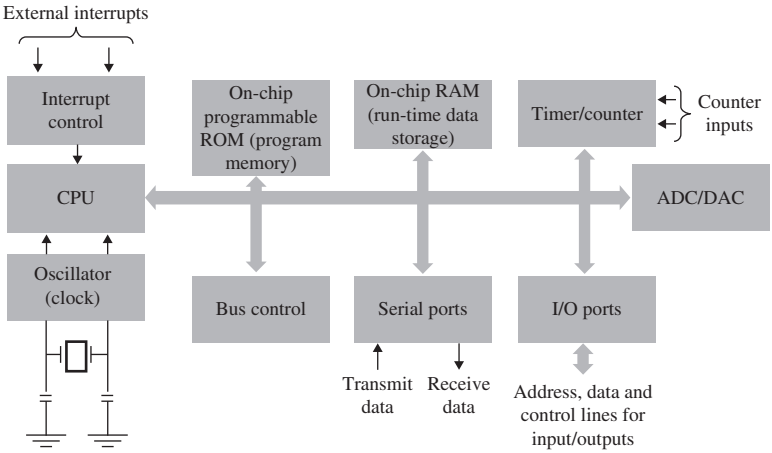


Figure 1.16 Block diagram of a typical microcontroller

1.4.2 Microcontroller

Microcontroller provides a customized solution as compared to the microprocessor that requires external peripherals like memory, I/O ports, timers, etc. A typical microcontroller has a CPU with program and data memory, I/O port interfaces, ADC/DAC, timer(s) that can be used as counter(s) and interrupt handling unit. A block diagram of a typical microcontroller is shown in Figure 1.16. With a few peripherals on chip, the microcontroller can be a standalone solution for controller implementations in many applications. The program memory can support the control-law, and ADC/DAC with I/O ports can provide interface to actuators and sensors. The computation capabilities are limited by the sizes of data registers. For example, an 8-bit microcontroller can perform 8-bit operations efficiently, but the performance degrades with computations requiring complex requiring floating-point operations. Similarly, program memory limits the code length corresponding to the control-law implementation.

Popular microcontrollers in the market are Altera Nios-16 bit, Intel 8051, Atmel AT89 series (Intel 8051 architecture), Atmel ATxmega series (AVR architecture), Cypress PSoC series (ARM Cortex), Freescale (ARM architecture), etc.

1.4.3 Field Programmable Gate Arrays (FPGA)

The FPGAs support system-on-chip design by facilitating hardware reconfigurable facility. It is easy to understand the reconfigurability of FPGAs by learning its architecture features. Figure 1.17 shows basic building blocks of FPGA

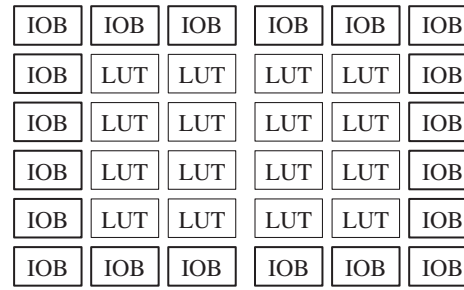


Figure 1.17 Typical architecture layout of an FPGA; LUT stands for look-up table and IOB stands for input–output block

architecture. Basic building blocks are Look-Up Tables (LUTs) at the center and Input–Output Blocks (IOBs) at the periphery. The LUTs are programmable to contain logic. The IOBs interface with sensors/actuators and programmable to act as input or output or both. Each FPGA has many LUTs and IOBs. For example, the smallest FPGA chip in Xilinx Spartan 3E family, XC3S100E has 2160 logic cells and 108 user IO pins. The programmable LUTs and IOBs provide support for hardware reconfigurability of FPGAs. The FPGAs have been widely used in designing hardware accelerators by exploiting the parallel processing in hardware. Any hardware module (programmed through LUTs) can be copied multiple times to create parallel functionality in FPGA for real-time performance. Likewise, parallel processing of multiple sensors and actuators (may not be of same kind) is easily reconfigured in FPGA avoiding time delays in servicing events generated by respective input and/or output device.

Popular choices are Xilinx Spartan series, Xilinx Vertex series, Altera Cyclone-V, Altera Stratix-V, etc. For the controller design perspective, FPGA is a good choice when there is a need of (i) real-time applications, (ii) I/O requirement is high, and/or (iii) taking benefits of parallel processing.

1.4.4 Digital Signal Processor

The DSPs are specifically designed for the needs of signal processing using specialized microprocessor architecture. The architecture comprises large data paths and memory with facility of matrix manipulations. The DSPs in control systems are typically used to deal with large amount of sensor data and specialized needs on processing the data to provide tangible feedback to the controller.

A few popular DSPs are Texas Instruments TMS320 series and C6000, Blackfin family, Embedded general purpose processor like OMAP3 includes ARM Cortex-A8 and C6000. The high-level programming environments are available that use “C-language” like constructs for designing algorithms using DSP.

1.5 Notes and Further Readings

The basic concepts on mobile robotics covered here are to connect with the control theory. Various kinematic models used with 2D and 3D mobile robots are expressed as system representations. The suggested reading on the detailed kinematic and dynamic models of mobile robots is Siciliano and Khatib (2016) or Fossen (1994). One may refer to Olfati-Saber (2002) for further reading on unicycle model to single integrator. The reader is encouraged to develop system representations defining states, inputs, outputs, and relations using dynamic and kinematic models. For the embedded controller design, the basics of embedded processor technology are presented for a naive person to select the embedded processor from the design perspective. Suggestions for further reading on embedded systems are Givargis and Vahid (2012) and Wescott (2006).

There has been a growing interest in learning embedded control concepts. Demonstrating the embedded control concept with application benefits has always been an effective course delivery method. The concepts developed in this chapter can supplement developing course modules on "Embedded Control for Mobile Robotics." There are many concepts on developing embedded control laboratory for giving hands-on experience on embedded platforms like microcontroller (Moallem, 2004) and Real-Time Embedded Linux (Martí et al., 2010). The control education with a low-cost solution using real-time execution of control-law (Krauss and Croxell, 2012) uses PC for the calculations and the interfaced microcontroller for the execution of control law in real time and analog-to-digital/digital-to-analog conversions avoiding the need of real-time operating system.

Any embedded implementation is in discrete-time or event-driven. The computations are based on the system clock and require designing of digital logic. Once the digital logic is designed, the effects of quantization and processing delay in sequential logic account for the performance of control loop. While the accuracy and resolution of computations depend on the quantization, the control loop time depends on the processing and sampling times. Therefore, these effects limit the faithful implementation of designed controller.

As explained in Section 1.1, the controller is implemented on an embedded platform while the system is typically continuous-time in nature. The controller design in discrete-time is covered next to understand the interfacing with the analog world and embedded implementation.