Chapter 2

Theoretical Backgrounds

In this chapter, theoretical backgrounds, to support implementation of robot control system, are presented. There are three main theoretical backgrounds: Fuzzy Logic, Transformation Matrix, and ERSP Robot Scorpion.

2.1 Fuzzy Logic

Fuzzy logic system was first proposed by an American professor, Lotfi A. Zadeh, in [Zadeh, 1965]. Zadeh showed that fuzzy logic unlike classical logic can realize values between false (0) and true (1). Basically, he transformed the crisp set into the continuous set. Fuzzy sets thus have movable boundaries, i.e., the elements of such sets not only represent true or false values but also represent the degree of truth or degree of falseness for each input.

Fuzzy logic is the part of artificial intelligence or machine learning which interprets a human's actions. Computers can interpret only true or false values but a human being can reason the degree of truth or degree of falseness. Fuzzy models interpret the human actions and are also called intelligent systems.

2.1.1 Fuzzy Sets

Unlike classical set theory that classifies the elements of the set into crisp set, fuzzy set has an ability to classify elements into a continuous set using the concept of degree of membership. The characteristic function or membership function not only gives 0 or 1 but can also give values between 0 and 1.

Consider the outside ambient temperature. Classical set theory can only classify the temperature as hot or cold (i.e., either 1 or 0). It cannot interpret the temperature between 20 °F and 100 °F. In other words, the characteristic function for the classical logic is given by:

$$\mu(x) = \begin{cases} 1 & \text{if } x \geq 50°F \quad \text{classifies as hot} \\ 0 & \text{if } x < 50°F \quad \text{classifies as cold} \end{cases} \tag{2.1}$$

The boundary 50 °F is taken because classical logic cannot interpret intermediate values. On the other hand, fuzzy logic solves the above problem with a membership function as given by:

$$\mu(x) = \begin{cases} 0 & \text{if} \quad x < 20°F \\ \frac{x-20}{80} & \text{if} \quad 20°F \leq x \geq 100°F \\ 1 & \text{if} \quad x > 50°F \end{cases} \tag{2.2}$$

A graph of the membership function for the fuzzy temperature variable is shown in Figure 2.1. The degree of coldness is taken as the complement of the degree of hotness.

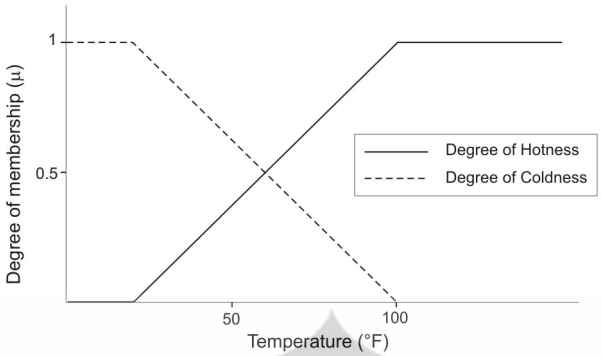

Figure 2.1: Fuzzy Sets Membership

2.1.2 Fuzzy Rule

Fuzzy rules are linguistic IF-THEN- constructions that have the general form 'IF A THEN B' where A and B are (collections of) propositions containing linguistic variables. A is called the premise and B is the consequence of the rule. In effect, the use of linguistic variables and fuzzy IF-THEN- rules exploits the tolerance for imprecision and uncertainty. In this respect, fuzzy logic mimics the crucial ability of the human mind to summarize data and focus on decision-relevant information. The fuzzy rule is written as:

If <fuzzy proposition> then <fuzzy proposition>

A fuzzy proposition can be an atomic or compound sentence. For example, 'Temperature is hot' is an atomic fuzzy proposition. 'Temperature is hot and humidity is low' is a compound fuzzy proposition.

In fuzzy rule, the linguistic proposition is derived from human perception, an assessment of a physical condition that is not measured with precision, but is assigned an intuitive value. For examples, temperature measured by a temperature transducer can be classified as 'cold', 'warm' and 'hot'. These classifications depend entirely on human perception. Perception from one human being to another human being might be different, thus these are called fuzzy human perception. A fuzzy rule system, incorporating above proposition example, can be defined as:

If temperature is cold then turn on heater
If temperature is hot then turn off heater
If temperature is warm then do nothing

It can be easily determined, based on human perception, that objective of above example system is to maintain the room temperature to be at 'warm' condition.

2.1.3 Defuzzification Technique

The input to the fuzzy system is a scalar value that is fuzzified. The set of rules is applied to the fuzzified input. The output of each rule is fuzzy. These fuzzy outputs

need to be converted into a scalar output quantity so that the nature of the action to be performed can be determined by the system. The process of converting the fuzzy output is called defuzzification. Before an output is defuzzified all the fuzzy outputs of the system are aggregated with an union operator. The union is the max of the set of given membership functions and can be expressed as:

$$\mu_A = \bigcup_i (\mu_i(x)) \tag{2.3}$$

There are many defuzzification techniques but primarily only three of them are in common use. These defuzzification techniques are discussed below in detail.

Maximum Defuzzification

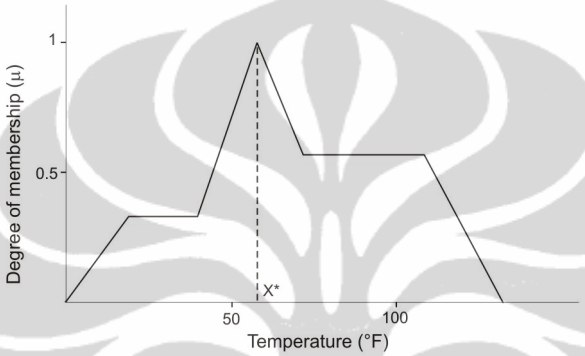

Figure 2.2: Maximum Defuzzification

This method gives the output with the highest membership function. This defuzzification technique is very fast but is only accurate for peaked output. This technique is given by algebraic expression as:

$$\mu(x*)_A \geq \mu(x) \quad \text{for all } x \, \varepsilon \, X \tag{2.4}$$

where x* is the defuzzified value. This is shown graphically in Figure 2.2.

Centroid Defuzzification

This method is also known as center of gravity or center of area defuzzification. This technique was developed by Sugeno in 1985. This is the most commonly used technique and is very accurate. The centroid defuzzification technique can be expressed as:

$$x* = \frac{\int \mu_i(x) x \, dx}{\int \mu_i(x) \, dx} \tag{2.5}$$

where x* is the defuzzified output, $\mu_i(x)$ is the aggregated membership function and x is the output variable.

Weighted Average Defuzzification

In this method the output is obtained by the weighted average of the each output of the set of rules stored in the knowledge base of the system. The weighted average

defuzzification technique can be expressed as:

$$x* = \frac{\sum_{i=1}^{n} m^i w_i}{\sum_{i=1}^{n} m^i} \tag{2.6}$$

where x* is the defuzzified output, m^i is the membership of the output of each rule, and w_i is the weight associated with each rule.

2.2 Transformation Matrix

There are several common transformations used in computer graphics. This section discusses geometric transformations: translation, scaling, and rotation.

2.2.1 Translation

Translation involves moving the element from one location to another. In the case of a point, the operation would be

$$\begin{aligned} x' &= x + d_x \\ y' &= y + d_y \end{aligned} \tag{2.7}$$

where (x', y') are the coordinates of the translated point, (x, y) are the coordinates of the original point, and (d_x, d_y) are the movements in the x and y directions. In matrix notation this can be represented as

$$P' = P + T \tag{2.8}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} d_x \\ d_y \end{bmatrix} \tag{2.9}$$

where T is the translation matrix.

Any geometric element can be translated in space by applying Equation 2.9 to each point that defines the element. For a line, the transformation matrix would be applied to its two end points.

2.2.2 Scaling

Scaling of an element is used to enlarge it or reduce its size. The scaling need not necessarily be done equally in the x and y directions. For example, a circle could be transformed into an ellipse by using unequal x and y scaling factors.

The points of an element can be scaled by the scaling matrix as follows:

$$P' = S \times P \tag{2.10}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} s_x & 0 \\ 0 & s_y \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.11}$$

This would produce an alteration in the size of the element by the factor s_x in the x-direction and by the factor s_y in the y direction. It also has the effect of repositioning the element with respect to the Cartesian system origin. If the scaling factors are less than 1, the size of the element is reduced and it is moved closer to the origin. If the scaling factors are larger than 1, the element is enlarged and removed farther from the origin.

2.2.3 Rotation

In this transformation, the points of an object are rotated about the origin by an angle 0. For a positive angle, this rotation is in the counterclockwise direction. This accomplishes rotation of the object by the same angle, but it also moves the object. In matrix notation, the procedure would be as follows:

$$P' = R \times P \tag{2.12}$$

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\alpha & -\sin\alpha \\ \sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{2.13}$$

where positive angle rotates the coordinates in counter-clockwise direction and negative angle rotates the coordinates in clockwise direction.

2.3 ERSP Scorpion Robot

The Evolution Robotics Software Platform (ERSP) is a Software Development Kit (SDK) that provides many implementations in a wide range robotics system, which are useful for vision, navigation and system development. ERSP and Scorpion Robot were used throughout the development and implementation in this thesis.

This section provides only some selected information of ERSP and Scorpion Robot, to give some overview explanation about the implementation in the next chapters. Refer to ERSP documentations for a complete insight about the software and the robot. The ERSP documentation consists of the following four documents: Getting Started [Evolution, 2004b], Tutorials [Evolution, 2004c], User's Guide [Evolution, 2004d], and API Documentation [Evolution, 2004a]. Documentation about the Scorpion Robot can be found in [Evolution, 2004e].

2.3.1 ERSP Architecture

The ERSP Architecture consists of three layers that can be used to run a collection of operations, from a simple task such as making the robot move forward, to complex tasks such as following a certain object in some environment. These layers are Hardware Abstraction Layer (HAL), Behavior Execution Layer (BEL), and Task Execution Layer (TEL). The system architecture is modular, with well-defined interfaces between its layers and software modules.

Hardware Abstraction Layer

The Hardware Abstraction Layer (HAL) is the lowest layer of the architecture, which provides interface between software applications and the robotics hardware resource. The HAL drivers are responsible on the robot's interactions between physical world and operating system (OS). Physical input from real world is received through infrared range sensors, bump sensors and cameras. The HAL is also able to interact with physical world, by changing its current state through effectors, for example, turning on only the left motor to make a rotation.

The HAL provides a way of interaction between a particular hardware device with other resources. A resource is a physical device that connects the software with the external environment. Through a resource, a robot or a software is aware of physical world and enables to interact with it. Resources include sensors and actuators, motors, cameras, infrared range sensors, bump sensors, or a battery.

Behavior Execution Layer

The Behavior Execution Layer (BEL) is the next layer of the architecture, on top of the HAL. This middle layer communicates with drivers and sensors through the HAL. Autonomous robotics system, built in this thesis, was using the Behavior Execution Layer of ERSP as its framework. The BEL is used, in order to acquire sensory input from the HAL, make a decision based on the input it received, and take an appropriate action.

The basic principle of BEL implementation is the behavior, which is defined as a block of computational unit that maps its inputs to its outputs. Behaviors provided by BEL of ERSP, cover a variety of the robot's functions, from driving sensors and actuators to mathematical operators, vision algorithms, and state machines. Behaviors are highly reactive and suitable for critical control loops.

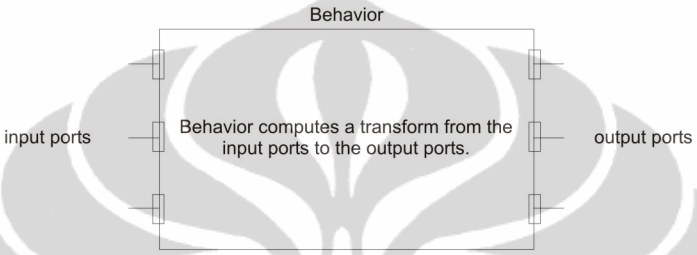

Figure 2.3: A Generic Behavior [Evolution, 2004d]

Behavior inputs and outputs are defined as ports. In Figure 2.3, the ports in a behavior are represented by a small rectangular. The ports on the left side of the behavior are defined as inputs, and the ports on the right side of the behavior are defined as outputs. Each output port can have connections to one or more input ports, represented by a line between ports. Each port has its own data type, data size, and semantic type, indicating the data structure that passing through that port. Connection between ports can only be done if they have a matching types. For example, a sensor measurement from IR sensors cannot be connected to a port expecting an image data type.

Interconnected behaviors form behavior networks. Behaviors in the network are executed sequentially by ERSP at the same rate, for every predefined interval (invocation). Behavior execution, in sequential mode, occurs in two stages: the behavior receives and waits data from all its input connections, and then it computes and pushes its output.

A behavior, in a network, operates in cycle manner. This means, every behavior will be executed in every given time interval. A behavior, which has received all necessary inputs must be executed first, if not, that behavior must wait until all necessary inputs received. Accordingly, ERSP uses a partial ordering so where behavior A is connected to behavior B, A executes before B. After behavior B received necessary input from behavior A, then B may run its execution.

Further, throughout this thesis, term of 'subsystem' is used to express 'behavior' and 'system' to express 'network'.

Task Execution Layer

The Task Execution Layer (TEL) is the top layer of the architecture. It provides a task-oriented method of programming an autonomous robot. By sequencing and combining certain tasks, a flexible plan for a robot to execute can be created, by writing the code in a classical procedural style. Tasks are useful to express a high-level execution knowledge and to coordinate multiple behavior actions, while behavior are suitable for creating robust repetitive actions. For example:

1. A robot using vision to follow an object with a certain color is best written as behavior.
2. A sequential robot actions such as moving forward 20 inches, taking a picture, rotating clockwise 1.57 radian, moving forward 10 inches, and then taking a picture again, is best written as a task.

2.3.2 Coordinate System

This subsection provides some convention definition about several different coordinate systems, which are used by ERSP and implemented in this thesis. There are three coordinate systems that are useful to mention in this subsection: Robot Coordinate System, Global Coordinate System, and Mapping Coordinate System.

Robot Coordinate System

Evolution Robotics Software Platform (ERSP) uses the standard robot coordinate system to describe the position and orientation of robot's actuators and sensors. This standard coordinate system is always used to define the robot's incremental of relative motion from the starting point where the robot is turned on. The origin of the system is located at the floor level, directly below the center of the drive wheelbase. Assume that the robot is traveling on flat surface and facing North. The positive x-axis extends in North direction. The positive y-axis extends in West direction and the positive z-axis is straight up, perpendicular to the floor surface. Figure 2.4 shows this coordinate system.

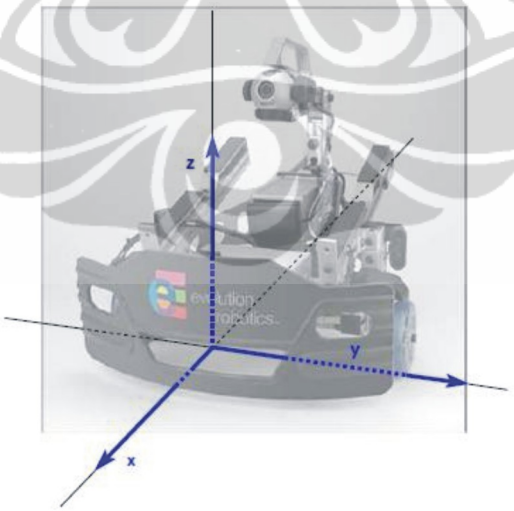

Figure 2.4: Robot's Coordinate System[Evolution, 2004b]

Global Coordinate System

Position and orientation can also be specified in the global coordinate system. The robot coordinate system and the global coordinate system are the same, when the first time the robot is turned on. The position of the robot in both reference frame coordinate system is (0,0,0). The first two numbers are position of the robot in (x,y) and the third number is the orientation, or the heading of the robot, in radian. The orientation angle, the positive angle of the robot is measured from the positive x-axis in counter clockwise direction.

In the robot coordinate system, the position of the robot is always (0,0,0). The global coordinate system is fixed to a particular starting point on the floor, as the robot is turned on. This particular point is always fixed all the time until the robot is turned off.

ERSP keeps track of the position of the robot, whenever the robot moves, in the global coordinate system through its wheel odometry information. The current position of the robot determined by ERSP, is based on the reference point (0,0,0), a starting point when the robot is turned on.

Mapping Coordinate System

Another coordinate system that is important to mention is the mapping coordinate system. This coordinate system is used, in the implementation, in the next chapter, to describe the position of obstacles from the robot's origin. The robot's origin, in the mapping coordinate system, is always (0,0). The origin of the mapping coordinate system is always at the center of the mapping image, as illustrated in Figure 2.5. The orientation of the mapping coordinate system always follows the orientation of the robot.

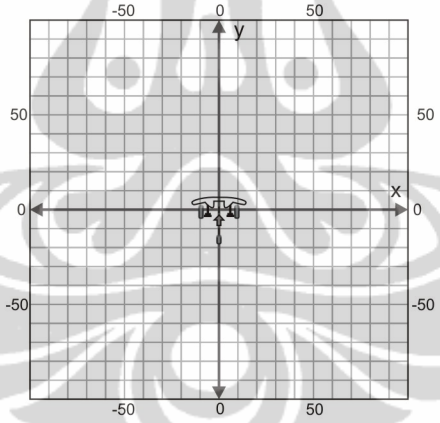

Figure 2.5: Mapping Coordinate System

Assumption of that the robot moves on a flat surface and the robot facing North, are also used in this mapping coordinates. The positive y-axis extends in the North direction. The positive x-axis extends in the East direction. These definition differ from robot coordinate system used by ERSP.