Chapter 4

Robot Control System

This chapter explained a detailed description of Robot Control System. The system consists of six implemented subsystems: Build-Map Subsystem,Odo-Map Subsystem, Command Subsystem, Add-Image Subsystem, Movement and Check-Region Subsystem . They are described into three parts: Description, Implementation, and Testing.

4.1 Build-Map Subsystem

Build-Map subsystem produces an image based on information it gets from sensors. This implemented subsystem is described in the following subsections.

4.1.1 Build-Map Subsystem Description

The main task of this subsystem is to produce an image, based on values received in its input ports. This subsystem is intended to receive a distance measurement values, as its input, from infrared range sensors.

Build-Map subsystem, as illustrated in Figure 4.1, has 9 input ports and 1 output ports. All input ports are *double* data type and the output port is an *image* data type.

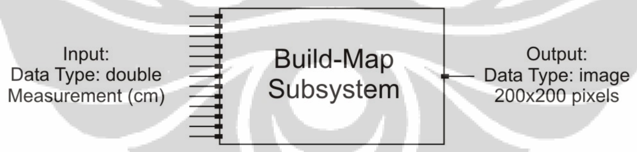

Figure 4.1: Build-Map Subsystem Diagram

Thirteen sensors are used in this subsystem and each sensor has its own mounting position (x- and y-values) and pointing direction (yaw-value) properties, as shown Figure 4.2. The sensors used, are assumed to have same height (z-value) and zero pitch value. Roll value is not taken into account because it does not affect the distance measurement. The properties of thirteen sensors used in this subsystem are given in Table 4.1.

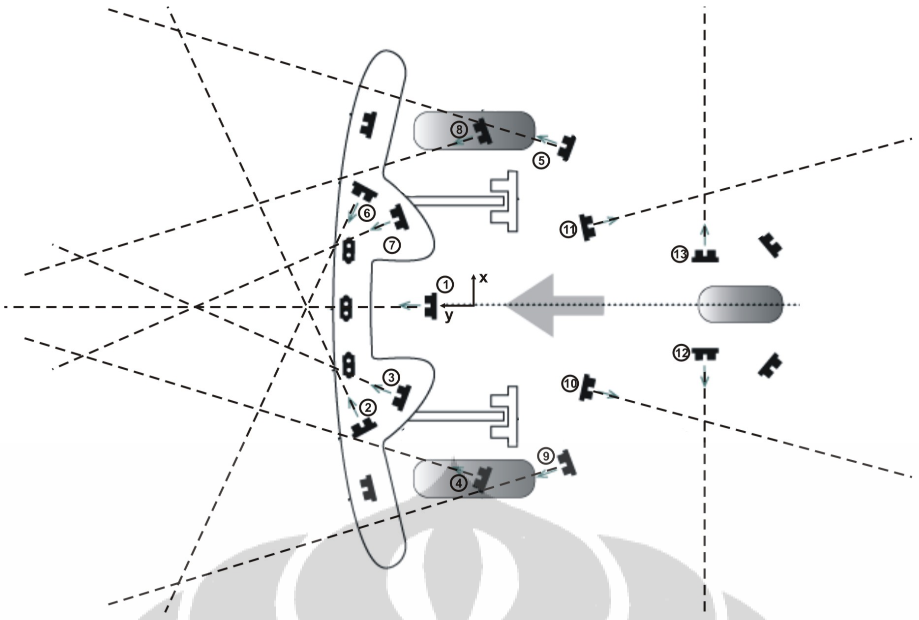

Figure 4.2: Pointing Direction of Scorpion Sensor

Table 4.1: Sensor Configuration Properties [Evolution, 2004e]

No.	Sensor's Name	x-value	y-value	yaw-value
1.	IR_bn_n	0.0	4.9	0.000
2.	IR_bn_ene	-9.2	12.6	-1.134
3.	IR_bn_ne	-6.4	9.2	-0.611
4.	IR_tw_nne	-17.3	1.6	-0.175
5.	IR_te_nne	16.9	-6.5	-0.262
6.	IR_bn_wnw	9.2	12.6	1.134
7.	IR_bn_nw	6.4	9.2	0.611
8.	IR_te_nnw	17.3	1.6	0.175
9.	IR_tw_nnw	-16.9	-6.5	0.262
10.	IR_bw_s	-6.9	-7.7	3.054
11.	IR_be_s	6.9	-7.7	-3.054
12.	IR_bs_w	-1.9	-17.2	1.571
13.	IR_bs_e	1.9	-17.2	-1.571

4.1.2 Build-Map Implementation

Build-Map subsystem is implemented, based on matrix rotation and translation in 2D environment. It needs a distance measurement value and three constant properties (x-value, y-value, yaw-value) from each sensor, as shown in Table 4.1.

First step of the procedure used in this subsystem, as illustrated in Figure 4.3, is to obtain the distance to an obstacle, and treat it as a value in y-axis:

$$\mathbf{X}_{obstacle} = \begin{bmatrix} 0 \\ distance \end{bmatrix} \tag{4.1}$$

And then, the coordinate of an obstacle is rotated based on orientation property of

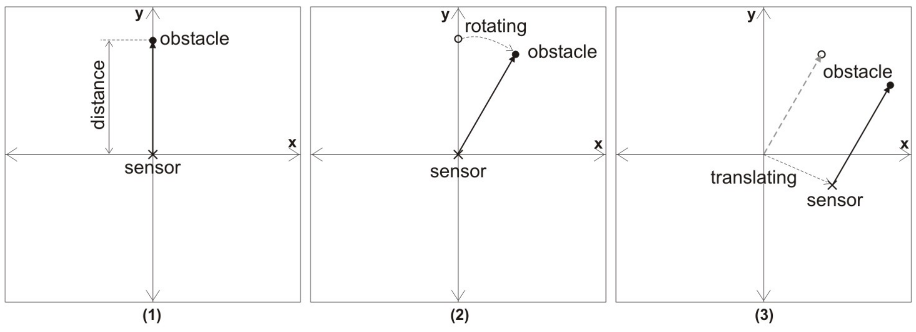

Figure 4.3: (1) Obstacle-Sensor Distance, (2) Rotation, (3) Translation

a sensor, by multiplying with the rotation matrix:

$$\mathbf{X}_{rotated} = \mathbf{X}_{obstacle}^{T} \underbrace{\begin{bmatrix} \cos yaw & -\sin yaw \\ \sin yaw & \cos yaw \end{bmatrix}}_{\text{rotation matrix}} \tag{4.2}$$

Finally, the real position of an obstacle in the image is calculated by adding the sensor's position properties:

$$\mathbf{X}_{real} = \mathbf{X}_{rotated} + \underbrace{\begin{bmatrix} x_{sensor} \\ y_{sensor} \end{bmatrix}}_{\text{sensor position}} \tag{4.3}$$

This procedure is applied to all thirteen sensors according to its own sensor reading and sensor properties. At the end, the results of the real obstacle coordinate are combined into a single output image. Everything behind the obstacle (from sensor point of view) is considered as an extension of that obstacle and marked as white area (pixels).

During experiment, maximum reading of each sensor is approximately around 70 cm. This is the reason why the size of the image output is set to 200×200 pixels. It is to represent 200×200 cm^2 area in the real world. Color format of image used in this subsystem is an 8-bit grayscale. The size and color format of this image are also used in other subsytem. An output example of this subsystem is shown in Figure 4.4. Bright pixels indicates the presence of an obstacle and dark pixels indicates the absence of an obstacle.

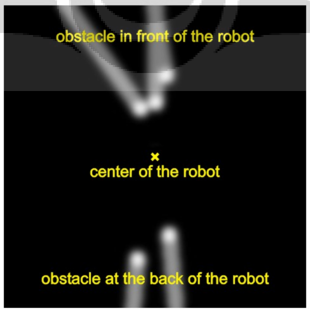
obstacle in front of the robot
center of the robot
obstacle at the back of the robot

Figure 4.4: Build-Map Output Example

4.1.3 Build-Map Subsystem Testing

Build-Map subsystem testing is performed, based on configuration shown in Figure 4.5. In this testing configuration, the constant is set to a certain value and connected to only one input port at a time. The other twelve sensors are connected to infinite constant. The output of the subsystem will be displayed by Image Display as soon as the result is available in the output port.

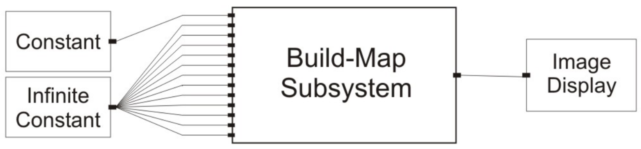

Figure 4.5: Build-Map Subsystem Testing Configuration

The subsystem is tested by applying several constant values to each input port, one at a time, to simulate the presence of an obstacle from each sensor. In this case, the constant values are set to 10, 50 and 80. Table 4.2 shows the calculated coordinate of an obstacle with a distance of 10 cm, 50 cm, and 80 cm, based on each sensor mounting and pointing properties. Figure 4.6 shows the output of Build-Map subsystem testing for sensor IR_bn_n (above) and IR_bw_s (below). It also shows that, every pixels behind the obstacle from sensor's point of view are also marked as an extension of an obstacle, represented by bright pixels. Bright circle in the middle of the image is not counted as an obstacle, it used to indicate the center of the robot.

Table 4.2: Build-Map Subsystem Testing Result

No.	Sensor's Name	Coordinate of a distance		
		10 cm	50 cm	80 cm
1.	IR_bn_n	(0,14)	(0,54)	(0,84)
2.	IR_bn_ene	(0,17)	(36,34)	(63,46)
3.	IR_bn_ne	(-1,18)	(22,50)	(39,75)
4.	IR_tw_nne	(-16,11)	(-9,51)	(-4,80)
5.	IR_te_nne	(18,4)	(28,43)	(36,72)
6.	IR_bn_wnw	(-1,17)	(-37,34)	(-64,46)
7.	IR_bn_nw	(0,18)	(-23,50)	(-40,75)
8.	IR_te_nnw	(15,11)	(8,51)	(3,80)
9.	IR_tw_nnw	(-19,4)	(-29,43)	(-37,72)
10.	IR_bw_s	(-7,-16)	(-11,-56)	(-13,-86)
11.	IR_be_s	(6,-16)	(10,-56)	(12,-86)
12.	IR_bs_w	(-11,-17)	(-51,-17)	(-81,-17)
13.	IR_bs_e	(10,-17)	(50,-17)	(80,-17)

4.2 Odo-Map Subsystem

Odo-Map subsystem uses previous information (previous obstacle map) to improve current Build-Map subsystem output by incorporating odometry information of robot movement. Its purpose is to refine the output of Build-Map subsystem, and it is an

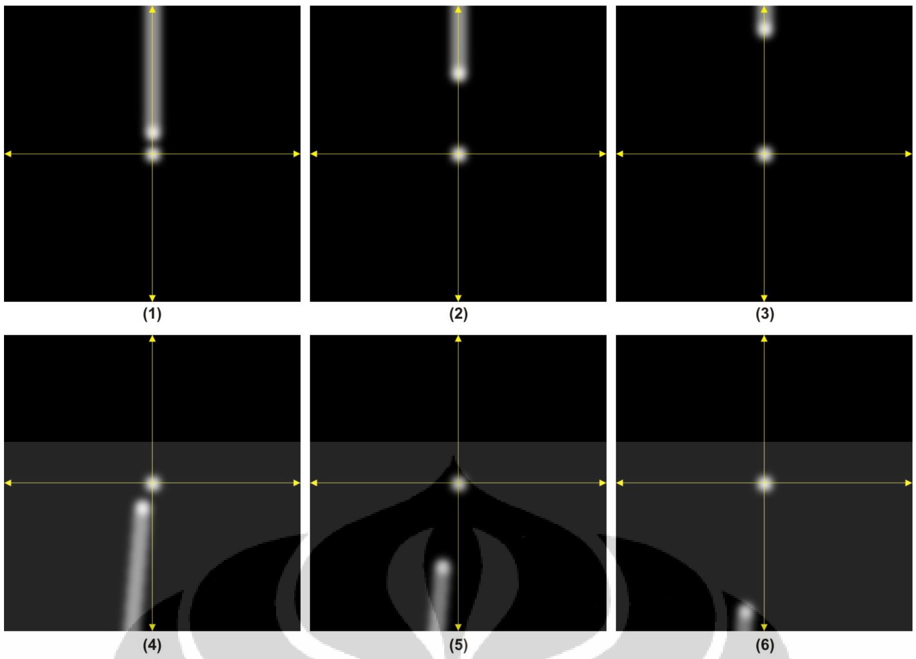

Figure 4.6: Testing Result for IR_bn_n (above) and IR_bw_s (below)

optional subsystem. This implemented subsystem is described in the following subsections.

4.2.1 Odo-Map Subsystem Description

The main task of this subsystem is to combine previous map and current map of Build-Map subsystem output, by incorporating odometry information of the robot movement. This subsystem produces a more detailed map image, by preserving previously mapped obstacle in its current output.

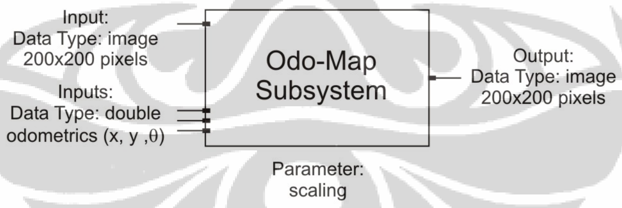

Figure 4.7: Odo-Map Subsystem Diagram

Odo-Map subsystem, as illustrated in Figure 4.7, has 4 input ports, 1 output port and 1 parameter. One of the input ports is an *image* data type, and it is intended to be connected with Build-Map subsystem output. The other three input ports are *double* data type. These ports receives movement information of the robot (current x-position, current y-position and current heading) from Odometry. The output port is *image* data type, which provides a more detailed map to Check-Region subsytem. Parameter scaling used in this subsystem is a *double* data type, and takes value from 0 to 1. This parameter is used as a grayscale multiplier of previously received image.

4.2.2 Odo-Map Implementation

The implementation of this subsystem is based on combining previously received image with newly received image into a single image. An adjustment to previously received image need to be made before the combination take place, by rotating and translating all pixels according to current odometry information (x-position, y-position and heading).

Rotating and translating procedure in this implementation is similar to Build-Map implementation procedure. In this case, the procedure is implemented to shift every pixel to another coordinate according to its odometry information. The procedure is described by the following equation:

$$\begin{bmatrix} x \\ y \end{bmatrix} = \underbrace{\begin{bmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{bmatrix}}_{\text{rotation matrix}} \begin{bmatrix} x' \\ y' \end{bmatrix} + \begin{bmatrix} x_c \\ y_c \end{bmatrix} \tag{4.4}$$

where θ is the heading of the robot, and (x_c, y_c) is the center of rotation of the robot.

Every pixel intensity $I(x', y')$ is shifted to $I(x, y)$ based on current odometry information of the robot (x-position, y-position and heading), as illustrated in 4.8.

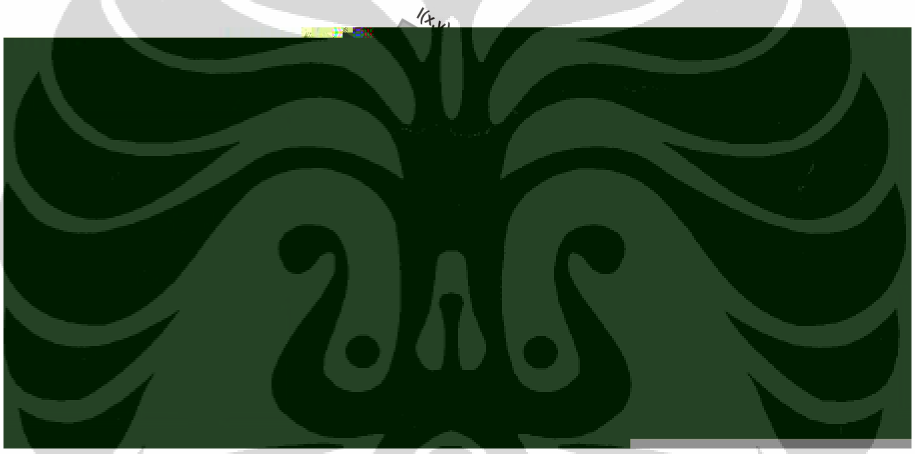

Figure 4.8: Odo-Map Subsystem Image Rotation Illustration

Once the previous image is being adjusted, Odo-Map subsystem combines the grayscale intensity information of this image with the current image based on per-element maximum intensity between two images, as described by:

$$I(x,y)_{output} = max(scaling \times I(x,y)_{previous}, I(x,y)_{current}) \tag{4.5}$$

Grayscale value of previous image is multiplied by scaling parameter. This multiplication makes the previous image fades away every cycle, if the scaling value is below 1.

4.2.3 Odo-Map Subsystem Testing

Odo-Map subsystem testing is performed, based on configuration shown in Figure 4.9. Load Image is used to simulate input image sent by Build-Map subsystem. There are three pairs Constants and Addition to simulate the increment of robot movement. The first pair is used to simulate y-position (cm), second pair to simulate x-position

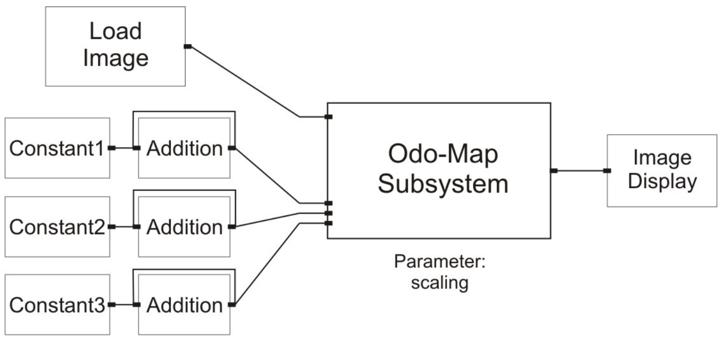

Figure 4.9: Odo-Map Testing Configuration

(cm), and third pair to simulate the current heading (rad) of the robot. The result of a refined map of obstacle is displayed by Image Display.

The subsystem is tested by using images from Figure 4.10(1) as input for test #1 through #3. Figure 4.10(2) is used for test #4 through #6. Values of constants in every testing are shown in Table 4.3. In the first test, Constant1 is set to 20, to simulate movement in y-axis. Second test, Constant2 is set to 20, to simulate movement in x-axis. Third test, Constant1 and Constant2 are set to 20, to simulate diagonal movement. Fourth test, Constant 3 is set to 0.785 (or 45°), to simulate rotational movement. Constants in fifth and sixth test is set to simulate combination of all movements.

Table 4.3: Constant Value for Odo-Map Testing

Test No.	Constant 1 y-axis (cm)	Constant 2 x-axis (cm)	Constant 3 heading (rad)
1.	20	0	0
2.	0	20	0
3.	20	20	0
4.	0	0	0.785
5.	15	15	0.25
6.	15	15	-0.25

The results of Odo-Map subsystem testing are shown in Figure 4.11. It is shown in 4.11(1), that the previous image is shifted downward if y-position of the robot moving forward. In Figure 4.11(2) is shown, that changes in x-position of the robot does not shift the previous image leftward nor rightward, but forward or backward, depends on its current heading orientation. This is because the robot can only move forward or backward, and sideways movement is not possible. The effect of changes in both x-position and y-position is shown in Figure 4.11(3), the distance of each circle is increased as the result of resultant of x- and y-position. Pure full rotation movement is shown in Figure 4.11(4). Counter-clockwise movement shifted the previous image in clockwise direction, and clockwise movement shifted previous image in counter-clockwise direction. Results of movement combination using different heading direction are shows in Figure 4.11(5) and (6).

The effect of scaling parameter can also be seen in the results. In this subsystem testing, the scaling value is set to 0.8, this means the previous image gets darker

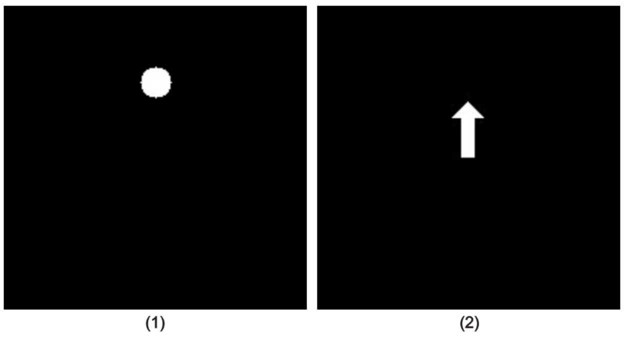

Figure 4.10: Odo-Map Testing Input

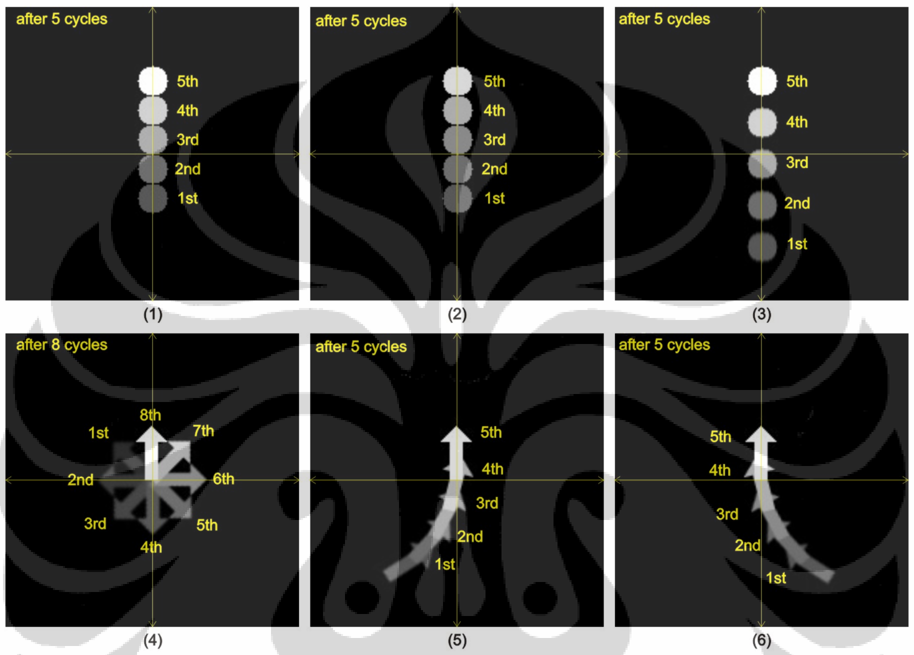

Figure 4.11: Odo-Map Testing Output Results

(fades away) every cycle. And eventually, the previous images will be disappeared after 24 cycles.

4.3 Check-Region Subsystem

Check-Region subsystem is used, to check the presence of an obstacle within specific region. This implemented subsystem is described in the following subsections.

4.3.1 Check-Region Subsystem Description

The main task of this subsystem is to check an input image for the presence of bright pixels, calculate the sum of pixels within a specific region and determine density of that region. A subsystem is assigned to check a region in an image, to determine whether that region is empty or not. Therefore, in order to check multiple regions in an image, more than one subsystem can be used to perform multiple regions checking.

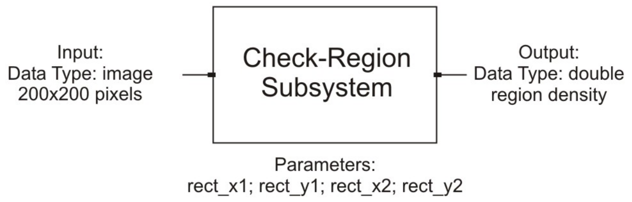

Figure 4.12: Check-Region Subsystem Diagram

Check-Region subsystem, as illustrated in Figure 4.12, has two ports (input and output) and 4 parameters. The input port is an *image* data type and the output port is a *double* data type. All parameters used in this subsystem are also *double* data type. These parameters are used to draw a rectangular mask region in an image.

4.3.2 Check-Region Implementation

The implementation of this subsystem is based on masking mechanism, applied to an image input. In this subsystem, the mask shape used is a rectangular and it has bright value of grayscale, in this case, the value is 255. The size and location of a mask in an input image are specified by two coordinate points, A (rect_x1,rect_y1) and B (rect_x2, rect_y2), as illustrated in Figure 4.13(a). These parameters (rect_x1, rect_y1, rect_x2, rect_y2) are required by the subsystem in its process, to be able to generate a mask image.

The masking mechanism in this subsystem is using pixel-by-pixel or element-by-element multiplication between the input image and the generated mask image, which is described by matrix operation:

$$[\mathbf{A}] .^* [\mathbf{B}] = \begin{bmatrix} a_{11} \cdot b_{11} & a_{12} \cdot b_{12} & \dots & a_{1j} \cdot b_{1j} \\ a_{21} \cdot b_{21} & a_{12} \cdot b_{22} & \dots & a_{2j} \cdot b_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i1} \cdot b_{i1} & a_{i2} \cdot b_{i2} & \dots & a_{ij} \cdot b_{ij} \end{bmatrix} \tag{4.6}$$

In this case, matrix **A** is the input image and matrix **B** is the mask image. The result of this operation is a masked image, which shows only pixels inside the mask region of the source image.

Figure 4.13(b) shows the interested region in an image, when a mask is applied to Figure 4.4. It shows only bright pixels inside the mask region, and pixels outside the region are all black, because it was multiplied by zero. Therefore, the density of the interested region can be calculated based on the amount of bright pixel wihtin the region.

A density of a region are required by Command subsystem, in order the subsystem to be able to determine whether the interested region is empty or not. The equation to obtain the region density is defined by:

$$region\_density = \frac{\sum_x \sum_y I(x,y)}{\sum_x \sum_y 255} \tag{4.7}$$

Region density value is the output of Check-Region subsystem and the range interval of this output is [0 , 1]. Output value from Check-Region subsystem can be further enumerated into two categories depends on some threshold value. For example, it

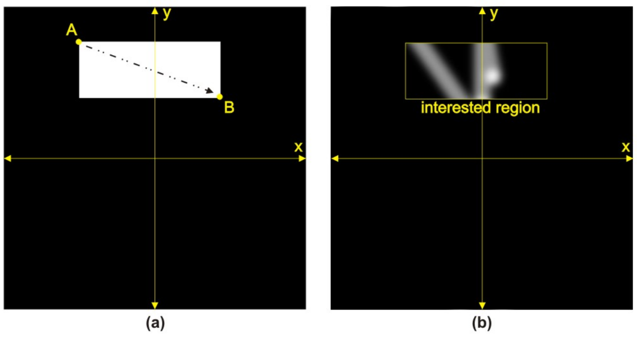

Figure 4.13: Mask Image (a) and Interested Masked Region(b)

can be categorized as 'empty' if the value is in interval [0 , 0.1] and 'not empty' if the value is in interval (0.1 , 1]. This kind of enumeration is used by Command subsystem, to determine what action should be taken, based on the region density value.

4.3.3 Check-Region Subsystem Testing

Check-Region subsystem testing is performed, based on configuration shown in Figure 4.14. Image used as input, in this testing, is taken from Figure 4.15. The image is loaded by Load Image and send it to Check-Region input port. The input image has equal amount of black and white pixels. Half part of the image is black (grayscale value is 0) and the other half part is white (grayscale value is 255). Input Collector is used to display subsystem output, region density, as soon as the value is available.

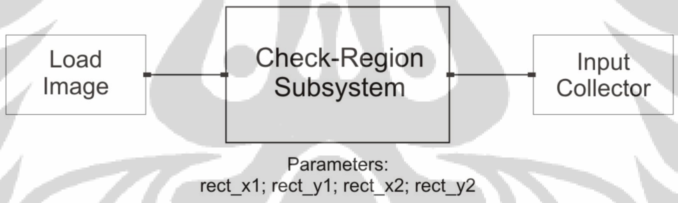

Figure 4.14: Check-Region Subsystem Testing Configuration

The subsystem is tested by applying different sets of parameters, which determine the size and location of a mask region. There are six sets of parameters, as shown in Table 4.4. All sets of parameters represented mask regions with the same size. The size of this mask region in the image is 60 × 30 pixels. The location of the mask region is set differently for each set of parameters.

The results of this testing are shown in Figure 4.16. In the results, locations of the mask region are illustrated by yellow grid boxes. It can be seen in the image that portion of bright pixels which is covered by the mask. Figure 4.16(1) showed that the applied mask did not cover any bright pixels of image input from Figure 4.15. Figure 4.16(1) showed that the mask covered some bright pixels. Figure 4.16(3) showed that half of the mask region covered bright pixels. The more bright pixels are covered by the mask region, the larger is the value of region density, as can be seen in Figure 4.16(3), (4), and (5).

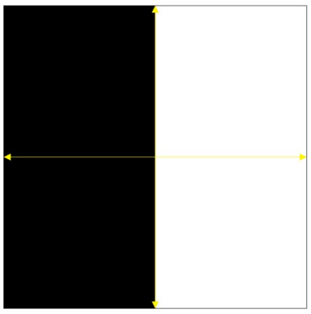

Figure 4.15: Check-Region Subsystem Testing Input Example

Table 4.4: Check-Region Subsystem Testing Parameters and Output

Test	Point A		Point B		output
No.	rect_x1	rect_y1	rect_x2	rect_y2	(region density)
1.	-100	50	-41	21	0
2.	-50	50	9	21	0.16667
3.	-30	50	29	21	0.5
4.	-20	50	39	21	0.66667
5.	-10	50	49	21	0.83333
6.	40	50	99	21	1

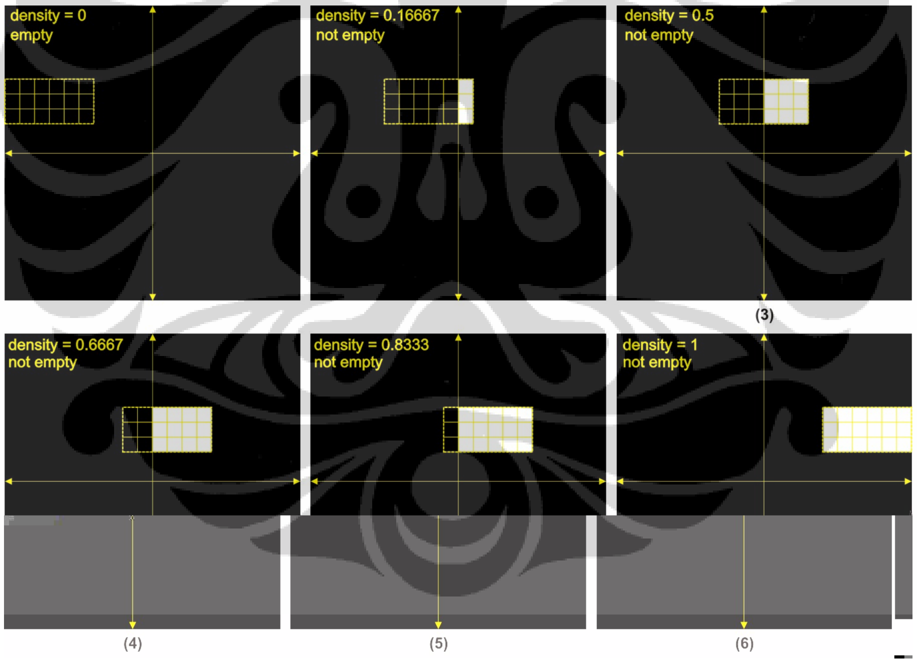

Figure 4.16: Check-Region Subsystem Testing Results

4.4 Command Subsystem

Command subsystem is used, to generate a specific command, and give the command with some weighting value based on region density value. This implemented subsystem is described in the following subsections.

4.4.1 Command Subsystem Description

The main task of this subsystem is to generate a specific command to its output port if the value (region density) in its input port satisfies some condition (threshold). This specific command contains information about angular and linear movement of the robot (angular and linear velocity), which is embedded into an image. One subsystem is used to generate one command. Therefore, more than one Command subsystem is needed, in order to perform multiple command combination.

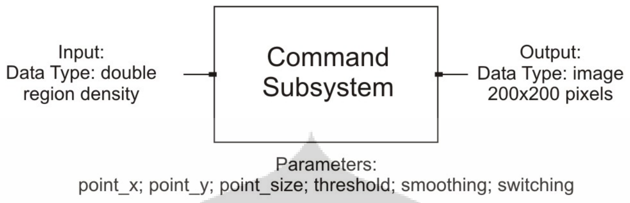

Figure 4.17: Command Subsystem Diagram

Command subsystem, as illustrated in Figure 4.17, has two ports (input and output) and 6 parameters. The input port is a *double* data type and the output port is an *image* data type. Parameter smoothing and switching are *boolean* data type and the other four parameters(point_x, point_y, point_size and empty_threshold) used in this subsystem are *double* data type.

4.4.2 Command Implementation

The implementation of this subsystem is based on embedding information about angular and linear velocity into an image. This information is represented by a circle drawn into image, as illustrated in Figure 4.18. Location of each circle in Cartesian coordinate is determined by parameter point_x and point_y.

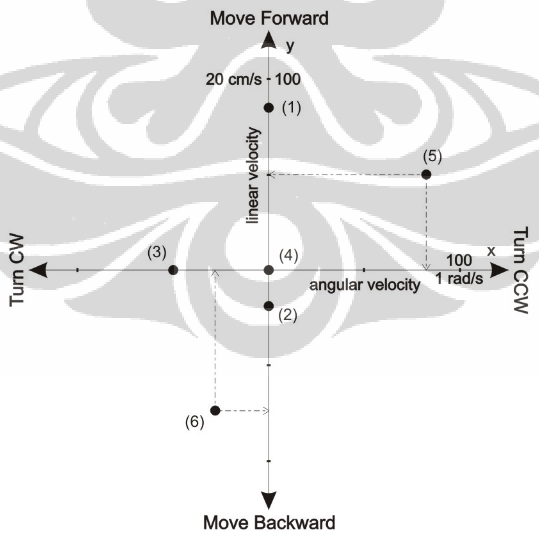

Figure 4.18: Graph Command Implementation

Six graphical command examples in Figure 4.18 can be translated into linguistic commands and a linguistic command can also be translated to a graphical command.

It can be translated when to 'move forward' or 'move backward' when there is only a linear component, and to 'turn counter-clockwise (CCW)' or 'turn clockwise (CW)' when there is only an angular component involved. Combination of both movement is used to translate a graphic command, when the command is consisted of linear and angular component, i.e command(5) and command(6). All six graphical commands in the figure can be translated as follow:

1. Move forward fast.
2. Move backward slowly.
3. Turn clockwise (CW) moderately.
4. Stop!
5. Turn counter-clockwise (CCW) fast while moving forward moderately.
6. Move backward fast while turning clockwise slowly.

These commands will be sent to output port as an image if the parameter switching is 'true' and the value it received in the input port is less than or equal to threshold value ('empty') , else it will send a blank image. By changing the parameter switching to 'false', the subsystem will send the given command to output port if the value it received is larger than threshold ('not empty'). The Command subsystem function is similar to IF-THEN-ELSE statement with CASE scheme, as described in following pseudocode:

```
Case true
   If region is 'empty'
      Then send the command with some weighting value
   Else send nothing
Case false
   If region is 'not empty'
      Then send the command with some weighting value
   Else send nothing
```

The weighting of a command mentioned in above pseudocode, can be regulated manually through the changing of radius of the circle (parameter point_size). Therefore, a larger circle is more important and has more influence effect than a smaller circle. A command weighting mechanism also depends on its grayscale color. The grayscale value of a command is determined by the value of region density received in the input port. The distribution of grayscale percentage of a region density is based on sigmoid curve function, defined by:

$$f(x) = \frac{1}{1+e^{-x}} \tag{4.8}$$

Then, the function is modified based on two density categories, 'empty' or 'not empty'. In this implementation, 'empty' is defined as value of region density in interval [0, threshold] and 'not empty' as value of region density in interval (threshold, 1]. Equation to calculate the percentage of grayscale value of an 'empty' region density is defined by:

$$f_1(x) = 1 - \frac{1}{1+e^{-(10\frac{x}{threshold}-5)}} \tag{4.9}$$

and for a 'not empty' region density, is defined by:

$$f_2(x) = \frac{1}{1 + e^{-(10\frac{x-threshold}{1-threshold}-5)}} \tag{4.10}$$

Curves for both equation to determine the grayscale value of a command are shown in Figure 4.19. Shape of each curve depends on the value of the threshold.

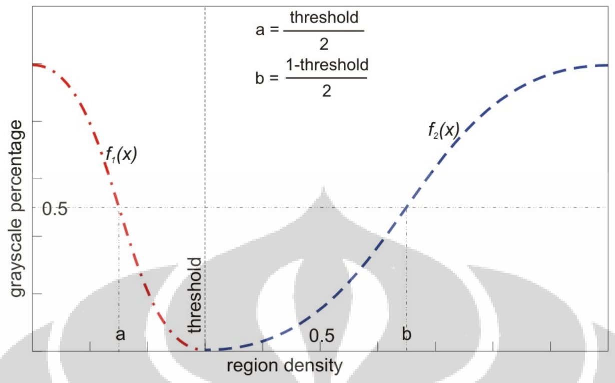

Figure 4.19: Regiond Density - Grayscale Percentage Relation

4.4.3 Command Test

Command subsystem testing is performed, based on configuration shown in Figure 4.20. Constant is used to simulate the density of a region from Check-Region subsystem. Output of Command subsystem will be displayed by Image Display. Parameter threshold, in this testing, is set to 0.1. This means, that region density less or equal 0.1 will be considered as 'empty' and region density greater than 0.1 as 'not empty'.

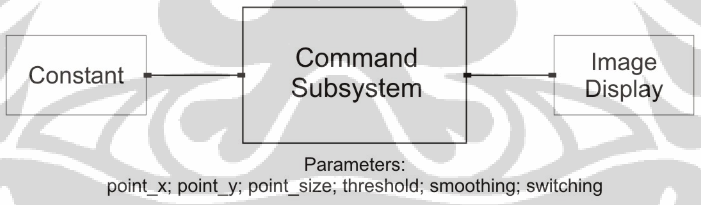
Constant
Command Subsystem
Image Display
Parameters:
point_x; point_y; point_size; threshold; smoothing; switching

Figure 4.20: Command Subsystem Testing Configuration

The subsystem is tested by applying several sets of parameters, in order to test Command subsystem. Sets of parameters in this testing is given in Table 4.5. Parameter switching is set to 'true' in testing #1 through testing #3 and set to 'false' in testing #4 through testing #6. Parameter smoothing set to 'true' only in testing #2 and #4. For the other testings, parameter smoothing is set to 'false'.

The testing results for all sets of parameters are shown in Figure 4.21. The grayscale value of a circle is calculated based on the value of region density it received in the input port. For a region density less or equal 0.1, the calculation is given by equation 4.8 and for greater than 0.1, the calculation is given by equation 4.9. Figure 4.21(2) and (4) show the result, when a circle is smoothed using Gaussian blur.

By referring explanation about Figure 4.18, the results can be translated into linguistic command, such as 'move forward', 'move backward', 'turn ccw', 'turn cw',

Table 4.5: Command Subsystem Testing Parameters

Test No.	Circle Coordinate (point_x)	(point_y)	Priority (point_size)	Constant Value (region density)
1.	0	75	10	0.02
2.	0	-25	15	0.05
3.	75	0	10	0.06
4.	0	0	40	0.08
5.	-50	50	20	0.55
6.	-25	0	20	0.45

‘stop’ and ‘move forward while turn clockwise’. Command ‘move forward’ and ’stop’ has the brightest grayscale color. This means, these command are more important and has more effect than any other command. Size of a circle has effect in a command weighting, as seen in the result, that command ‘stop’ is the largest. The effect of this weighting is explained in the next section (Add-Image Subsystem).

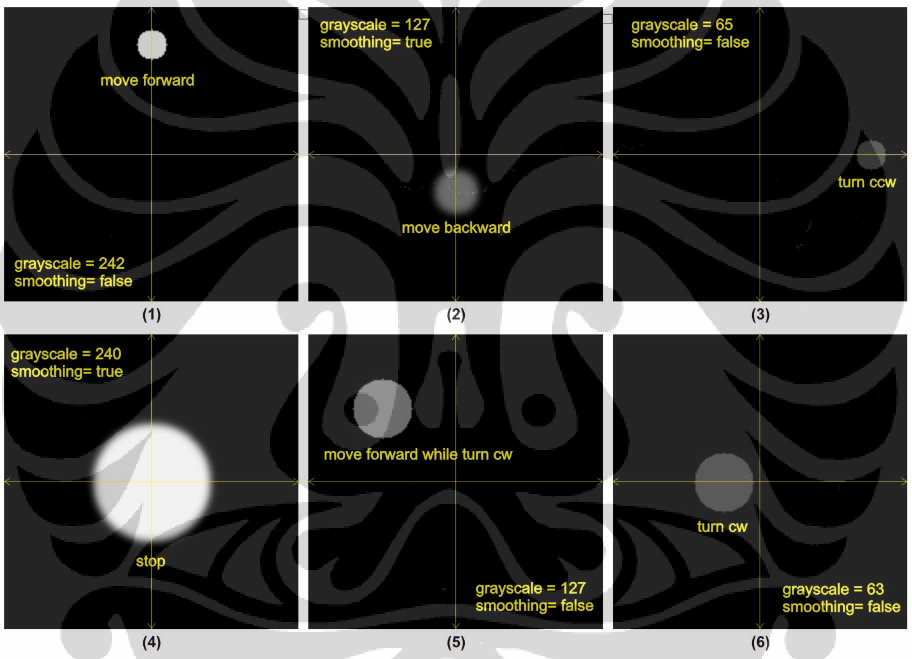

Figure 4.21: Command Subsystem Testing Result

4.5 Add-Image Subsystem

Add-Image subsystem is used to combine command images from Command subsystems into a single command image. The implemented subsystem is described in the following subsections.

4.5.1 Add-Image Subsystem Description

The main task of this subsystem is to combine several command images it received in input port and send the result to its output port. Number of received image in its input ports is not limited, therefore, more than one command images can be sent to the Add-Image subsystem input ports to be combined.

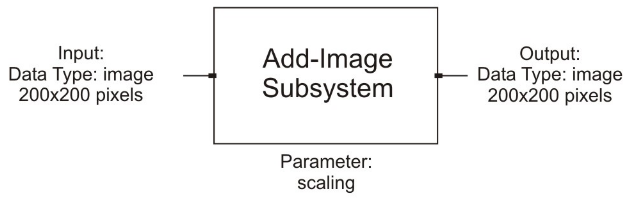

Figure 4.22: Add-Image Subsystem

Add-Image subsystem, as illustrated in Figure 4.22, has two ports (input and output) and one parameter. Input and output port are *images* data type. Parameter scaling used in this subsystem is *double* data type.

4.5.2 Add-Image Implementation

The implementation of this subsystem is based on adding multiple image into a single image. The subsystem will add all the incoming images and send it to its output port. The addition procedure is described by equation:

$$[\mathbf{A}] + [\mathbf{B}] = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} & \dots & a_{1j} + b_{1j} \\ a_{21} + b_{21} & a_{12} + b_{22} & \dots & a_{2j} + b_{2j} \\ \vdots & \vdots & \ddots & \vdots \\ a_{i1} + b_{i1} & a_{i2} + tb_{i2} & \dots & a_{ij} + b_{ij} \end{bmatrix} \tag{4.11}$$

In this case, matrix **A** and **B** are the image inputs, which have the same size (200×200 pixels) and color format (grayscale).

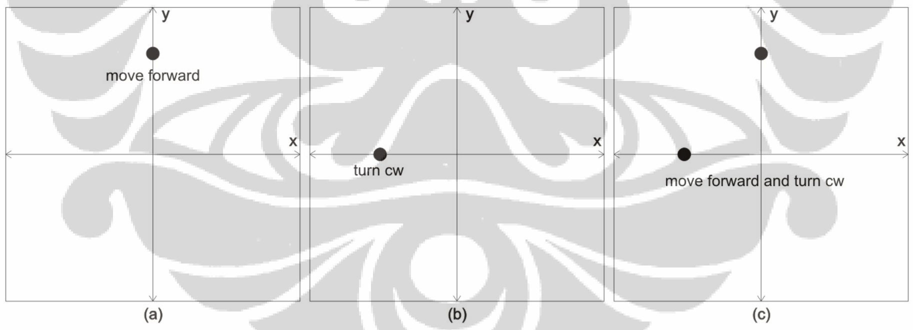

Figure 4.23: Move Forward(a), Turn CW(b), and Result (c)

Figure 4.23 shows an example of graph command addition, between command ‘move forward’ (a) and ‘turn clockwise’ (b). In a linguistic manner, the combination of these two command is described as ‘move forward and turn clockwise’, but the exact quantity of the combination need to be calculated, by determining the centroid of the combined command, which will be explained in the next section (Movement Subsystem).

In this subsystem. parameter scaling is used to scale down grayscale value of every command image before the addition process. Every input image will be multiplied by the scaling factor:

$$\frac{scaling}{255}\,[\mathbf{A}] + \frac{scaling}{255}\,[\mathbf{B}] + \ldots \tag{4.12}$$

If the scaling is not used, every addition element which exceeded 255, will be considered as a grayscale value of 255 (white).

4.5.3 Add-Image Subsystem Testing

Add-Image subsystem testing is performed, based on configuration shown in Figure 4.24. Load Image is used to simulate incoming images at the input port, by loading a predefined command image. The result of command image combination is displayed by Image Display.

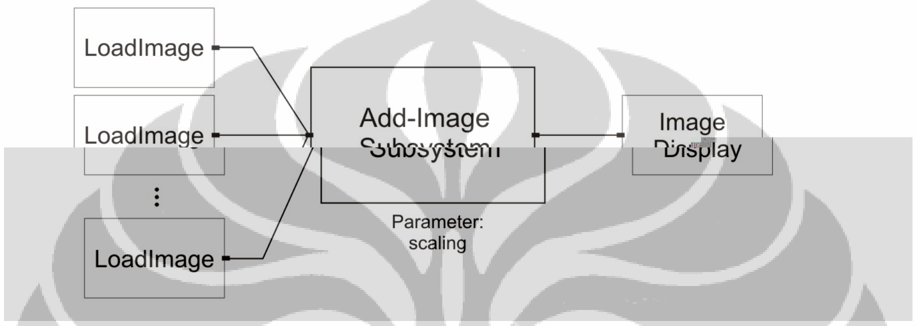

Figure 4.24: Add-Image Testing Configuration

The subsystem is tested by loading several sets of command images into Add-Image subsystem input port. Command image inputs for this testing, are taken from Figure 4.21. The sets of command images used to test the subsystem are given in Table 4.6. Testing #1 through #3 are addition of 2 command images; testing #4 and #5 are the addition of 3 command images; and the last testing is the addition of 5 command images.

Table 4.6: Adding Commands from Figure 4.21

Test No.	Adding Figure 4.21	Scaling Value
1.	(1) + (2)	255
2.	(1) + (3)	255
3.	(5) + (6)	10
4.	(4) + (5) + (6)	20
5.	(1) + (2) + (4)	255
6.	(1) + (2) + (4) + (5) + (6)	50

The results of Add-Image subsystem testing are shown in Figure 4.25. Parameter scaling is useful when there are overlapping command images, as can be seen in Figure 4.25(4) and (6). When the parameter scalling is not used, scalling = 255, the overlapping command most likely will cannot be distinguished, unless there are differences in their grayscale value. As can be seen in Figure 4.25(5), which is the addition of 3 command images, one of the command almost cannot be distinguished from other command.

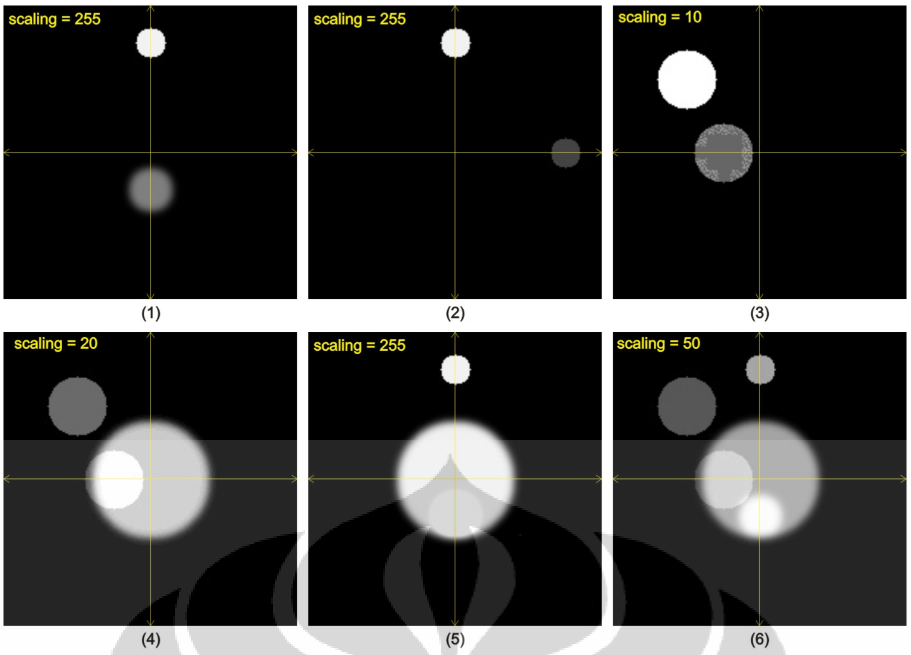

Figure 4.25: Add-Image Addition Result

4.6 Movement Subsystem

Movement subsystem is used to calculate the centroid of a combined command image, and controls angular and linear movement of the robot. The implemented subsystem is described in the following subsections.

4.6.1 Movement Subsystem Description

The main task of this subsystem is to calculate centroid of an image, convert the value into angular and linear velocity scales, and then send the results to its output ports. The outputs of this subsystem are angular and linear velocity values, which regulates the movement of the robot.

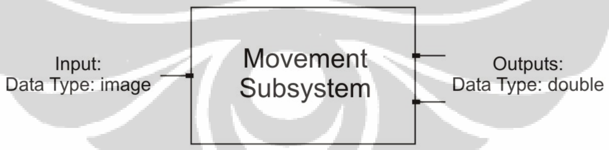

Figure 4.26: Movement Subsystem

Movement subsytem, as illustrated in Figure 4.26, has 1 input port and 2 output ports. The input is an *image* data type and the outputs are *double* data. This subsystem has no parameter.

4.6.2 Movement Implementation

The implementation of this subsystem is based on the calculation of image moments to determine its centroid. Equation of raw moments of an image, M_{ij}, is defined by:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y) \tag{4.13}$$

In a grayscale image, pixel intensity $I(x, y)$ value varies from 1 to 255. Centroid of an image, $(\bar{x}, \bar{y})$, can be calculated based on its moments, defined by:

$$\bar{x} = \frac{M_{10}}{M_{00}} \tag{4.14}$$

$$\bar{y} = \frac{M_{01}}{M_{00}} \tag{4.15}$$

The centroid value $(\bar{x}, \bar{y})$ has to be scaled down to limit the maximum velocity. Maximum value for angular velocity is 1 (radian per second) and maximum value for linear velocity is 20 (cm per second). The scalings are defined by:

$$angular\_velocity = \frac{\bar{y}}{100} \times 1 \text{ rad/s} \tag{4.16}$$

$$linear\_velocity = \frac{\bar{x}}{100} \times 20 \text{ cm/s} \tag{4.17}$$

These value directly control the movement of the robot. Positive value of angular velocity makes the robot turn counter-clockwise, and negative value makes the robot turn clockwise. Positive value of linear velocity makes the robot move forward, and negative value makes the robot move backward.

4.6.3 Movement Test

The Movement subsystem is performed, based on configuration shown in Figure 4.27. Image input is loaded by Load Image and the ouput of this subsystem is displayed by Input Collector.

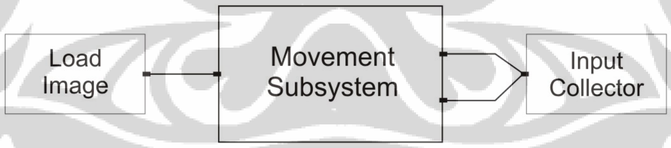

Figure 4.27: Movement Subsystem Testing Configuration

The subsystem is tested by using images from Figure 4.25 as its input, to determine the image centroid. The results of the subsystem testing are given in Table 4.7, which show coordinate of image centroid, and the value of angular and linear velocity. Figure 4.28 shows the location of the centroid in input image. It can be seen in the result, that a larger circle has the most effect in centroid calculation. It drags the centroid of the image near to its own center, as can be seen in Figure 4.28 (4), (5), and (6). It also shows that a cicrle with greater grayscale value will have more influence effect int the centroid calculation, as can be seen in Figure 4.28 (1), (2), and (3).

Table 4.7: Testing Result By Using Figure 4.25 As Input

Image	Centroid Coordinate		Output	
No.	$\bar{x}$	$\bar{y}$	angular vel.	linear vel.
1.	-0.00254942	21.1063	-2.54942e-005	4.22127
2.	16.6433	58.342	0.166433	11.6684
3.	-42.3837	34.835	-0.423837	6.96699
4.	-6.88728	5.51369	-0.0688728	1.10274
5.	0.000935116	3.94386	9.35116e-006	0.788771
6.	-6.20652	7.15113	-0.0620652	1.43023

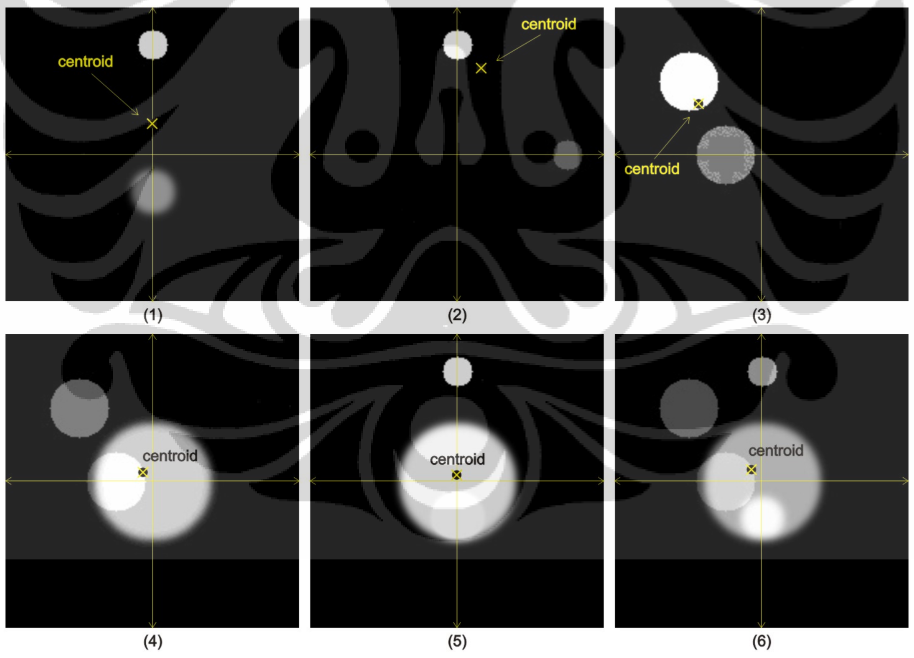

Figure 4.28: Movement Subsystem Testing Results

Chapter 5

Composite System Experiment

This chapter explains the application of Robot Control System in several experiments. The experiments were conducted by implementing rules and their combinations to produce a meaningful movement. The experiments are described into three sections: first section of this chapter describes experiments to produce simple movements, enhancement of simple movements is described in the second section, and last section of this chapter describes experiments to produce a more complex movements.

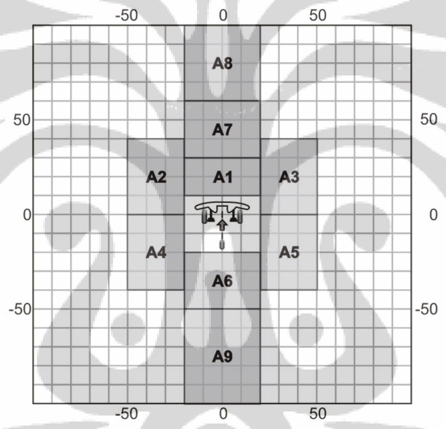

Figure 5.1: Configuration of Mask Regions

Throughout this chapter, size and location of all mask used in the rule statement and system configuration are defined in Figure 5.1. There are nine mask regions with different size and location, from A1 to A9. The mask configuration setup is specific for experiments in this chapter, but other mask configuration is possible to use.

5.1 Simple Movement

A simple movement of Scorpion Robot can be controlled by using only one single rule. There are at least eight simple movements: forward, backward, clockwise rotation, counter-clockwise rotation, clockwise forward, counter-clockwise forward, clockwise backward and counter-clockwise backward.

System configuration to produce simple movement only need one Check-Region subsystem and one Command subsystem, as illustrated in Figure 5.2. This simple rule configuration represents one IF-THEN statement. Check-Region subsystem is assigned to determine the existence of an obstacle within one of nine mask region

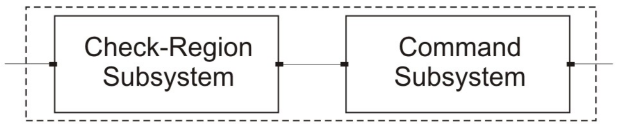

Figure 5.2: One Rule System Configuration

shown in Figure 5.1. Command subsystem produces one simple movement command if the condition about certain mask region density is satisfied. Default function of Command subsystem is to asserts ‘a region is empty’. By changing the parameter switching to ‘false’, the function of Command subsystem will change to assert ‘a region is not empty’.

5.1.1 Forward Movement

Behavior of forward movement of the robot is shown by this experiment. The experiment of forward movement is described as follows.

Objective

The objective of this experiment was to make the robot move forward if there is no obstacle in front of the robot.

Rule System Configuration

A rule statement, to control the forward movement of a robot if there is no obstacle within A7 region, was defined by:

$$\textit{If A7 region is empty then move forward} \tag{5.1}$$

The rule was created by setting parameters of Command subsystem (point_x, point_y, point_size, switching) to draw a sized 10 circle at coordinate (0,30) in a ‘case true’ scheme (by default, the Command subsystem was in a ‘case true’ scheme, indicated by ‘true’ value of switching parameter, refer to 4.4.2).

Result

The behavior of robot movement was based on whether A7 region is empty or not empty. The system sent ‘move forward’ command when there was no obstacle within A7 region. This ‘move forward’ command, as shown in Figure 5.3(1), made the robot moves forward. When there was an obstacle within A7 region, the ‘move forward’ command was not send out, instead, a ’blank’ command (completely black image) was sent out. This ’blank’ command made the robot stop its movement. The behavior of robot movement, in this experiment, is illustrated in Figure 5.3(2). Bright image of the robot shows the initial starting position of the robot, and darker image of the robot shows the latest position of the robot.

5.1.2 Backward Movement

Behavior of backward movement of the robot is shown by this experiment. The experiment of backward movement is described as follows.

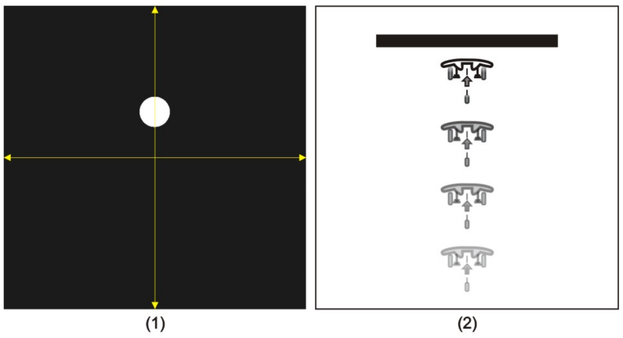

Figure 5.3: Forward Movement Command

Objective

The objective of this experiment was to make the robot move backward if there is no obstacle at the back of the robot.

Rule System Configuration

A rule statement, to control the backward movement of a robot if there is no obstacle within A6 region, was defined by:

$$\textit{If A6 region is empty then move backward} \tag{5.2}$$

The rule was created by setting parameters of Command subsystem to draw a sized 5 circle at coordinate (0,-20) in a 'case true' scheme.

Result

The behavior of robot movement was based on whether A6 region is empty or not empty. The system sent 'move backward' command when there was no obstacle within A6 region. This 'move backward' command, as shown in Figure 5.4(1), made the robot moves backward. When there was an obstacle within A6 region, the 'move backward' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its movement. The behavior of robot movement, in this experiment, is illustrated in Figure 5.4(2).

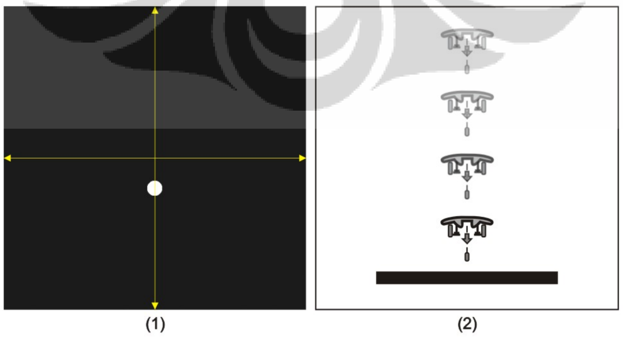

Figure 5.4: Backward Movement Command

5.1.3 Clockwise Rotation Movement

Behavior of clockwise rotation of the robot is shown by this experiment. The experiment of clockwise rotation is described as follows.

Objective

The objective of this experiment was to make the robot rotate clockwise is there is an obstacle in front of the robot.

Rule System Configuration

A rule statement, to control the clockwise rotation of a robot if there is an obstacle within A1 region, was defined by:

$$\textit{If A1 region is not empty then rotate clockwise} \tag{5.3}$$

This pure rotation rule was created by setting parameters of Command subsystem to draw a sized 10 circle at coordinate (-30,0) in a 'case false' scheme.

Result

The behavior of robot rotation was based on whether A1 region is empty or not empty. The system sent 'rotate clockwise' command when there was an obstacle within A1 region. This 'rotate clockwise' command, as shown in Figure 5.5(1), made the robot rotate clockwise. When there was no obstacle within A1 region, the 'rotate clockwise' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its rotation. The behavior of robot rotation, in this experiment, is illustrated in Figure 5.5(2).

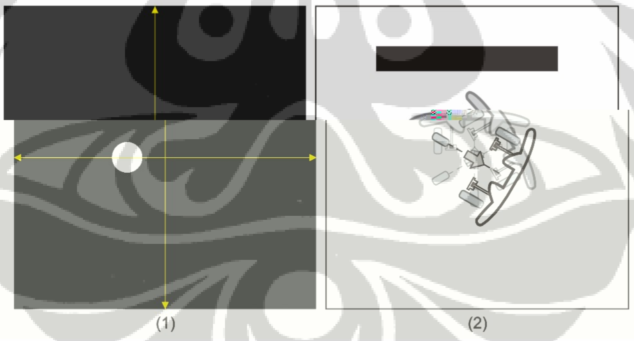

Figure 5.5: Clockwise Rotation Command

5.1.4 Counter-Clockwise Rotation

Behavior of counter-clockwise rotation of the robot is shown by this experiment. The experiment of clockwise rotation is described as follows.

Objective

The objective of this experiment was to make the robot rotate counter-clockwise if there is an obstacle in front of the robot.

Rule System Configuration

A rule statement, to control the counter-clockwise rotation of a robot if there is an obstacle within A1 region, was defined by:

$$\textit{If A1 region is not empty then rotate counter-clockwise} \tag{5.4}$$

This pure rotation rule was created by setting parameters of Command subsystem to draw a sized 10 circle at coordinate (30,0) in a 'case false' scheme.

Result

The behavior of robot rotation was based on whether A1 region is empty or not empty. The system sent 'rotate counter-clockwise' command when there was an obstacle within A1 region. This 'rotate counter-clockwise' command, as shown in Figure 5.6(1), made the robot rotate counter-clockwise. When there was no obstacle within A1 region, the 'rotate counter-clockwise' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its rotation. The behavior of robot rotation, in this experiment, is illustrated in Figure 5.6(2).

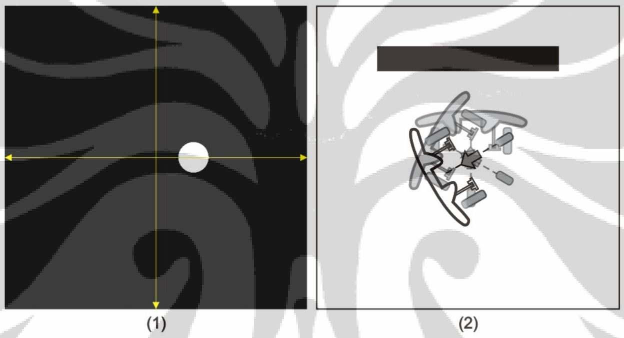

Figure 5.6: Counter-Clockwise Rotation Command

5.1.5 Forward-Right Movement

Behavior of forward-right movement of the robot is shown by this experiment. The experiment of forward-right movement is described as follows.

Objective

The objective of this experiment was to make the robot move forward and slightly to the right if there is an obstacle at front-right side of the robot.

Rule System Configuration

A rule statement, to control the forward-right movement of a robot if there is an obstacle within A2 region, was defined by:

$$\textit{If A2 region is not empty then move forward-right} \tag{5.5}$$

The rule was created by setting parameters of Command subsystem to draw a sized 10 circle at coordinate (-25,30) in a 'case false' scheme.

Result

The behavior of robot movement was based on whether A2 region is empty or not empty. The system sent 'move forward-right' command when there was an obstacle within A2 region. This 'move forward-right' command, as shown in Figure 5.7(1), made the robot move forward with slightly to the right. When there was no obstacle within A2 region, the 'move forward-right' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its movement. The behavior of robot movement, in this experiment, is illustrated in Figure 5.7(2).

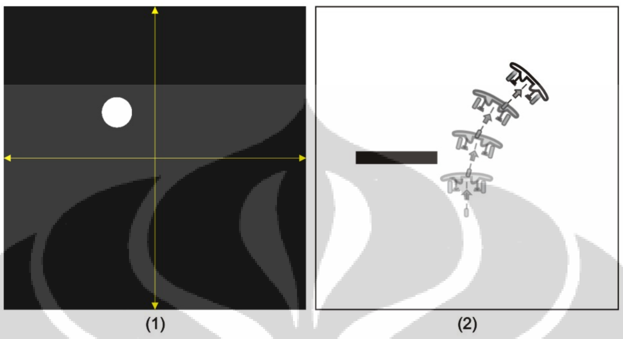

Figure 5.7: Forward-Right Movement Command

5.1.6 Forward-Left Movement

Behavior of forward-left movement of the robot is shown by this experiment. The experiment of forward-left movement is described as follows.

Objective

The objective of this experiment was to make the robot move forward and slightly to the left if there is an obstacle at front-left side of the robot.

Rule System Configuration

A rule statement, to control the forward-left movement of a robot if there is an obstacle within A3 region, was defined by:

If A3 region is not empty then move forward-left (5.6)

The rule was created by setting parameters of Command subsystem to draw a sized 10 circle at coordinate (25,30) in a 'case false' scheme.

Result

The behavior of robot movement was based on whether A3 region is empty or not empty. The system sent 'move forward-left' command when there was an obstacle within A3 region. This 'move forward-left' command, as shown in Figure 5.8(1), made the robot move forward with slightly to the right. When there was no obstacle within A2 region, the 'move forward-right' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its movement. The behavior of robot movement, in this experiment, is illustrated in Figure 5.8(2).

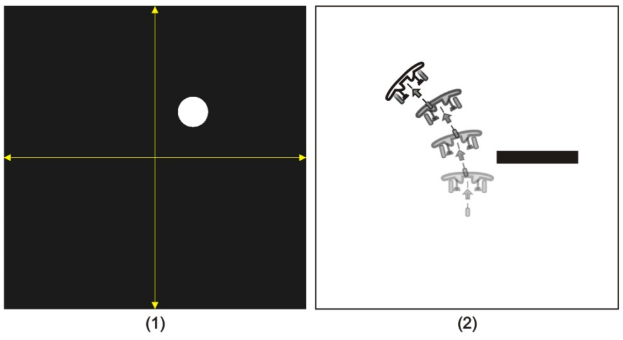

Figure 5.8: Forward-Left Movement Command

5.1.7 Backward-Right Movement

Behavior of backward-right movement of the robot is shown by this experiment. The experiment of backward-right movement is described as follows.

Objective

The objective of this experiment was to make the robot move backward and slightly to the right if there is an obstacle in front of the robot.

Rule System Configuration

A rule statement, to control the backward-right movement of a robot if there is an obstacle within A1 region, was defined by:

$$\textit{If A1 region is not empty then move backward-right} \tag{5.7}$$

The rule was created by setting parameters of Command subsystem to draw a sized 10 circle at coordinate (25,30) in a 'case false' scheme.

Result

The behavior of robot movement was based on whether A1 region is empty or not empty. The system sent 'move backward-right' command when there was an obstacle within A1 region. This 'move backward-right' command, as shown in Figure 5.9(1), made the robot move backward with slightly to the right. When there was no obstacle within A1 region, the 'move backward-right' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its movement. The behavior of robot movement, in this experiment, is illustrated in Figure 5.9(2).

5.1.8 Backward-Left Movement

Behavior of backward-left movement of the robot is shown by this experiment. The experiment of backward-left movement is described as follows.

Objective

The objective of this experiment was to make the robot move backward and slightly to the left if there is an obstacle in front of the robot.

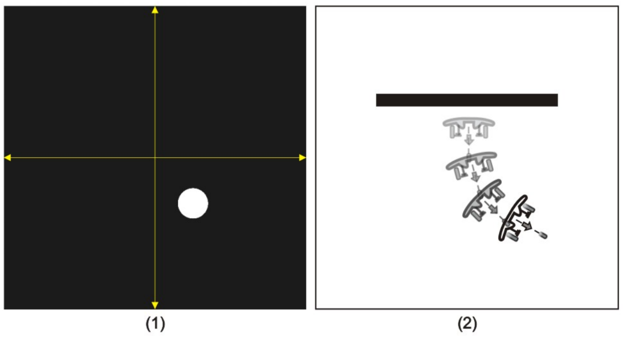

Figure 5.9: Backward-Right Movement Command

Rule System Configuration

A rule statement, to control the backward-left movement of a robot if there is an obstacle within A1 region, was defined by:

If A1 region is not empty then move backward-left (5.8)

The rule was created by setting parameters of Command subsystem to draw a sized 10 circle at coordinate (-25,30) in a 'case false' scheme.

Result

The behavior of robot movement was based on whether A1 region is empty or not empty. The system sent 'move backward-left' command when there was an obstacle within A1 region. This 'move backward-left' command, as shown in Figure 5.10(1), made the robot move backward with slightly to the left. When there was no obstacle within A1 region, the 'move backward-left' command was not send out, instead, a 'blank' command was sent out, and the robot stopped its movement. The behavior of robot movement, in this experiment, is illustrated in Figure 5.10(2).

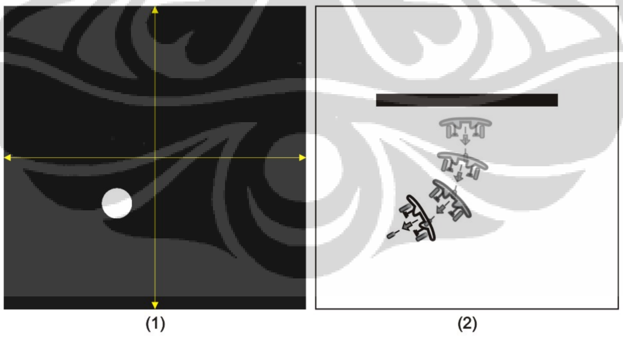

Figure 5.10: Backward-Left Movement Command

5.2 Enhanced Movement

An additional rule was added, in order to enhance the behavior of simple movement, thus the system configuration comprised two rules. Configuration of a system with two rule is illustrated in Figure 5.11(1). System configuration in Figure 5.11(2) was used, when both rules are checking the same region, thus, only one Check-Region subsystem was needed.

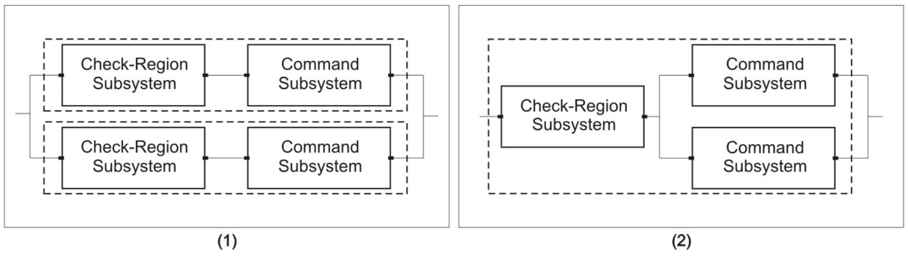

Figure 5.11: Two Rule System Configuration

Basically, the purpose of this enhancement was to make the robot avoid obstacle while moving forward. Therefore, there are, at least 6 combination of 'forward movement' with other movement, described in this sections: combination 'move forward' and 'move forward-right', combination 'move forward' and 'move forward-left', combination 'move forward' and 'move backward-left', combination 'move forward' and 'move backward-right', combination 'move forward' and 'rotate clockwise', and combination 'move forward' and 'rotate counter-clockwise'.

5.2.1 Combination With Forward-Right

Behavior of forward movement enhancement, by adding forward-right movement, was shown by this experiment. The experiment of this enhancement is described as follows.

Objective

The objective of this experiment was to make the robot, while moving forward, avoid an obstacle around its front-left, by moving slightly to the right.

Rule System Configuration

In this system configuration, 'move forward' command was enhanced by 'move forward-right' command. The rules statement to control the movement combination is defined by:

If A1 region is empty then move forward
If A2 region is not empty then move forward-right (5.9)

These rules were created, by setting parameters of Command subsystems to draw 'move forward' command (circle at coordinate (0,30) with size of 10) in a 'case true' scheme and 'move forward-right' command (circle at coordinate (-40,30) with size of 10) in a 'case false' scheme, as illustrated in Figure 5.12(1) and (2) respectively.

Result

The behavior of robot movement was based on the presence of an obstacle within A1 and A2 regions. The system sent 'move forward' command when there was no obstacle within A1 regions, and sent 'move forward-right' command when there was an obstacle within A2 region. The robot kept moving forward after avoided an obstacle around its front-left by moving forward with slightly to the right. The behavior of robot movement, in this experiment, is illustrated in Figure 5.12(3).

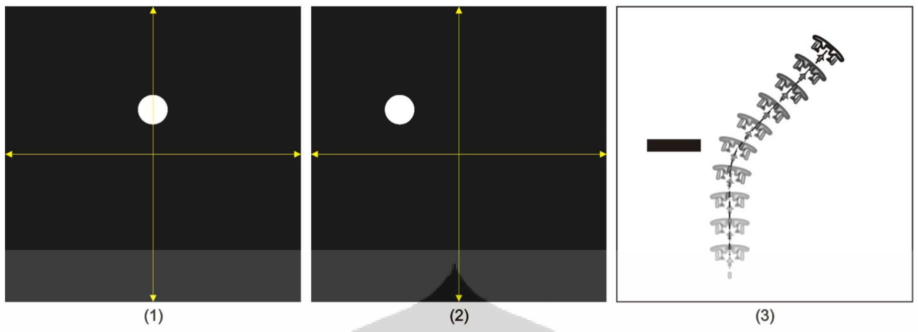

Figure 5.12: Combination with Forward-Right Movement

5.2.2 Combination With Forward-Left

Behavior of forward movement enhancement, by adding forward-left movement, was shown by this experiment. The experiment of this enhancement is described as follows.

Objective

The objective of this experiment was to make the robot, while moving forward, avoid an obstacle around its front-right, by moving slightly to the left.

Rule System Configuration

In this system configuration, 'move forward' command was enhanced by 'move forward-left' command. The rules statement to control the movement combination is defined by:

If A1 region is empty then move forward
If A3 region is not empty then move forward-left (5.10)

These rules were created, by setting parameters of Command subsystems to draw 'move forward' command (circle at coordinate (0,30) with size of 10) in a 'case true' scheme and 'move forward-left' command (a sized 10 circle at coordinate (40,30) with size of 10) in a 'case false' scheme, as illustrated in Figure 5.13(1) and (2) respectively.

Result

The behavior of robot movement was based on the presence of an obstacle within A1 and A3 regions. The system sent 'move forward' command when there was no obstacle within A1 regions, and sent 'move forward-left' command when there was an obstacle within A3 region. The robot kept moving forward after avoided an obstacle around its front-right by moving forward with slightly to the left. The behavior of robot movement, in this experiment, is illustrated in Figure 5.13(3).

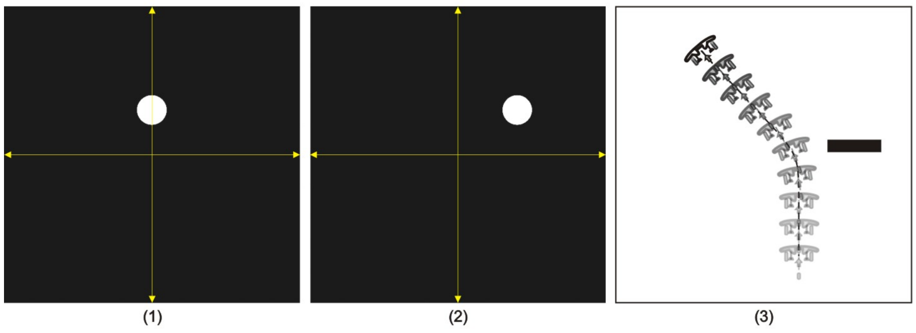

Figure 5.13: Combination With Forward-Left Movement

5.2.3 Combination With Backward-Left

Behavior of forward movement enhancement, by adding backward-left movement, was shown by this experiment. The experiment of this enhancement is described as follows.

Objective

The objective of this experiment was to make the robot, while moving forward, avoid an obstacle in front of it, by moving backward with slightly to the left.

Rule System Configuration

In this system configuration, 'move forward' command was enhanced by 'move backward-left' command. The rules statement to control the movement combination is defined by:

$$\begin{aligned}&\textit{If A1 region is empty then move forward}\\&\textit{If A1 region is not empty then move backward-left}\end{aligned}\qquad(5.11)$$

The rules checked the same region, thus configuration of two rules system in 5.11(2) was used. These two rule were redefined as a single rule statement:

$$\textit{If A1 region is empty then move forward else move backward-left}\qquad(5.12)$$

The rule was created, by setting parameters of Command subsystems to draw 'move forward' command (circle at coordinate (0,30) with size of 10) in a 'case true' scheme and 'move backward-left' command (a sized 10 circle at coordinate (-60,30) with size of 10) in a 'case false' scheme, as illustrated in Figure 5.14(1) and (2) respectively.

Result

The behavior of robot movement was based on whether A1 region is empty or not empty. The system sent 'move forward' command when there was no obstacle within A1 regions, otherwise 'move forward-left' command were sent when there was an obstacle. When encountered with an obstacle while moving forward, the robot avoided the obstacle in front of it, by moving backward with slightly to the left, and then continued to move forward if there was no obstacle. The behavior of robot movement, in this experiment, is illustrated in Figure 5.14(3).

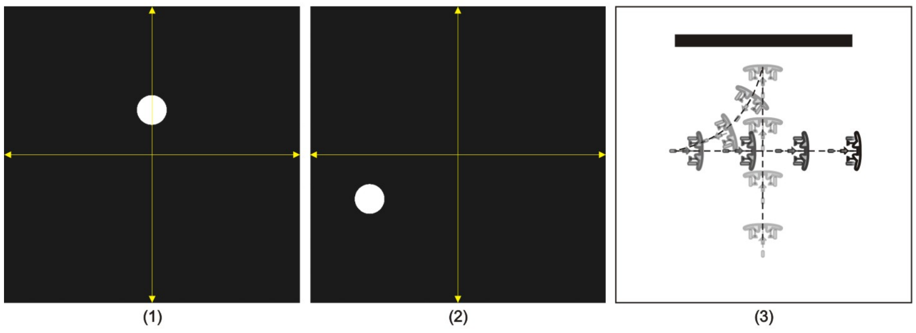

Figure 5.14: Combination with Backward-Left Movement

5.2.4 Combination With Backward-Right

Behavior of forward movement enhancement, by adding backward-right, movement was shown by this experiment. The experiment of this enhancement is described as follows.

Objective

The objective of this experiment was to make the robot, while moving forward, avoid an obstacle in front of it, by moving backward with slightly to the right.

Rule System Configuration

In this system configuration, 'move forward' command was enhanced by 'move backward-right' command. The rules statement to control the movement combination is defined by:

If A1 region is empty then move forward
If A1 region is not empty then move backward-right (5.13)

The rules checked the same region, thus configuration of two rules system in 5.11(2) was used. These two rule were redefined as a single rule statement:

If A1 region is empty then move forward else move backward-right (5.14)

The rule was created, by setting parameters of Command subsystems to draw 'move forward' command (circle at coordinate (0,30) with size of 10) in a 'case true' scheme and 'move backward-right' command (a sized 10 circle at coordinate (60,30) with size of 10) in a 'case false' scheme, as illustrated in Figure 5.15(1) and (2) respectively.

Result

The behavior of robot movement was based on whether A1 region is empty or not empty. The system sent 'move forward' command when there was no obstacle within A1 regions, otherwise 'move forward-right' command were sent when there was an obstacle. When encountered with an obstacle while moving forward, the robot avoided the obstacle in front of it, by moving backward with slightly to the right, and then continued to move forward if there was no obstacle. The behavior of robot movement, in this experiment, is illustrated in Figure 5.15(3).

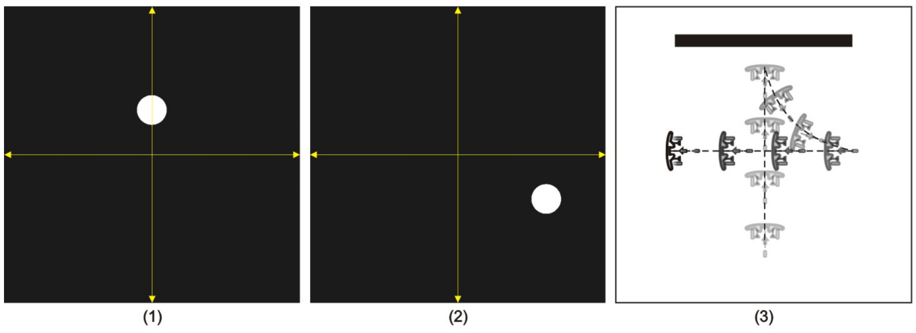

Figure 5.15: Combination with Backward-Right Movement

5.2.5 Combination With Clockwise

Behavior of forward movement enhancement, by adding clockwise rotation, was shown by this experiment. The experiment of this enhancement is described as follows.

Objective

The objective of this experiment was to make the robot, while moving forward, avoid an obstacle in front of it, by rotating in clockwise direction.

Rule System Configuration

In this system configuration, 'move forward' command was enhanced by 'rotate clockwise' command. The rules statement to control the movement combination is defined by:

$$\begin{aligned} &\textit{If A1 region is empty then move forward} \\ &\textit{If A1 region is not empty then rotate clockwise} \end{aligned} \tag{5.15}$$

The rules checked the same region, thus configuration of two rules system in 5.11(2) was used. These two rule were redefined as a single rule statement:

$$\textit{If A1 region is empty then move forward else rotate clockwise} \tag{5.16}$$

The rule was created, by setting parameters of Command subsystems to draw 'move forward' command (circle at coordinate (0,30) with size of 10) in a 'case true' scheme and 'rotate clockwise' command (a sized 10 circle at coordinate (-85,0) with size of 10) in a 'case false' scheme, as illustrated in Figure 5.16(1) and (2) respectively.

Result

The behavior of robot movement was based on whether A1 region is empty or not empty. The system sent 'move forward' command when there was no obstacle within A1 regions, otherwise 'rotate clockwise' command were sent when there was an obstacle. When encountered with an obstacle while moving forward, the robot avoided the obstacle in front of it, by rotating in clockwise direction, and then continued to move forward if there was no obstacle. The behavior of robot movement, in this experiment, is illustrated in Figure 5.16(3).

Figure 5.16: Combination with Clockwise Rotation

5.2.6 Combination With Counter-Clockwise

Behavior of forward movement enhancement, by adding counter-clockwise rotation, was shown by this experiment. The experiment of this enhancement is described as follows.

Objective

The objective of this experiment was to make the robot, while moving forward, avoid an obstacle in front of it, by rotating in clockwise direction.

Rule System Configuration

In this system configuration, 'move forward' command was enhanced by 'rotate counter-clockwise' command. The rules statement to control the movement combination is defined by:

$$\begin{aligned} &\textit{If A1 region is empty then move forward} \\ &\textit{If A1 region is not empty then rotate counter-clockwise} \end{aligned} \tag{5.17}$$

The rules checked the same region, thus configuration of two rules system in 5.11(2) was used. These two rule were redefined as a single rule statement:

$$\textit{If A1 region is empty then move forward else rotate counter-cw} \tag{5.18}$$

The rule was created, by setting parameters of Command subsystems to draw 'move forward' command (circle at coordinate (0,30) with size of 10) in a 'case true' scheme and 'rotate counter-clockwise' command (a sized 10 circle at coordinate (85,0) with size of 10) in a 'case false' scheme, as illustrated in Figure 5.17(1) and (2) respectively.

Result

The behavior of robot movement was based on whether A1 region is empty or not empty. The system sent 'move forward' command when there was no obstacle within A1 regions, otherwise 'rotate counter-clockwise' command were sent when there was an obstacle. When encountered with an obstacle while moving forward, the robot avoided the obstacle in front of it, by rotating in counter-clockwise direction, and then continued to move forward if there was no obstacle. The behavior of robot movement, in this experiment, is illustrated in Figure 5.17(3).

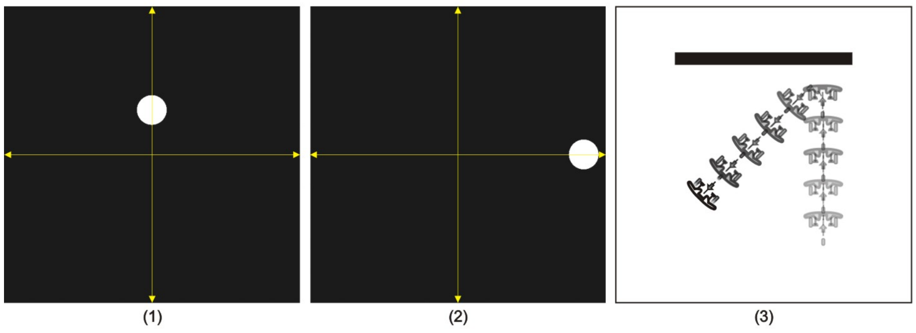

Figure 5.17: Combination with Counter-Clockwise Rotation

5.3 Complex Movement

In this section, several experiments, which were using more than two rule to control the movement of the robot, are presented. The purpose of these experiments was to see the behavior of the robot, which were controlled by several sets of rules, in a narrow dead-end alley, as illustrated in Figure 5.18. The effects of replacing some initial rules with other rules and the effect of adding a rule to a set of rules were shown by these experiments.

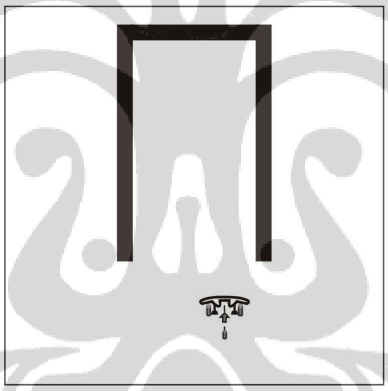

Figure 5.18: Dead-End Alley Obstacles Configuration

5.3.1 Experiment 1 - Navigating With Normal Speed

Behavior of complex robot movement, by using initial set of rules, was shown by this experiment. The experiment of this complex movement is described as follows.

Objective

The objective of this experiment was to make the robot enter and then come out from a narrow dead-end alley without hitting the obstacle.

Rule System Configuration

In this experiment, the initial set of rules to control the movement of the robot with the given obstacle (as illustrated in Figure 5.18) is defined by:

If A1 region is empty then move forward
If A8 region is empty then move forward
If A2 region is not empty then rotate clockwise
If A3 region is not empty then rotate counter-clockwise (5.19)

Configuration to construct a system for these rules is illustrated in Figure 5.19. Regions of the masks used in these rules can be seen in Figure 5.1. Parameters of each command used in the rules are given in Table 5.1. The purpose of the second forward in the rules is to speed up the first forward command if there is no obstacle within A8 region.

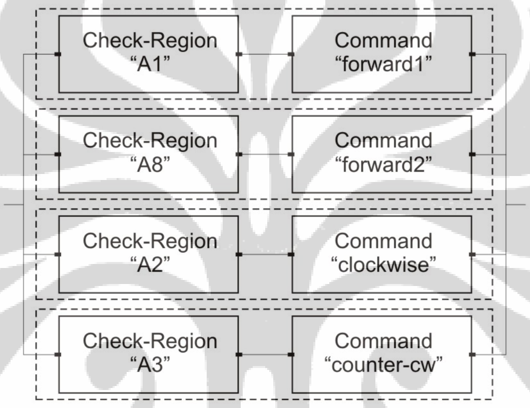

Figure 5.19: Configuration of Complex Rule System 1

Table 5.1: Command Parameters in Experiment 1

No.	Command	Circle Coordinate	Circle Size	Case Scheme
1.	Forward 1	(0,30)	10	true
2.	Forward 2	(0,70)	5	true
3.	Clockwise	(-40,0)	10	false
4.	Counter-cw	(40,0)	10	false

Result

Experiment with this set of rules showed that the robot immediately moved as there were no obstacle in front of it. The robot moved from left to right and from right to left, for several times. This is because, the robot always tried to avoid obstacle at one side by rotating to the other side. For example, when the position of the robot was closer to the left side obstacle then moved to the right side, and the other way around, it moved to the left side when the position of the robot was closer to the left side obstacle.

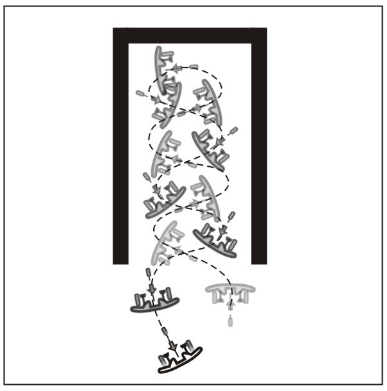

Figure 5.20: Robot Movement Behavior in Experiment 1

The behavior of the robot in this experiment is illustrated in Figure 5.20. The illustration of robot movement path does not depict the exact movement of the robot, because the path taken by the robot varied for every run. Basically, it depended on the initial position and heading of the robot.

5.3.2 Experiment 2 - Navigating With Faster Speed

Behavior of complex robot movement, by replacing several rules, was shown by this experiment. The experiment of this complex movement is described as follows.

Objective

The objective of this experiment was to make the robot enter and then come out from a narrow dead-end alley without hitting the obstacle, with a faster movement.

Rule System Configuration

In this experiment, some of initial rules to control the movement of the robot were replaced with other similar rules. Command 'rotate clockwise' and 'rotate counter-clockwise' were replaced with command 'move forward-right' and 'move forward-left' respectively. The new set of rules is defined by:

$$\begin{aligned}&\textit{If A1 region is empty then move forward}\\&\textit{If A8 region is empty then move forward}\\&\textit{If A2 region is not empty then move forward-right}\\&\textit{If A3 region is not empty then move forward-left}\end{aligned}\qquad(5.20)$$

System configuration in this experiment was the same with configuration in Experiment 1. The were four rules to control the movement of the robot. The differences were in the last two rules. The new commands had more forward linear velocity than the previous commands in Experiment 1. This new system configuration was illustrated in Figure 5.21. Parameters of each command used in the new rules were given in Table 5.2.

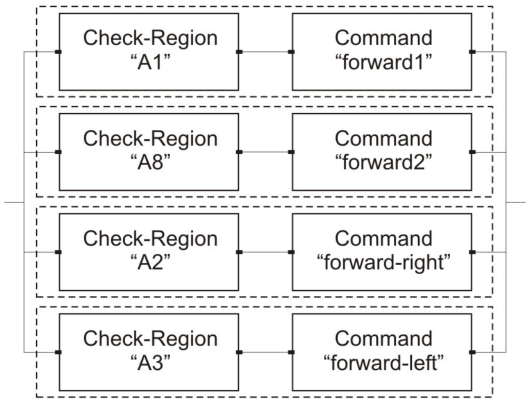

Figure 5.21: Configuration of Complex Rule System 2

Table 5.2: Command Parameters in Experiment 2

No.	Command	Circle Coordinate	Circle Size	Case Scheme
1.	Forward 1	(0,30)	10	true
2.	Forward 2	(0,70)	5	true
3.	Forward-Right	(-40,30)	10	false
4.	Forward-Left	(40,30)	10	false

Result

Movement of the robot in this experiment had the same behavior with the previous experiment. The robot immediately moves forward when were no obstacle in front of it. In this experiment, the robot also moved from left to right and from right to left, for several times. But the frequency of side-to-side movement was much less than in experiment 1. This is because, the new set of rules in this experiment produced more forward linear velocity than in previous experiment. The behavior of the robot in this experiment is illustrated in Figure 5.22. The illustration does not depict the detailed movement of the robot, because it only shows the generalization movement of the robot.

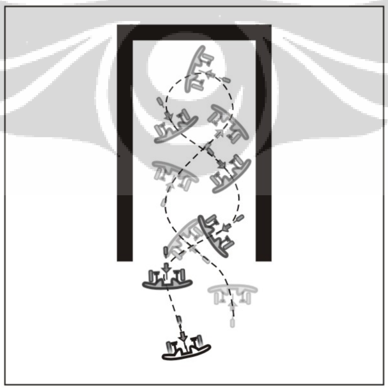

Figure 5.22: Robot Movement Behavior in Experiment 2

5.3.3 Experiment 3 - Solving Problem With Corners

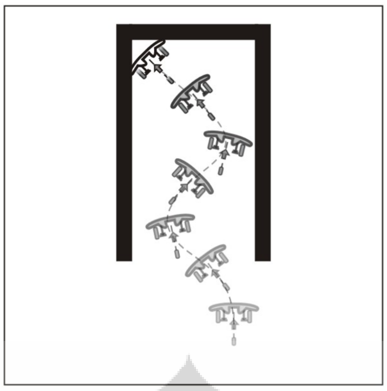

Figure 5.23: Problem With Corners in Experiment 1 and 2

In experiment 1 and experiment 2, there were problem when the robot headed toward the corner, as illustrated in Figure 5.23. The robot did not stop, it continued to move toward the corner. While heading toward the corner, the robot detected the same amount of obstacle in its right and left side, thus the set of rule in those experiment produced the forward movement as its resultant of two commands. Therefore the current set of rules needed an additional rule to solve this problem. Behavior of complex robot movement, by adding a rule, was shown by this experiment. The experiment of this complex movement is described as follows.

Objective

The objective of this experiment was to make the robot enter and then come out from a narrow dead-end alley without hitting the obstacle, with a faster movement, additionally, to fix the problem in experiment 1 and 2, when the robot headed toward the corner.

Rule System Configuration

This problem was fixed by adding one additional rule to control the backward movement of the robot, when there was an obstacle in front of the robot. The new set of rules to fix the problem is defined by:

If A1 region is empty then move forward
If A8 region is empty then move forward
If A2 region is not empty then move forward-right
If A3 region is not empty then move forward-left
If A1 region is not empty then move backward (5.21)

In this system configuration, there were five rules to control the movement of the robot. The purpose of additional rule was to compensate the forward movement, in case, the robot moved toward the corner. As can be seen in Table 5.3, the linear velocity of backward command was 50% of full throttle and the size of the circle was 50% larger than 'forward-left' or 'forward-right' commands. The new system configuration, to deal with corners problem, is illustrated in Figure 5.24.

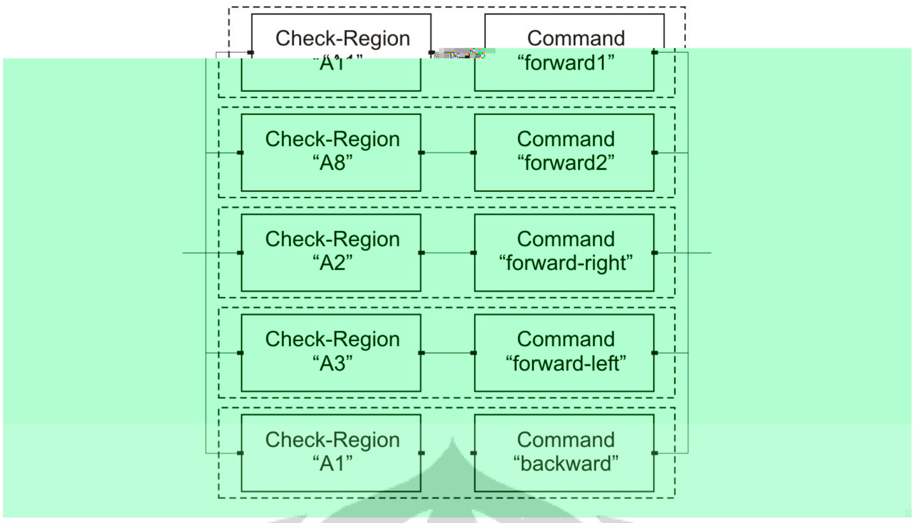

Figure 5.24: Configuration of Complex Rule System 3

Table 5.3: Command Parameters in Experiment 3

No.	Command	Circle Coordinate	Circle Size	Case Scheme
1.	Forward 1	(0,30)	10	true
2.	Forward 2	(0,70)	5	true
3.	Forward-Right	(-40,30)	10	false
4.	Forward-Left	(40,30)	10	false
5.	Backward	(0,-50)	15	false

Result

Movement of the robot in this experiment had the same behavior with the previous experiments. The robot immediately moved forward when were no obstacle in front of it. In this experiment, the robot also moved from left to right and from right to left, for several times. In this experiment, the problem with corner in experiment 1 and 2 was solved, by using additional backward rules. When headed toward the corner, the robot did not continue its forward movement, instead, the 'backward' command was triggered. This 'backward' command reduced the forward velocity of the robot, and eventually, resulted a small amount of backward velocity. The robot also made a rotational movement while moving backward. Therefore, the robot started to moved forward again when there was no obstacle in front of it.