Abstract

This project started with the assumption that by integrating reinforcement learning algorithm and embedded vision algorithm, improved, intelligent robot performance would emanate. In fact, a robot commonly performs given roles in specific environments (In-Cheol, 2002). However, by adapting reinforcement learning algorithm, a robot could have a more autonomous and intelligent performance. To demonstrate, this project composed an embedded system using a mobile robot, camera and embedded board. The progress of the project was separated into two main components. The first is the introduction of object tracking and reinforcement learning algorithms. Within those algorithms, the project proposed the combination of colour tracking algorithm and K-mean clustering algorithm with added weight distribution for the object tracking process. Moreover, as the reinforcement learning algorithm varied Q-learning methods, it was supported through QL and MQL. Secondly, the project produced the programs using the above algorithms and tests based on them. The experiment following the program had various features of progress with different parameters such as learning rate, discount factor, and environmental conditions. As a result, the project offers support for the assumption that the robot showed improved performance through learning from past actions, despite limitations such as memory space shortage and error results.

Acknowledgements

Actually, for me, the period of doing the final project and dissertation has been meaningful but painful. During the whole period of fulfilling this project, there have been lots of challenges and difficulties, and I have needed to implement various ways to achieve my aims: sometimes, there was success, sometimes failure. But fortunately, with the help of so many people, I have finally accomplished my project. Here, I would like to say thanks to everyone. Firstly, I am so grateful to my personal supervisor, Dr. Walterio Mayol-Cuevas, who always supported me and gave me suggestions to solve the hardware problems. Moreover, I also want to give appreciation to Dr. Tilo Burghardt and Dr. Andrew Calway, both of whom gave me useful feedback on how to better accomplish my project. In addition, it is due to the help of Dr. Tim Kovacs in specific relation to 'reinforcement learning' that I applied this aspect to my project. Furthermore, I also need to say thanks to my friends and classmates, because they encouraged me by offering their own opinions and suggestions which urged me to improve the quality of my project. Finally, I want to thank my family member who always loves me and support me.

Table of contents

1. Introduction

1.1 Overview

Although video and robot technologies have seen tremendous leaps of progress in the past few decades (with the inclusion of motors, sensors and cameras) it is still difficult to create more autonomous and artificial intelligent humanoid or mobile robots, largely because of the current inability to apply varied algorithms to the system (Peters, Sethu, and Stefan, 2003). For this reason, the creation of algorithms which can lead to an improved system has received plenty of attention from developers and scientists. This is particularly the case in the field of embedded systems, where vision algorithms in combination with artificial intelligent theories have arisen as one of the most important areas of development.

The development of embedded systems has always been connected to enhancing the quality of human life. Robots and mobile phones, for example, are part and parcel of our everyday life, and have been integrated into plenty of useful functions and attachments such as cameras and multi-touched screens. To achieve improved embedded system technologies, scientists and developers aim to integrate already existing technologies with new and creative technological developments.

Figure 1 FIRA world cup, Source: http://www.fira.net

The FIRA robot cup is a notable example of combination reinforcement learning and embedded vision. The FIRA (Federation of International Robot-Soccer Association) was established in June 1997 with the objective to promote the development of autonomous multi-agent co-operated robotic systems. It has supported developers who have attempted to improve Micro-Robots with varied theories. These theories were typically researched or invented by participants in advance of the competition. Under the guidance of FIRA, the participants began attempting to use ‘reinforcement learning’ as a way of developing more intelligent robot movements (In-Cheol, 2002). Moreover, within various activities of robotic system, a number of improvements were suggested in order to integrate reinforcement learning into the embedded vision theories. There were a number of successful cases, particularly with attachments that used sensor technologies (In-Cheol, 2002).

In the case of embedded systems which have a time or spatial limit for the system configuration and performance, the participants have tried to solve the limitation and to improve their activities by using not only the embedded vision but also a diversity of beam sensors and Bluetooth technologies. In addition, combining different theories offered another alternative. One example was a solution that combined embedded vision algorithms and reinforcement learning without other attachments. Hence, these improvements offer potentially effective methods to solve current limitations and improve system functions.

In fact, Asada M. and Noda H. (1996) designed an embedded robot by vision-based reinforcement leaning (Asada, Noda, Tawaratsumida, and Hosoda, 1996) and subsequently In-Cheol (2002) suggested the reinforcement learning approach, particularly in regard to the dynamic positioning of each robot agent in different environments (In-Cheol, 2002). Moreover, Arleo, A. and Smeraldi F. (2001) have delineated the hippocampus model configuration by using vision and reinforcement learning (Arleo, Smeraldi, Hug, and Gerstner, 2001).

The positive results from these diverse of fields, as well as earlier studies and research; suggest that integrating reinforcement learning and vision algorithm offers a promising way to effectively improve system performance.

1.2 Aims and objectives

Under the assumption that integrating algorithms can improve system performance, this project attempts to effectively combine reinforcement learning and vision algorithms. In particular, the most important aim of this project is to adapt combined algorithms to an embedded mobile robot and to observe and analyze the resulting change in robot movements.

In describing the main aim of this project, Figure 2 shows the general system configuration.

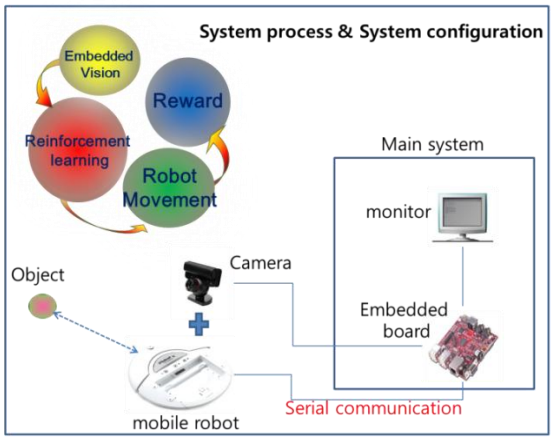


Figure 2 system process & system configuration

In addition, this project can be separated into two parts: general and specific aims.

General aims

- Composing an embedded system based on vision with a camera.

- Composing development environment depending on the operating system to manage the whole system including board, camera and robot.

- Reviewing embedded vision and reinforcement learning algorithms.

- Adapting combined theories to the mobile robot, and testing resulting robot movements.

Specific aims

- Specific object tracking by using vision algorithms.

- Accurate measurement of size and location of the object.

- Adapting diversity reinforcement learning algorithms to the embedded mobile robot.

- Handling a mobile robot movement by using serial communication.

- Accurate measurement of the object distance by the use of image frame.

- Comparison and analysis of reinforcement learning algorithms through observation of robot movements.

2. Theoretical background

To prove hypotheses through experiments that use attachments (such as a camera, a robot and an embedded board) it is necessary to have a certain amount of theoretical background knowledge. This knowledge should be related to the embedded vision for specified object tracking and reinforcement learning for the optimization of robot movements. In the first case, embedded vision using a camera necessitates that the robot needs to be able to distinguish and track specified coloured objects in an intricate environment. Moreover, the robot needs to be able to search for the object, as well as determine its size and location. In the second case, fundamental theories about reinforcement learning should be applied to the robot, so that it can potentially decide and make relevant movements for itself. In line with these aims, the next part will introduce the fundamental algorithms or theories for this project, specifically with regard to embedded vision and reinforcement learning.

2.1 Embedded vision

Embedded vision methods are of vital importance. They allow image availability of high quality to be obtained, including the ability to withstand an embedded system using an inexpensive camera: how to measure the shape of an object; how to handle images in order to obtain the object size; and how to position the image; all of which will be the main tasks of this project. Therefore, this area will introduce possible vision algorithms to effectively track objects and calculate the exact object size and position. As such, embedded vision is to include: object tracking, target position and the measurement of distance.

2.1.1. Object tracking

Object tracking is one of the important tasks within embedded vision fields of this project (Yilmaz, Javed, and Shah, 2006). Under the framework of a video frame analysis, it can be separated into three key steps: detection of moving objects, object tracking from frame to frame, and tracking object analysis in order to recognize moving object (Yilmaz, Javed, and Shah, 2006). The essential tasks in this project are thus to effectively and correctly detect the movement of specific objects. For this task, it is proposed that the main features for object tracking should include: template matching, colour, shape, edges, optical flow and texture tracking.

2.1.1.1 Object representation

In the object tracking method, an object can be represented as anything that is of interest such as colour, shape and optical flow (Yilmaz, Javed, and Shah, 2006). In addition, how to represent the object image information could stimulate changes of tracking algorithm. As a result, interpreting object representation forms part of the important background knowledge for image processing and object detection.

Firstly, the object is represented as a single point (called the centroid) (Veenman, Reinders, and Backer, 2001) or as a few set of points (Serby, Koller-Meier, and Gool, 2004) such as (a) and (b) in Figure 3. It is generally suitable for object tracking to occupy a small tracking area in a camera image frame (Yilmaz, Javed, and Shah, 2006).

Secondly, the object can manifest its general shape as a rectangle or ellipse, such as (c) and (d) in Figure 3 (Comaniciu, Ramesh, and Meer, 2003). In fact, the precise tracking of motion in an image frame is not simple. Because of this, object motion can be generated into models by translation, affine or projective transformation.

The third method is to define object contour region such as the boundary of an object (Figure 3 (g), (h). Moreover, there is a representation called a silhouette, which is the method to draw only the silhouette of the object with dots or line. Figure 3 (i), shows the example of drawing inside part of the object contour. These methods are useful for tracking the object which has complex non-rigid shapes (Yilmaz, Li, and Shah, 2004).

Fourthly, one of the methods that characterize object representation is the use of articulated objects, such as human and animal shapes. In addition, these shapes (such as the human body) are composed of torso, hands, feet, heads and legs connected by joints. This method can be described as the expression of the relationship between governed parts by using a model of kinematic motion, with ellipses or cylinders. (Figure 3 (e) (Yilmaz, Shafique, and Shah, 2003)).

Finally, one of the representation methods termed ‘skeletal models’, can extract the adaptation of the transformations of medial axis to the silhouetted object (Ballard, and Brown, 1982). Generally speaking, the shapes of these objects can combine both articulated and rigid object models as in (f) in Figure 3 (Ali and Aggarwal, 2001).

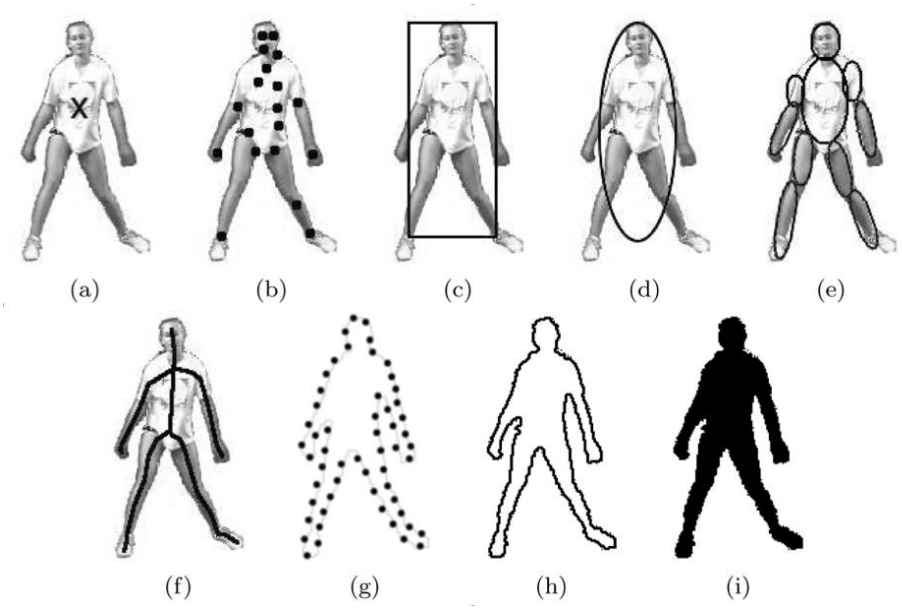


Figure 3 object representations: (a) centroid, (b) multiple points, (c) rectangular patch, (d) elliptical patch, (e) part-based multiple patches, (f) object skeleton, (g) complete object contour, (h) control points on object contour, (i) object silhouette (Ali and Aggarwal, 2001).

In short, the interpretation of various representations offers a starting point for image processing in video frames. Because this project uses specified simple objects, objects characterized by point, rectangle and ellipse shapes are stable and effective methods.

2.1.1.2 Feature Selection

Correctly extracting the features of objects for image-recognition plays a critical role in the tracking process. Moreover, it can be argued that in a given frame, the most desirable point and unique property of the object will be how it is distinguished from the perceived other space of the object's feature (Yilmaz, Javed, and Shah, 2006). In addition, the methods for the feature selection may have a deep relationship with the object representation; that is the reason why selecting the features can be used alone or in combination with object representation methods. For instance, coloured feature selection using a specific colour range based on a histogram of the image, could be said to incorporate colour appearance representation. Edges tracking method is also based on point or contour for object representation.

Many algorithms for tracking commonly use the combined feature selection based on the object representation. The fundamental factors for feature selection which are used in this project include: colour, edges, optical flow and texture selections.

The specific details of these features are as follows:

**** Colour***

There are two physical factors that influence the apparent colour of an object. These are: surface reflecting properties and the spectral power distribution of the illuminant (Yilmaz, Javed, and Shah, 2006). The RGB colour space is able to represent colour in image processing. Normally, however, the RGB is not based on colour space. Thus, it probably means that there is a difference between colours in the RGB space and colours perceived by humans (Paschos, 2001). In addition, the perceptual uniform for colour spaces, such as *L*u*v** and *L*a*b**, may not have to be completely correlated with each other, but the RGB dimensions are quiet opposite. For example, HSV of HSB (Hue, Saturation, Brightness value) is the approximate uniform of colour space. Because of this, the HSV colour spaces have vulnerabilities to noise and other external factors (Song, Kittler, and Petrou, 1996). As a result, it could not be argued that tracking by using colour space is extremely efficient. Furthermore, a diversity of colour spaces may be a useful method of selection which offers a great deal of possibilities in tracking.

**** Edges***

Canny Edge detector I is the most popular edge detection algorithm because of its simplicity and accuracy (Canny 1986). Usually algorithms trace object boundaries by using edges as the representative feature of the object. Object boundaries are probably one of the strongest factors by which changes and differences between different objects can be identified. With regard to recognizing edge features, the most important thing is that the algorithm is not easily influenced by illumination changes, when it is compared to colour features as noted above (Yilmaz, Javed, and Shah, 2006).

**** Optical Flow***

Yilmaz (2006) states that an optical flow is a dense field of displacement vectors translating each pixel in an image region (Yilmaz, Javed, and Shah, 2006). To obtain the optical flow, the main feature is typically the brightness constancy, which corresponds to pixels in successive frames like a video (Horn, and Schunk, 1981). Commonly, the optical flow factors are used as a motion feature in dynamic motion segmentation and object tracking programs (Black and Anandan, 1996). Because of this, the optical flow is relevant for recognizing the direction of moving objects.

**** Texture***

Recognizing features with regard to textures commonly measures the intensity modification of a surface such as regularity or smoothness properties. When texture is compared to colour, there is one difference in the processing step, which involves the use of descriptors. With regard to texture descriptors, the various methods often include Law's texture measures, wavelet and Steerable pyramids (Yilmaz, Javed, and Shah, 2006; Laws, 1980; Mallat, 1989; Greenspan, Belongie, Goodman, Perona, Rakshit, and Anderson, 1994).

Law's texture measures uses 25 masks obtained from 5 one-dimensional vectors, which stand for level, spot, wave, edge and ripple (Laws, 1980; Yilmaz, Shafique, and Shah, 2003). Wavelets decomposition, however, which includes specifically scaled and targeted information, can be attained through orthogonal banks (Mallat, 1989). Finally, the steerable pyramids are composed by the wavelet-based function of over-complete type, which is directly derived from operators in different conditions, such as size or orientated information.

In short, feature selection methods incorporate a number of factors: colour, edges, optical flow and texture. These are the fundamental factors in order to process images.

2.1.1.3 Tracking using colour model

As mentioned before, colour is one of the major factors for feature selection (Yilmaz, Javed, and Shah, 2006). In spite of changeable accuracy due to the brightness of environments, a colour-based extracting method can be used to track not only a non-rigid object but also rigid human faces or motions.

Generally, a colour-model can be obtained by a represented histogram of colours in the range of HSV (Hue Saturation Value) colour space. The HSV colour space is used in order to separate chromatic image data from shading part (Perez, Hue, Vermaak, and Gangnet, 2002).

For this, tracking by using a colour model may need to use a rigid object which has firstly not too small a saturation capacity, and secondly, brightness value in HSV colour space. That is the reason why the colour information is reliable and sensitive to the smallest of difference in the colour values. Therefore, this method could be used as standard practice to ignore colour points under specific thresholds. Thus, these parts can be populated by a black or white colour.

When the pixel has satisfied the HSV values between two specific thresholds, the HS histogram can be generated as a graph composed of $N_h N_s$ bins and an additional N_v value. Hence, the HS histogram for each pixel can be defined as $N = N_h N_s + N_v$ in such thresholds (Perez, Hue, Vermaak, and Gangnet, 2002; Perez, Hue, Vermaak, and Gangnet, 2002).

In short, colour-based tracking uses the difference of the colours between specific thresholds in HSV colour space. In addition, each pixel in the range of the thresholds can be filtered by these thresholds and can remain as white or black states as well as grey level states.

2.1.2 Target position

The exact target position is an equally important part in measuring the distance between an object and the observer. Additionally, in order to calculate the position of the object in this project the tracking algorithm needs to be applied. However, as mentioned above, tracking colour based on non-rigid objects has weakness because it is extremely sensitive to noise. This means that recognizing coloured objects using HSV or RGB colour spaces is affected by the environments surrounding the object. For this reason, recognizing image information which allows for a high error rate is one way of developing more accurate object tracking. This part introduces classifications for pattern recognition and analysis.

2.1.2.1 K-means clustering

According to the classification of algorithms as suggested by MacQueen (1967), K-Means clustering can classify object attributes or features into K number of groups (Teknomo, 2007; MacQueen, 1967). The K number is an integer type, and grouping can be done by using a K number. The cluster centroid and groups can be decided by minimizing the sum of the distance between data and the randomly chosen cluster within the grouping process.

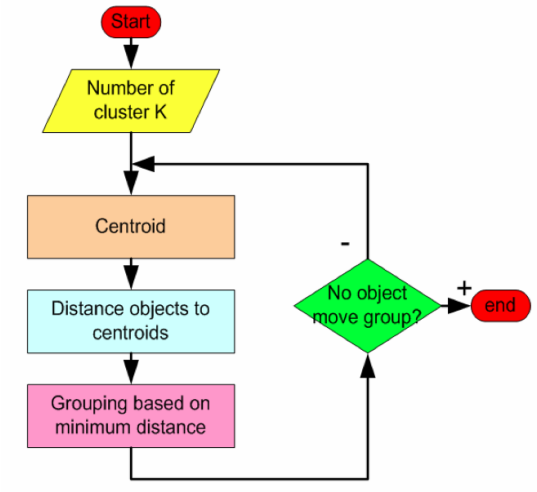


Figure 4 Flow chart of the K-Means clustering (Teknomo, 2007).

As can be seen from the above flow chart, the steps of K-Means clustering are simple. Firstly, it is necessary to determine the number of clusters before assuming the random centroid of clusters. The next step is to measure the distance from the centroid cluster to each data (Cortes and Vapnik, 1995).

Finally, the grouping process can be performed based on the minimum distance. Unless the program has a restraining condition to control infinite repetition, the process will be repeated unlimitedly. Because of this, if there is no change of the data in each group, it should possess a general alternative as it forces the process to stop within a few cycles of repetition (Chen, Luo, and Parker, 1998).

The objective function for K-means clustering can be defined as follows.

$$J = \sum_{j=1}^{k} \sum_{i=1}^{n} \left\| x_i^{(j)} - c_j \right\|^2 \quad (1)$$

Where $\left\| x_i^{(j)} - c_j \right\|^2$ is a calculated distance between a data point $x_i^{(j)}$ and a randomly chosen centroid cluster c_j this indicates the sum of the distance from the chosen centroid cluster to each data point.

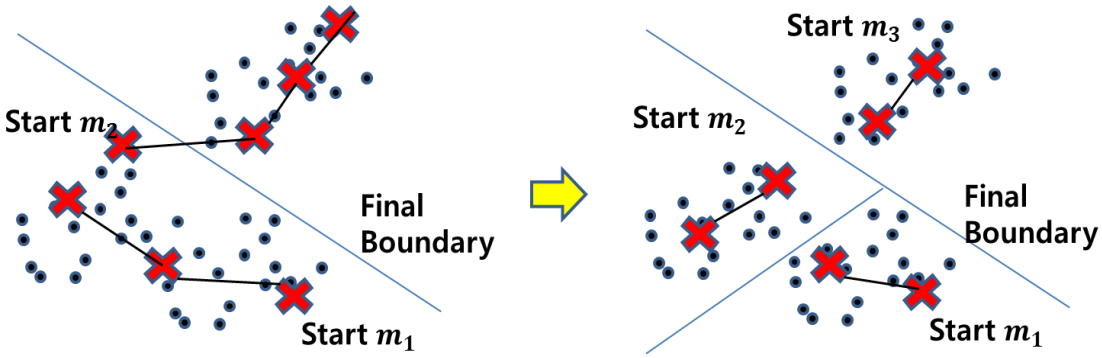


Figure 5 examples to show how the points move into the centre

As an example, Figure 5 above shows how the mean value m_1 and m_2 moves into the central area of data space (Chen, Luo, and Parker, 1998). Furthermore, it can be seen from the right side of Figure 5 that in each grouped space, the centroid cluster can be determined by the repetitive process which calculates the sum of the minimum distances between the centroid cluster and other data points in a group (Matteucci, 2003).

Despite the simplicity of using the K-means clustering algorithm, this method has the following weakness (Matteucci, 2003). Firstly, the initializing method for the means in each data could not be specified, which is to say that generally the samples of means are chosen randomly. Because of this, it is unable to determine the length of time it will take to decide final boundaries and centroid clusters. Furthermore, its results either frequently make sub-optimal partitions or one group reveals empty sub-data (MacQueen, 1967). To solve this, many people commonly use different starting points, but this does not solve the problem perfectly (Matteucci, 2003). Finally, this process needs to decide upon a convoluted K number. For instance, while a video includes plenty of image frames and the noise of each image needs to be different. Thus, the process cannot decide on a fixed K number for K-Means clustering

algorithm. Therefore, it needs a flexible change of K number depending on each image (Teknomo, 2007).

In conclusion, the K-Means clustering algorithm is a potentially effective method for the classification. However, the K number can generate completely different results. As a consequence, this method requires the consideration of specific conditions depending on the particular settings of different data situations.

2.1.2.2 SVM (support vector machines)

In the case of an image obtained by an embedded camera, the image can be stored and termed as in a 'state with plenty of noises'. Hence, sometimes, in classifying it by linear classification, the algorithm is limited (Cortes and Vapnik, 1995). Specifically, the data filtering by colour-tracking method is distributed over a wide range, and it includes similar points which can be found in the colour range of HSV. As a result, effectively distinguishing the data in reference to its exceptions, thus using the linear classification method with a margin in the SVM is advised. Because of this, the method will introduce the linear classifications of an SVM algorithm.

As a linear classification, SVMs algorithm use the maximum margin, which was suggested by Cortes and Vapnik in 1995, in order to find the optimal boundary hyper-plane by using margin value within at least two classes (Bennett and Campbell, 2000; Meyer, 2009; Gunn, 1998). In addition, it is used to calculate the maximum margin value with the closest points on the support vectors boundaries. This process allows the mean value of the margin to be expressed as the optimal hyper-plane (Cortes and Vapnik, 1995).

Figure 6 shows a simple example by using the linear classification. In this case, the optimal boundary can be defined as the furthest hyper-plane from each data group. In other words, it could be a hyper-plane passed between two data groups, and the closest vectors from the hyper-plane can be defined as the support vector.

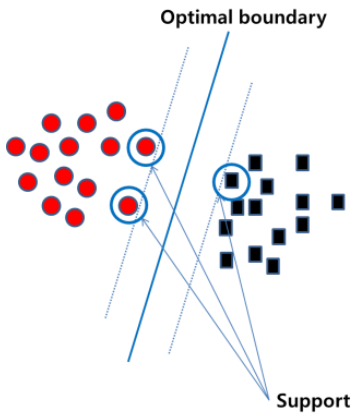


Figure 6 the best plane for the maximized margin (Cortes and Vapnik, 1995)

Firstly, one separating hyper-plane can be formulated as follows, when w is a weight coefficient vector, and b is bias (Gunn, 1998).

$$w^T x = b + 1 \; or \; w^T x = b - 1 \qquad (2)$$

As the distance between the training vector x_i and separating hyper-plane, the margin can thus express the following formula (Cortes and Vapnik, 1995).

$$r = 2/\|w\|_2 \tag{3}$$

Finally, the maximum distance between the support vector and optimal separating hyper-plane can be calculated by minimizing $\|w\|_2/2$ within the quadratic program, as the following shows:

$$\mathbf{min}_{w,b}\, \frac{1}{2}\|w\|^2 \tag{4}$$

$$\boldsymbol{s.t} \quad y_i(wx_i - b) \geq 1 \tag{5}$$

However, the above case can only follow the methods of a linear classification, and so could have the errors demonstrated in Figure 7.

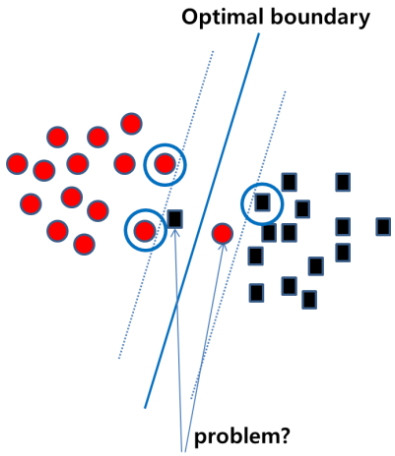


Figure 7 hyper-plane with error of maximize margin

However, these errors can be solved by adding weight to the quadratic program with the variable z_i for an error (Cortes and Vapnik 1995; Meyer, 2009; Gunn, 1998).

$$\mathbf{min}_{w,b,z}\, \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{l} z_i \tag{6}$$

$$\boldsymbol{s.t} \quad \begin{cases} y_i(wx_i - b) + z_i \geq 1 \\ z_i \geq 0 \; i = 1, \dots, m \end{cases} \tag{7}$$

Therefore, as the Lagrangian dual of the quadratic program, finding the closest data point can be summarized in the following total equation:

$$\min_a \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} y_1 y_j a_i a_j x_i x_y \; - \sum_{i=1}^{m} a_i \tag{8}$$

$$\boldsymbol{s.t} \quad \begin{cases} \sum_{i=1}^{m} y_1 a_i = 0 \\ C \geq \boldsymbol{a_i} \geq 0 \quad i = 1, \dots, m \end{cases} \tag{9}$$

Besides this, there is also a non-linear SVM method. However, in the entangled case without specific characteristics which should use non-linear SVM, the case state could mean that the image obtained from the camera has attracted a large amount of noises or that the detected co-ordination could be wrong. Hence, the image information should be ignored.

In conclusion, image data with plenty of noises is distinguished by introducing a linear soft margin in the SVMs algorithm.

2.1.3 Measurement of the distance

The method of efficient application of reinforcement learning is one of the most important tasks, and requires a method that can measure the exact distance between an object and a robot. However, a vision measurement based on exact distance values is difficult to achieve because of the impact of noise on the measuring methods. Therefore, a method based on this strategy should try to approach an accurate value by using an algorithm as specific as possible. To achieve this task, this part will introduce methods related to the measurement of distances. This includes such processes as: comparisons with the template images and the comparison of approximate object sizes by counting the number of filtered pixels.

2.1.3.1 Measurement by using template

A template matching algorithm is commonly used to detect an object (Hajdu and Pitas 2007). However, this has the possibility of measuring the distance to an object using a similar template matching method with the assumption that we can know a specific object size (Hajdu and Pitas 2007). The following figure shows general template matching process using correlations x, y.

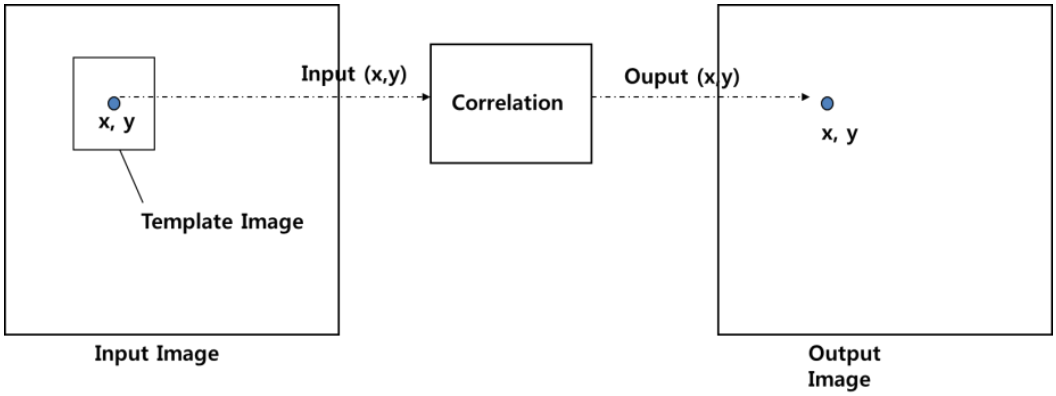


Figure 8 template matching architecture (Cole, Austin and Cole, 2004)

The above template matching method can be used to measure the Euclidean distance, pixel by pixel if the template forms some part of an image. In fact, it is also possible to measure the distance to the object by using normalized cross correlation (Lewis, 1995). However, it may be useful when an image includes plenty of similar object shape with template image. Because of this, we can think of another way of comparing template images with filtered object features, which can be similar to the template matching method.

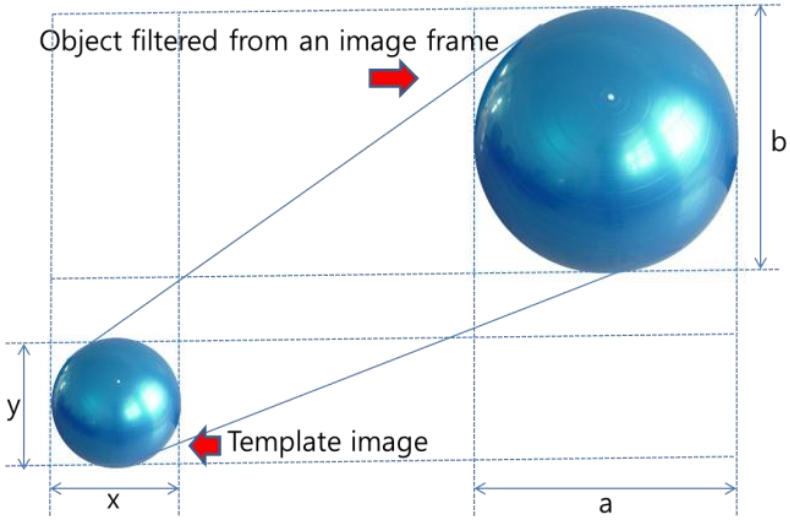


Figure 9 Distance measurement by using saved template

The above figure describes the architecture of the measuring method by a comparison of a filtered object image and stored template image. In this case, the method should be used under the assumption that we know the distance d of template image size. The following formula summarizes the above process.

$$\begin{cases} x : d = a : D \\ D = \frac{a \cdot d}{x} \end{cases} \quad or \quad \begin{cases} y : d = b : D \\ D = \frac{b \cdot d}{y} \end{cases} \tag{10}$$

Where the distance of object is D, the distance can be calculated by the above proportional formulation. For this, however, it requires prior measurement for the template distance, as well as a fixed memory for the template image.

2.1.3.2 Measurement by using filtered out pixels

In another method, the object distance can potentially be approximately measured by calculating the number of filtered object image pixels. This method requires absolute accuracy for object detection but may be faster and easier than using the template method because it only compares the number of pixels. Thus, using the template method involves preparing a prior template image, and while comparing the number of pixels in an image can be used without preparation of templates, there is a considerable risk to disruptive elements such as noise and image brightness (Gupta and Pati, 2009).

Typically, the 'cvHoughCircles' function of the 'OpenCV' library can approximately describe the filtered object size with circles by column and rows that include the number of lines detected as specified colour pixels (Intel Corporation, 2001).

```
public:
static IntPtr cvHoughCircles(
          IntPtr image,
          IntPtr circleStorage,
          HOUGH_TYPE method,
          double dp,
          double minDist,
          double param1,
          double param2,
          int minRadius,
          int maxRadius
)
```

Figure 10 'cvHoughCircles' function for Visual C++ in OpenCV libraries
Source: *http://www.emgu.com/wiki/files/1.3.0.0/html/0ac8f298-48bc-3eee-88b7-d2deed2a285d.htm*

However, the accuracy of this method actually relies on the image and the captured environment. The next figure shows the wrong measurement of the image size.

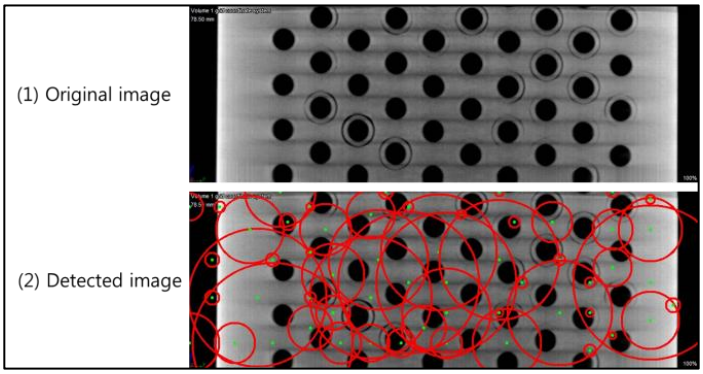


Figure 11 example of unwise use of 'cvHoughCircles' function
Source: *http://cafe.naver.com/opencv.cafe?iframe_url=/ArticleRead.nhn%3Farticleid=9205&*

As can be seen from the above picture (2), the size of detected circles as shown in the two images is completely different. In this case, the colour of grey level may reveal serious similarity, indicating that the colours are too similar to distinguish.

In short, the method using filtered pixels has an uncertain rate of accuracy, but its advantage is that it can readily adapt to the embedded system, so that its processing speed is very fast, and that it takes up a little memory space.

2.2 Reinforcement learning

An embedded vision-based robot can be seen as an agent in reinforcement learning algorithm, however its recognizing range is limited. In spite of this, the robot should be able to repeat behavioral acts in real time. However, it is quite difficult to distribute commands considering the varied environments open to the robot. Because of this, the robot may need to find its optimal behavior by itself in reaction to different environments.

In unspecified environments, adapting reinforcement learning is one of the most effective methods to optimize robot movement. In contrast to supervised learning, a reinforcement learning algorithm does not include specified representative inputs or outputs (In-Cheol, 2002).It could mean that the robot could approach the optimal action policy as distributing reward in specified tasks. For example, Mahadevan and Connel (1993) proposed reinforcement learning as a method of rapid task learning for real robots (Connel and Mahadevan, 1993). As a result, they observed successful performances in tasks involving goals such as carrying an object to an appointed location (Asada, Noda, Tawaratsumida and Hosoda, 1996). In such a scenario, we may assume that reinforcement learning can help improve robot movement. Thus the next part of the process will try to introduce reinforcement learning algorithms which can be adapted to robot movements.

2.2.1 Basic concept of reinforcement learning

Reinforcement learning can be defined as a learning method from which one or more agents can obtain optimal policy from rewards. This optimization is accumulated by distributing award or punishment to the agents in unspecified environments (Peters, Sethu and Stefan 2003; Dayan and Watkins 1992). As a goal-oriented learning method, reinforcement learning is characterized by the trial and error method, which involves delayed reward and interaction within environments. Generally speaking, with reinforcement learning the real performance of an agent is normally decided by following action policy π in the given state S_t (In-Cheol, 2002). After that, the reward about the real action, scalar value r_t, is supported to reward the agent, and then it will be changed to new state s_{t+1}. All of these above processes are subsequently repeated, and can be reflected by the following formula (In-Cheol, 2002; Dayan and Watkins, 1992; Connel and Mahadevan, 1993; Ono and Fukumoto 1996).

$$V^x(S_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots\cdots = \sum_{i=0}^{\infty} \gamma^i r_{t+i} \quad (11)$$

$$\pi^* \equiv \text{argmax}_\pi V^x(s), (\forall_s) \quad (12)$$

, where the discount rate is represented as γ.

In the first formula, the $V^x(S_t)$ represents a value function as a set of the value at the specific state s_t, which can calculate the expected reward with the sum of rewards following action policy π. The second formula defines optimal action policy π^*, and means that the value function can be optimized at every possible state (Watkins, 1989).

2.2.2 Q-learning

As one algorithm of value function estimation, Q-learning (Watkins, 1989) cannot have specific models. This means that Q-learning algorithm can mostly approach an optimised action policy without pre-selected specific state models if the learning process repeats infinitely or sufficiently. Because of this, it is possible for reinforcement learning to be adapted to this project. For the purpose of including a Q-learning algorithm, it requires the Q-value function to determine the action policy. Thus, the function to update Q-value is as follows (Ono and Fukumoto, 1996).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha_t(s_t, a_t)\left[R(s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right] \quad (13)$$

Where discount factor is γ and reward is $R(s_{t+1})$ from the state s_t and action a_t, the old value $Q(s_t, a_t)$ can be obtained from the above formula. In this case, the discount factor γ is between 0 and 1. Due to this the above formula can be rewritten as follows (Ono and Fukumoto, 1996; Kaelbling, Littman and Moore, 1996).

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t)(1 - \alpha_t(s_t, a_t)) + \alpha_t(s_t, a_t)\left[R(s_{t+1}) + \gamma \max_{a_{t+1}} Q(s_{t+1}, a_{t+1})\right] \quad (14)$$

In fact, however, the Q-learning algorithm could face real-world problems, where Q-table is too large or complex. The reason is that Q-learning may not be converged to optimal policy for the action if the size of elements such as each state or action is too large to store at the Q-table (In-Cheol, 2002; Kohri, Matsubayashi, and Tokoro, 1998).

2.2.3 Modular Q-Learning(MQL) and Automatic Modular Q-Learning (AMQL)

In a diversely changeable image environment, storing each state with real-world problems is one of the most difficult aspects of this project. To solve the above problem, it is suggested that diverse methods reduce the number of models in state environments by sub-dividing the stored elements of the Q-table (Kohri, Matsubayashi, and Tokoro,1998) Within them, one of the most effective methods is the Modular Q-learning algorithm developed by Whitehead (1990); Whitehead, and Ballard 1990; Ki-Duk, and In-Cheol, 2001). This algorithm uses a method in which agents are individually learned by using separate and fewer modules. Based on this process, the optimal policy is decided by combining the results that have been learned (Littman, Cassandra, and Kaebling, 1994). However, this method can be difficult to adapt to extreme changes in the environment because it also uses steady method called GM (Greatest Mass Strategy). The formula, which describes the greatest mass strategy, is as follows (In-Cheol, 2002).

$$a^* \leftarrow \text{argmax}_{a \in A} \sum_{i=1}^{n} Q_i(s, a) \quad (15)$$

Where the present state is s, the optimal action value a^* is selected by a maximized sum of each module i, thus, Q_i. The architecture of MQL is the next figure.

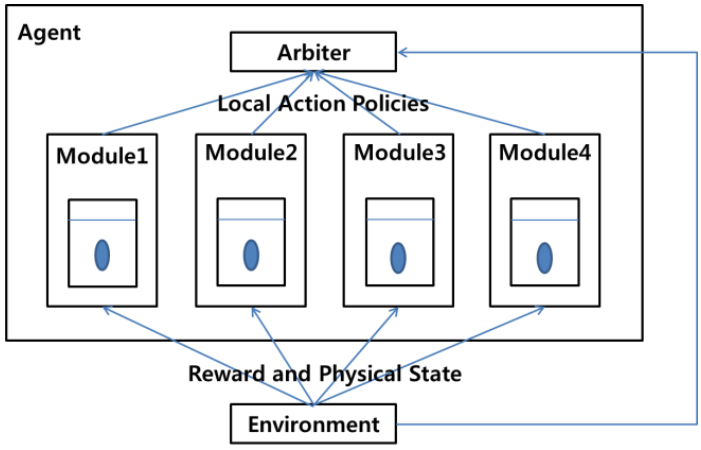


Figure 12 Modular Q-Learning architecture (Ki-Duk and In-Cheol, 2001; Kohri, Matsubayashi and Tokoro, 1998).

In addition to this method, Kohri (1998) suggested that AMQL (Automatic Modular Q-Learning) used in combination with MQL could offer an extended solution for mirroring real-world problems. The AMQL algorithm is one in which the agent makes suitable module groups by itself with respect to changeable environments (Kohri, Matsubayashi, and Tokoro, 1998).This method can be divided into three steps. The first is to compose each module selected from environmental factors which describe a state space. The second step is to evaluate the suitability of the modules depending on the level of contribution at policy, while it processes the MQL. Finally, the algorithm filters modules that have a lower suitability than the specified threshold in order to find an optimal policy, and then reconstructs a module group with the new selected environmental factors. This method is highly relevant for the agent which reacts quickly to changes in a diverse environment. However, this method may experience problems if it expects speedy effects to the embedded robot in real-time. This is because this process requires the use of extra learning tasks and time to decide the relevant modules (Kostiadis and Hu, 1999; Kui-Hong, Yong-Jae, and Jong-Hwan, 2001).

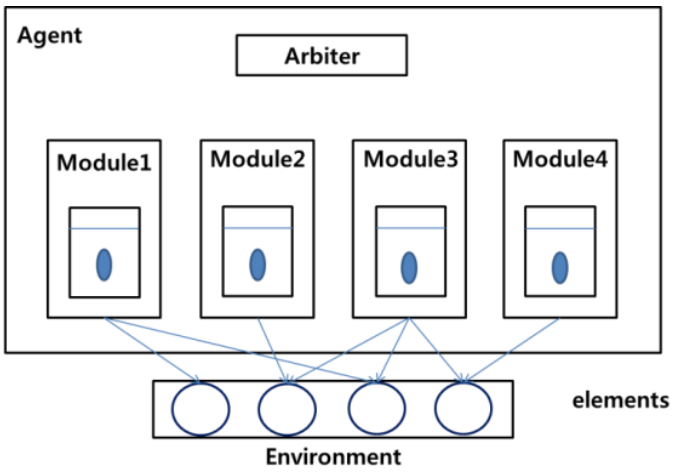


Figure 13 AMQL architecture (Ki-Duk and In-Cheol, 2001; Kohri, Matsubayashi and Tokoro, 1998).

In short, both MQL and AMQL are kinds of Q-learning methods that help to solve the state space problem of Q-learning. However, MQL when using fixed modules and AMQL when using unspecified modules still have weakness with regard to their efficiency and adaptability to diverse environments. Nevertheless, if this method efficiently reduces the number of modules, then the adaptation of MQL or AMQL to this project may not be a big problem.

2.2.4 Adaptive Mediation based Modular Q-Learning (AMMQL)

Beyond MQL and AMQL, it was suggested that an AMMQL based on the adaptive mediation might be a possible method to solve too big a Q-table (Ki-Duk and In-Cheol, 2001). Typically, AMMQL has been used within robot soccer simulation which often needs to handle a great deal of information data in real time. In line with this, this project should also be able to handle plenty of image data within a changeable environment. Hence, using these techniques in this project may offer a possible solution for the real-world problem.

In particular, the AMMQL is a method which dynamically determines the combining method of modules in order to adapt to a changeable environment rapidly and responsively. Moreover, the method supports higher adaptability to environment and achieves a greater efficiency with low cost than MQL (In-Cheol, 2001). The next figure shows the architecture of the AMMQL composed with n modules. The rewards from the environment depending on the agent's actions are passed to each module and adaptive module. After that, the adaptive module judges Q-value weight based on the reward contribution of each module. Thus, it could be said that AMMQL uses a double learning structure.

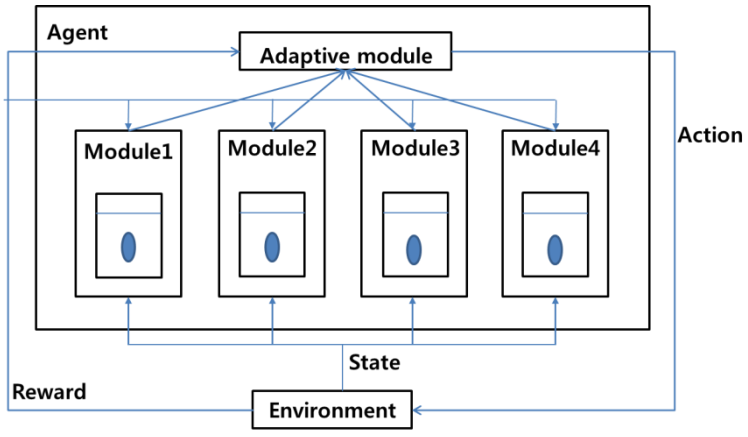


Figure 14 AMMQL architecture (Kohri, Matsubayashi and Tokoro, 1998)

When each state before and after modularising pre-supposes as $s_x \in$ S and $s_{1x}, s_{2x}, \cdots, s_{3x}$, the next formula represents optimal policy a^* which combines the weight with Q-value of each module.

$$a^* = \text{argmax}_{a \in A}\left(\omega_1 Q_1(s_{1x}, a) + \cdots + \omega_n Q_n(s_{nx}, a)\right) = \underset{a \in A}{arg\max} \sum_i^n \omega_i Q_i(s_{ix}, a) \qquad (16)$$

In this case, the sum of each weight ω_i is 1 (Ki-Duk and In-Cheol, 2001).

Subsequently, each module can compare Q-value rate between a^* and possibly every policy with an obtained reward R.

$$r_i = \frac{Q_i(s_{ix},a^*)}{\sum_{a_m \in A} Q_i(s_{ix},a_m)} \tag{17}$$

Where a is the learning rate, the temporal weight ω_i can be calculated, and used as new weight through normalization, as follows.

$$\omega'_i \leftarrow \omega_i + a \cdot r_i \cdot R \tag{18}$$

$$\omega_i = \frac{\omega'_i}{\omega'_1+\omega'_2+\cdots+\omega'_n} \tag{19}$$

As a result, with repetition of the above process, AMMQL can process modular Q-learning, which adapts more sensitively to changeable environment, without a large amount of extra cost (In-Cheol, 2002; Ki-Duk, and In-Cheol, 2001).

In this project, one of the biggest problems is how to control the size of the Q-table for each state. Therefore, using advanced AMQL or AMMQL methods may lead to more improved robot performance based on vision algorithm.

3. System configuration

A system's development environment is one of the most important parts of a system's configuration. Different development environments are able to express a diversity of development processes and consequences. For example, with regard to the specifications of memory, various attachments or MCU can display different capabilities such as varied system speed and image quality in the same software. In this respect, the following part will explain the essential factors for a system configuration, primarily as they relate to the Beagle-board and IRobot.

3.1 Beagle-board

Figure 15 Beagle-board xM Rev B (Tran, Gerdzhey and Ferwom, 2010)

The main part in configuring an embedded system is choosing a stable board which can handle the demands of the whole system. As a powerful single board computer based on the Linux system, the Beagle-board has a number of attractive advantages and plays a pivotal role for the system (Tran, Gerdzhey and Ferwom, 2010). For example, it includes an essential proper form to support 4 USB host ports, Ethernet, Serial port, HMDI and SD card slot. Furthermore, by using a low voltage in the board it satisfies one of the conditions of a composite embedded system, which is that the Beagle-board must be able to manage all of its components with only a 5V external power supply (excluding the power demands of a robot (Osier-Mixon, M. J., 2010).) Therefore, it could be said that the board is an adequate and effective system to replace the personal computer for the embedded system.

After choosing the board device, this process requires the installment of an operating system and the setting-up of other attachments to make a complete system model. As such, detailed tasks will be simply explained; in particular, the following section will explain what operating system to choose, how to boot the board, and what kinds of camera will be used. Detailed explanations of the board interface can be found in the Appendices 1 section (Beagle-board.org, 2011).

3.1.1 Operating system: Natty 11.04 of Ubuntu

The operating system is able to influence the program by different compiling methods with libraries or controlling system kernels. The Ubuntu based on the Linux system, can provide three advantages for this project. Firstly, it makes the developer work under a GUI platform. Another thing is that the Ubuntu supports the varied versions with compatibility for embedded boards. Finally, controlling and updating system kernels are one of the biggest advantages of the Ubuntu. Moreover, with the Ubuntu it is easy to download essential applications and libraries as packages. One

disadvantage, however, is that it needs to update to the latest version continually for smooth development.

3.1.2 Booting Method: SD card (8GB)

To boot the Beagle-board, the board supports three methods by using SD card, serial port by the RS-232 transceiver or Ethernet networking. However, because the board supports only one serial port, it could not use the port for both the serial communication with a robot and booting. Besides, the Internet should be considered in terms of using limited place. Because of these limited factors in the computing environment, the booting method by SD card was seen as the best option for this project. Booting with SD card has an advantage in that it is flexible in memory size.

To boot the Beagle-board by SD card more than 2GB, it needs a special formatting process in order to select complex options correctly within the process, depending on the operating systems. In this case, because the operating system was chosen as the Ubuntu natty, it will describe the installing method related to the Ubuntu.

Firstly, for the operating system of the Ubuntu or other Linux version, it needs to divide and format SD card simultaneously as two partitions under any operating system. The first thing is to create the FAT 32 space for system booting and transferring files from Windows (Beagle-board, "Linux Boot Disk Format). Another partition requires the root file system as the Linux partition. The FAT 32 partition does not demand big space, but it could be the general method which is to assign the space as approximately 63 sectors in the total 255 sectors. As another step for the SD card booting, after dividing partition and formatting, it needs to set up necessary booting factors directly for the mounting of a root file system for the Linux partition. Most important, the location of root priority, baud-rate for the communication and serial port name for the console are essential factors if the Linux and Ubuntu use U-Boot as a boot-loader. However, it could be possible whenever setting up or changing those factors to other values. To sum up, the SD card booting method is as yet the most suitable and easiest method on the Beagle-board.

3.1.3 Camera: PS3 USB Camera

Figure 16 appearance of PS3 eye USB camera

This project includes the main roles of obtaining and handling video stream in real time from camera. To use embedded vision algorithm, the essential process is the checking capability of the camera; this is the main reason why some products cannot be supported by drivers to recognize and work by the operating system. Because of

this, the system needs to be checked by the driver after connecting the camera to the board, in order to establish whether the camera has a problem to input and output frames or not, because this factor may influence the total vision algorithms capacity, which could result in failure or inaccurate results. To sum up, for the checking process, two simple steps are required: firstly, check whether the operating system has a PS3 USB Camera module of a correct version by using a 'dmesg' command in the terminal. Secondly, test the capability of image frames by simple media application such as 'vlc', 'Cheese' or 'Mplayer'. In this project, the 'Mplayer' application was used to test and the device was mounted as 'video0'.

3.2 Robot: IRobot create

Furthermore, in this project a robot is defined as having substantive existence by showing visual movements and standing up to the testing of integrated algorithm. Because of this, background knowledge about a pre-producing robot is essential for smooth robot performance to be exhibited.

Figure 17 Appearance of the IRobot (IRobot Corporation, 2006)

As demonstrated by the above basic figure. However, unlike its appearance, its usable functions vary as well as optional music play, Bluetooth or serial communication.

To adapt this robot to the project, making movements and communicating with the main board will be the essential main tasks. Firstly, to make communication with the Beagle-board, a serial port name is required with the inclusion of baud-rate if using serial communication. In order to make a moving robot, the control of robot components such as motor and sensor need to be known. Fortunately, this project will not use the sensors to calculate distance between an object and a robot. Because of this, for the movement factor, it will focus only on how to control the servo-motors and communication function.

In the next 'Robot programming' part, the essential robot interface requirements will be explained in detail, how to set serial port and motors, and program for the moving robot.

4. Robot programming

The following 'Robot programming' part will include two large works of serial communication programming and robot movement programming.

4.1 Serial communication

Generally, serial communication is defined as a kind of sending signal method which can directly connect to computer components with various serial cables. Meanwhile, it can also send each 1 bit immediately through TX and RX. Here, the TX and RX can be seen as the Transmitter and Receiver Crystal (X-tal). Hence, to conduct the serial communication, firstly, it needs to examine the interface (or so called port) type, because this will affect the method setting as well as values (Kui-Hong, P., Yong-Jae, K., and Jong-Hwan, K., 2001; Axelson, J., 1998).

In this project, the Beagle-board can support 9 pins mini DSUB RS-232, and the common usage of each pin number are summarized in the below table.

Mini DSUB connector	Signal title	Simple using method
1	CD(Data Carrier Detect)	Input, not use
2	RD(Receive Data)	Input, connect to other TD
3	TD(Transmit Data)	Output, connect to other RD
4	DTR(Data Terminal Ready)	Output, not use
5	SG(Signal Ground)	Ground, connect to other SG
6	DSR(Data Set Ready)	Input, not use
7	RTS(Request to Send)	Output, connect to other CTS
8	CTS(Clear to Send)	Input, connect to other RTS
9	RI(Ring Indicate)	Input, not use

Table 1 Pin number and signal of mini DSUB connector, and simple using method

After examining the serial ports type, the name of mounted serial port name will be known. Thus use the relevant communication speed ('baud-rate') to connect to the system, as it will be easier to conduct the serial communication programming. However, the programming method can vary on different operating systems. For example, serial programming on Windows does not need to use the 'termios' structure because the Windows can support and catch the serial port automatically when the serial port is linked with RS232 cable After examining the serial ports type, the name of mounted serial port name will be known. Thus use the relevant communication speed ('baud-rate') to connect to the system, as it will be easier to conduct the serial communication programming. However, the programming method can vary on different operating systems. For example, serial programming on Windows does not need to use the 'termios' structure because the Windows can support and catch the serial port automatically when the serial port is linked with RS232 cable (libcreateoi, Create Open Interface Library, 2011; UNIX, 2001, "Specification, Version 2: <termios.h>").

Nevertheless, on the Linux environment, using the 'termios' structure is imperative, because the project is based on Ubuntu. Thus the 'termios' structure will support 5 controlling modes of 'input', 'output', 'control', 'local', and 'specific' (shown in Figure 18), and is defined as an interface standard by POSIX.

```
#define NCCS 19
struct termios {
    tcflag_t c_iflag;       /* input mode flags */
    tcflag_t c_oflag;       /* output mode flags */
    tcflag_t c_cflag;       /* control mode flags */
    tcflag_t c_lflag;       /* local mode flags */
    cc_t c_line;            /* line discipline */
    cc_t c_cc[NCCS];        /* control characters */
};
```

Figure 18 'termios' structure in <asm/termbits.h>

This can check the mounted port name as '/dev/tty**', and normally it can be mounted between ttyO0 and ttyO3 on the Beagle-board (Beagle-board.org, 2011, "Beagle-Board-xM Product Details). The simple code for the IRobot using 57600 baud rate is as follows (IRobot Corporation, 2006).

```
if (0 == fd)
{
    fd = open (serial, O_RDWR | O_NOCTTY | O_NDELAY);
    if (fd < 0)
    {
        perror ("Could not open serial port");
        return -1;
    }
    fcntl (fd, F_SETFL, 0);
    tcflush (fd, TCIOFLUSH);

    //get config from fd and put into options
    tcgetattr (fd, &options);
            //give raw data path
    cfmakeraw (&options);
    //set baud
    cfsetispeed (&options, B57600);
    cfsetospeed (&options, B57600);
    //send options back to fd
    tcsetattr (fd, TCSANOW, &options);
}
```

Figure 19 Simple source codes for the serial communication (libcreateoi, 2011)

To summarize, the key point of the serial communication is to compose with the baud rate and device name mounted on the board. Those can be composed easily by using the 'termios' structure.

4.2 Robot movements

After setting the serial port, the achieved state will be ready for a moving robot through the bytes transmission of serial communication. However, before the robot programming, in order to avoid interference in the serial communication, the asynchronous serial communication needs to be considered. Hence, for preventing the interference, the effective way of using and controlling 'multi-thread' is the standard practice (Powell, Kleiman, Barton, Shah, Stein, and Weeks, 1991; Mueller, 1993; IRobot Corporation, 2006). Below the overall process of a robot program adopting multi-thread will be shown.

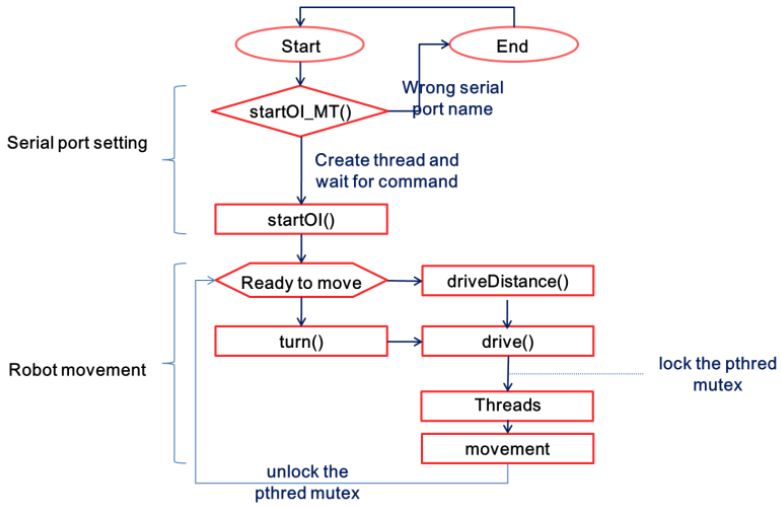


Figure 20 flow chart of the robot performance

This can be seen clearly in Figure 20. It is a simple flow of robot program using threads. In fact after setting the serial port, the robot will enter into a waiting state, which can be understood as the preparation state in waiting for the next command and movement. Hence, the following steps relate to both turning and straight movements, which have different processes in the robot program as follows: straight and turning movements.

4.2.1 Straight and turning movements

To test the example sources of the straight and turning movement, samples can be easily obtained through IRobot open interface library or other coding sites, such as 'createoi.c' and 'square.c' (IRobot Corporation, 2006). **Figure 21** below can be seen as a 'drive' function in the sample code called 'createoi.c'. In the source code, through 'drive' function, both 'driveDistance' and 'turn' function can be controlled, which in turn, can affect the straight and the turning movement. Besides, it is also possible to make movements directly through 'directDrive' function that is different from the above mentioned functions. In fact, the example source of IRobot moving can not only support the straight and turning movement but also the controlling sensors. However, within the context of this project, it is unnecessary to use the censor control and make sounds. Thus the focus will be on the explanation of moving parts (libcreateoi, Create Open Interface Library, 2011).

As to controlling the methods of robot movements, there are two ways: indirect and direct. Firstly, for the indirect control method, ‘driveDistance’ is used with ‘turn’ functions based on ‘drive’ function which can control robot components with the serial communication of four data bytes.

```
int drive (short vel, short rad)
{
    byte cmd[5];

    //keep args within Create limits
    vel = MIN(500, vel);
    vel = MAX(-500, vel);
    rad = MIN(2000, rad);
    rad = MAX(-2000, rad);

    if (0 == rad)   //special case for drive straight (from manual)
        rad = 32768;

    cmd[0] = OPCODE_DRIVE;
    cmd[1] = (vel >> 8) & 0x00FF;
    cmd[2] = vel & 0x00FF;
    cmd[3] = (rad >> 8) & 0x00FF;
    cmd[4] = rad & 0x00FF;

    pthread_mutex_lock( &create_mutex );

    if (cwrite (fd, cmd, 5) < 0)
    {
        perror ("Could not start drive");
        pthread_mutex_unlock( &create_mutex );
        return -1;
    }
    pthread_mutex_unlock( &create_mutex );
    return 0;
}
```

Figure 21 'drive' function in 'createoi.c' supported by IRobot interface library
Source: *http://libcreateoi.googlecode.com/svn/trunk/src/createoi.c*

The above Figure 21 shows the essential ‘drive’ function to make straight and turning movements. To specify, the ‘drive’ function uses a velocity for straight moving and radius for turning. The given range of velocity is between -500 and +500 (mm/s), and the radius range is between -2000 and 2000 (mm). The negative values of the given velocity and radius can mean the backward and right side movements of the robot direction (libcreateoi, Create Open Interface Library, 2011). For example, in the given radius, -1 can make the robot turn in place clockwise, but 1 turns anti-clockwise. In the case of 0, the robot recognizes it as a command for the straight movement. Therefore, the radius value at the straight movement should be 0.

The following two figures, Figure 22 and Figure 23, are ‘turn’ and ‘driveDistance’ functions using ‘drive’ function. These functions may support safe progress towards the goal, while incorporating the uses of ‘waitDistance’ and ‘waitAngle’ functions. Because of this, the robot can make stable movements similar to an asynchronous thread control.

```
int turn (short vel, short rad, int angle, int interrupt)
{
    int ret = 0;

    if (drive (vel, rad) == -1 ||
        (ret = waitAngle (angle, interrupt)) == INT_MIN ||
        drive (0, 0) == -1)
        return INT_MIN;

    return ret;
}
```

Figure 22 'turn' function in 'createoi.c' supported by IRobot interface library
Source: *http://libcreateoi.googlecode.com/svn/trunk/src/createoi.c*

```
int driveDistance (short vel, short rad, int dist, int interrupt)
{
    int ret = 0;

    if (drive (vel, rad) == -1 || (ret = waitDistance (dist, interrupt)) == INT_MIN || drive (0, 0) == -1)
        return INT_MIN;

    return ret;
}
```

Figure 23 'driveDistance' function in 'createoi.c' supported by IRobot interface library
Source: *http://libcreateoi.googlecode.com/svn/trunk/src/createoi.c*

Another method directly manages the robot movements with 'directDrive' function. As shown in Figure 24, the function is quite similar to the 'drive' function, but it does not allow the performing of different commands at the same time. Thus, it is assumed impossible to make unsynchronized and malfunctioning situations (libcreateoi, Create Open Interface Library, 2011).

```
int directDrive (short Lwheel, short Rwheel)
{
    byte cmd[5];

    //keep args within Create limits
    Lwheel = MIN(500, Lwheel);
    Lwheel = MAX(-500, Lwheel);
    Rwheel = MIN(500, Rwheel);
    Rwheel = MAX(-500, Rwheel);

    cmd[0] = OPCODE_DRIVE_DIRECT;
    cmd[1] = (Rwheel >> 8) & 0x00FF;
    cmd[2] = Rwheel & 0x00FF;
    cmd[3] = (Lwheel >> 8) & 0x00FF;
    cmd[4] = Lwheel & 0x00FF;

    pthread_mutex_lock( &create_mutex );

    if (cwrite (fd, cmd, 5) < 0)
    {
        perror ("Could not start direct drive");
        pthread_mutex_unlock( &create_mutex );
        return -1;
    }
    pthread_mutex_unlock( &create_mutex );
    return 0;
}
```

Figure 24 'directDrive' function for direct robot control in the 'createoi.c'
Source: *http://libcreateoi.googlecode.com/svn/trunk/src/createoi.c*

To implement these codes to the robot, both of them basically need serial communication to send 5 bytes including one-byte operation code. The first byte should be an operation code, and other codes are composed by separated 16-bit signed values. Hence, it is in total 5 bytes as a one-byte operation code and each two-byte velocity and radius value, as shown in Table 2.

Command Sequence	**1**	Operation code
	2	High byte of the velocity value
	3	Low byte of the velocity value
	4	High byte of the radius value
	5	Low byte of the radius value

Table 2 Serial command sequence for the robot movement

As a result, the overall robot program, which includes serial communication, can be illustrated as the following sample code (Figure 25).

```
#include <createoi.h>
#include <stdio.h>

int main()
{
    startOI_MT ("/dev/ttyUSB0");

    while(1)
    {
        driveDistance (300, 0, 200, 1);
        turn (20, 1, 90, 1);
    }

    stopOI_MT();
}
```

Figure 25 sample source code for the indirect control of IRobot movement
Source: http://libcreateoi.googlecode.com/svn/trunk/samples/square.c

4.3 Summary

To compose the robot program, serial communication and hardware control of the robot is essential parts. Firstly, the serial communication program between the robot and board is composed by using the 'termios' structure. Secondly, robot program can use an indirect or direct method. For the indirect method is to use the 'drive' function through 'driveDistance' and 'turn' function, but direct method use 'directDrive' function. Both of them seem to use similar programming architecture. However, indirect method might support more synchronized the serial communication than the direct method.

5. Embedded vision programming

In this project, the camera plays the role of the 'eye' of the robot. This means that an embedded vision program can decide on the direction of the robot movements. For this reason, the embedded vision programming, including the experiment algorithms can be separated into three parts: object tracking, weight distribution and measurement of the distance.

5.1 Object tracking

In standard practice, using the 'OpenCV' library for the vision programming is considered to be the most useful method. In particular, it could be said that it is assumed suitable for the object tracking algorithms. The reason is that the 'OpenCV' library supports a diversity of functions for the vision algorithms depending on the operating system and developing software. In addition, it continuously updates its version for the improved algorithms. Because of this, the object tracking part relies on the 'OpenCV' library in this project.

For the object tracking, the method using the HSV colour space can be divided into three steps. Firstly, the program for the object tracking needs to obtain an image frame through a camera. After that, in the case of using HSV colour space, the colour range for specific colour extraction should determine as 'hsv_min' and 'hsv_max' values. Finally, it checks and extracts the location and size with the colour range. The summarised process of object tracking part can check the following flowchart.

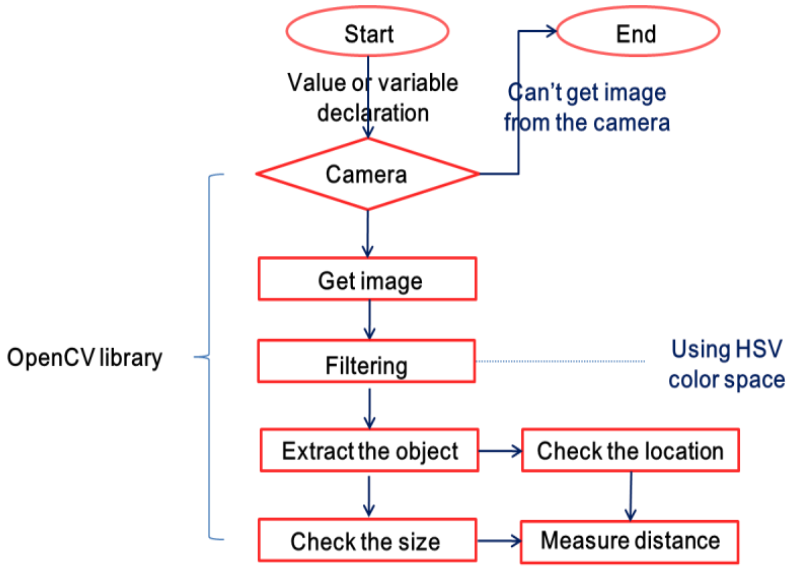


Figure 26 flow chart of object tracking using HSV color space

To specify the above Figure 26, in the 'OpenCV' library, each image frame can be obtained by using 'cvCaptureFromCAM' function, and the code as follows.

```
//Obtain image fram from the camera
CvCapture* capture = cvCaptureFromCAM( 1 );
if( !capture )
{
        fprintf( stderr, "ERROR: capture is NULL ₩n" );
        getchar();
        return -1;
}
```

Figure 27 Source code to obtain the image from the camera

Thus, turning to the next step, it is necessary to set the colour range and filter out colours with it. To find the specific colour range, the colour table of window application or image colour histogram is usually used.

The following table shows the examples of yellow and pink colour ranges in RGB colour space.

HSV	Pink	Yellow
HSV_min	cvScalar (0,50,170)	cvScalar (20,100,50)
HSV_max	cvScalar (10,180,255)	cvScalar (30,255,100)
HSV_min2	cvScalar (170,50,170)	cvScalar (20,100,100)
HSV_max2	cvScalar (255,180,255)	cvScalar (30,255,255)

Table 3 Color rage for yellow and pink color

Finally, filtering out the specific colours in an image frame can be done easily by using 'cvCvtColor', 'cvInRangeS' and 'cvOr' function in 'OpenCV' library. The 'cvCvColor' function is to convert colour space to HSV colour space, and the purpose of 'cvInRangeS', is to find the colours in the range of colour space. With these two functions, 'cvOr' function extracts original common colour into the changed image frames and original image frame. In this part, the reason for using the 'cvCvColor' function is to filter colours in the HSV colour space.

As a result, the 'cv CvtColor' and 'cvInRangeS' functions are assumed to be the basements before the filtering by using the 'cvOr' function. The filtered example images by using HSV colour space are as follows.

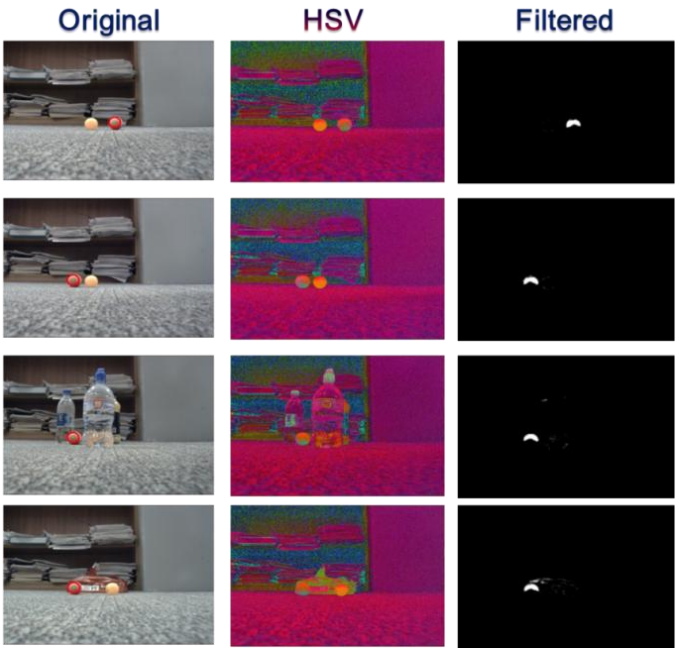


Figure 28 object extracting examples, when it used pink color range.

As can be seen from Figure 28, the pink ball can be extracted easily by HSV colour space. However, using HSV colour space has a large amount of noise depending on the environment around the pink ball. In this project, normally it uses the pink ball to extract and track the colour. However, this way only had low accuracy of 40.34% on 1000000 times in a complex environment, but averagely with 97.24% in a specified simple environment. Their basic source code can see in Appendices 2.

In short, using 'OpenCV' library is helpful for the realization of colour tracking method. However, colour tracking method relies on environmental conditions. Hence, colour tracking method should be used, based on the specified conditions to acquire more accurate results of the test.

5.2 Weight distribution to pixel data

Because of low accuracy within an experiment in a changeable environment, there is sometimes a great quantity of noises in the filtered images. In this case, distinguishing the original object point in the noises can be difficult. Because of this, filtered image data should be specified by using classification algorithms or distributing weight. As a result, this project wrote programs for the k-mean clustering program to separate out image data and flexible weight distribution code.

For the first process, the program for K-means clustering should require the number of k clusters. Thus, it needs to be decided how many groups it divides before grouping image data. However, the number of k clusters might be flexible in this project. The reason is that we cannot know the amount of noises within filtered image data, and cannot predict it. Because of this, this project suggests that the default number of k clusters decides 2 and the k number should depend on the weights of the image data. The image frames from the camera follows 30 frames per

one second in the above tracking program. Therefore, it was decided to check the position, size and robot movement through 90 frames for 3 seconds. Figure 29 shows centre positions of a drawn image, when it obtained frames during 3 seconds and filtered them.

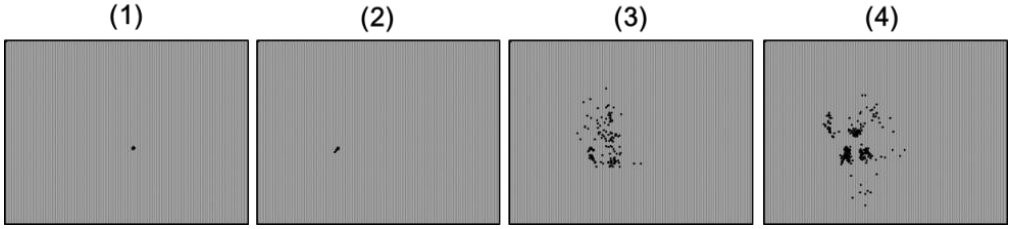


Figure 29 the examples of 90 frames obtained in Figure 28

In the case of (3) and (4) of in Figure 29, it could not be known where the centre is co-ordinated in the original object. Because of this, the different weight adapted to each co-ordinate. If the data point includes any other data points in 10 by 10 pixels around it, its position will be distributed as (1), relevant to the highest weight. Moreover, if the data co-ordinates include more than 20 other data at the same point, it will be one of the cluster points used as K-mean clustering algorithm. As a result, the calculated number of data points is the K number in the program.

Figure 30 shows the tested results based on filtering image data points by adapting K-mean clustering algorithm to weight distribution.

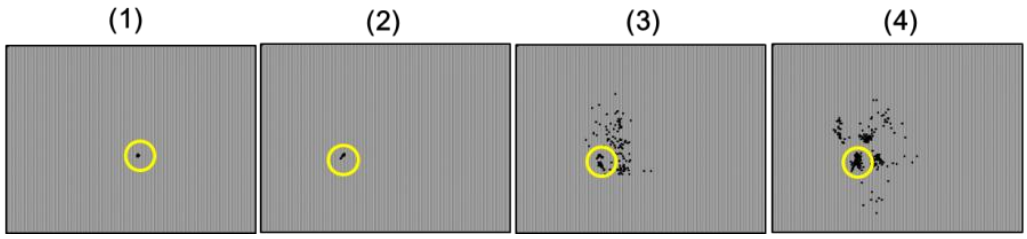


Figure 30 results of combination k-mean clustering and weight

In the case of (1) and (2), they do not need K-mean clustering because the number of clusters is only one. The K-mean clustering algorithm is to classify the data groups. Only the cases such as (3) and (4) can be required to use K-mean clustering and weight distribution.

5.3 Measurement of the distance

After checking the co-ordinates x, y of the object, the program for the distance between the object and robot could be written by using 'cvHoughCircles' and 'cvGetSeqElem' function in 'OpenCV' library. The 'cvGetSeqElem' has the function to extract co-ordinates information and the approximate object size which the data point has. Thus, using this function enables the possibility to calculate the distance through comparison between the original object size and the measured object size on image frame.

As mentioned in 2.1.3.1, prior measurement is required for the object distance based on the specific model. In this project, the size of image frame is 640 by 480. Based on this condition, the distance approached to approximately 20.23 *cm* when the pink ball was used and its appearance was drawn as 50 by 50 pixels on an image frame. Hence, calculating a program to include the distance can be calculated as follows.

```
// Memory for hough circles
CvMemStorage* storage = cvCreateMemStorage(0);
// hough detector works better with some smoothing of the image
cvSmooth( thresholded, thresholded, CV_GAUSSIAN, 9, 9 );
CvSeq* circles = cvHoughCircles(thresholded, storage, CV_HOUGH_GRADIENT, 2,
                                thresholded->height/4, 100, 40, 20, 400);
//get the image information
float size_circle = 0.0;
float p[3] = {};
for (int i = 0; i < circles->total; i++)
{
    float* tmp = (float*)cvGetSeqElem( circles, i );

    if(tmp[2] >= size_circle)
    {
        size_circle = tmp[2];
        p[0] = tmp[0];
        p[1] = tmp[1];
        p[2] = tmp[2];
    }
}
cvCircle( frame, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, CV_RGB(0,255,0), -1, 8, 0 );
cvCircle( frame, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );
cvCircle( rutin, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, CV_RGB(255,255,0), -1, 8, 0 );

//calculating distance
float distance = 0.0;
distance = ((p[2]*2)*20.23)/50;
```

Figure 31 the source code to calculate the object distance

5.4 Summary

In this project almost complete embedded vision algorithms were achieved by using ‘OpenCV’ library. However, the performance included a quantity of noises contingent on changeable environmental conditions. In spite of this, the performance was able to overcome the impediments through added K-mean clustering algorithm and weight distribution.

6. Experiments based on reinforcement learning

For intelligent robot performance, it is necessary to add reinforcement learning algorithm to the embedded vision program. However, making a reinforcement learning program is not a simple process. In addition, the embedded system has limited memory space and the system speed is also quiet slow. Because of these reasons, effective use of memory space has to be considered, as well as suitable programming style. Thus it is now necessary to describe how to make the program for the Q-learning program. This will be demonstrated as one of the reinforcement learning algorithms, along with experiment processes which engage the adapted robot to the reinforcement learning program.

6.1 Program for the Q-learning

To adapt Q-learning algorithm to embedded vision, separation is needed in order to make Q-table and effectively determine Q-value action policy with Q-tables. First of all, Q-learning algorithm is based on the Q-table. For the Q-learning algorithm, there are lots of programs made by Java or Matlab languages. In Java and Matlab, it is possible to use hash table function which Java or Matlab supports. However, to make Q-learning algorithm in C language, it could be essential to make Q-table. The program for the Q-table can be composed of structure groups by using SLL (singly linked list) in this project instead of hash table. Furthermore, each structure describes each state, action, result, and reward.

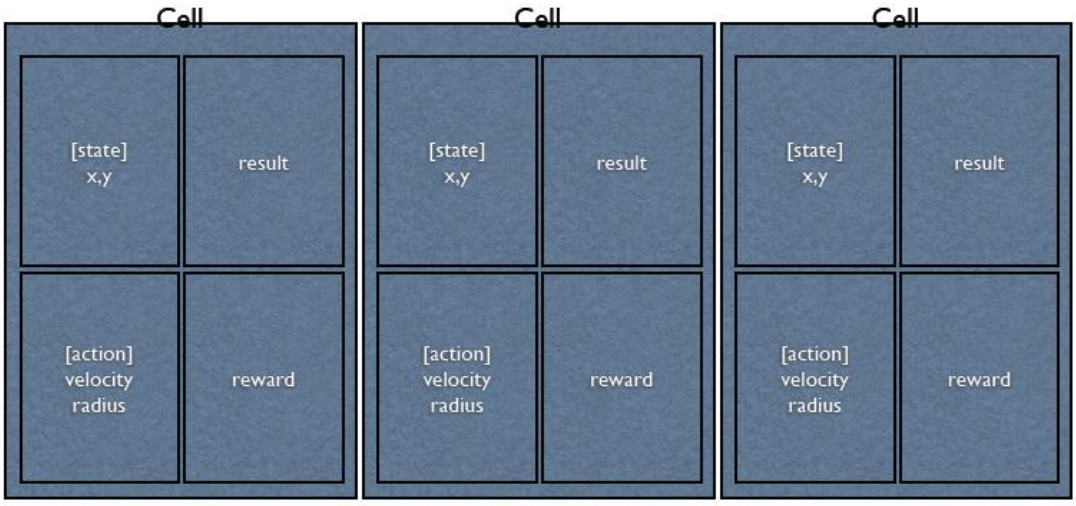


Figure 32 singly linked list for the Q-table

The above Figure 32 shows how and with what each Q-table connects, and furthermore within the Q-tables is included a state structure, an action structure, results as an old state value, and reward value concerning the result. Furthermore, to handle each Q-table and a linked list of the Q-table, additional functions need to be included such as an appending function for the new states and a destroyer function in a linked list. Following Figure 33 is a Q-table part of a Q-learning program used in this project. The program includes the functions which control and not only the linked list but also each cell.

List controlling functions

```
typedef struct _list{
    void *data;
    struct _list *next;
}u_list;

// basic functions
int u_list_count(u_list *root);
u_list *u_list_get_last(u_list *root);
u_list *u_list_append(u_list *root, void *data);
u_list *u_list_new(void *data);

//index :: start with 0
u_list *u_list_get_at(u_list *root, int index);
void *u_list_get_data_at(u_list *root, int index);


// custom functions
void u_list_print_string(u_list *root);

void u_list_free_all(u_list *root);
void u_list_free_all_with_data(u_list *root);

u_list *u_list_remove_first(u_list *root);
u_list *u_list_remove_first_with_data(u_list *root);

u_list *u_list_remove_last(u_list *root);
u_list *u_list_remove_last_with_data(u_list *root);

u_list *u_list_remove_at(u_list *root, int index);
u_list *u_list_remove_at_with_data(u_list *root, int index);

u_list *u_list_remove(u_list *root, void *target);
u_list *u_list_remove_with_data(u_list *root, void *target);
```

Q-table controlling functions

```
//structures for the Q-table
typedef struct _state{
    int x;
    int y;
}t_state;

typedef struct _action{
    int velocity;
    int radius;
}t_action;

typedef struct _cell{
    t_state state;
    t_action action;

    double result;
    double reward;
}cell;

void cell_add(t_state state, t_action action, double result, double reward);
void cell_clear(void);
cell *cell_get(int index);  //index start 0

void cell_all_plus_x(int _x);
void cell_all_plus_y(int _y);
void cell_all_plus_velocity(int _v);
void cell_all_plus_radius(int _r);
void cell_all_plus_result(double _result);
void cell_all_plus_reward(double _reward);

double update_Q (double distance ,u_list *Q_table,int x, int y, int radius,int velocity);
double object_distance(double action_policy);
double getMaxQ(int x,int y);
```

Figure 33 the functions to control linked list and each cell

Another step in the reinforcement learning, is to determine Q-value for the optimal action policy with Q-table. As mentioned in 2.2.2, it is firstly necessary to set up the learning rate and discounting rate. Generally, the learning rate is set at 0.9 and the discounting rate set at 0.1 as between 0 and 1. After that, it is necessary to find the maximized Q-value of last states. Each Q-value of past states was stored as a result in the Q-table. By using these, an overall Q-learning program can be written and achieved, as shown in Appendices 3. In short, the programming for the Q-learning could be composed by using the 'OpenCV' library and a singularly linked list of C language instead of the hash table.

6.2 Program for the MQL, AMQL and AMMQL

As mentioned in 2.2, use of modules in the Q-learning may have the purpose that Q-learning could have effective decision and reduce the weakness of the real-world problems. Based on the purpose, how to compose modules with data in the Q-table could be one of the key points for the program of the MQL, AMQL and AMMQL in the Q-learning program (Ki-Duk and In-Cheol, 2001). Hence, this project attempted to divide the Q-table into three kinds of modules.

Firstly, each module can be separated with co-ordinates of the object. The x, y co-ordinates of detected object can be 3072006 (640*480) cases in an image frame. Composing Q-table with 3072006 cases might be heavy on the memory space and system performance. Thereby, the modular co-ordinates can manage in the Q-table as 48 parts. In addition, each part can follow the separated co-ordinates as follows.

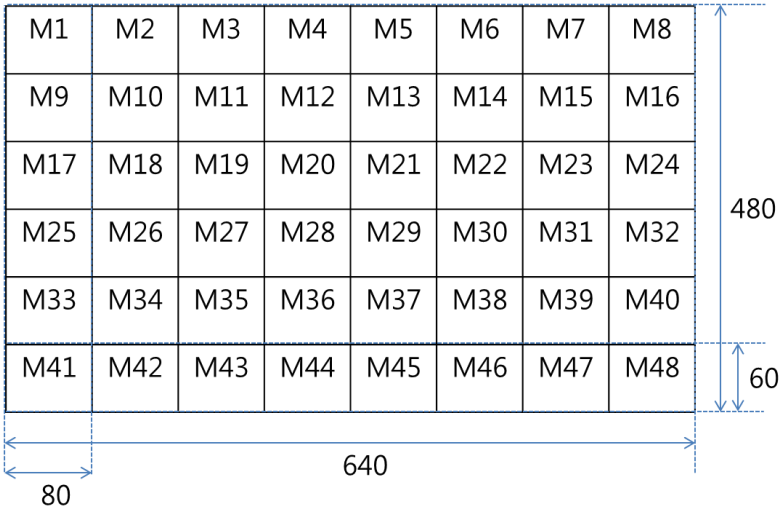


Figure 34 Modularization following separated co-ordinates

Secondly, the modularization can be done by whether the robot can detect the object or not. Sometimes the robot may not find the object position when it had movements to wrong direction, which could not find the object, or error of the object detection. Thus, the Q-table considers of modules having distance 0 and others (Tan, 1993). Configuration of this modularization can compose as separates Q-table into 2 groups and following table shows the example.

Module 1 Result = 0	**Module 2** Result ≠ 0
[state] x, y	[state] x, y
[action] velocity, radius	[action] velocity, radius
reward	reward

Table 4 example of Modular Q-table

The third method of modularization is to combine above two modularization methods. In addition, each module firstly can be divided by the co-ordinates, and then it stores modules depending on the result value.

To sum up, dividing modules in MQL, AMQL and AMMQL is based on Q-learning program. However, the methods to modularise depend on the result of robot performance. Thus, the result of distance and co-ordinates can consider of the modularization methods

6.3 Experiment conditions

As is mentioned in the forthcoming sections, robot performance can be influenced by environmental conditions. In addition, the results might not be regular and the learning level is probably flexible. Because of this, to check the robot performance based on the reinforcement learning, this project basically separated out into 2 conditions in order to observe robot performance at the same or in a random position.

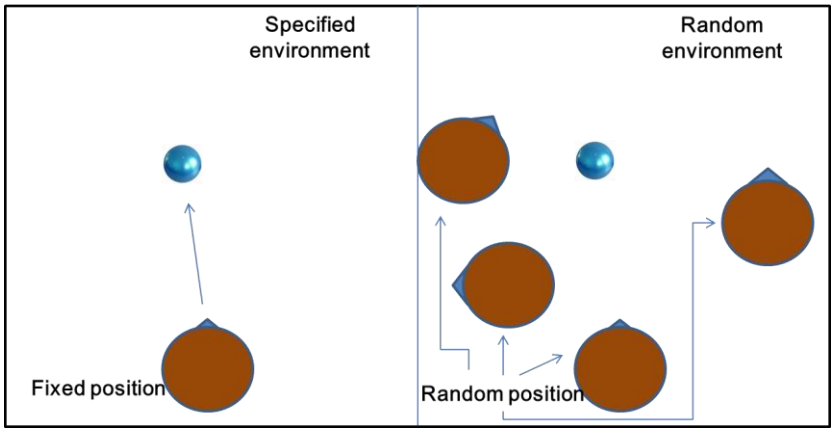


Figure 35 experiment conditions for the robot movements; left side of the picture is the condition which has fixed point as starting point of the robot, however right side has random starting point.

The first condition of experiments is to set up an environment where the robot can obviously find the object, and its performance will always start at the same point. In this case, the object location should have a delicate difference between the object and the robot position. However, in this project it presumed that their locations are the same, and then it observed the learning process of the robot. However, on the right side of the picture in Figure 35, the robot can have a different position and start object tracking. In this case, their condition is only saved by the initial distance between the object and robot. Furthermore, this project assumes that the robot performance can also have different results following the changes of the learning rate and the discount rate parameters. In standard practice, the learning rate and discount value might be between 0 and 1. Because of this, the conditional rates had a 0.1 gap between 0 and 1, such as $0.1, 0.2, \cdots, 0.9$. Finally, the experiment has to consider if the robot used each algorithm for the object tracking or not. This project suggested not only Q-learning or MQL, but also the weight distribution method. Hence, the experiment can process with or without algorithms for each part.

In conclusion, the robot performance can be compared within the conditions which were divided by the starting points of the object and robot, including the different learning rate and discount values. Furthermore, it procedes with or without each algorithm for the object tracking and reinforcement learning.

7. Analysis and Evaluation of results obtained by trial and error

The main aim of this project has been to evaluate the combination of reinforcement learning algorithm and vision algorithms. To achieve this aim, this project has needed to prove the effectiveness of combined algorithms with the tested results of robot performance. In addition, to test the robot performance with added reinforcement learning algorithm, the robot was tested on average more than twenty thousand times. As a result, this aspect concerns the results with the training standard for the robot experiment. As mentioned above, the results will be described by following the order of the experiment conditions. Furthermore, most of them will be based on the reinforcement learning results.

7.1 Use of the algorithms

This part will be analysed and evaluated with the results concerning the robot performance when the robot uses the K-mean clustering algorithm, weight distribution or not. For the reinforcement learning part, the learning rate and discount value set up default parameter values as 0.9 and 0.1, and the robot started from a fixed position. As the first comparison tested with or without a K-mean clustering algorithm and weight distribution, the next figure shows the calculated distance based on vision results and robot movements.

(a) Non-use K-mean clustering and Weight distribution				(b) Use K-mean clustering and Weight distribution			
x	y	r	Distance	x	y	r	Distance
0	0	0	0	358	166	50.16	40.59
100	178	17.69	14.32	354	166	47.06	38.55
0	0	0	0	360	166	50.57	40.92
0	0	0	0	358	166	48.47	39.22
0	0	0	0	358	166	48.37	39.14
0	0	0	0	360	166	50.61	40.95
0	0	0	0	360	166	50.96	41.24
0	0	0	0	360	168	52.24	42.27
0	0	0	0	362	166	50.25	40.66
0	0	0	0	360	164	47.38	38.34
0	0	0	0	360	162	46.1	37.3
0	0	0	0	356	164	44.55	36.05
100	178	17.69	14.32	360	162	47.1	38.11
102	180	20.25	16.39	358	164	47.01	38.04
0	0	0	0	358	164	46.62	37.72
0	0	0	0	358	164	46.65	37.75
0	0	0	0	358	164	46.69	37.78
102	184	23.77	19.23	358	164	46.84	37.9
100	178	17.69	14.32	356	164	45	36.41
0	0	0	0	362	162	48.17	38.98
0	0	0	0	356	164	46.39	37.54
0	0	0	0	360	162	47.01	38.04

Table 5 compared results depending on use of K-mean clustering and weight distribution, when the brightness condition to track the pink ball was better than other places, before making robot movements.

As can be seen from Table 5, the result at (b) shows the fixed co-ordinates and distance of the object, much more than (a) at the same position of the robot. Thus, the robot could not recognise the object position correctly in the (a) case. After all, the robot can have the wrong results in the performance. Hence, it could be said that using a K-mean clustering and weight distribution strongly influences the object tracking results.

To show the more specific differences, the following Figure 36 explains the first 100 times of the robot learning states, dependent on the use of the above algorithms.

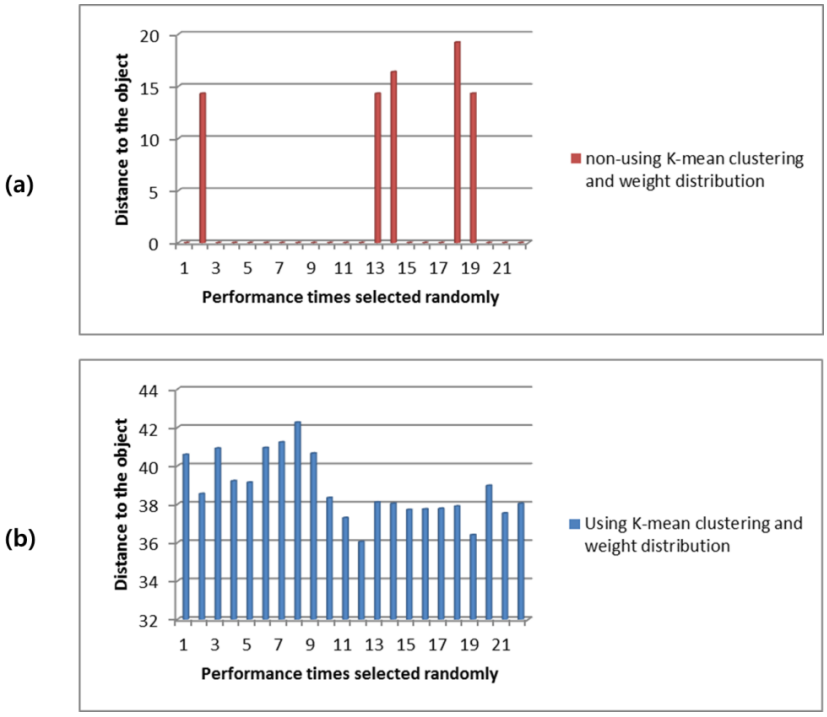


Figure 36 the distance results at robot learning; (a) is when a K-mean clustering algorithm is used with weight distribution, (b) is without these based on Table 5.

As can be seen from the above figure, both (a) and (b) showed the irregular distance result following robot performance. In particular, (a) has a part where the distance value is 0. Thus, it might mean that the robot could not recognise the object shape, due to failure to extract an object colour in the image frames. In addition, the robot when programmed should perform randomly at the starting point, and specifically when the robot has 0 distance value. Because of this, in the case of (a), the robot shows random movement after 0 distance value. The above figure shows only 100 times robot performances. However, actual robot movement normally runs more than 20,000 times. As a result, whenever the situation in (a) occurs, it could not be said that the robot can learn from robot performance. Moreover, in the case of (b) it appears that its distance value approaches closely to optimal distance value. However, its change of distance value is unstable and fluctuated. In conclusion, using a K-mean clustering algorithm and weight distribution might be helpful for the object tracking process as the first step is to learn from its performances.

7.2 Changeable learning rate and discount factor

The experiment results could be changed by the learning rate and discount value. As mentioned before, the learning rate and discount value is set up as 0.9 and 0.1. In addition, its regular change could decide how much the robot can learn from the past actions, rewards and past Q-values. Probably, these are useful to know the relationship between the conditions and robot learning performance, even if all of the robot performance engages in a changeable environment (Storey, 2003). For the comparison of the difference between learning rate and discount value, there are the changes of the Q-values and robot performances. This experiment was progressed on the fixed starting points, thus, the robot made performances to approach to about $20cm$.

Firstly, to check the difference following the change of the learning rate, the learning rate decreased each 0.1 from 0.9 to 0.1. As can be seen from Figure 37, when the learning rate is 0.1, the distance which the robot checked could not reach a converged distance result. However, in the case where the learning rate is 0.9, we could see the narrow range of the distance change finally. This might mean the robot can learn from past performances. In addition, in the case of the learning rate 0.5, it may not distinguish the big difference between learning rate 0.1 and 0.5. Hence, when the learning rate is 0.9, the degree of the learning performance may be better than a low learning rate as demonstrated through the results.

In fact, a higher learning rate is generally used for an indulgent action policy (Sutton, 1996). However, the experiment in this project has a high error rate because of image noises and tracking methods. Because of this, it may not be said that a higher learning rate always leads to a better learning performance of the robot. As a result, the learning rate should be chosen through a great deal of experiments and it should be used flexibly.

Figure 37 the distance results following the learning rate; when the learning rate is 0.1, 0.5 and 0.9, it shows the distance change following the robot performances.

Secondly, comparing learning performance with different discount factors also has similar progress to the learning rate experiment based on the learning rate 0.1. Before the explanation of the result, the discount value could be defined by the decision factor on how much it adapts past Q-value for the new action policy (Littman, 1994). It probably means that learning performance can bring different results depending on the value of the discount factor. The actual result concerning the changes of the discount factor is demonstrated as follows:

Figure 38 the results depending on different discount factor at fixed starting position.

From Figure 38, the distance results, which are the following robot movements, could not distinguish easily enough to evaluate the difference between different discount values. In spite of this, the fluctuation of the distance factor 0.7 is less flexible at the discount factor than under discount factor 0.2. The reason may be that each new action policy could not correctly evaluate the past Q-values. As mentioned in the case of the experiment, based on the different learning rates, the object tracking result in this project could demonstrate the wrong result many times. In this scenario, the robot should always make a randomly new action performance. In addition, the robot with discount factor attempts to consider its optimal performance within the past Q-values, which are the Q-values when the robot could not recognize the object in an image (Tan and Weihmayer, 1992). However, the Q-values have different alternatives and rewards. Furthermore, it could be said that it was the optimal action policy. Therefore, choosing the higher value as the discount factor could make the wrong optimal policy for new robot movement.

In conclusion, selecting a lower discount factor and higher learning rate strongly or weakly influences the result of the learning performance, but it need not be said that the parameter value is optimal for the performance. Therefore, the parameter values should be selected through plenty of repeated experiments.

7.3 Change of robot positions

Robot performances could be influenced by the starting position. The starting position of the robot performance could have the intention of how to decide the direction in which the robot will move. Thus, the robot may learn from a robot performance which has conditions such as fixed or random action states. As mentioned in 6.2, robot performances were tested on the fixed and random starting position.

To evaluate those cases, the robot also has different results as follows:

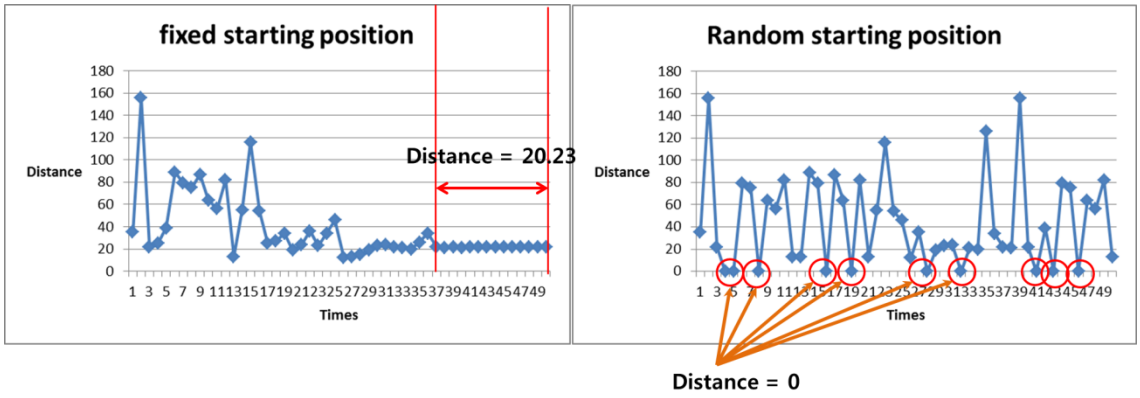


Figure 39 the results at the different starting point; the left side of figure is the result at the fixed starting point, otherwise right side shows the result when the robot moved at random starting point.

As can be seen from Figure 39, when the robot started at the fixed starting point, the robot could approach the optimal distance result 20.23. Although its performance fluctuated at the beginning part, after 27 times of the robot performance the robot slowly started finding a stable distance and maintained its distance from the learning. Otherwise, in the random starting position, it may not be said that the robot had learned from its performance (Gullapalli, 1990).Furthermore, from the right side of the Figure 39, it can be seen that the robot frequently repeated random movements. The reason might be that the robot had lot of cases where the robot could not find the object when the distance values are 0. After all, the robot was not able to learn correctly from each of its actions when it started at the random point. Hence, it may be said that the robot learned when it progressed at the fixed starting point.

7.4 Limitations

From the experiment, it is possible to deduce the result of learning performance contingent on the different learning rate, discount factor and starting point for both robot movement and algorithmic object tracking. However, there were various limitations such as unlimited robot learning processes and low accuracy in object tracking.

Firstly, there are a huge amount of environmental changes when recognising the object in an image frame. To track the object, the robot uses the co-ordinates of the object. In addition, the size of the image is 640 by 480, thus there are 307200 cases.

As a result, the Q-table space, in order to create the object information might be required with at least 4915200 bytes. In fact, the memory size in the embedded system was not enough for every case depending on the changeability of the environment. Therefore, this project captured and analysed only the times when the memory of the robot program did not overflow.

Secondly and similarly to above limitation, the robot required unlimited learning processes when the robot could not track the object correctly. In addition, when the robot could not track the object shape, the robot program handles the object distance as 0 and starts making random performance. Because of this, the robot sometimes could not learn from its performance.

Thirdly, it was not easy to find a relevant learning rate and discount factor for the experiment. As mentioned before, the changeable environment brought unstable results. Because of this, it was difficult to find effective results from the robot performance.

In short, while the experiment was progressing, there were various limitations such as memory shortage, changeable environment conditions and unstable results. Furthermore, this adversely affected the experiment results such as flexible distance value and not being able to reach the optimal distance.

8. Conclusion

This project started with the assumption of integrating reinforcement, learning algorithm and embedded vision algorithm in order to improve more intelligent robot performance. As influenced by the above assumption, this project suggested varied Q-learning algorithms and object tracking methods, with added K-mean clustering and weight distribution. Within the context of this project, the role of Q-learning algorithm was to learn from past action performances, while colour tracking added K-mean clustering and weight distribution which related to how to extract and track the specific object. Thus, these procedures have been explained by the theoretical background. Furthermore, this project demonstrated an embedded system including a camera, mobile robot and Beagle-board to illustrate how vision algorithm implemented reinforcement learning can make system performance more intelligent.

In order to achieve its aims, by conducting an embedded system, this project was separated into three parts. Firstly, it comprised of hardware programming for the communication between basic hardware components. To exemplify, it needed the serial communication to transmit or receive the command between the board and robot. Hence, robot movement was composed of controlling servo-motors in the robot. Thus, the colour tracking method added the Q-learning, which was embodied and based on the hardware programming in C language. In the process for the tracking method, it used the 'OpenCV' library. Based on these programs, the robot performances were finally tested with different parameters and experimental conditions, such as learning rate, discount factor, use of algorithms and different starting points. As a result, when the robot added K-mean clustering and weight distribution method it had a learning rate of 0.9 and a discount factor of less than 0.3, and started at the fixed position, thus revealing the results that the robot learned from past actions and therefore contained intelligent movement.

However, the experiment had a few limitations in the project. For instance, memory space shortage and repeatable wrong object detection brought flexible and furthermore wrong results due to changeable environments. In spite of the limitations as observed, the robot learning performance in this project showed the possibility that the embedded vision system adapted through reinforcement algorithm could bring more intelligent performance.

9. Future prospects and potentiality in these fields

This project has progressed with a number of limitations within poor testing environments, and has presented the hypothesis that with the combination of reinforcement learning and vision algorithm, intelligent embedded robot performance is possible. Various embedded vision systems are deeply close to our life at present. For example, the video camera in the smart mobile phone currently has the function of an embedded vision algorithm (Dietrich and Garn, 2007). In particular, the convolutional face detector application in real-time has become one of the most popular applications on the mobile phone. The humanoid has attempted to adapt an effective embedded vision system in order to improve performance. Thus, in an effort to adapt reinforcement learning to embedded vision, better performance on the system has been attained. However, using reinforcement learning often consumes a huge amount of time to find the optimal action value, and includes the limitation of real-world problems. Hence, the effective alternatives for reinforcement learning including real-world problems and more accurate object tracking method will be one of the key points for future intelligent performance of the embedded system.

10. Bibliography

Ali, A., and Aggarwal, J., (2001), "Segmentation and recognition of continuous human activity", *IEEE Work-shop on detection and Recognition of Events in Video*, pp. 28-35.

Arleo, A., Smeraldi, F., Hug, S., and Gerstner, W., (2001),"Place Cells and Spatial Navigation based on Vision, Path Integration, and Reinforcement Learning", *Centre for Neuro-Mimetic Systems, MANTRA, Swiss Federal Institute of Technology Lausanne, CH-1015 Lausanne EPFL, Switzerland*, pp.1-7.

Asada, M., Noda, S., Tawaratsumida, S., and Hosoda, K., (1996), "Purposive Behavior Acquisition for a real Robot by vision-Based Reinforcement Learning", *Kluwer Academic Publishers, Boston*, Machine Learning, 23, pp. 279-303.

Axelson, J., (1998), "Serial Port Complete: Programming and Circuits for RS-232 and RS-485 Links and Networks", *Lakeview Research, ISBN 0-9650819-2-3*, pp. 1-10.

Ballard, D., and Brown, C., (1982), "Computer Vision", *Prentice-Hall. Chap. 8.*

Beagle-board.org, (2011), "Beagle-Board-xM Product Details", *available: http://beagleboard.org/hardware-xM*.

Beagle-board, N.D., "Linux Boot Disk Format", *available: http://code.google.com/p/beagleboard/wiki/LinuxBootDiskFormat*.

Bennett, K. P., and Campbell, C., (2000), "Support Vector Machines: Hype or Hallelujah?", *available: http://www.sigkdd.org/explorations/issue2-2/bennett.pdf, SIGKDD Explorations, Vol. 2, Issue 2*, pp. 1-13.

Black, M., and Anandan, P., (1996), "the robust estimation of multiple motions: Parametric and piecewise-smooth flow fields", *Comput. Vision Image Understand. 63*, 1, pp. 75-104.

Canny, J., (1986), "A computational approach to edge detection", *IEEE Trans. Patt. Analy. Mach. Intell. 8*, 6, pp. 679-698.

Chen, C. W., Luo, J., and Parker, K. J., (1998), "Image Segmentation via Adaptive K-Mean clustering and Knowledge-Based Morphological Operations with Biomedical Applications", *IEEE Tran. Image process, Vol.7, No.12*, pp. 1673-1682.

Cole, L., Austin, D., and Cole, L., (2004), "Visual Object Recognition using Template Matching", *Australasian Conference on Robotics and Automation 2004*.

Comaniciu, D., Ramesh, V., and Meer, P., (2000), "Real-time tracking of non-rigid objects using mean shift", In *Proc. Conf. Comp. Vision Pattern Rec*., pp. 142-149.

Comaniciu, D., Ramesh, V., and Meer, P., (2003), "Kernel-based object tracking", *IEEE Trans. Patt. Analy. Mach. Intell. 25*, pp. 564-575.

Connel, J. H., and Mahadevan, S., (1993), "Rapid task learning for real robot", *In J. H. Connel and S. Mahadevan, editors, Robot Learning*, chapter 5.

Cortes, C., and Vapnik, V., (1995), "Support-vector network", *Machine Learning, 20*, pp. 1-25.

Dayan, P., and Balleine, B. W., (2002), "Reward, Motivation, and Reinforcement Learning.", *Cell press, Neuron, Vol. 36*, pp. 285-298.

Dayan, P., and Watkins, J., (1992), "Machine Learning", *Encyclopedia of Cognitive Science, Vol. 8, Numbers 3-4, DOI: 10.1007/BF00992698*, pp. 279-292.

Dietrich, D., and Garn, H., (2007), "Editorial Embedded Vision System", *Hindawi Publishing Corporation EURASIP Journal on Embedded Systems, Vol. 2007, Article ID 34323, doi:10.1155/2007/34323*, pp. 1-3.

Greenspan, H., Belongie, S., Goodman, R., Perona, P., Rakshit, S., and Anderson, C., (1994), "Overcomplete steerable pyramid filters and rotation invariance", *In IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 222-228.

Gullapalli, V., (1990), "A stochastic reinforcement learning algorithm for learning real-valued functions", *Neural Networks, Vol. 3, Issue 6*, pp. 671-692.

Gupta, A., and Pati, K. A., (2009), "A Project Report on Finger Tips Detection and Gesture Recognition", *available: http://home.iitk.ac.in/~ashis/CS676Report.pdf*, pp. 1-21.

Gunn, S. R., (1998), "Support Vector Machines for Classification and Regression", *ISIS Technical report, university of Southampton*, pp. 1-27.

Hajdu, A., and Pitas, I., (2007), "Optimal approach for fast object-template matching", *IEEE Trans. on Image Processing 16*, pp. 2048-2057.

Horn, B., and Schunk, B., (1981), "Determining optical flow", *Artific. Intell. 17*, pp. 185-203.

In-Cheol, K., (2002), "Design and Implementation of Robot Soccer Agent Based on reinforcement Learning", *Korea information processing society*, Rev. 9-B, pp.139-142.

Intel Corporation, (2001), "Open Source Computer Vision Library: Reference Manual", *available: http://developer.intel.com, chapter 14*, pp. 4- 69.

IRobot Corporation, (2006), "IRobot Create open interface Manual", *available: http://www.IRobot.com*, pp. 9-13.

Jarupan, B., N.D., "Introduction to Input/Output Interface", *available: http://www2.ece.ohio-state.edu/xilinx/667/ECE667-doc/Lec/WK5/serial_rs232_usb.pdf*, pp. 1-30.

Kaelbling, L. P., Littman, M. L., and Moore, A. W., (1996), "Reinforcement Learning: A Survey", *Journal of Artificial Intelligence Research 4*, pp. 237-285.

Ki-Duk, K., and In-Cheol, Kim, (2001), "Reinforcement Learning Approach to Agents Dynamic Positioning in Robot Soccer Simulation Games", *Korea simulation society*, pp.321-324.

Kohri, T., Matsubayashi, K., and Tokoro, M., (1998), "An Adaptive Architecture for Modular Q-Learning", Journal of AI Research, pp. 1-6.

Kostiadis, K., and Hu, H., (1999), "Reinforcement Learning and Co-operation in a Simulated Multi-agent System", *RoboCup-98: Robot Soccer World Cup II, Berlin*, pp. 366-377.

Kui-Hong, P., Yong-Jae, K., and Jong-Hwan, K, (2001), "Modular Q-learning based multi-agent cooperation for robot soccer", *Robotics and Autonomous Systems, 35*, pp. 109-122.c

Laws, K., (1980), "Textured image segmentation", PhD thesis, *Electrical Engineering, University of Southern California*.

Lewis, J. P., (1995), "Fast Template Matching", *Vision Interface 95, Canadian Image processing and Pattern Recognition Society*, pp. 120-123.

libcreateoi, Create Open Interface Library, (2011), "Create Open Interface Library", *available: http://libcreateoi.googlecode.com/svn/trunk/.*

Littman, M. L., (1994), "Markov games as a framework for multi-agent reinforcement learnin", eleventh international conference on machine learning, pp. 1-7.

Littman, M. L., Cassandra, A. R., and Kaebling, L. P., (1994), "Learning Policies for Partially Observable Environments: Scaling up.", *Proceedings of the Eleventh International Conference on Machine Learning*, pp. 157-163.

MacQueen, J. B., (1967), "Some Methods for classification and Analysis of Multivariate observations, proceedings of 5-th Berkeley Symposium on Mathematical Statistics and Probability", *Berkeley, University of California Press*, 1:281-297.

Mallat, S., (1989), "A theory for multi-resolution signal decomposition: The wavelet representation", *IEEE Trans. Patt. Analy. Mach. Intell. 11*, 7, pp. 674-693.

Matteucci, M., (2003), "A Tutorial on Clustering Algorithms", *available: http://home.dei.polimi.it/matteucc/Clustering/tutorial_html/kmeans.html.*

Meyer, D., (2009), "Support Vector Machines ***** The Interface to **libsvm** in package e1071", *Technic University Wien, Austria*, pp.1-5.

Mueller, F., (1993), "A library Implementation of POSIX Threads under UNIX", In *proceedings of the 1993 Winter USENIX Conference, available: http://www.informatik.hu-berlin.de_tmueller_projects.html*, pp. 29-41.

Ono, N., and Fukumoto, K., (1996), "Multi-agent Reinforcement Learning : A Modular Approach,", *Proceedings of the Second International Conference on Multi-Agent Systems, AAAI Press*, pp. 252-258.

Osier-Mixon, M. J. (2010), "Booting Linux on the BeagleBoard-xM", *IBM developerWoks*, pp. 1-14.

Paschos, G., (2001), "Perceptually uniform color spaces for color texture analysis: an empirical evaluation", *IEEE Trans. Image Process. 10*, pp. 932-937.

Perez, P., Hue, C., Vermaak, J., and Gangnet, M., (2002), "Color-Based Probabilistic Tracking", *Computer Vision-ECCV 2002, LNCS 2350*, pp. 661-675.

Peters, J., Sethu, V., and Stefan, S., (2003), "Reinforcement Learning for Humanoid Robotics", *IEEE-RAS International Conference on Humanoid Robots*, pp. 2-17.

Powell, M. L., Kleiman, S.R., Barton, S., Shah, D., Stein, D., and Weeks, M., (1991), "SunOS Multi-thread Architecture", In *Proceedings of the USENIX Conference*, pp. 65-80.

Serby, D., Koller-Meier, S., and Gool, L. V., (2004), "Probabilistic object tracking using multiple features", *IEEE International conference of Pattern Recognition(ICPR)*, pp. 184-187.

Song, K. Y., Kittler, J., and Petrou, M., (1996), "Defect detection in random color textures", *Israel Verj. Cap. J. 14*, 9, pp. 667-683.

Storey, J.D., (2003), "The positive faslse discovery rate: A Bayesian interpretation and the Q-value", *Annals of Statistics, 31*, pp. 2013-2035.

Sutton, R. S., (1996), "Generalization in Reinforcement Learning: Successful Examples Using Sparse Coarse Coding", *Advances in Neural Information Processing Systems 8*, pp.1038-1044.

Tan, M., (1993), "Multi-Agent Reinforcement Learning: Independent vs. Cooperative Agents", *In proceedings of the tenth International Conference on Machine Learning*, Amherst, Masschusetts. Morgan Kauf-mann.

Tan, M. and Weihmayer, R., (1992), "Integrating agent-oriented programming and planning for cooperative problem solving", *Proceedings of the AAAI-92's Work-shop on Cooperation among Heterogeneous Intelligent Agents, San Jose, CA.*

Teknomo, K., (2007), "K-Means Clustering Tutorials", *available: http://people.revoledu.com/kardi/tutorial/kMean*, pp. 1-12.

Tran, J., Gerdzhev, M. and Ferworn, A., (2010), "Continuing Progress in Augmenting urban Search and Rescue Dogs", *Ryerson University*, pp. 1-5.

UNIX, (2001), "Specification, Version 2: <termios.h>", *available: http://pubs.opengroup.org/onlinepubs/007908799/xsh/termios.h.html.*

Veenman, C., Reinders, M., and Backer, E., (2001), "Resolving motion correspondence for densely moving points", *IEEE Trans. Patt. Analy. Mach. Intell. 23*, 1, pp. 54-72.

Watkins C.J.C.H. (1989), "Learning from delayed rewards*", PhD Thesis, University of Cambridge, England*.

Whitehead, S. D., and Ballard, D. H. (1990), "Active Perception and Reinforcement Learning", *Neural Computation 2, Massachusetts Institute of Technology*, pp. 409-419.

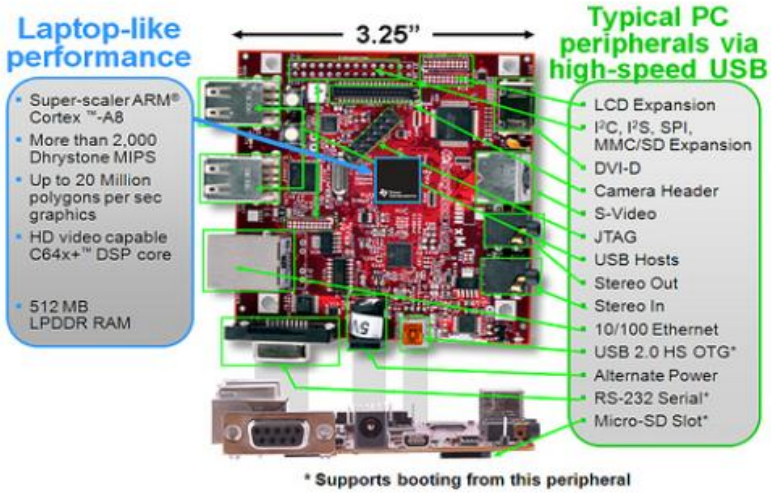
Yilmaz, A., Javed, O., and Shah, M., (2006), "Object Tracking: A Survey", ACM *Compute. Survey*, 38, 4, Article 13 (Dec. 2006), pp.1-45.

Yilmaz, A., Li, X., and Shah, M., (2004), "Contour based object tracking with occlusion handling in video acquired using mobile cameras", *IEEE Trans. Patt. Analy. Mach. Intell. 26*, 11, pp. 1531-1536.

Yilmaz, A., Shafique, K., and Shah, M., (2003), "Target tracking in airborne forward looking infrared imagery", *Image and Vision Computing, 21*, pp. 623-635.

11. Appendices

Appendices 1 Beagle-board Hardware details



- Super-scalar ARM Cortex ™ -A8
- 512-MB LPDDR RAM
- High-speed USB 2.0 OTG port optionally powers the board
- On-board four-port high-speed USB 2.0 hub with 10/100 Ethernet
- DVI-D (digital computer monitors and HDTVs)
- S-video (TV out)
- Stereo audio out/in
- High-capacity microSD slot and 4-GB microSD card
- JTAG
- Camera port

Source: http://beagleboard.org/hardware-xM

Appendices 2pink ball filtering in an image frame.

```
// Covert color space to HSV easily to filter colors in the HSV color space.
cvCvtColor(frame, hsv_frame, CV_BGR2HSV);

// Filter out colors which are out of range.
cvInRangeS(hsv_frame,hsv_min, hsv_max, thresholded);
cvInRangeS(hsv_frame, hsv_min2,hsv_max2, thresholded2);
cvOr(thresholded, thresholded2, thresholded);

// image smoothing by using hough detector
cvSmooth( thresholded, thresholded, CV_GAUSSIAN, 9, 9 );
CvSeq* circles = cvHoughCircles(thresholded, storage, CV_HOUGH_GRADIENT, 2, thresholded->height/4, 100, 40, 20, 400);

for (int i = 0; i < circles->total; i++)
{
    float* tmp = (float*)cvGetSeqElem( circles, i );

    if(tmp[2] >= size_circle)
    {
        size_circle = tmp[2];
        p[0] = tmp[0];
        p[1] = tmp[1];
        p[2] = tmp[2];
    }
}
cvCircle( frame, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, CV_RGB(0,255,0), -1, 8, 0 );
cvCircle( frame, cvPoint(cvRound(p[0]),cvRound(p[1])), cvRound(p[2]), CV_RGB(255,0,0), 3, 8, 0 );
printf("Ball position  x=%f y=%f r=%f\n\r",p[0],p[1],p[2] );
cvCircle( rutin, cvPoint(cvRound(p[0]),cvRound(p[1])), 3, CV_RGB(255,255,0), -1, 8, 0 );

cvShowImage( "Camera", frame ); // Original stream with detected ball overlay
cvShowImage( "HSV", hsv_frame); // Original stream in the HSV color space
cvShowImage( "After Color Filtering", thresholded ); // The stream after color filtering
cvShowImage( "rutin", rutin );
```

Source: http://www.aishack.in/2010/07/tracking-colored-objects-in-opencv/

Appendices 3

the program functions code for the Q-learning algorithm

```
#include<stdlib.h>
#include<stdio.h>
#include<string.h>
#include"createoi.h"       //using basic source which irobot interface
supported

#definediscount_rate 0.1
#define alpha 0.9

typedefstruct _list{
        void *data;
        struct _list *next;
}u_list;

// basic functions
intu_list_count(u_list *root);
u_list *u_list_get_last(u_list *root);
u_list *u_list_append(u_list *root, void *data);
u_list *u_list_new(void *data);

//index :: start with 0
u_list *u_list_get_at(u_list *root, int index);
void *u_list_get_data_at(u_list *root, int index);

// custom functions
voidu_list_print_string(u_list *root);

voidu_list_free_all(u_list *root);
voidu_list_free_all_with_data(u_list *root);

u_list *u_list_remove_first(u_list *root);
u_list *u_list_remove_first_with_data(u_list *root);

u_list *u_list_remove_last(u_list *root);
u_list *u_list_remove_last_with_data(u_list *root);

u_list *u_list_remove_at(u_list *root, int index);
u_list *u_list_remove_at_with_data(u_list *root, int index);

u_list *u_list_remove(u_list *root, void *target);
u_list *u_list_remove_with_data(u_list *root, void *target);

//structures for the Q-table
typedefstruct _state{
int x;
int y;
}t_state;

typedefstruct _action{
int velocity;
int radius;
}t_action;

typedefstruct _cell{
t_state state;
t_action action;

double result;
double reward;
}cell;

voidcell_add(t_state state, t_action action, double result, double reward);
voidcell_clear(void);
cell *cell_get(int index); //index start 0

voidcell_all_plus_x(int _x);
voidcell_all_plus_y(int _y);
voidcell_all_plus_velocity(int _v);
voidcell_all_plus_radius(int _r);
voidcell_all_plus_result(double _result);
voidcell_all_plus_reward(double _reward);

doubleupdate_Q (double distance ,u_list *Q_table,int x, int y, intradius,int
velocity);
doubleobject_distance(doubleaction_policy);
doublegetMaxQ(intx,int y);
```

```
u_list *u_list_new(void *data){
u_list *t = (u_list *)malloc(sizeof(u_list));
memset((void *)t, 0, sizeof(u_list));

return t;
}

//get list count
intu_list_count(u_list *root){
        int n = 0;
        u_list *t = root;

        if(t == NULL)
                return 0;

        while(t!=NULL){
                t = t->next;
                n++;
        }
        return n;
}
//get last element
u_list *u_list_get_last(u_list *root){

        u_list *t = root;

        if(root == NULL)
                return NULL;

        while(t->next!=NULL){
                t = t->next;
        }

        return t;
}
//append
u_list *u_list_append(u_list *root, void *data){
        u_list *dest;
        u_list *t = (u_list *)malloc(sizeof(u_list));

        t->data = data;
        t->next = NULL;

        if(root == NULL)
                return t;

        dest = u_list_get_last(root);
        dest->next = t;
        return root;
}

u_list *u_list_get_at(u_list *root, int index){
        u_list *t = root;

        while( (t != NULL) && (index >= 0)){
                if(index == 0){
                        return t;
                }
                index--;
                t = t->next;
        }
        return NULL;
}

void *u_list_get_data_at(u_list *root, int index){
        u_list *t = u_list_get_at(root, index);
        if(t == NULL)
                return NULL;

        return t->data;
}

//u_list_remove_first
staticu_list *_u_list_remove_first(u_list *root, intis_with_data){
        u_list *t = root;

        if(root == NULL)
                return NULL;
```

```
        root = t->next;
        if(is_with_data&& (t->data != NULL)){
                free(t->data);
        }

        free(t);

        return root;
}

u_list *u_list_remove_first(u_list *root){
        return _u_list_remove_first(root, 0);
}

u_list *u_list_remove_first_with_data(u_list *root){
        return _u_list_remove_first(root, 1);
}

//u_list_remove_last
staticu_list *_u_list_remove_last(u_list *root, intis_with_data){
        u_list *t = root;
        u_list *t_next;

        if(t == NULL)
                return NULL;

        if(t->next == NULL){
                if(is_with_data&& (t->data != NULL)){
                        free(t->data);
                }
                free(t);

                return root;
        }

        while(t->next->next != NULL){
                t = t->next;
        }

        t_next = t->next;
        t->next = NULL;

        if(is_with_data&& (t_next->data != NULL)){
                free(t_next->data);
        }

        free(t_next);

        return root;
}
u_list *u_list_remove_last(u_list *root){
        return _u_list_remove_last(root, 0);
}

u_list *u_list_remove_last_with_data(u_list *root){
        return _u_list_remove_last(root, 1);
}
//u_list_remove_at
staticu_list *_u_list_remove_at(u_list *root, int index, intis_with_data){
        u_list *t = root;
        u_list *t_next;

        if(index < 0)
                return root;

        if(t == NULL)
                return NULL;

        if(index == 0){
                t_next = t->next;
                if(is_with_data&& (t->data != NULL)){
                        free(t->data);
                }
                free(t);
                returnt_next;
        }

        if(t->next == NULL){
                if(is_with_data&& (t->data != NULL)){
```

```
                        free(t->data);
                }
                free(t);

                return NULL;
        }

        t = u_list_get_at(root, index-1);   //prev

        if(t == NULL)
                return root;

        t_next = t->next;

        if(t_next == NULL){
                return root;
        }
        else{
                t->next = t_next->next;
                if(is_with_data&& (t->data != NULL)){
                        free(t_next->data);
                }
                free(t_next);
        }
        return root;
}

u_list *u_list_remove_at(u_list *root, int index){
        return _u_list_remove_at(root, index, 0);
}

u_list *u_list_remove_at_with_data(u_list *root, int index){
        return _u_list_remove_at(root, index, 1);
}

u_list *_u_list_remove(u_list *root, void *target, intis_with_data){
        u_list *t = root;
        u_list *t_next;
        if(root == NULL || target == NULL)
                return root;

        if(t->data == target){
                root = t->next;

                if(is_with_data&& (t->data != NULL)){
                        free(t->data);
                }

                free(t);
                return root;
        }

        while(t != NULL){
                t_next = t->next;

                if(t_next != NULL &&t_next->data == target){
                        t->next = t_next->next;
                        if(is_with_data){
                                free(t_next->data);
                        }
                        free(t_next);
                        return root;
                }
                t = t->next;
        }

        return root;
}

u_list *u_list_remove(u_list *root, void *target){
        return _u_list_remove(root, target, 0);
}

u_list *u_list_remove_with_data(u_list *root, void *target){
        return _u_list_remove(root, target, 1);
}
//======================list free====================//
staticvoid _u_list_free_all(u_list *root, intis_with_data){
        u_list *t = root;
        u_list *bt;
```

```
        while(t != NULL){
                bt = t;
                t = t->next;
                if(is_with_data&& (bt->data != NULL)){
                        free(bt->data);
                }
                free(bt);
        }
}

voidu_list_free_all(u_list *root){
        _u_list_free_all(root, 0);
}

voidu_list_free_all_with_data(u_list *root){
        _u_list_free_all(root, 1);
}

//*********** custom function ********//
voidu_list_print_string(u_list *root){
        u_list *t = root;
        intcnt = 0;

        while(t != NULL){
                printf("%d :: %sWn", cnt++, (char *)t->data);
                t = t->next;
        }
}

staticu_list *head = NULL;

voidcell_add(t_state state, t_action action, double result, double reward){

cell *c = (cell *)malloc(sizeof(cell));
c->state = state;
c->action = action;
c->result = result;
c->reward = reward;

head = u_list_append(head, c);

return;
}

cell *cell_get(int index){
cell *ret = NULL;
u_list *list = head;

while(list){
ret = (cell *)list->data;

if(index == 0){
return ret;
        }
index--;
    }

return NULL;
}

voidcell_clear(void){
u_list_free_all_with_data(head);
head = NULL;
}

voidcell_all_plus_x(int _x){
u_list *list = head;
while(list){
cell *c = (cell *)list->data;
c->state.x = _x;

list = list->next;
    }
}
voidcell_all_plus_y(int _y){
u_list *list = head;
while(list){
cell *c = (cell *)list->data;
c->state.y = _y;
```

```
list = list->next;
    }
}
voidcell_all_plus_velocity(int _v){
u_list *list = head;
while(list){
cell *c = (cell *)list->data;
c->action.velocity = _v;

list = list->next;
    }
}
voidcell_all_plus_radius(double _r){
u_list *list = head;
while(list){
cell *c = (cell *)list->data;
c->action.radius = _r;

list = list->next;
    }
}
voidcell_all_plus_result(double _result){
u_list *list = head;
while(list){
cell *c = (cell *)list->data;
c->result = _result;

list = list->next;
    }
}
voidcell_all_plus_reward(double _reward){
u_list *list = head;
while(list){
cell *c = (cell *)list->data;
c->reward = _reward;

list = list->next;
    }
}
```

```
doublegetMaxQ(intx,int y){
        doubletmp=0.0;

        u_list *list = head;
while(list){
cell *c = (cell *)list->data;
if(tmp<= c->result)
                {
                        tmp = c->result;
                }
list = list->next;
    }

        returntmp;
}

doubleobject_distance(doubleaction_policy){
        doubledistance_reward;
        distance_reward = robot_movement(action_policy);
        returndistance_reward;
}

doubleupdate_Q (double distance ,u_list *Q_table,int x, int y, intradius,int
velocity){
        doublereward_alpha;
        doubleoldStateValue;
        doublenewStateValue;
        t_state *new_state;
        t_action *new_action;
        u_list *list = head;
        cell *c = (cell *)list->data;

        //find the last Q-value
        while(list->next != NULL){

list = list->next;
    }
        oldStateValue = c->result;
```

```
        reward_alpha = object_distance(oldStateValue);
        //calculate new Q-value
        newStateValue = oldStateValue +
alpha*(reward_alpha+discount_rate*getMaxQ(x,y)-oldStateValue);
        //new state
        new_state->x = x;
        new_state->y = y;
        //new action
        new_action->radius = radius;
        new_action->velocity = velocity;
        //update cell
        cell_add(*new_state,*new_action,newStateValue,reward_alpha);

        returnreward_alpha;
}
```