1 Introduction

Human perception is not determined only by stimuli the person received. It seems to be an agreement in cognitive psychology that our perception is a result of interacting two distinct processes, i.e., a bottom-up process that is driven by the stimuli and a top-down process that is controlled by memory[3, 5]. Therefore what a person perceives from particular stimuli depends on who the person is.

The stimuli can be as simple as a picture of a black dot on white background, or as complex as a situation where one should decide a right way to cure a cancer. Although it is usually called *interpretation*, we can even regard a text describing the situation as stimuli to be perceived, because interpretation and perception are similar in the sense that both of them translate something outside of one's mind to something inside of the mind. Both can be seen as a mental level modeling.

With this viewpoint, it is easily understood that ways of perception heavily affect problem solving skills, because the first step of problem solving is to create a *correct* mental model of the initial state of the problem, that requires an appropriate perception. We often experience instances of the statement in our real life. To learn how to ride a bicycle, one needs to know where to focus her/his attention to keep balance. To be a shogi (or chess) master, one needs to know how to eliminate less promising lines of plays. A good fund manager should know what is information to be concerned and what is not from flood of financial information.

If a human needs a good perception to be a good problem solver, why not an intelligent software? In this paper, we formulate the software's perception as translation of input stimuli into a vector space. Then we show how to implement it when the data to be perceived are text documents, and its application to information retrieval. We also show our preliminary implementation for still images and experiments with it. Finally we examine future directions of the study.

2 Vector Space as Artificial Mental Space

2.1 Artificial Mental Space

Perception is translation of input stimuli to internal representations. There are several approaches to formalize the internal representation for an intelligent software.

In traditional AI, the input stimuli and the internal representation often employ the same formalization, such as predicate logic, attribute-value pairs, whatever it is. This approach is prone to be criticized as *single representation trick*[12], namely the software's programmer can design the formalization with knowledge of a task to be done so that the intelligent software can easily perform the task.

In neural network approaches, weights between neurons are trained by input-output pairs of data. A set of the weights can be seen as the neural network's internal representation of the situation defined by the training data. However, what the set means cannot be known in a direct manner[2]. Manipulating the set by means of factors other than the training data, for example, by memory, is yet to be investigated.

---

[1] Takehiko Yokota had been a member of the Laboratory until March 2000.

[2] There is an indirect way to know the meaning of the set[14].

We propose to use a vector space as a scheme of the internal representation of an intelligent software. A situation perceived by the software corresponds to a point in the vector space. This approach has the following advantages.

- Similarity between situations in the software's mental space can be easily measured by means of Euclidean distance between corresponding points in the vector space.
- It is easy to manipulate the points by something other than the input stimuli, if it is related to the axes composing the vector space.
- A set of points in the vector space can be seen as a private concept of the software. This view allows us to give it a simple means to form a new concept, i.e., by standard clustering techniques.

In this approach, a way of the perception is characterized by what are axes of a vector space and how input stimuli are transformed into points in the space. Ideally the software should be able to construct the axes on demand. In this paper we simplify the problem by giving the software a reasonably large space and letting the software to use its subspaces at will[3].

As for the transformation, behavior of transformer should be systematically controllable because we implicitly want to adjust the behavior to optimize efficiency of the software at a task, so that the software can gradually improve its perception through the task executions. This can be done by parameterizing the procedures.

2.2 Good Perception vs. Complex Problem Solving

In the previous section, we argued that a good problem solver needs a good perception. The statement was a bit inaccurate. One can still be a good problem solver with a mediocre perception — though it would require greater effort. For example in chess playing, you need not eliminate any lines of plays if you have sufficiently strong computing capability.

Thus we had better say that a good perception makes problem solving *easier*. We assume that the easiness can be measured in terms of easiness to recognize similarity between *similar* situations in the internal representations. Here situations are said to be similar if the same problem solving tactics is applicable to them. Considering our internal representation is a vector space, the similarity is recognized most easily if the similar situations are geometrically close each other in the space. In other words, in the easiest case a region consists of similar situations is defined by means of distance. More difficult cases can involve definitions of the region using fancy functions.

We believe that having a good perception has advantages over complex recognition in the internal representations, because we assume that similar problem solvings can benefit from similar definitions of the similarities and therefore a similarity definition which is useful for problem solving A can be reused for problem solving B if A and B are similar. This view can explain our experiences with skill acquisition. We have to make a lot of trial-and-errors until we obtain an appropriate viewpoint for the problem. Once we have one, everything looks clear, and we can get it done effortlessly, usually even similar problems that we have not particularly tackled with.

---

[3] There is an analogy between the two methods to create the vector space and two approaches to inductive learning, i.e., *constructive* induction and *selective* induction[6].

The view is also useful in a context of learning. Inductive learning achieves an object-level generalization, meaning that it gives classification of unknown objects for the given concept. If an appropriate vector space is learned, it will achieve a concept-level generalization, meaning that it gives insight of unknown objects even for unknown concept, by means of clustering.

In Figure 1, our view of perception and problem solving is summarized. Input stimuli are translated into internal representations, namely, points in a vector space, by means of parameterized transformers. The points become an input for a problem solver. An output of the problem solver is evaluated with the environment and feedback is given to the transformers and the problem solver. Conventional machine learning has been only dealing with adaptation of the problem solver. We are thinking that adaptation of the transformers will be more important and investigating its possibility.

In the next two sections, we will show implemented methods of the approach for text documents and still images.

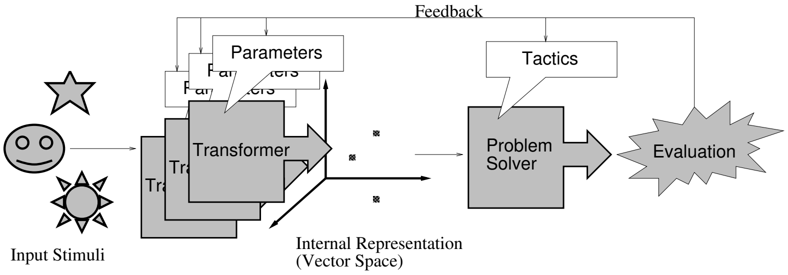

Figure 1: Perception and problem solving

3 Text Document Perception and IR Application

3.1 Vector Space Model in IR

In the area of information retrieval (IR), representing a text document as a point in a vector space is a common practice. *Tf-idf* method, which assigns values to each vector element based on frequency of occurrence of a word in a set of the text documents, is often employed[13]. First one must specify a *dictionary vector*, which is in fact a list of words one should take into account. For a dictionary vector $(w_1, \ldots, w_r)$, the i-th document D_i is represented by a r-dimensional vector

$$\vec{D_i} \overset{def}{=} (W(D_i,1), W(D_i,2), \ldots, W(D_i,r))$$

where $W(D_i,k)$ represents the tf-idf weight of the w_k in a document D_i, which is defined as the followings where $tf(D_i,k)$ denotes the number of occurrence of w_k in D_i and N as the number of documents we have.

$$W(D_i,k) \overset{def}{=} \frac{tf(D_i,k) \cdot idf(k)}{\sqrt{\sum_l (tf(D_i,l) \cdot idf(l))^2}}$$
$$idf(k) \overset{def}{=} log(N/|\{D_j | tf(D_j,k) > 0\}|)$$

A query statement Q is also transformed into a query vector $\vec{Q}$ in the same way as D_i. A task of IR is to find the nearest $\vec{D_i}$ to $\vec{Q}$ in the vector space.

The model even gives a way to adapt for each user, called *relevance feedback by query refinement*[1]. In this method, an IR system initially returns several documents for a query. A user classifies the returned documents into *positive* documents, which are useful for the user's purpose, and *negative* documents, which are not. Let Q be an original query, E_i be a positive document and F_i be a negative document. Given Q, $\{E_i\}$ and $\{F_i\}$, the query is refined to obtain a new query $Q^{'}$ as follows.

$$W(Q^{'}, j) = W(Q, j) + \sum_{i=1}^{m} W(E_i, j)/m - \sum_{i=1}^{n} W(F_i, j)/n$$

The relevance feedback by query refinement is a local modification of the retrieval behavior in the sense that it can give the system better performance only for the current query. Despite it is likely that the user tends to make similar abbreviations or exaggerations over retrieval sessions, user's viewpoint captured by the query refinement cannot be reused for later retrieval sessions.

3.2 Restructuring Document Space

One way[4] to preserve user's preference over retrieval sessions is relevance feedback by restructuring the document space[15, 9, 8]. This can be implemented by changing the vectorization procedure's behavior, using the dictionary vector as a parameter. Here we only investigate cases the list is shrunken[5]. Instead of physically shrinking the list, we regard a $\{0, 1\}$ vector, with the same length as the list, in the same light as a dictionary vector which only consists of words whose corresponding element of the vector is 1.

The relevance feedback is to find the optimal dictionary vector for a user. This is not a simple subspace search if we take correlations of words into account, against the original tf-idf weighting, like the followings. We use weights U defined below, instead of W.

$$U(q, i) \overset{def}{=} \frac{\sum_j [tf(q,j) \cdot idf(j) \cdot M(i,j)]}{\sqrt{\sum_k [\sum_j [tf(q,j) \cdot idf(j) \cdot M(k,j)]]^2}}$$

Here $M(i, j)$ denotes a correlation between w_i and w_j. The value can be determined either statistically by frequency of co-occurrence of the words, or semantically by distance between the words in a thesaurus.

3.3 Empirical Results

We have implemented an experimental IR system to see effectiveness of our method. Here M is statistically determined. First a user submit a query and the system returns documents based on the initial dictionary vector in which all the elements are 1. Then the user is asked to rank positive documents for arbitrary number.

---

[4] The other ways include adaptation in a distance function, though it is difficult to deal with correlations of words in this approach[9].

[5] If the original list contains a large number of words, this is an insignificant restriction.

A new dictionary vector which is optimally appropriate to the user is searched among the space of the dictionary vectors. A candidate of the new dictionary vector d' is created from the current dictionary vector d using heuristics. d' is regarded better than d if

$$distance_{d'}(D_1, Q) \leq \cdots \leq distance_{d'}(D_m, Q) \qquad (1)$$

$$\sum_{i=1}^{m} distance_{d'}(D_i, Q) \leq \sum_{i=1}^{m} distance_{d}(D_i, Q) \qquad (2)$$

stand, where Q denotes the query, D_i denotes a ranked positive document (i being the rank) and $distance_d(X, Y)$ denotes distance between documents X and Y based on a dictionary vector d. The space of the dictionary vectors is searched by depth-first manner with backtracking.

We use FAQ (Frequently Asked Questions) documents extracted from a VC++ mailing list archive which consist of approximately 12,000 messages, as the document set for the experiment. We choose five queries to be tested. Each query is associated with the reference set chosen by a VC++ expert to evaluate quality of returned documents by the IR system.

3.3.1 Effect of Correlation in Weighting

To evaluate our weighting scheme we compare the difference of the retrieval result between the tf-idf weighting and the weighting based on U. The experiment is done with the initial dictionary vector. Table 1 shows precision and recall of tf-idf method and of our weighting scheme for the five queries. The result shows advantage of our method over tf-idf for all the queries.

	tf-idf method		our weighting scheme	
	precision	recall	precision	recall
$query_1$	35.25%	75.0%	50.0%	100.0%
$query_2$	60.0%	75.0%	80.0%	100.0%
$query_3$	12.5%	50.0%	25.0%	100.0%
$query_4$	50.0%	50.0%	75.0%	75.0%
$query_5$	75.0%	100.0%	75.0%	100.0%

Table 1: Difference with respect to weighting schema

3.3.2 Relevance Feedback by Restructuring Document Space

We examine effectiveness of the relevance feedback by the document space restructuring that is carried out as optimization of a dictionary vector. The weighting scheme U is used in this experiment. We compare the difference between retrieval results according to the initial dictionary vector and to the optimized one through the search.

Figure 2 shows a process of search for the optimal dictionary vector. The space of dictionary vectors is searched as follows. Starting with the initial dictionary vector, a candidate of a new dictionary vector is generated using heuristics. If the candidate is evaluated as superior than the current one in terms of inequalities (1)(2), the candidate will be assigned a numerical ID in ascending order, where the initial dictionary vector has the ID being 1. In Figure 2, the IDs are shown in the X-axis and values of the evaluation function used in inequality (2) are shown in the Y-axis. The lower the value of the evaluation function of a dictionary vector, the more highly it is evaluated. The evaluation is locally improving and temporarily worsened when the backtracking occurs. In Figure 2 the 28-th dictionary vector is finally determined as the best one through the search where the search tree has 42 nodes and the maximum depth is 13. Precision and recall of retrieval with the optimized dictionary vector are shown in Table 2. The result shows relevance feedback by the document space restructuring through the optimization of the dictionary vector is effective. Although the searched space is relatively small as compared with the whole space of the dictionary vectors, the optimal dictionary vector from the search yields significantly better performance.

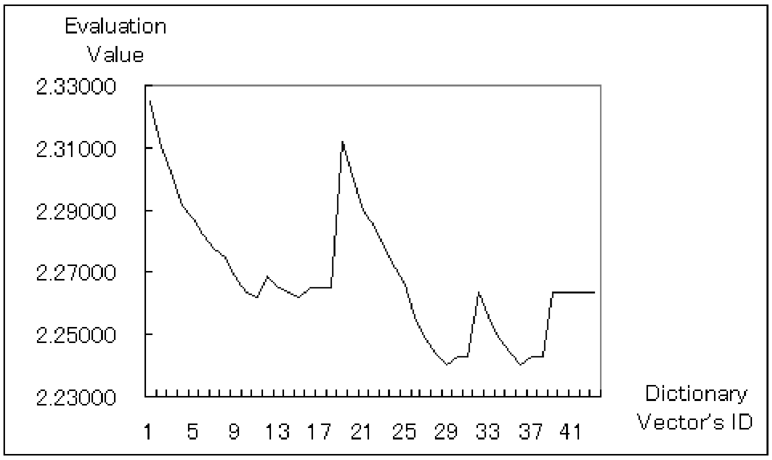

Figure 2: Search process of optimal dictionary vector

	initial dictionary	optimized dictionary
precision	50.0%	100.0%
recall	66.6%	100.0%

Table 2: Improvement by dictionary optimization

3.3.3 Visualizing Document Space

The document space is multi-dimensional and the state of affairs in it is difficult to understand. We have created a simple visualizer to help users understand the restructuring of the document space. A user specifies two words from the list of index terms, and the visualizer shows a 2-D map which is a projection of the document space into a plane defined by the two words. In Figure 3, how the adaptation of the document space was going in one of the experiments is shown. The left map displays the document space after restructuring and the initial document space is shown in the right. In the maps, each document is represented by a dot. The query is also represented by a dot, using a different color, and specified by an arrow in Figure 3.

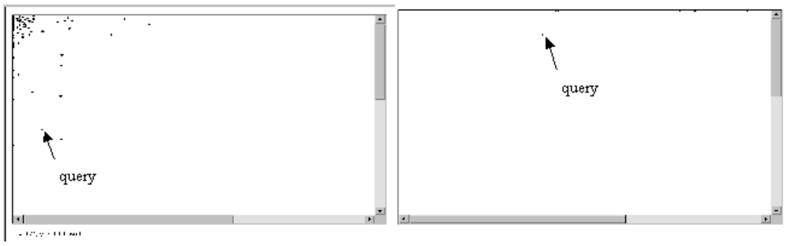

Figure 3: Visual map of document space

4 Still Image Perception

A human obtains millions of information from vision and simulating all of them exceeds this study's scope. In this paper we focus on color information.

Usually a lot of colors are included in input stimuli. However we do not care most of them. We *selectively* see colors, possibly based on our purposes. A previous study in art suggests that an index color need not to occupy a large area in an image. For example, when we see a red rose on a white background, we feel the red is the index color[7].

The previous study also proposes an algorithm to extract index colors from an image[7]. The algorithm has several parameters to be tuned for a purpose. Our approach employs the algorithm as a transformer for color information. The parameters of the algorithm are adjusted to make a space of index colors useful for a task.

4.1 Index Color Extraction

The algorithm consists of the following three steps.

4.1.1 Reducing the Number of Colors

Usually a number of colors are used in an image and it is computationally prohibitive to treat all of them respectively. The algorithm first reduces the number of colors in an image with the following procedures.

1. All colors in an image are converted from RGB space into $L^*a^*b^*$ space. The distance of two colors is closer to human intuition in the $L^*a^*b^*$ space than in the RGB space. Appendix A shows the details of the conversion.
2. The color list, which consists of a color and its frequency, is created as the following. When a new color is added to the list, if the distance between the color and one of colors already in the list is less than a threshold p, then the new color and the color in the list are mixed each other regarding their frequencies.
3. Less frequent colors in the color list are likely to be noise. If a color's frequency is less than 0.5% of all the pixels in the image then the color is mixed with the nearest color in the list regarding their frequencies.
4. The number of colors in the list is reduced by mixing the two nearest colors regarding their frequencies. This step is iterated until the number of colors becomes m.

5. A histogram is created for each of L^*, a^*, b^* by partitioning the axis into $2m$ regions and summing the frequencies in the region. The maximum number of convexes m^* in the histograms represents the *true* number of colors[6]. The number of colors in the list is reduced by mixing the two nearest colors regarding their frequencies until it becomes m^*. Then the histograms are recreated and m^* is recalculated. The procedure is iterated until m^* is not changed by the recalculation.

4.1.2 Prominence of Colors

An index color must be prominent. Prominence L of a color is defined as the following.

$$L \overset{def}{=} \sqrt{(e \cdot C_1)^2 + (f \cdot C_2)^2 + (g \cdot C_3)^2}$$

where C_1 denotes frequency of the color, C_2 denotes temptation of the color and C_3 denotes contrast of the color, and e, f and g are parameters. C_1, C_2 and C_3 are defined as the following.

- Frequency

  $$C_1 \overset{def}{=} \frac{\text{Frequency of the color}}{\text{The number of pixels in the image}}$$

- Temptation
  First, the color is converted into HSV space. Appendix B shows the details of the conversion. A value H^* is used instead of H. H^* is determined as shown in Appendix C. Then

  $$C_2 \overset{def}{=} \frac{s \cdot H^* + t \cdot S + u \cdot V}{s + t + u}$$

  where s, t and u are parameters.

- Contrast
  Let V^* denote the V value of the color created by mixing all the colors in the list but the color under consideration. Then

  $$C_3 \overset{def}{=} |V - V^*|$$

4.1.3 Specifying Index Colors

The L is calculated for all the colors in the list. Index colors are n colors with larger L values in the list.

4.2 CF Image Clustering

We have implemented the algorithm which has e, f, g, s, t, u and m as controllable parameters. We applied the program to 128x88 still images captured from TV commercial films. The number of index colors, n, is fixed to 2.

As a preparation, we specified a set of products which attracts people at the same time, using market research data. Let us call the set P. In our experiment, P consists of computer game products and portable audio sets. Now, we are trying to extract common features of CFs advertising P, which can be the appeal of P. The closer those images are placed in an

---

[6] The number of different colors that the designer intended to use.

internal representation, the easier to see their common features, thus the representation is more suitable for the task.

First the program with randomly chosen initial parameters was applied to 78 images advertising P. Then we performed 2-means clustering in 2-fold HSV space[7] of index colors of the images. We tuned the parameters so that all the images of P come closer, by picking an image from each clusters and changing the parameters according to heuristics based on the difference between the two images. After tuning, the program is reapplied to the images.

As the result, variance of the images of P in the 2-fold HSV space is decreased. We can say that the space after the tuning is better representing common features of the images of P than one before the tuning.

4.3 Interpretation Using Kansei Scale

We also attempted to express the common features of the images of P using Kansei scale[2]. The method allows us to see intuitive descriptions of the images as a whole. Table 3 shows the result. In the table, a larger negative number shows stronger relationship towards a word in the right hand, while a larger positive number shows stronger relationship towards a word in the left hand.

Warm – Cold	0.328002
Soft – Solid	0.25045
Natural – Artificial	-0.47079
Bright – Dark	0.36981
Flamboyant – Plain	1.03151

Table 3: Kansei scale value

From the table, we can say that images of P's CF are artificial and flamboyant. This is quite reasonable regarding that P consists of computer game products and portable audio sets.

5 Conclusion and Future Works

We pointed out the importance of good perceptions in problem solving for intelligent softwares and formalized a perception scheme using vector spaces. The method can be seen as a perceptive-level version of attribute selection in inductive learning[10, 11]. We also constructed a prototype system for text documents and empirically evaluated the effectiveness of our method in IR context. The result is so far promising. Furthermore, we implemented an experimental system for still image perception.

However we have to say that our study is still in its preliminary stage. First, in the IR context, the restructuring is much more computationally expensive comparing to the query refinement. We need not only to make effort to find an efficient approximate algorithm, but also to show the effect of learning over retrieval sessions, which is yet to be shown in our experiment and cannot be achieved by the query refinement. Second, we have

---

[7] A 6-dimensional space with H, S and V of each of 2 index colors.

not fully utilized the advantage of a vector space as an internal representation scheme of an intelligent software. We should investigate possibility of concept formation in the space and its impact to real world applications.

Our final goal is to construct a decision support system, which can analyze multimedia data from various viewpoints that are implemented as parameters of transformers.

References

[1] C. Buckley and G. Salton. Optimization of relevance feedback weights. In *Proceedings of the 18th Annual International ACM SIGIR Conference on research and Development in Information Retrieval*, pages 351–357, 1995.

[2] Yamazaki H. and Kondo K. A method of changing a color scheme with kansei scales. In *Proc. of 8th International Conference on Engineering Computer Graphics and Descriptive Geometry*, pages 210–214, 1998.

[3] D. R. Hofstadter et al. *Fluid Concepts and Creative Analogies: computer models of the fundamental mechanisms of thought.* BasicBooks, 1995.

[4] M. Ikeda. *Foundation of Color Engineering.* Asakura Shoten, 1980. in Japanese.

[5] B. Indurkhya. *Metaphor and Cognition*, volume 13 of *Studies in Cognitive Systems.* Kluwer Academic, 1992.

[6] R.S. Michalski. A theory and methodology of inductive learning. In Michalski et al., editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, chapter 4, pages 83–134. Tioga Publishing Company, 1983.

[7] Y. Morohara et al. Automatic picking of index colors in textile pictures for designers. *Trans. Inf. Process. Soc. Jpn.*, 36(2):329–337, 1995. in Japanese.

[8] T. Murakami et al. Friendly information retrieval through adaptive restructuring of information space. In *Proc. of AIE/IEA 2000*, 2000.

[9] T. Murakami and R. Orihara. Friendly information retrieval through adaptive restructuring of information space. *New Generation Computing*, 18(2):137–146, 2000.

[10] R. Orihara. Explanation-based feature subset selection for decision tree induction. *Journal of JSAI*, 14(2), 1998.

[11] R. Orihara. *Knowledge Description and Semantics in Non-Deductive Reasoning.* PhD thesis, University of Tsukuba, 1999.

[12] S.J. Russell. *The use of knowledge in analogy and induction.* Pitman Press, 1989.

[13] G. Salton et al. Automatic structuring and retrieval of large text files. *Communications of the ACM*, 37(2):97–108, 1994.

[14] H. Tsukimoto. Extracting propositions from trained neural networks. In *Proc. of IJCAI '97*, pages 1098–1105, 1997.

[15] C.T. Yu. A formal construction of term classes. *Journal of the ACM*, 22(1):17–37, 1975.

A Conversion from RGB to $L^*a^*b^*$[4]

Let R, G and B denote R, G and B coordinates of the color. Then X, Y, Z, X_0, Y_0 and Z_0 are defined as the following.

$$\begin{aligned}
X &\overset{def}{=} 2.7689R + 1.7517G + 1.1302B \\
Y &\overset{def}{=} 1.0000R + 4.5907G + 0.0601B \\
Z &\overset{def}{=} 0.0565G + 5.5943B \\
X_0 &\overset{def}{=} 95.045 \\
Y_0 &\overset{def}{=} 100 \\
Z_0 &\overset{def}{=} 108.892
\end{aligned}$$

The $L^*a^*b^*$ coordinates are defined using the values.

$$\begin{aligned}
L^* &\overset{def}{=} 116\left(\frac{Y}{Y_0}\right)^{\frac{1}{3}} - 16 \\
a^* &\overset{def}{=} 500\left(\left(\frac{X}{X_0}\right)^{\frac{1}{3}} - \left(\frac{Y}{Y_0}\right)^{\frac{1}{3}}\right) \\
b^* &\overset{def}{=} 200\left(\left(\frac{Y}{Y_0}\right)^{\frac{1}{3}} - \left(\frac{Z}{Z_0}\right)^{\frac{1}{3}}\right)
\end{aligned}$$

B Conversion from RGB to HSV[4]

Let R, G and B denote R, G and B coordinates of the color. Then the H, S and V coordinates of the color, H, S and V respectively, are defined as the following, using $r \overset{def}{=} \frac{V-R}{V-X}$, $g \overset{def}{=} \frac{V-G}{V-X}$ and $b \overset{def}{=} \frac{V-B}{V-X}$.

- $V \overset{def}{=} max(R, G, B)$
- $S \overset{def}{=} \frac{V-X}{V}$ where $X \overset{def}{=} min(R, G, B)$
- H is decided depending on how V is decided.
  - If $R = V$
    - If $G = X$ then $H \overset{def}{=} \frac{5+b}{6}$
    - Otherwise $H \overset{def}{=} \frac{1-g}{6}$
  - If $G = V$
    - If $B = X$ then $H \overset{def}{=} \frac{1+r}{6}$
    - Otherwise $H \overset{def}{=} \frac{3-b}{6}$
  - If $B = V$
    - If $R = X$ then $H \overset{def}{=} \frac{3+g}{6}$
    - Otherwise$H \overset{def}{=} \frac{5-r}{6}$

C Determining H^*[2]

H represents hue of the color. H^* is determined using Table 4 shown below, depending on the hue.

Hue	H^*
Red	1.00
Yellowish red	0.89
Yellow	0.93
Yellowish green	0.61
Green	0.28
Blue green	0.22
Blue	0.18
Blue violet	0.08
Violet	0.00
Red violet	0.35

Table 4: Hue and H^*