UNIVERSITY OF WATERLOO
Faculty of Engineering and Mathematics
Department of Software Engineering

Design and Implementation of Speed Detector and Speeder Chaser Robot

University of Waterloo
Waterloo, ON

Prepared by

Mariam Al-Aziz; mmalaziz; #20526491
Arash Mortazavi; a2mortaz; #20354252
Jiazhou Li; j449li; #20512864
Ernest Ho; e35ho; #20534420
Ye Li; y665li; #20532765
Difei Zhang; d87zhang; #20530592
Daniel Joseph; d8joseph; #20502475

1A Software Engineering
November 28, 2013

Mariam Al-Azizi
328 Regina St. N
Waterloo, ON
N2J 3J6

November 28, 2013

Dr. Krzysztof Czarnecki, Professor
SE 101, Fall 2013
University of Waterloo
Waterloo, ON
N2L 3G1

Dear Prof. Czarnecki:

The enclosed report, “Design and Implementation of Speed Detector and Speeder Chaser Robot”, is related to the Robot project in SE 101 course. It is based on our team's (team 11) work on the project.

The problem addressed in the project is automatic detection of objects moving at a higher speed than the limit and programming a robot to follow such object. Hence the project's goal became constructing a model for an autonomous robot that is able to assist police in enforcing speed limits. The realization of this model can contribute to greatly reduce accidents, and keep roads safe from accidents caused by speeding vehicles.

The project's outcome was an implementation that fulfils the goal that was initially set. The Scribbler 2 robot is successfully able to detect the object characterized by a distinguishable physical description (a colour), that is moving higher than a speed limit. It also is able to follow this object effectively in a non-linear course.

This report explains the design constraints, decisions and the implementation steps that the team went through to succeed in the project. It also includes in-depth analysis of the possible solutions for implementing each phase of the project and supporting data on the tests executed.

The team would also like to thank the teaching assistants of the course, as well as yourself, for assisting the team when facing various issues.

The team hereby confirms that the team did not receive any help, other than what is mentioned above, on the project and to write this report. The team also confirms that this report was not previously submitted for academic credit at an academic institution.

Sincerely,

SE 101 Team 11

Mariam Al-Azizi (205206491), Arash Mortazavi (2035425), Jiazhou Li (20512864), Ernest Ho (20534420), Ye Li (20532765), Difei Zhang (20530592), Daniel Joseph (20502475)

Executive Summary

The motivation behind this project is to help contribute in the construction of the autonomous robot that is able to assist police in enforcing speed limits. The realization of such a goal will greatly reduce accidents, and keep our roads safe. The purpose of this project is to produce a robot capable of detecting and pursuing other speeding vehicles. This project is aimed at prototyping for future autonomous police robots that are capable of assisting police in their duties.

The team has gone through stages of assessing methods used by the robot to detect speed limit violation and to chase the speeder object. The problem was further divided into three parts: speed detection, object recognition and tracking to clarify the functions required to be implemented. Hence, each part was analyzed and designed separately, with problems and benefits in each method. In the end, after evaluations performed on the solutions, solution algorithms that fit most with the criteria were implemented for the final deliverable.

For the speed detection phase, an algorithm using the Scribbler 2's built-in infrared sensors and a customized mathematical model was implemented. For the object recognition phase, image processing on the colour of the object was considered to be the best solution. Finally, the tracking phase was implemented by adjusting the robot's direction according to the perceived position of the object. During the final tests, the completed program demonstrated the required functionality with considerably accurate results. Nonetheless, recommendations were given for future projects of a similar nature.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background

Speed limits are often enforced by the presence of police cars monitoring roads or highways for speed offenders. In the case where a car is detected to pass the speed limit, it needs to be tracked down so that the driver gets penalized. As a simulation to the real life police car, the scribbler robot will follow the general steps of a road-monitoring police. It will be placed on the road's side, detect the speeds of cars passing by, and when a speed offender has been recognized, the robot will try to catch it.

1.2 Purpose

The purpose of this project is to produce a robot capable of detecting and pursuing other speeding vehicles through the use of basic programming principles via Python programming language.

2 Criteria and Constrains

2.1 Criteria Description

The scribbler police project has the following functional requirements as well as non-functional one.

Functional requirements consist of speed detection, chasing a speeding object through colour recognition, adjusting the robot's path according to the object's maneuvers, and reporting

the detected speed. If speed exceeded a certain threshold, the scribbler has to effectively chase the object until it was within proximity of the object.

Non-functional requirements include detecting speed reliably and with a small percentage error. It was also desirable that the chase algorithm could detect different colors to improve the functionality of the robot.

2.2 Design Constraints

The team encountered a number of constraints while developing the design control system for the scribbler robot. The camera on the fluke card has a resolution of 256x192 pixels. The low resolution caused issues while developing and testing the image processing algorithm. The camera also had poor image quality which imbued non uniform changes in hue throughout the image. In addition to that, the computer had limited processing power and sending image information over Bluetooth slowed it down to a point where the robot was not able to function in real time. The image processing method for detecting speed was discarded due to this particular constraint. However, the team found a work around for the chase algorithm by compressing the image taken by the camera on the fluke card to enable faster transmission over Bluetooth. Among other hardware limitations, the team faced difficulties while working with the obstacle sensors on the fluke card. The data received had a high error rate. This reduced the functionality of the chase algorithm. The stall sensor on the scribbler had trouble functioning, and its use was curtailed early on in the design process.

3 Analysis and Design

To complete the tasks described in the criteria, the scribbler needs to execute three distinct procedures that include: speed detection, image processing and tracking. The implementation of these procedures will be discussed separately in the following subsections.

3.1 Speed Detection

To determine whether a passing object is going over the speed limit or not, the scribbler police needs to be able to calculate the object's speed using input from its sensors. This task is especially vital to the project's functionality as errors made at the speed detection phase can result in inaction against speed offenders or unwanted action done against law-abiding travelers. Therefore, it is essential that the calculated speed is both reliable and accurate. As for this task, the evaluation criteria will focus mainly on reliability and accuracy.

Three solutions exist to implementing the algorithm for this task and they are mainly distinguished by their choice of input methods. The first method entails using the three infrared sensors on the fluke card to detect the passing object. Given the time the object used to pass through the infrared sensors' detection range, a correlation between the speed and the used time can theoretically be found. This is possible since the distance the object has to travel inside the detection range should be constant as long as the distance between the robot and the imaginary road is controlled, as shown in **Figure 3-1**.

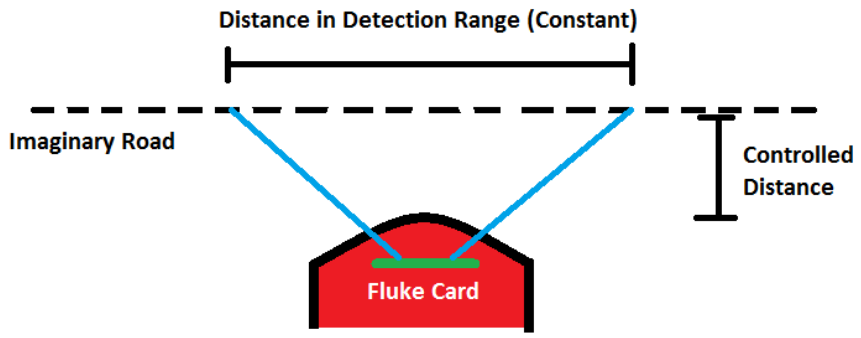


Figure 3-1. First Solution's Scenario Illustration

However, the problem with this solution is that the input from the fluke card sensors is highly unclear and unreliable. The range of the sensor input is between 0 and 6400 [1], which means the team has to define a threshold to determine whether an object was detected or not. Tests showed that sensor inputs fluctuate significantly even with stationary objects and that

detection thresholds change with differently coloured objects. Hence, this solution is not satisfactory in its reliability and accuracy.

Another alternative is similar to the first solution, but uses the Scribbler robot's two built-in infrared sensors. These sensors have a smaller detection angle, again giving a theoretically constant distance between the detection ranges as illustrated in **Figure 3-2**.

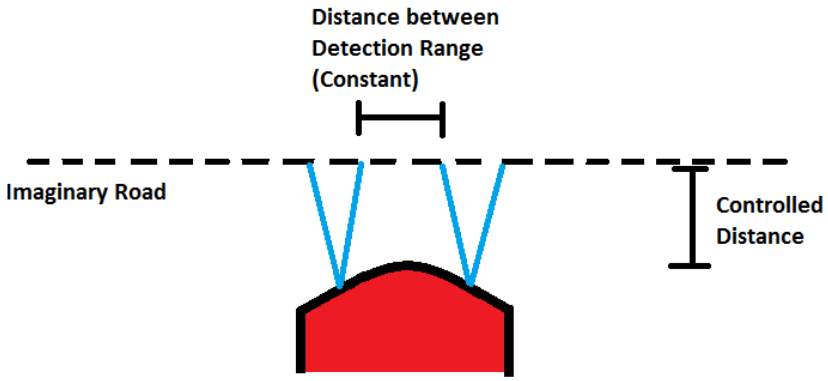


Figure 3-2. Second Solution's Scenario Illustration

The input from the built-in sensors are much more reliable, since they return only 0 or 1, distinguishing clearly between a detected-state and a non-detected state. Testing efforts also showed that these sensors reliably detect passing objects of various colours and shapes. Thus solution 2 with the built-in sensors meets the evaluation criteria of reliability and accuracy more closely than solution 1.

The third possible solution requires image processing to recognize that an object is passing through. The speed of the object can be calculated by the distance of camera range divided by the time taken for the head of the object to pass through the camera range, as illustrated in **Figure 3-3**.

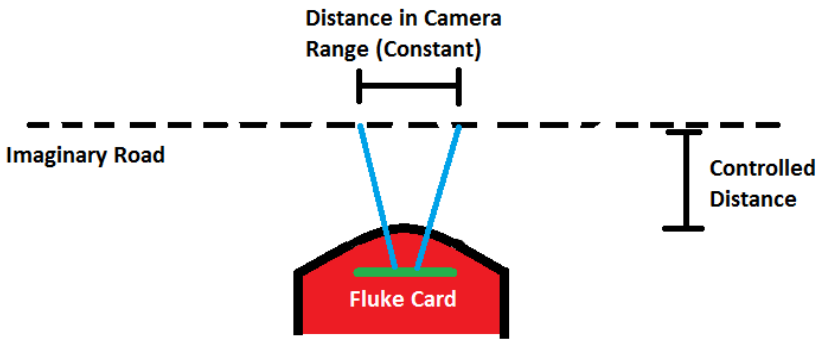


Figure 3-3. Third Solution's Scenario Illustration

Since the angle of the camera range is arguably more consistent than those of the infrared sensors, a more reliable correlation should be obtained. However, image processing takes significantly longer time than the sensors' inputs hence, precision with the time measured will be greatly compromised. In consideration of the precision loss and hence accuracy loss, and the longer time required to implement this algorithm. Therefore, solution three was abandoned.

Comparing the three solutions, solution two is more satisfactory given the importance of reliability and accuracy at the speed detection phase. Solution two was thus implemented, using a model that maps a time measured by the robot to a corresponding speed. A comprehensive field test was done to provide data for this model (see **Appendix A-1**), and the graph is shown below:

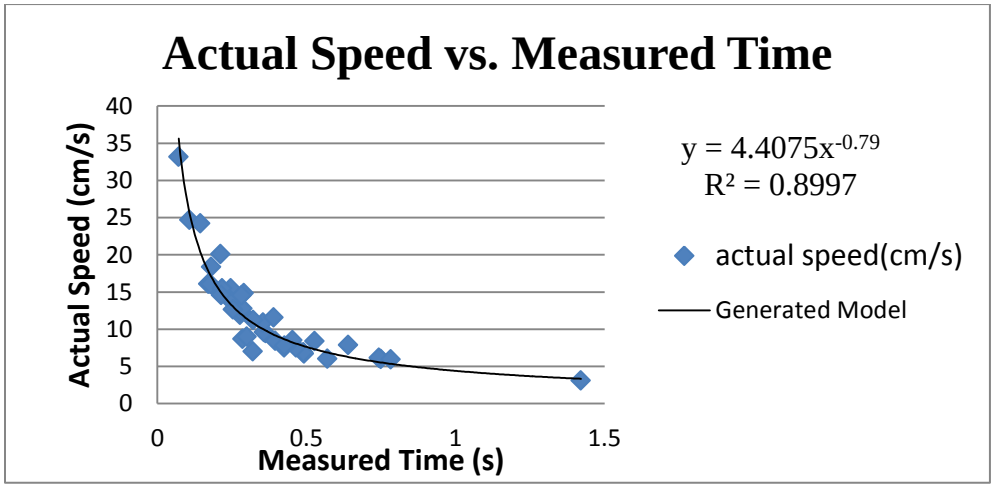


Figure 3-4. Speed vs. Time Correlation Graph

Here, Microsoft Excel suggests a model where time taken to the power of -0.79 would be proportional to the speed. Ideally, speed should be inversely proportional to the measured time, given a constant distance. However, the team decided to use this model for its practical representation of real world data and its reasonably high R^2 value.

3.2 Image Processing

3.2.1 Development of prototypes

There were several different prototypes the group experimented prior to the final algorithm, but each method has their problems. Initially, the line recognition is considered. By making the target object leave a trail marked by pen, the robot would analyse the mark using the line sensor. This approach is quite simple and effective in most situations, especially when combined with obstacle avoidance function. However, the main drawback is the practical viability, since typical objects would not leave noticeable mark through its path, thus using this method defeats the purpose of the proposal.

As the line detection option closes, the only possible methods of tracking are through infrared sensor on the scribbler, infrared obstacle sensors on the fluke, and camera and image processing. The sensors on the scribbler and fluke are very basic and provide little and unreliable results. The scribbler IR sensor only output whether an obstacle is present or not, while the fluke chip sensor outputs a number indicating the distance. However, the number has little correlation with actual distance, and rapidly fluctuates while no changes are made. Apart from distance, no characteristic of the obstacle is returned. Therefore, the group decided to use camera since it provides the most information.

After conducting research within the Myro library, as well as the image manipulation libraries available for Python, two potential methods of following an object were discovered. The first method was to use a built-in function in the Myro library, called ‘blob’, which essentially takes input values based on the YUV color space, returning a black and white image with the specified colours filtered out. The second method was to program an algorithm to filter out the

specified input colours, but using the Python Image Library (PIL) rather than the Myro library. After thorough testing and comparisons, the latter method was chosen to be used for extracting a colour from an image, enabling the Scribbler robot to follow an object.

3.2.2 Method 1: Blob (Myro Library)

The first method involves utilizing the built-in image processing functions in the Myro library, specifically the 'blob' object. The two functions required for extracting a specific colour from an image are listed above: *takePicture()* and *configureBlob()*. In order to filter out a colour from an image, the *takePicture()* function is called first, essentially capturing an image from the Scribbler Robot's camera. Next, the picture is analyzed, based on the average colours found after pixel analysis. Through a conversion from RGB to YUV, the new values are used to parameterize the *configureBlob()* function, and thus, a new image with the desired colours are outputted.

3.2.3 Method 2: Custom Algorithm (Python Image Library)

The main difference in the second method involves using the HSV colour space, rather than the YUV colour space, enabling colours to be analyzed through hue, saturation, and value. The main function used is called *getObjectColor(picture, clrL, clrH)*. This function takes a picture and the colour range to process the image and return a black and white filtered picture, as well as the average X and Y coordinates of the colour. The algorithm for this method is quite simple but effective. First, an image is taken by the Scribbler Robot, and input into the *getObjectColor()* function, with a desired colour range to filter out (e.g. clrL = 175, clrH = 260 for blue). Next, each pixel in the image is processed, using the condition that the colour in HSV must have a Hue in between the inputted colour range, with a saturation and value greater than 0.5. The hue analysis enables the colour to be obtained, while the saturation and value condition filters out faint colours. Once the image processing is completed, the average X and Y coordinates of the colour are returned into the main program, to be used in object path recognition. Another function utilized in this algorithm is *compressImage(picture, factor)*, which reduces the resolution of an image by a certain factor, such as 4. This reduces the image size

from 256 by 192 to 64 by 48 pixels, effectively reducing the processing time required to get the colour from an image, while maintaining an accurate reading.

Sample Images (See Appendix B for a full list) for Comparisons

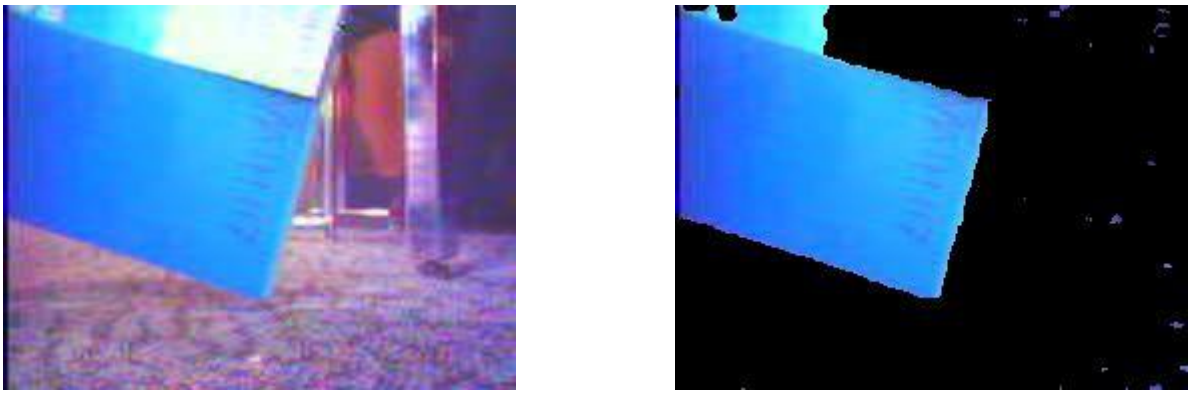


Figure 3-5. Blue Object

Figure 3-6. Red Object

3.2.3.1 Comparison

3.2.3.1.1 Complexity

The first comparison that can be made is the complexity of each algorithm. When the 'blob' object was used, it required many steps before the desired final output could be obtained. It first needed to take a picture, analyze it to adjust the blob configuration, obtain a blob image, and finally, reanalyze the new image for the average X and Y coordinates of the colour. In addition, since the function was built into the Myro library, the YUV colour space had to be

used, which involved manually programming a function to convert between RGB and YUV. In the second method, the algorithm involved much simpler and faster steps. First, the image was taken and compressed, then the pixel colours were analyzed, and then the output was obtained. For conversion between the RGB and HSV colour spaces, the "colours" library already has built-in functions to perform the conversion.

3.2.3.1.2 Computation Time

In this specific project, reducing calculation time is important as when the Scribbler Robot is chasing an object, it needs to be able to locate the object quickly, in order to accurately follow the object. If too much time is needed in performing imaging processing, the object would outrun the robot, and thus, render chasing impossible. Comparing the two methods of image processing, the second algorithm runs much more efficiently than the first. Since the first method uses the built in 'blob' functions from the Myro library, it restricts the input to a "blob image" that must be obtained by the camera. In the actual method, a regular image is used, enabling additional image processing to be done. A specific example is using the *compressImage*() function. After performing a multitude of tests on processing time, when the image is compressed, it processes colour much faster. In 50 trials, the uncompressed image required an average of 1.98 seconds in computation time, relatively slow compared to the 0.251 seconds on the compressed image (See Appendix A for the full list of trials).

3.2.3.1.3 Colour Space

Colour spaces are used to describe colours in an image. In each method of image processing, a different colour space is used. The first method uses the YUV colour space, where Y represents luminance and UV represents two colour components [2]. In the second method, the HSV colour space is used for colour detection, where H stands for hue (colour), S for saturation (intensity), and V for value (brightness) [3]. Comparing the two colour spaces, for the purposes of this project, the HSV colour space is much more effective. Firstly, it only has one value describing colour, compared to the two in YUV, enabling a more consistent method of colour detection. When two values are used, detecting a specific colour becomes more difficult, as the values change more dynamically. Secondly, HSV uses saturation and value, two values to describe the shade of a colour, compared to one in YUV. This allows for a more accurate method

to differentiate between the colour of the object and the background, since background colours are commonly darker and less colourful.

3.2.3.1.4 Overall Observation

After careful analysis of the problem and thorough testing of the different algorithms, it is evident that the second algorithm triumphs over the first in this specific solution. The above explanation outlines the steps used in the comparison between the two methods of image processing and the criteria used to evaluate the best solution for this project.
Describe image processing issue

3.3 Tracking

3.3.1 Design

The purpose of this algorithm is to allow the scribbler robot to follow an object of desired colour.

The "Tracking" algorithm relies heavily on the image processing algorithms discussed above. The algorithm starts off by taking continuous streams of pictures, compressing the pictures using the compressImage() function and then calling getObjectColor() to process the compressed image for the desired coloured pixels. The getObjectColor() function returns the average x and y position of the target pixels, which is used to estimate the object's position relative to the robot's camera. The robot would then adjust itself (if necessary) so that the object is directly in front of the camera. This adjustment is done by directly controlling the speed of the left and right motors. By setting the left motor to be slower than the right motor would cause the robot to turn left, and vice versa. Note that the speed of the turns can be adjusted by setting the speed of the motors closer or further apart, this is to allow the robot to adjust the speed of its turns so that should the object make an abrupt turn, the robot would not lose sight of it.

The getObjectColor() function also returns the number of coloured pixels in the image. Should there not be enough target coloured pixels, the robot would interpret it as there is no object in front for it to follow.

3.3.2 Design Justification and Prototypes

Code Organization:

We decided to keep the image processing algorithm and the tracking algorithm separate because it is a good style for developing modular programs and allows for easier testing and debugging.

Turning:

In the early phase of development, the turning was discontinuous. Every time the robot needed to adjust its position, it stopped and turned. We soon realized that with this algorithm the robot cannot keep up with the object it is tracking. To solve this, we decided to adjust the speed of individual motors for continuous turning. We implemented this better alternative in our final code which allowed the robot to adjust its position without stopping.

Also, in our testing stages we observed that the robot could not turn fast enough if the object quickly turned left/right. To perfect the turning, we made the speed of the turning dependent on the average x-position of the object, so that if the object is about to move out of the robot's view, the robot would turn faster.

3.4 Current problems

The current object chasing algorithm allows scribbler to follow the object whenever it is within range of the camera. There are, however, many limitations to the current algorithm.

1. **Colour requirement of tracking:** Since tracking is entirely dependent on camera and colour identification, if the object does not have a predominant colour or if the

background contains similar colour to the object, the image processing will fail to distinguish the object.

2. **Lighting and reflection:** If the object is highly reflective and placed into an atmosphere with limited source of light, the object would experience major colour changes with different lighting, which would cause the tracking to malfunction.

3. **Obstacles and walls:** The current algorithm cannot deal with obstacles, since blockage caused by obstacles would hide the colour from the camera, and robot will stop since object is not within vision range.
4. **Camera range:** This would also happen when target is outside the small camera range, which is problematic since sharp turns done by the target can easily lose the scibbler police.

4 Conclusion

Scribbler Police is a project that aims to simulate a real life scenario of a police car detecting and chasing a speeding vehicle. The project utilised the scribbler's inbuilt hardware capabilities to achieve the design criteria. The control software design was realized through python. Inbuilt infrared sensors were adjudged optimal for object detection. The tracking algorithm employed identifying the centerline of the moving object's coloured pixels to redirect the robot.

Results shown by this implementation successfully solved the problem and carried out the functions specified in the criteria descriptions. Non-functional requirements were also met to a reasonable extent. Therefore, despite the constraints posed by hardware and resource limitations, the scribbler police project created a constructive prototype towards achieving autonomous police enforcement.

5 Recommendations

To develop a relatively more complete and functional algorithm that can withstand the difficulties discussed in the previous section, better hardware is required. One possible approach is to use more accurate infrared obstacle sensors that can detect the shape of the target object and output reliable distance. Additionally, the infrared sensors and the camera should be both facing the object. By detecting and following the shape and distance in conjunction with colour, many limitations mentioned can be resolved. As an object exceeds the speed limit, the robot would record the object's colour and approximate shape, reducing the chance that it will follow the wrong object in the tracking phase. During the process of tracking, the robot will decide its movement based on distance, shape movement, and colour movement, which greatly increase accuracy. High fluctuation in distance indicates the presence of an obstacle, and the robot can use obstacle avoidance algorithms to bypass it.

Map and coordination system will also be valuable to the system. Two dimensional arrays can be created to plot the area of travel in similar fashion to object avoidance, while tracking system could store the position of the target into the memory. Utilizing this system during tracking, if the target goes outside the range of sensors, the robot can use the data to predict the position of the object on the map and make necessary adjustment like rotating and steering to gain vision of the object again. This algorithm, however, requires enough transmission and processing speed for the robot to detect, store, and access the coordination data without delaying its image processing and obstacle detection, which cannot be achieved with the current robot.

References

[1] IPRE Wiki, “Myro Reference Manual,” (Institute for Personal Robots in Education) [online] 24 April, 2013,

http://wiki.roboteducation.org/Myro_Reference_Manual (Accessed: 28 Nov., 2013).

[2] Christopher Wright, “YUV Colorspace,” [online] 2004,

http://softpixel.com/~cwright/programming/colorspace/yuv/ (Accessed: 28 Nov., 2013).

[3] Carey Bunks, “5.2 The HSV Colorspace,” [online] 2000,

http://ie.technion.ac.il/CC/Gimp/node51.html (Accessed: 28 Nov., 2013).

Appendix A

Table A-1. Data obtained during speed detecting tests

Entry Number	measured time(s)	actual speed(cm/s)
1	0.749000072	5.984555985
2	1.421000004	3.075396825
3	0.743000031	6.138613861
4	0.640000105	7.848101266
5	0.569999933	6.019417476
6	0.363000154	9.465648855
7	0.394999981	8.43537415
8	0.492000103	6.695464363
9	0.782999992	5.950095969
10	0.171000004	16.06217617
11	0.246000051	15.46134663
12	0.289999962	14.83253589
13	0.10800004	24.70119522
14	0.071000099	33.15508021
15	0.27699995	11.90019194
16	0.286000013	8.683473389
17	0.49000001	6.85840708
18	0.211999893	20.06472492
19	0.246000051	13.83928571
20	0.144000053	24.21875

Entry Number	measured time(s)	actual speed(cm/s)
21	0.282999992	12.25296443
22	0.319999933	7.013574661
23	0.286000013	12.75720165
24	0.465000153	7.533414338
25	0.357999802	9.567901235
26	0.354000092	10.87719298
27	0.178000212	16.02067183
28	0.213999987	14.72684086
29	0.282999992	14.51990632
30	0.217000008	15.46134663
31	0.180000067	18.34319527
32	0.52699995	8.378378378
33	0.299999952	8.972503618
34	0.424999952	7.542579075
35	0.453000069	8.516483516
36	0.42599988	7.769423559
37	0.320999861	11.23188406
38	0.213999987	14.55399061
39	0.390000105	11.54562384
40	0.253000021	12.65306122

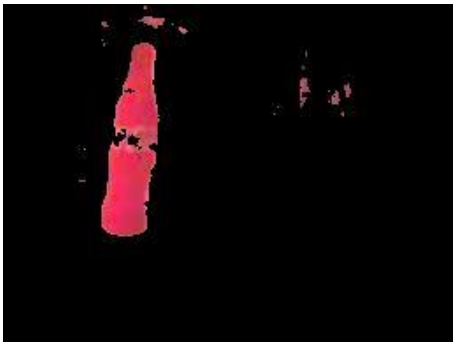
Table A-2. Data Comparing Computation Time (s) Between Compressed and Uncompressed Image Algorithms

Trial	Compressed (s)	Uncompressed (s)	Trial	Compressed (s)	Uncompressed (s)
1	0.281000137	2.003000021	**26**	0.239000082	1.963999987
2	0.288999796	1.984999895	**27**	0.24000001	1.974999905
3	0.273000002	1.980000019	**28**	0.239000082	1.984999895
4	0.242999792	1.980000019	**29**	0.232000113	2.023999929
5	0.269000053	2.006000042	**30**	0.236000061	1.981000185
6	0.246999979	2.023000002	**31**	0.244999886	1.957999945
7	0.246999979	1.99000001	**32**	0.246999979	1.970000029
8	0.284000158	1.976999998	**33**	0.234999895	1.9599998
9	0.251000166	1.963000059	**34**	0.233999968	1.976000071
10	0.259000063	1.963000059	**35**	0.257000208	1.968999863
11	0.286999941	1.96600008	**36**	0.242000103	1.986999989
12	0.245000124	1.985999823	**37**	0.23300004	1.970999956
13	0.23300004	1.980999947	**38**	0.243999958	1.980999947
14	0.234999895	1.968999863	**39**	0.243000031	1.970000029
15	0.233999968	1.994999886	**40**	0.236000061	2.011999846
16	0.243000031	1.955999851	**41**	0.237000227	1.993999958
17	0.253999949	1.974999905	**42**	0.239000082	1.95299983
18	0.322999954	1.980000019	**43**	0.239000082	1.965999842
19	0.285999775	1.969000101	**44**	0.233999968	2.026000023
20	0.253000021	1.973999977	**45**	0.236000061	1.976000071
21	0.26699996	1.960999966	**46**	0.234999895	1.998999834
22	0.303999901	1.968999863	**47**	0.24000001	1.968999863
23	0.25999999	2.024999857	**48**	0.233999968	1.969000101
24	0.247999907	1.973999977	**49**	0.239000082	1.981999874
25	0.244000196	1.997000217	**50**	0.235000134	2.057999849

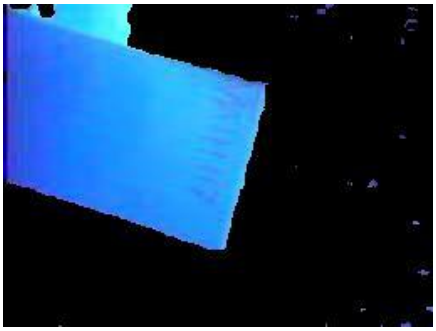
Appendix B

List of Pictures Showing Colour Recognition

Red

Blue

Green