

ROBOTRACKER – A SYSTEM FOR
TRACKING MULTIPLE ROBOTS IN
REAL TIME

by Alex Sirota, alex@elbrus.com

Project in intelligent systems
Computer Science Department
Technion – Israel Institute of Technology

Under the supervision of Noam Gordon, Ronen Keidar

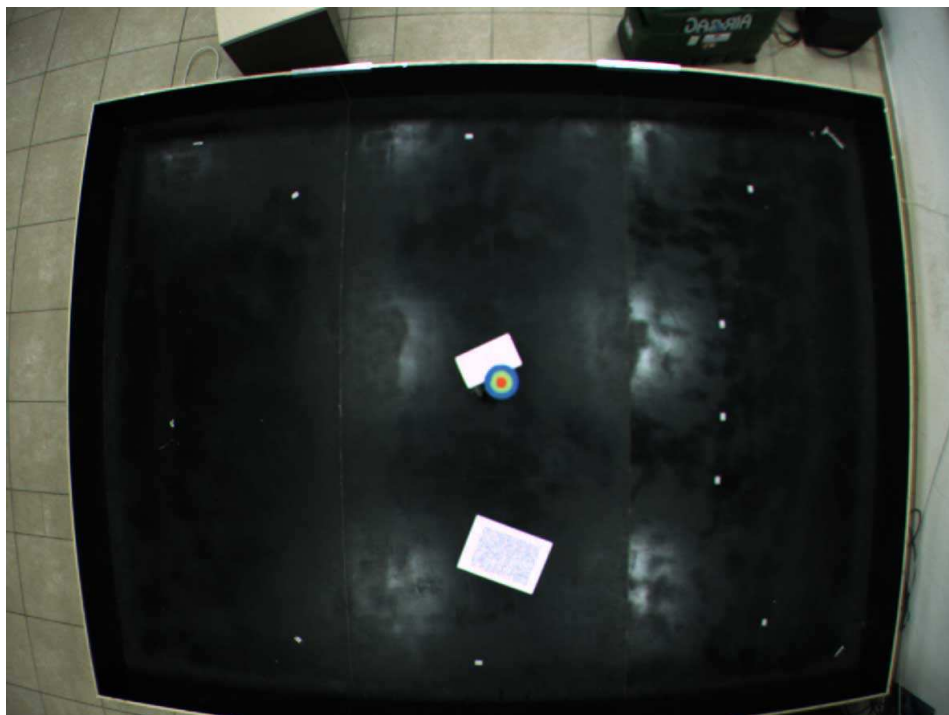
December 2004

TABLE OF CONTENTS

1.	INTRODUCTION.....	3
2.	SOLUTION AND ALGORITHMS	4
2.1.	GENERAL.....	4
2.2.	IMAGE SEGMENTATION	4
2.3.	ROBOT IDENTIFICATION	5
3.	SOFTWARE ARCHITECTURE	9
3.1.	GENERAL.....	9
3.2.	SOFTWARE LIBRARIES USED	9
3.2.1.	<i>MFC</i>	9
3.2.2.	<i>Carnegie Mellon University Vision Library (CMVision)</i>	9
3.2.3.	<i>Additional classes</i>	9
3.3.	MAIN SOFTWARE MODULES	10
3.3.1.	<i>CWatcher Class</i>	10
3.3.2.	<i>CRoboTrackerEngine Class</i>	10
3.3.3.	<i>CRoboTrackerSever Class</i>	10
3.3.4.	<i>CRoboTrackerDoc/CRoboTrackerView Classes</i>	10
3.3.5.	<i>CCamera Class</i>	10
3.4.	MODULE DYNAMIC RELATIONS	11
4.	THE ROBOTRACKER APPLICATION	12
4.1.	GENERAL.....	12
4.2.	LOADING IMAGES AND MOVIE CLIPS	12
4.3.	SAVING IMAGES AND MOVIE CLIPS	13
4.4.	ANALYZING THE INPUT	13
4.5.	SETTING THE MASK	13
4.6.	SETTING THE THRESHOLDS	14
4.7.	MOVIE CLIP PLAYBACK	15
4.8.	CAMERA GRABBING	15
4.9.	CHANGING THE CAMERA SETTINGS	16
4.10.	SAVING AND LOADING THE SETTINGS	16
4.11.	SAMPLING THE PIXEL VALUES	16
5.	THE TCP/IP SERVER.....	17
6.	RESULTS AND CONCLUSIONS.....	18
6.1.	RESULTS	18
6.2.	FUTURE IMPROVEMENTS	18
7.	REFERENCES.....	19

1. INTRODUCTION

The goal of the project is to track multiple robots in real time using a video camera. The robots are small LEGO Mindstorms vehicles that are moving inside an arena. The video camera is positioned on the ceiling above the arena. The video feed from the camera is analyzed in real time and the robot positions are determined.



2. SOLUTION AND ALGORITHMS

2.1. General

In order to locate the robots in the image they are marked with colorful "hats". The "hat" is actually a color coded marker that uniquely identifies the robot. The floor of the arena and other non-robot objects are assumed to be achromatic. Colorful stationary objects can be masked out from the analysis (see below).

2.2. Image segmentation

Image segmentation is done using a color based thresholding in the HSV color space, followed by a connected components analysis.

First, the image is converted from the RGB color space to HSV. This is done using pre-calculated lookup tables to increase performance.

If a mask is specified and a mask pre-processing mode is selected (see below) all the masked-out pixels of the input frame are set to black and will be ignored by the algorithm.

During the thresholding stage the image is divided into robot and non-robot pixels according to their Hue, Saturation and Brightness values. Robot pixels are assumed to be of Red/Blue/Green hues and having high saturation. The default thresholds are set accordingly. Segmenting using the hue and the saturation values allows the algorithm to be more robust to changes in lighting conditions. The choice of only the primary hues for color coding allows high separation between the different color codes, as

these values divide the color spectrum into three distinct regions, with the primary color in the middle of each region. This allows the algorithm to identify each color code with high level of certainty.



The Hue scale

Note - the red hue has two regions, as the hue scale is cyclic with the red hues at the upper and the lower edges of the scale. Thus, two sets of thresholds are needed for identifying the red pixels.

After the image has been thresholded and each pixel classified as Background/Red/Blue/Green pixel, a connected components analysis is performed on the classification map. The result of this analysis is a list of the found connected components (regions), each region having a color code (R/G/B), centroid coordinates and a calculated area.

2.3. Robot identification

As mentioned above, each robot is marked with a colorful marker which uniquely identifies it. The marker surface has any number of co-centric colorful areas (typically circles), with each consecutive surface having a different color (R/G/B). This is in a way a unique color "bar-code" of the robot. The code is calculated by appending the robot colors going from the

inner-most patch outwards. If a numerical representation is needed the color name is substituted with a digit – 1 for red, 2 for green, 3 for blue.



A sample marker "hat" for the R-G-B robot (123)

For example, a robot having a red circle surrounded by a blue ring, which in turn is surrounded by a green ring will have "R-B-G" code (132). Another robot with the same number of rings having the same colors but in a different order can have a code like "B-G-R" (321).

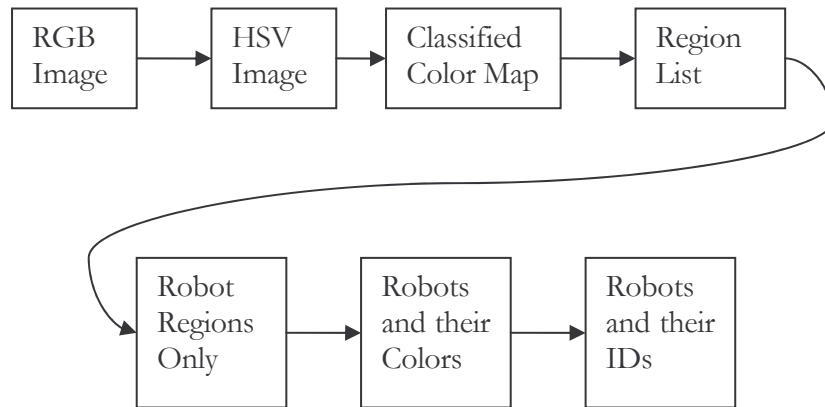
Under the above coding scheme with a maximum number of 3 rings, a total number of 21 possible robots can be coded (3 robots with a single color, 6 robots with two colors and 12 robots with 3 colors). If 4 rings are used, 45 possible robots (codings) are possible.

As with a round circle marker (having one middle circle and any number of surrounding rings) the areas of the consecutive color patches should increase going from the inner-most patch outwards, in order for the algorithm to uniquely identify the robot. This heuristic (area) can easily be substituted with bounding box analysis that can theoretically give more stable results.

The algorithm for identifying the robots given a list of regions in the image:

1. Cut-off:
 - a. Small regions (minimum area threshold)
 - b. Regions having very high ratio between their height and width (ratio thresholds). Robot markers are assumed to have a ratio that is close to 1:1, areas having higher ratios often appear on the image object edges as a result of demosaicing artifacts.
 - c. Regions that fall outside the interest of region mask (if specified). This is done only in mask post-processing mode (see below).
2. Sort and group all the regions by their centers. Two regions having centers that are close enough to each other (robot center distance threshold) will be classified as belonging to the same robot.
3. For each robot – sort its regions by area. The order of the found regions for any specific robot gives its unique color code. The principle used is that the inner-most patch will have the smallest area and so on. This is geometrically correct for circles and other shapes.

The complete analysis process:



3. SOFTWARE ARCHITECTURE

3.1. General

This section describes the software system implemented, its main modules and data flows between them.

The complete source code, documentation of the class and file hierarchies, etc. is available in a browsable HTML form.

3.2. Software Libraries Used

3.2.1. MFC

The system is implemented in C++, using MS Visual Studio 6.0 and MFC.

3.2.2. Carnegie Mellon University Vision Library (CMVision)

Carnegie Mellon University

Author: James R. Bruce

This software package was re-factored by me to support the HSV color model. It is used for segmentation and connected components detection.

3.2.3. Additional classes

CAviFile by P.GopalaKrishna

<http://www.codeproject.com/bitmap/createmovie.asp>

CIniReader by Aisha Ikram

<http://www.codeproject.com/file/ini.asp>

CSmartEdit, CLinkSlider by Rick York

<http://www.thecodeproject.com/editctrl/smartedit.asp>

CMutexRW by Joris Koste

<http://www.codeproject.com/threads/mutexrw.asp>

3.3. Main Software Modules

The main modules of the system are:

3.3.1. CWatcher Class

This class is responsible for image analysis. The theoretical grounds for this class' operation are laid in section 2.

3.3.2. CRoboTrackerEngine Class

This is the main system class. Responsible for interaction with the various input methods (camera, image files, movie clips), the CWatcher image analysis class and several other classes.

3.3.3. CRoboTrackerSever Class

Responsible for sending the current robots positions to any number of clients through TCP/IP sockets.

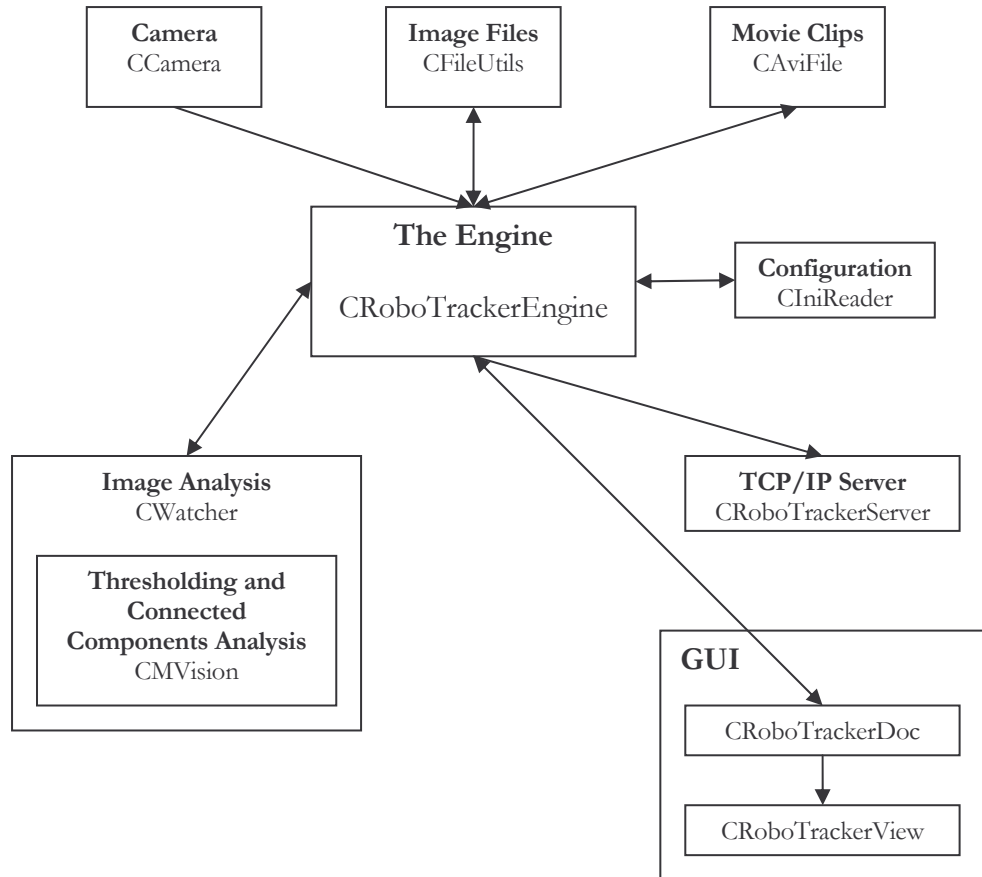
3.3.4. CRoboTrackerDoc/CRoboTrackerView Classes

The system uses the MFC document/view model. These classes are responsible for the UI logic and presentation respectively.

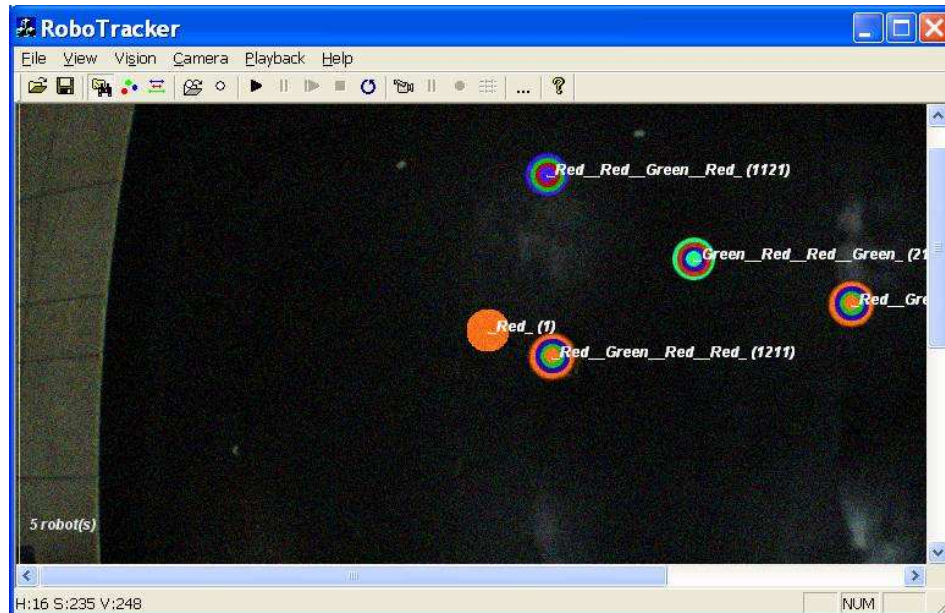
3.3.5. CCamera Class

Provides an interface to the PGR FlyCapture camera driver

3.4. Module Dynamic Relations



4. THE ROBOTRACKER APPLICATION




4.1. General

The RoboTracker application allows obtaining images from image files (BMP, PPM) movie clips (AVI) and getting a live video feed from the camera. The image data obtained is displayed, analyzed and the found robots are shown on the image. The application can also be used to see the classification bitmaps, set masks, record the camera feed, adjust the thresholds and the camera settings and to save and load these settings.


The application also acts as a server, sending the found robot positions to any number of clients using TCP/IP.

4.2. Loading images and movie clips


Use the **File > Open** command or the  icon to open BMP and PPM files or AVI movie clips.


4.3. Saving images and movie clips

Use the **File > Save** command or the  icon to save the current frame to a BMP file.


While grabbing images from the camera, you can use the **Camera > Record** command or the  icon to start recording the video feed to an AVI file. Press this button again to stop recording.

4.4. Analyzing the input

Use the **Vision > Analyze** command or the  icon to start analyzing the current frame(s) and displaying the found robots.

While analyzing, you can use the **Vision > Show classification** command or the  icon to display the classification results of the current frame. This mode is used for debugging and fine tuning, as it considerably slows down the algorithm.


4.5. Setting the mask

Use the **Vision > Mask > Load** command or the  icon to load a mask image. A mask image is a regular BMP image having the same dimensions as the input frame. All the Red (255, 0, 0) pixels in this image will be masked out.


In order to conveniently create a mask, you can save the current frame, open it in Paint and draw over the areas you want to mask out with a red pen or brush. Save this image as BMP and load it in RoboTracker.

After the mask image has been loaded you can change the mask mode using the **Vision > Mask > Preprocess** or the **Vision > Mask >**

Postprocess commands. In the pre-process mode, all the pixels that should be masked out are removed from the image (blackened) and the image analysis algorithm ignores them. In the post-process mode no pre-processing is done, and the robot regions that fall into the masked out areas are discarded. Each of the algorithms has its advantages and disadvantages and can be more effective under different circumstances. With pre-processing, the algorithm must go over the entire frame (on every frame) and blacken the needed pixels. While this slows the algorithm down, the speedup that can be obtained because the algorithm has fewer regions to consider can outweigh this slow down. On the other hand, considering all the regions and disregarding the ones that fall outside the mask can sometimes be considerably faster. I suggest testing the two algorithms under the given circumstances and seeing which one performs better.

After the mask has been loaded, it can be turned off using the **Vision > Mask > Off** command. Also, you can display a superimposed image of the mask using the **Vision > Mask > Show** command or the  icon.






4.6. Setting the thresholds

Use the **Vision > Thresholds > Adjust** command or the  icon to open the thresholds dialog. You can adjust the Hue, Saturation, Value thresholds for the four colors – Low Red, High Red, Green, Blue, as well as the other thresholds – the minimal region area to consider, the maximum height/width ratio for an element and the maximal distance between the region centers in order for the regions to be considered as belonging to the same robot. See section 2 for more details on these settings.


When you adjust the thresholds in the dialog, you see the segmentation and tracking results in real time on the screen. This way you can easily adjust the thresholds to obtain the best possible results for a given environment.


4.7. Movie clip playback

After loading a movie clip you can:


- use the **Playback > Play** command or the  icon to start the playback
- use the **Playback > Pause** command or the  icon to pause the playback.
- use the **Playback > Stop** command or the  icon to stop the playback.
- when paused, use the **Playback > One frame step** command or the  command to step one frame at a time.
- you can turn on and off the playback looping using the **Playback > Loop** command or the  icon.

4.8. Camera grabbing

Use the **Camera > Start grabbing** command or the  icon to connect to the camera and start grabbing frames.

Use the **Camera > Pause grabbing** command or the  icon to pause the grabbing and freeze the current frame

4.9. Changing the camera settings

After the camera is connected you can use the **Camera > Settings > Adjust** command or the  command to open the camera settings dialog. You can see the changes in the image and the analysis while changing the values, without the need for closing the dialog first.

4.10. Saving and loading the settings

Use the **File > Load Settings** or the **File > Save Settings** to load or save all the current settings. This includes thresholds, camera settings (if the camera is connected), the server port number etc.

To save or load only the thresholds, use the **Vision > Thresholds > Save** or the **Vision > Thresholds > Load** command.

To save or load only the camera settings, use the **Camera > Settings > Save** or the **Camera > Settings > Load** command. The camera must be connected first. The settings saved are:

4.11. Sampling the pixel values

While going over the image pixels with your mouse, you can see the current pixel HSV values on the status bar. These values can help correctly set the thresholds.

5. THE TCP/IP SERVER

The system listens for incoming TCP/IP connections on port 6666 (can be changed). Each client that connects to this port sends any number of bytes to the server to start receiving data.

After the initial connection has been established, and the server reads a data packet from the client, it will start sending data packets to the client every time new robot location data is available.

The data is sent as text. This allows easy debugging and diagnostics and allows a wide variety of clients – ones written using scripting languages, compiled languages, Java, .NET, telnet etc. The size overhead comparing to the binary communication is minimal.

The format of every packet sent by the server is as follows:

```
[new line] == "\r\n", the START/END markers are not a part of the packet.

----- START OF DATA PACKET -----
size of data that follows in bytes[new line]
timestamp in milliseconds[new line]
number of robots[new line]
robot1 id[space]x position[space]y position[space]optional data[new line]
robot2 id[space]x position[space]y position[space]optional data[new line]
.
.
robotN id[space]x position[space]y position[space]optional data[new line]
----- END OF DATA PACKET -----
```

6. RESULTS AND CONCLUSIONS

6.1. Results

The system successfully locates robots in the input video feed of large images (1024x768) in real time.

The choice of thresholds and the camera settings (gain, exposure) is important. If these settings are defined correctly, the system is quite robust to changes in lighting conditions and noise.

6.2. Future Improvements

The algorithms can be improved in several aspects. The area based sorting can be substituted with bounding box analysis to obtain better stability. Automatic threshold adjustment can be implemented. The information from the previous frames can be used to search in the vicinity of the previously found robots in order to improve performance.

7. REFERENCES

1. J.D.Foley, A.Van-Dam, S.K.Feiner and J.F.Hughes, Computer Graphics - Principles and Practice
2. B. K. P. Horn, Robot Vision, 1986.
3. A. K. Jain, Fundamentals of Digital Image Processing, 1989
4. James Bruce, Tucker Balch, Manuela Veloso, Fast and Inexpensive Color Image Segmentation for Interactive Robots, School of Computer Science Carnegie Mellon University
5. James Bruce, Realtime Machine Vision Perception and Prediction, 2000
6. A. Sirota, D. Sheinker, O. Yossef, Controlling a Virtual Marionette Using a Web Camera, Project in intelligent systems, Technion, August 2003