Discover Vision: A Framework for Building, Evaluating, and Testing Performance Based Machine Vision Applications

Brian C. Becker
Undergraduate Student
Robotics Laboratory
University of Central Florida
Phone: (407) 882-0293
Email: brian@briancbecker.com
URL: www.robotics.ucf.edu

Daniel Barber
Graduate Student
Robotics Laboratory
University of Central Florida
Phone: (407) 882-0293
Email: dbarber@ist.ucf.edu
URL: www.robotics.ucf.edu

Dr. Fernando Gonzalez
Associate Professor
Department of Engineering and Computer Science
University of Central Florida
Phone: (407) 823-3987
Fax: (407) 823-5835
Email: fgonzale@pegasus.cc.ucf.edu
URL: www.cecs.ucf.edu

Abstract

This paper presents Discover Vision, a framework for the fast creation, evaluation, and testing of machine vision applications used in real time systems such as autonomous vehicles. The framework utilizes user edited scripts describing what image processing and feature extraction techniques to employ. Users can easily and quickly build a vision system by altering these scripts without changing the underlying framework, thus saving time when testing new methods. A graphical user interface is used to display the real time performance in the form of visual displays, processed frame rates, and system accuracy based on validation sets. It is possible to evaluate the effectiveness of a script by loading in live or recorded video for visual or numerical analysis. Scripts developed in Discover Vision can be used within a custom framework through the use of the scripting engine. Extensibility is achieved through a plug-in architecture. This paper describes the Discover Vision framework, demonstrates its application in an autonomous ground vehicle, and analyzes the resulting performance.

Keywords: Machine Vision, Autonomous Vehicles, Unmanned Vehicles, Context-Based Reasoning, Machine Learning, Testing & Simulation

1. Introduction

Machine vision applications typically require a significant amount of trial and error in choosing algorithms and algorithm parameters necessary to extract the best data possible from the environment. In a situation where long testing times are required, such as analyzing the performance of a vision system on a pre-recorded video, any change to the vision system forces a recompile and restart of the testing. Discover Vision simplifies this process of developing machine vision applications by removing and alleviating these tedious problems.

Three components provide the core functionality of Discover Vision: an interactive GUI, a robust engine, and an extensible plug-in architecture. Being a machine vision tool, Discover Vision can accept image inputs in the form of live video from a webcam or camcorder, pre-recorded video stored on disk, or sequence of images in a directory. The code editor part of the interactive GUI allows a user to create a script describing the image processing and machine vision algorithms to execute on the image inputs. Another pane contains visual displays of user-selected intermediate processes in the vision system for real-time examination. Interaction happens through functions defined in the script, which are often used to respond to mouse events or perform initialization routines.

Discover Vision's engine comprises the core of the tool. The engine's main responsibility is to manage everything relating to the scripting portions of Discover Vision. It provides and ties the available machine vision and image processing algorithms to the GUI for use in the code editor. It handles compiling the script into an efficient, compact representation of the processes that form the vision system the user has created. Finally, it handles executing the custom vision system on the image inputs and reacting to events the GUI passes to it. In essence, it is the backend of Discover Vision. To enhance its usefulness, the engine may easily be decoupled and

embedded into any pre-existing system to allow for the execution of the script developed in Discover Vision. This provides a smooth transition from the rapid prototyping available in Discover Vision to a final vision system within any arbitrary framework.

While the GUI and engine give the user a set of pre-existing tools to solve machine vision problems, the plug-ins architecture allow for a user to extend the set of algorithms available in Discover Vision. This functionality is crucial when more complex solutions are required than those provided by the common algorithms built-in to Discover Vision. Whether a new simple algorithm is necessary, or an entirely new framework, the plug-in architecture allows any user to extend the functionality of Discover Vision with minimum work. The architecture is set up so that any new C function or any new C++ class (and member functions) a user creates can be loaded in dynamically and then used within scripts.

This paper discusses the three parts of the Discover Vision framework in greater detail. Section two contains background information on the necessity of building a framework such as Discover Vision. The three primary modules of Discover Vision are detailed in section three. Future enhancements are described in section four and the final section concludes with a summary of Discover Vision.

2. Background

The primary purpose for which Discover Vision was built is to ease the construction of vision systems for the autonomous, mobile robots the Robotics Laboratory at the University of Central Florida was building. Such a vision system would need to identifying obstacles in the environment and markings (signs, lane lines, etc). It was quickly realized that varying lighting conditions, occluded obstacles, a number of other conditions could change how well the designed vision system performed. Thus, a method of being able prototype and quickly analyze

the performance of a particular process was desired. While proprietary solutions such as MatLab and LabView existed, it was important for the solution to be open source and able to integrate well with other open source libraries the team was using. While open source vision system libraries did exist, most consisted of a collection of algorithms designed to simplify the coding part of a vision system. Since no suitable open source tool was available, the Discover Vision framework grew out of the requirements and goals of aiding the creation of vision systems for autonomous, mobile robots.

3. Framework Design

As mentioned earlier, Discover Vision is comprised of three distinct parts: the interactive GUI, the scripting engine, and the plug-in architecture. The scripting engine and the plug-in architecture can both be separated from Discover Vision and used within any third-party C++ program.

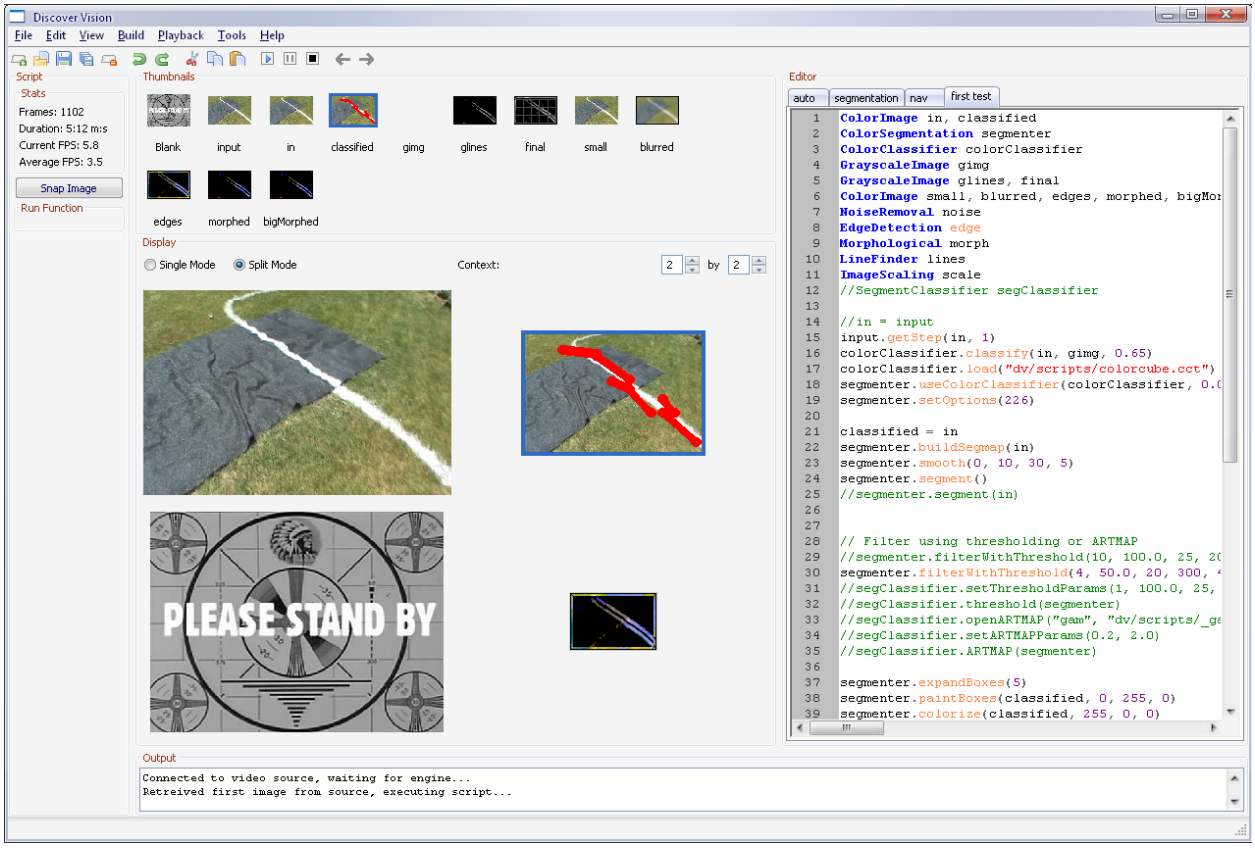
3.1 Platform Design and Dependencies

Discover Vision was implemented in C++ using the popular cross-platform wxWidgets [7] GUI library. Currently, Discover Vision is Windows-only because of better support for hardware on Windows. Discover Vision 2.0 requires Windows XP and DirectX 9 for full functionality; however future versions will support the Linux operating system.

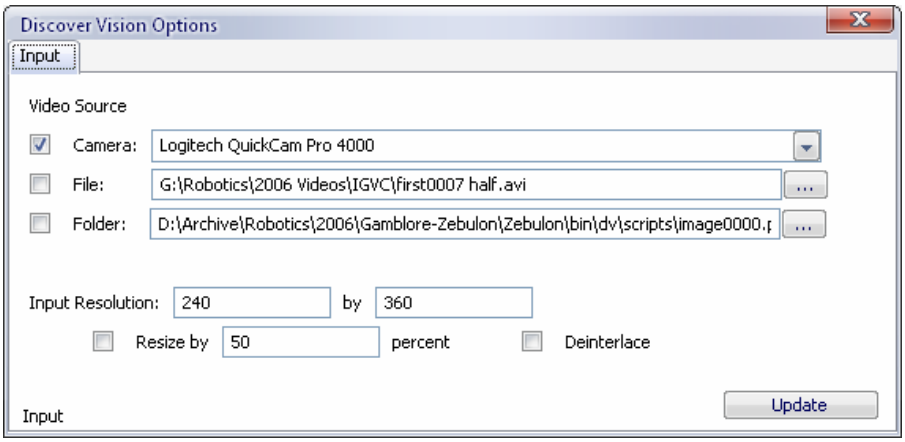
3.2. Interactive GUI

The Discover Vision GUI is laid out into several panes, each with its own functionality. The two major panes are the display and editor panes, in the middle and right sides respectively. The script information pane on the left shows performance specifications on the current script, including frames per second. It also shows the names of user-defined functions in button form. Clicking a button calls the corresponding function in the script. The bottom pane, the output

pane, gives real-time status information during script compilation and execution. Control occurs through the menus, the toolbar, and keyboard shortcuts.



3.2.1. GUI Image Inputs



Discover Vision options allow the user to specify which image inputs to use when testing the script. The user can select live video from a camera (a USB webcam or Firewire camcorder) connected to the computer. Another option is to load in a pre-recorded AVI file (optionally compressed). Finally, individual images (PNG or PPM) can be loaded in from disk. Controls for videos follow the standard start, pause, resume,

and stop. When loading in images, the user can move to the next or previous image in the directory. When dealing with video sources, input resolutions and the option to de-interlace the video can be specified. These image input options allows the user a wide range of abilities to test the performance of the script.

3.2.2. GUI Code Editor

The code editor facilitates the development of vision scripts via a simple programming language. A script has a variety of use, but it primarily describes the image processing or machine vision algorithms to execute on the image inputs. In the code editor, a user can create images to store intermediate output from algorithms or the results from multiple processes. Objects that perform specialized processes may be created as well. Examples include ImageScaling, LineFinder, ColorSegmentation, etc. To aid rapid prototyping, the code editor has syntax highlighting and Intellisense. Syntax highlighting segments different parts of the code into unique colors, so all valid functions are indicated with an orange color. This makes typos easier to locate and correct when developing the script. Intellisense presents a list of functions available after typing the "." after any object, making selecting the correct function to call a snap. The code editor also features multiple tabs for multiple scripts. While multiple scripts cannot be executed simultaneously, editing multiple scripts is beneficial in cases where a user wants to compare the performance and accuracy of more than one approach. Scripts and tabs in the editor are remembered and automatically reloaded when the program starts.

3.2.3. GUI Image Display

The image display forms a core feature of Discover Vision. When developing vision systems, visualization of the output and even intermediate steps is critical. Because of this, Discover Vision has robust means for visualizing the results of the script. The image display has

two components. First, a series of thumbnails are presented at the top of the display. Each thumbnail represents an image declared in the script. Furthermore, the thumbnail is labeled with the name the user specified in the script. During script execution, the thumbnails are updated in real-time and serve as a selection guide. The second component of the display is the larger display grid. Dynamically configurable from 1x1 images to 5x5 images, each cell in the grid can display a thumbnail in greater detail. The images are scaled to the available size, keeping the aspect ratio so the outputs of multiple processes can be viewed simultaneously. If only a single output needs to be viewed, a group box option will toggle the mode between “Single” and “Split.” By default, an empty cell displays the old TV “Please Stand By” Indian Head image.

3.3. Scripting Engine

The scripting engine manages the compilation and execution of the script. When a new frame arrives from the GUI via a video or new image file, the script is executed on the image.

3.3.1. Scripting Language

The scripting language most closely resembles a mixture of C++ and BASIC. Statements are line terminated instead of semi-colon terminated for ease of use. Variables are declared in a similar manner to C, but variable types may only be classes loaded by the plug-in architecture. DiscoverVision supports a wide range of built-in classes, including:

BackgroundSubtraction	ColorClassifier	ColorImage	ColorSegmentation
EdgeDetection	HSVColor	GrayscaleImage	Histogram
ImageEnhancement	ImagePainting	ImageScaling	LineFinder
MiscFilters	Morphological	NoiseRemoval	PyramidImage
YIQColor			

Each class has any number of functions that support the functionality of the class. For instance, EdgeDetection would have functions named “canny” and “sobel.” Additional classes can be added to the scripting language through the plug-in architecture discussed in section 3.4. Garbage management is performed simplistically: variables are created at the beginning of the script execution and automatically destroyed at when the script terminates. Since some variables require memory or time intensive initialization routines (such as calculating lookup tables), the keyword “preserve” maybe be added before the object type to indicate this variable should be preserved between compiles if possible, thus avoiding the lengthy initialization routines.

The majority of a script usually consists of algorithms to run. These algorithms are the class functions of the classes listed above. Organizing the image processing and machine vision algorithms into classes is beneficial because it makes finding a particular type of function easier. Furthermore, temporary memory used by an algorithm can be cached in the class, yielding faster scripts. All classes available from the scripting language are loaded from normal C++ classes built into DLLs. While it is impossible to declare or modify the data types common to C++, they can be passed as function parameters. Characters, shorts, integers, doubles, strings, and any object of a class type loaded by the plug-in architecture can be passed.

In addition to manipulating objects and images, custom functions, events, and pipelines may be defined. Pipelines consist of a group of algorithms being executed simultaneously in different threads. As multi-core processors become more popular, it is expected that significant speed increases could be seen by running several algorithms in parallel. Any number of algorithms can be run in parallel and any number of pipelines may be defined. Functions and events are slightly different in that they are not executed with the rest of the script. Instead, a

function or event is a set of special algorithms that run on some trigger. This allows the user to interact with the script. A number of events are available so the script can respond to mouse clicks or moves. These events may be used for training or identifying certain image features. Functions behave exactly except the trigger is the user. Functions can be called by clicking on the correspondingly named button in the script pane. Typically, functions handle special cases, such as saving or loading data (a Neural Network for example).

3.3.2. GUI Interaction

The engine is initially invoked when a user issues the command to compile the script via the menu or keyboard shortcut. When this occurs, Discover Vision passes the script contained in the current code editor to the engine for compilation. If errors are found within the script, the engine returns them to the GUI for display and correction. Otherwise, the engine readies the script for execution. If image inputs are available, the script is applied to the inputs; otherwise, the script is held until an image input becomes available. Using this design, a recompilation can occur at anytime – even during the playback of live video. Once compilation is finished, the new script replaces the old without the need to pause live video. This greatly facilitates tweaking algorithm parameters and immediately seeing the effects of the new value.

3.3.3. Engine Internals

When receiving the script, the engine breaks the script components into individual tokens. Once tokenized, the compiler matches declared variables and function calls to the classes available in the plug-in architecture. Compilation converts and compacts the text script to a byte code for faster execution by the interpreter. The interpreter creates a separate thread for the script to execute within and receives new image frames from the GUI. The engine uses a callback function to receive and pass back images.

3.3.4. Decoupling the Engine

The scripting engine can be easily decoupled from the GUI and used to execute the vision system developed through Discover Vision. The engine has a public interface for loading in scripts or script bytecode, executing the script, receiving image inputs, and returning variables within the script after each frame. With this feature, a user can prototype the vision system in Discover Vision and then use the exact same script within the application framework making use of their vision system.

3.4. Plug-in Architecture

While Discover Vision comes with a decent set of image processing and machine vision algorithms, many complex vision systems will require custom algorithms. Since Discover Vision is designed to accelerate the development of the entire vision system, an important aspect is its extensibility. The ability to easily add new algorithms and classes to Discover Vision is handled through the plug-in architecture. By encapsulating new C++ classes and functions within a Windows Dynamically Shared Library (DLL), Discover Vision can read most any custom C++ class for use within a script.

3.4.1. Dynamically Loading C++ DLLs

When creating a normal C++ DLL, the C++ function names are mangled by encoding the function's parameter types in the function name (this is also sometimes referred to as name decoration). While this makes linking to the DLL more difficult, it supplies all the information necessary to call the function. Conversely, a C DLL would provide only the name of the function, leaving the user no clue of the parameters to pass to the function without the aid of a header file. Using some of the Windows debugging libraries, it is possible to enumerate all the C++ functions within a DLL and undo the process of mangling the names and construct a

function declaration similar to that seen in a C++ header file. Parsing the list of functions in the DLL makes it possible to reconstruct a database of classes, functions, and parameters available in the DLL. Multiple classes may reside within a DLL or within multiple DLLs and classes dependencies may span DLL files. Thus, when Discover Vision first loads, all DLLs located in the “plugins” sub-directory are loaded and all the available classes and functions are reconstructed and made available to the scripting engine.

4. Application in Autonomous Robot

For several years the Robotics Laboratory at the University of Central Florida has participated in the Intelligent Ground Vehicle Competition (IGVC) [2] sponsored by the Association for Unmanned Vehicle Systems International (AUVSI). At this event, autonomous ground robots are required to identify road lane-lines and obstacles over different terrain using machine vision. Using Discover Vision [1], the UCF team was able to quickly update and modify the robots’ vision system quickly without modification to the underlying system. This ability was a major benefit during an outdoor event where conditions can change at any moment with little time for adjustments before competing on different courses.

5. Future Work

While robust in its current state, several aspects of the Discover Vision framework could be improved. Future work could include integrating commonly used machine vision libraries, increasing script efficiency with Just-In-Time compilation, and adding the ability to dynamically switch contexts to adjust for changing environmental conditions.

5.1. Additional Plug-In Libraries

Currently, Discover Vision has support for a number of the most common vision algorithms. However, integration with some of the commonly used image and vision C++

libraries could greatly increase the functionality of Discover Vision. For instance, the addition of OpenCV [3] or ImageMagick [4] to Discover Vision would greatly increase the capability of Discover Vision. In addition, users already comfortable using these frameworks would benefit from a reduced learning curve while enjoying the benefits of Discover Vision.

5.2. Just-In-Time Compilation

Because the Discover Vision scripting engine compiles the script to byte-code and uses an interpreter to execute it, there is a loss in efficiency when running scripts. A Just-In-Time (JIT) compiler that converts the script to machine code using a library such as GNU Lightning [5] could significantly increase the speed of the executed script.

5.3. Context Switching

When using Discover Vision with an autonomous, mobile robot, one difficulty in building a comprehensive vision system is the mobility of the robot. The unmanned robot can move in and out of different environments that should ideally be handled by different scripts. Even stationary vision systems can encounter this problem if lighting conditions change. Instead of building a vision system with a set of static algorithms that attempt to deal with changing environmental conditions, a better approach might be to design scripts for each anticipated environment [6]. Thus, the script being run can be switched dynamically in response to the environment so the most optimal script is being run at all times. This would require an overseeing script to analyze the current environment and switch to the vision script that best handles current context. Implemented this way, context switching could provide a means of responding on-the-fly to changes in the environment.

6. Conclusion

Designed to aid in developing vision systems for autonomous, mobile robots, Discover Vision gives system designers a robust framework to design, prototype, combine, and test image processing and machine vision algorithms. The Discover Vision framework's central components are the GUI, engine, and plug-ins. With its interactive GUI, creating scripts of machine vision algorithms and visualizing the performance and accuracy on live video streams is easy. The scripting engine is the workhorse of Discover Vision, powering the execution of the scripts and enabling Discover Vision scripts to be integrated and run in third party software. Through the plug-in architecture, any custom C++ class may be added to the scripting language, expanding the functionality of Discover Vision. These features make Discover Vision an ideal framework to use in the development of vision systems for autonomous, mobile robots.

7. References

[1] Barber, D.J., Gonzalez, F.G., Roberts, T., Becker, B.C., Software Design for an Autonomous Ground Vehicle for the 13th Annual Intelligent Ground Vehicle Competition. In proceedings of *Intelligent Robots and Computer Vision XXIII: Algorithms, Techniques, and Active Vision*, SPIE, Vol. 6006 p. 202-209 October, 2005.

[2] http://www.igvc.org – Intelligent Ground Vehicle Competition, Rules and Regulations.

[3] http://www.intel.com/technology/computing/opencv/index.htm - Intel OpenCV.

[4] http://www.imagemagick.org/script/index.php - ImageMagick software.

[5] http://www.gnu.org/software/lightning/index.html - GNU Lightning

[6] Perner P., Why Case-Based Reasoning Is Attractive for Image Interpretation. Proceedings of the 4th International Conference on Case-Based Reasoning, pp. 27-43, 2001.

[7] http://www.wxwidgets.org – wxWidgets Library