Control Surface

Project Proposal

By

Caleb Ott

# Executive Summary

Control Surface is software that performs tasks by using hand written commands. It utilizes webcam technology to view a whiteboard or piece of paper and recognize commands immediately while someone sketches those commands on the board or paper. The user can add plugins to allow the program to perform any number of tasks. The project consumes the JavaCV API to accomplish its goal. I will use the NEO waterfall methodology to develop Control Surface and will build and test the different components of the project over a period of ten weeks.

# Project Description

Programs with gesture-based controls are getting more popular. Microsoft’s Surface lets people control a computer by using a fluid touch screen. Likewise, Microsoft’s Kinect lets people play games by using their body as the controller. Using controls unlike the traditional mouse and keyboard is what futuristic programs are moving towards and the philosophy that Control Surface employs. Control Surface uses symbols written on a whiteboard (or other surface) to perform specific actions. The user can change the configuration of the program, which allows the system to recognize different symbols and actions. With these different configurations, a user can use Control Surface to perform any number of tasks from solving a problem to playing a game.

# Project Requirements

The project’s requirements are a list of objectives the program must accomplish. The bulleted list below separates each requirement into three separate pieces – functions, attributes, and constraints. The first part, the function (the main bulleted point), specifies that the program will be performing that function. The second part, the attribute (the sub points to the main bulleted points), gives parameters to its function and gives it a dimension. The third part, the constraint (which is part of the attribute), gives details on how exactly that function can perform and constrains it. “The system should” prefaces each function.

The system should…

* Recognize symbols
  + Types of symbols consists of monochrome images
  + System works better with simple symbols (preference)
* Activate commands associated with the symbol
  + Commands are programmable actions
  + 1 to 5 commands can be associated with a symbol
* Change program’s configuration
  + The configuration defines the set of symbols, the set of commands, and the associations between those symbols and commands
  + A symbol must have at least 1 command associated with it
  + The program configuration may only be changed while the program is running in configuration mode
* Allow users to manage symbols within a configuration
  + User can add symbols to the current configuration’s set of symbols
  + User can remove symbols from the current configuration’s set of symbols
  + User can associate or dissociate commands with symbols
  + Symbols may only be managed while the program is running in configuration mode

# Project Specification

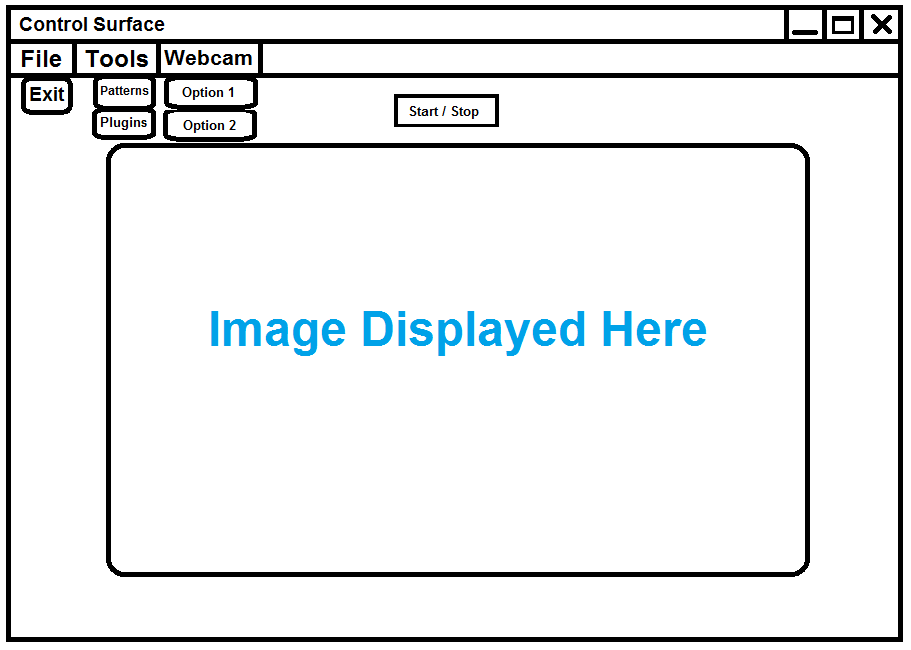
This program utilizes five separate interfaces to operate. It uses a graphical interface, a standard webcam, and a plugin API. The list below contains the interfaces and goes into details about each one.

## Graphical Interface - Windows

### Main Window

The bulleted list below identifies the pieces of this window and explains how the user will use it.

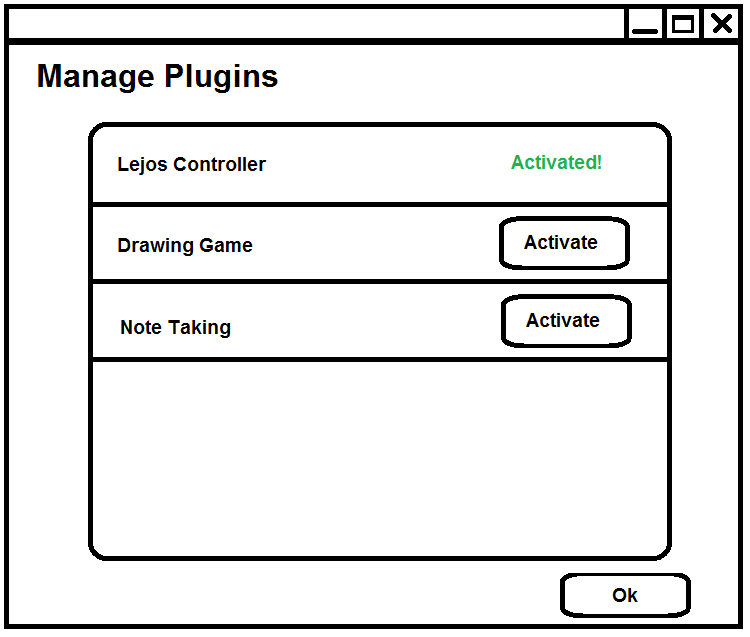
* Squares on top left are drop down menus
  + The file menu gives the user an option to exit the program
  + The Tools menu gives the user an option to navigate to other windows
    - Selecting “Patterns” opens the “Custom Patterns” window
    - Selecting “Plugins” opens the “Manage Plugins” window
  + The Webcam menu lists all the available webcams and the user can select which one he/she wants to use
* The user can use the start/stop button in the middle to tell the program to start or stop the pattern recognizing
  + The button says “Stop” when the program is started
  + The button says “Start” when the program is stopped
* The rounded square in the middle displays to the user the active webcam image with recognized patterns highlighted



### Manage Plugins

The bulleted list below identifies the pieces of this window and explains how the user will use it.

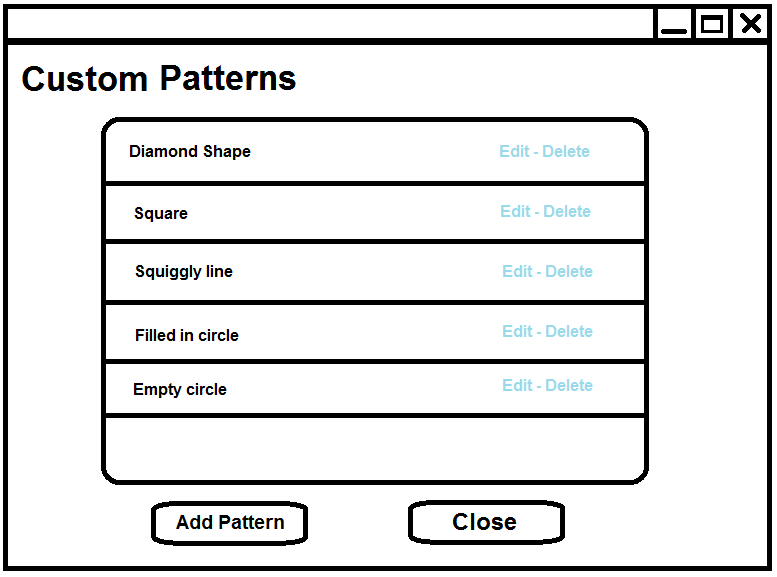
* The rounded rectangle in the middle shows the list of installed plugins
  + The plugin with the text “Activated!” is the currently activated plugin
  + When the user presses a plugin’s “Activate” button, that plugin is switched to “Activated!”
  + The program can only have one plugin active at any time
* The user installs a plugin by copying it into the ‘Plugins’ folder prior to opening this window



### Custom Patterns

The bulleted list below identifies the pieces of this window and explains how the user will use it.

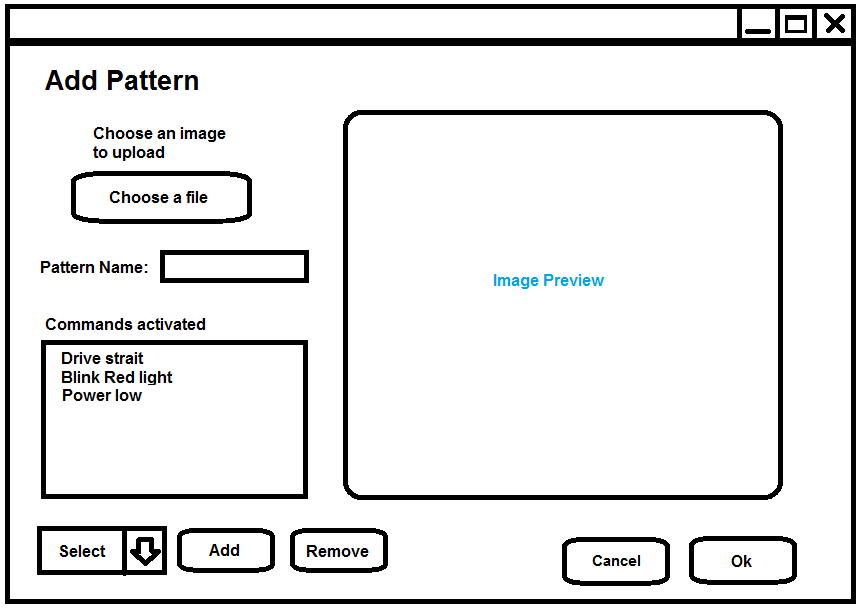
* Displays a list of custom patterns installed
  + The name is displayed to the user on the left
  + The user can edit or delete a custom pattern by selecting the buttons on the right
    - Editing a pattern opens the “Edit Pattern” window
  + Editing a pattern opens the “Add Pattern” window with the information already entered
* Can add a new pattern at the bottom which opens the “Add Pattern” window



### Add Pattern

The bulleted list below identifies the pieces of this window and explains how the user will use it.

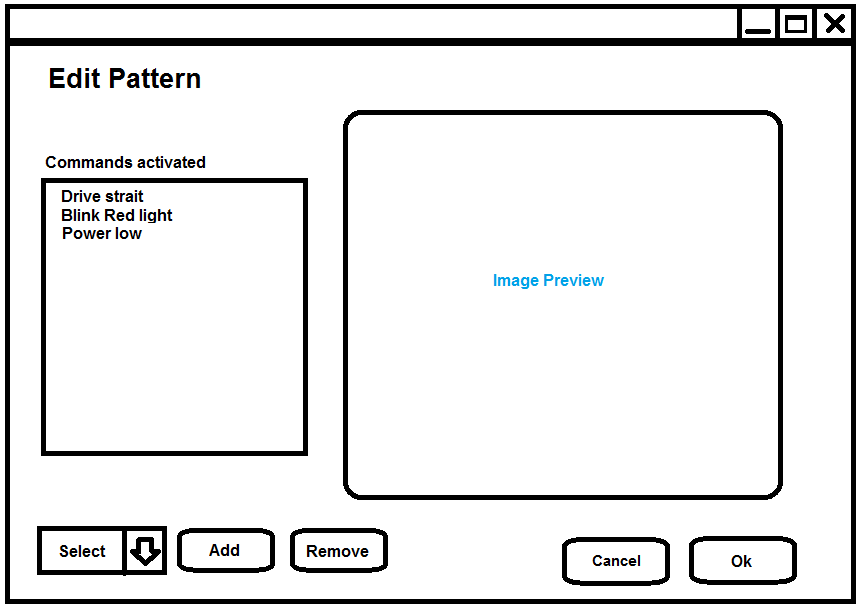
* On the top left, the user can upload a picture and preview it on the right
  + The window displays the picture to the user in high contrast
* On the middle left, the user can enter a name for this particular pattern
* On the bottom left, the user can select commands. The program will activate those commands when this pattern is recognized
  + There is a drop down list of commands the user can choose from



### Edit Pattern

The bulleted list below identifies the pieces of this window and explains how the user will use it.

* The user can preview the image for the pattern on the right
  + The window displays the picture to the user in high contrast
* On the left, the user can select commands. The program will activate those commands when this pattern is recognized
  + There is a drop down list of commands the user can choose from



## Webcam

The program needs an operational webcam installed. The “Main Window” will display all the operational webcams in the options list. The program uses the webcam to view the whiteboard, so the user must place the webcam viewing the whiteboard in an area with ample light.

## Plugin API

For the program to install plugins, the plugins need to follow a consistent standard. A class in the plugin must implement the “IPlugin” interface. When the user activates the plugin in the “Manage Plugins” window, the program will use that plugin for all the model functions. The “IPlugin” interface is defined as follows:

**void** **activate**(ott.controlSurface.Command command);

Iterator<Command> **getCommands**();

Iterator<Pattern> **getPatterns**();

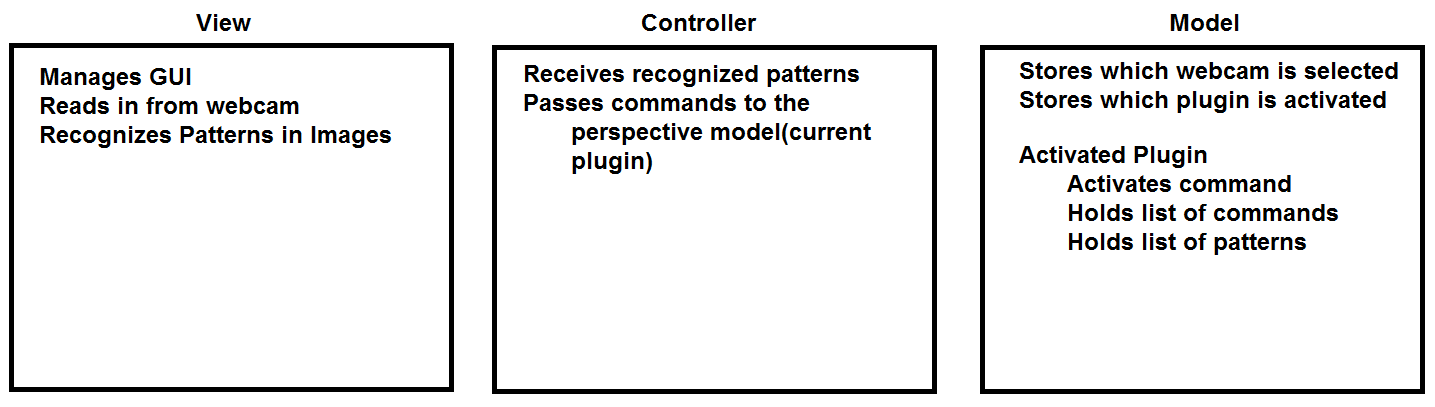
**void** **addPattern**(ott.controlSurface.Pattern pattern);

**void** **removePattern**(ott.controlSurface.Pattern pattern);

The plugin is responsible for persisting or not persisting custom Patterns.

# Project Architecture

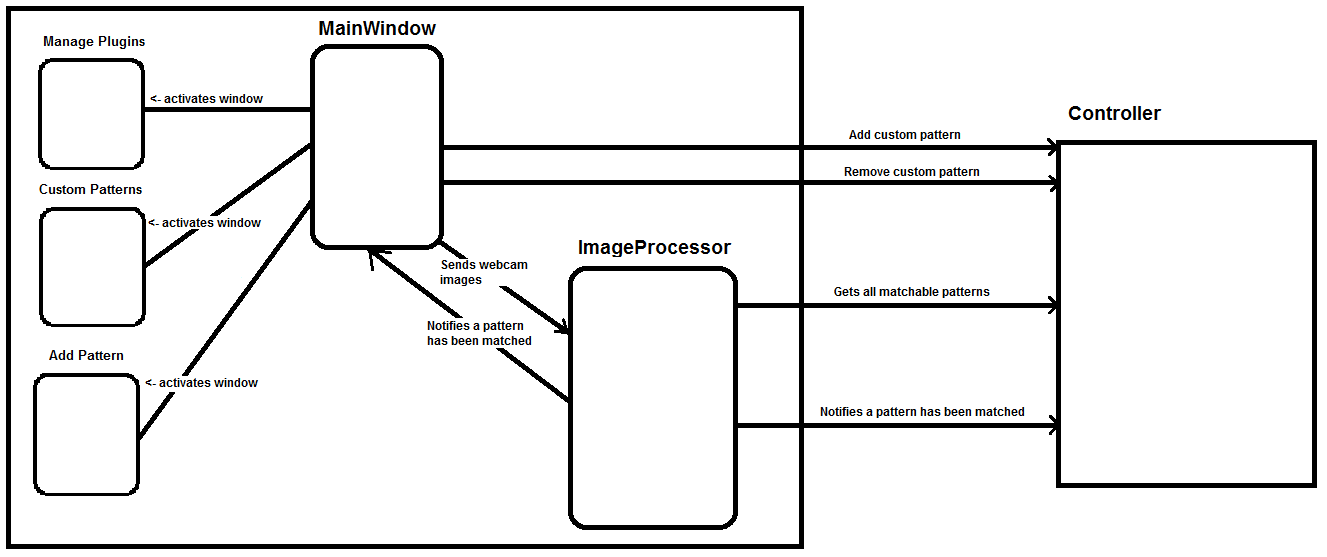
The design below defines the MVC architecture of the system and all the sub components. The first diagram shows the basic layout of the entire program and functions of each component. The next three diagrams show more details about each component as well as their sub-components’ attributes.



## View

The view manages how the user interacts with the program by the windows he/she interacts with and the webcam input.

* MainWindow
  + Sends images to the image processor
  + Adds and removes patterns from the controller
  + Activates other windows
* ManagePlugins
  + Window for managing plugins
* CustomPatterns
  + Window for managing patterns
* AddPattern
  + Window for managing patterns
* ImageProcessor
  + Gets all match able patterns from the controller
  + Processes images for matched patterns
  + Sends matched patterns to the controller
  + Sends matched patterns to the MainWindow



#### ImageProcessor

This section lists the design, pseudo code for the ImageProcessor class.

searchForPatterns()

for each pattern in the match able patterns…

search current image for pattern

if pattern is found,

notify Main Window pattern is found

notify Controller pattern is found

searchCurrentImageForPattern(Pattern)

find borders of image

scan image for pattern match

return if pattern is matched

findBordersOfImage()

// use Canny algorithm built into OpenCV to find edges in image

// <http://en.wikipedia.org/wiki/Canny_edge_detector>

scanImageForMatch(Pattern)

// will manually build one of the following algorithms to perform this task

// or a slight variation of the algorithm to account for variances in drawing

// Template matching - <http://en.wikipedia.org/wiki/Template_matching>

// Scale-invariant - <http://en.wikipedia.org/wiki/Scale-invariant_feature_transform>

// Or some homemade algorithm to match the pattern

## Controller

The plugin for this system can be dynamically changed during runtime, so the controller is responsible for changing the plugin and sending commands the appropriate plugin.

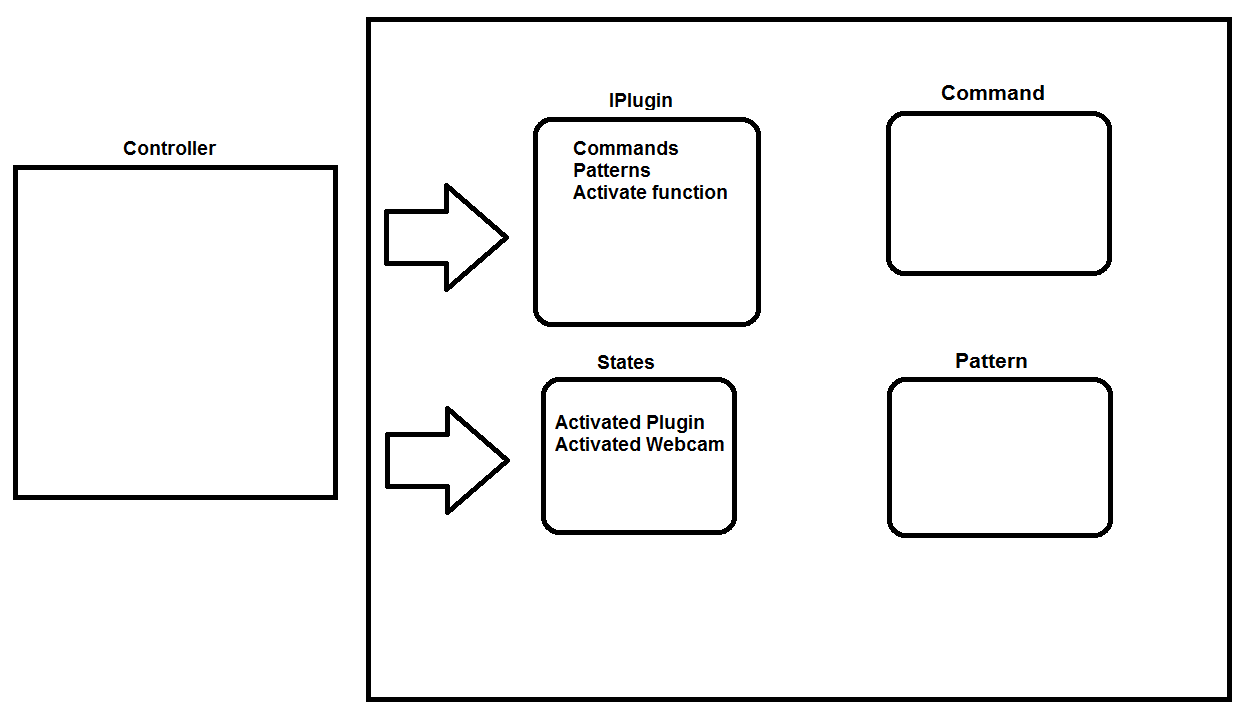
* ModelManager
  + Loads activated plugin from the file system
  + Passes the following functions to the correct plugin model
    - Add custom pattern
    - Remove custom pattern
    - Get custom patterns
    - Activate command
  + Stores activated plugin in the model
  + Stores activated webcam in the model

## C:\Users\cott\Documents\Past Homework\Quarter 6\Capstone\Architecture\ControllerLayout.png

## Model

Part of the model of this project changes dynamically based on the installed plugin. All future plugins will follow the ‘IPlugin’ interface. Along with the plugin, the model holds a ‘States’ class that stores the activated Plugin as well as the activated webcam.

* IPlugin
  + Interface for model classes
* Command
  + System uses this class to abstract out the idea of a command or function
* Pattern
  + System uses this class to abstract out the idea of a pattern that the system can use to match against an image
* States
  + Stores which plugin is currently active
  + Stores which webcam is currently active



# Technological Description

This section lists the technologies for the project. The main point explains the technology and what it is, whereas the sub point specifies how it will be used in the project.

* Eclipse Juno -  Software Development Environment
  + Main IDE for development
* Java 1.7 - Software Language
  + The project uses this language for coding the program
* OpenCV - Computer Vision API
  + The project uses this API for image processing
* JavaCV - Java specific Computer Vision API
  + A java specific version of OpenCV that gives an interface to OpenCV as well as webcam support
* JUnit – Java unit testing framework
  + The project uses JUnit to test the components
* TortoiseSVN – Version Control
  + The project uses TortoiseSVN to keep the versions of the program

# Novelty

I have always had an interest in images and computing them, so because this project has to process images to find patterns, it gave me a way to dive head first into this technology. With the popularity of video streaming and webcam usage, I want to be experienced in those technologies for the future, even if I am not required to use it in my job.

As a student, I am trapped in the scope of small, simple projects. They have no larger dependency and are usually contained to a small scope like a single language or API. This project lets me expand my project to something more than a minute assignment. In this project, I am using Java, but also have an API as well. This API will give me practice working with projects at a larger scope. I will also be making my own API for my program so plugins can be installed. I am excited for all the knowledge I am going to gain by working on this project, from the image processing to the API usage.

# Development Process

The developer will develop the project over a period of 10 weeks. The numbered list below shows the exact dates (milestones) that the developer will have completed each component of the project. Currently some dates are set as “TBD” (To Be Determined). Since this is a research project, the developer will have to plan the milestones in previous milestones. In addition, some milestones have “Tentatively:” to propose an item to have completed by that date; however, this is subject to change based off previous milestones.

At the end of this project, I am hoping to be able to recognize symbols that can be dynamically changed. I will start the process by researching algorithms for line segments, move to symbols with multiple line segments, and finally move to recognizing symbols using arced line segments as well as straight line segments.

1. Jan 11, 2013
   1. Research 2 algorithms for line/pattern detection
      1. Have prototype to display each algorithm and how it works
      2. Have understanding of how each algorithm works
2. Jan 18, 2013
   1. Be able to recognize a vertical and horizontal line on a whiteboard
      1. Use the better of the two algorithms from week 1
      2. Have a prototype to display the program working
      3. For all future points - being able to recognize a line is defined as:
         1. Furthest left coordinate
         2. Furthest right coordinate
         3. Line length
         4. Line width
   2. Have a plan for the development process for weeks 4, 5, and 6
3. Jan 25, 2013
   1. Be able to recognize lines in all directions
      1. Expand algorithm
4. Feb 1, 2013
   1. TBD
   2. Tentatively:
      1. Be able to recognize shape based off lines
5. Feb 8, 2013
   1. Have a plan for the development process for weeks 7, 8, 9, and 10
   2. TBD
6. Feb 15, 2013
   1. TBD
7. Feb 22, 2013
   1. TBD
8. Mar 1, 2013
   1. TBD
9. Mar 8, 2013
   1. TBD
10. Mar 15, 2013
    1. TBD

# Resources

The developer will use Eclipse Juno along with Java 1.7 to code the program. In addition, the developer will use the API’s OpenCV and JavaCV, which provide image-processing support. To keep track of program versions, the developer will use TortoiseSVN. The developer will test on Aaron Friedman’s whiteboard and has Bluetooth built into his laptop.

The user needs to supply a whiteboard or piece of paper to draw the commands on as well as a webcam device to view the board. The computer running the Control Surface needs to have OpenCV and Java 1.7 or later installed to operate.