SENG 499: Final Project Report

Group 499/14

Justin Sketchley - V00185454

Brandon Jacklyn - V00732611

(Richard) Morgan McKenzie - V00685820

Faculty Advisor - Alexandra Albu

July 31 2014

Table of Contents

---

1 Introduction

Our team, Motion Inc, aims to make any USB connected webcam capable of performing hand gesture recognition with an emphasis on controlling vehicle infotainment systems. Modern vehicles now have screens built into the dashboard capable of providing route-planning via maps integration as well as entertainment such as playing music and videos. Our product augments this experience with hand gestures that can control these functionalities. Some examples of using hand gestures to control an infotainment system include using sideways swiping motions to switch between songs, raising and lowering your hand to zoom in/out, or making panning motions to move the viewpoint on a map. Although controlling a real infotainment system is not part of our project, we have created an API that can emit events (when gestures are recognized) to all registered event listeners.

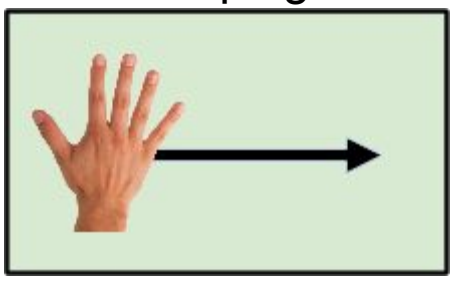
1.1 Project Motivation

One of our team members, Morgan, is a volunteer on the UVic EcoCAR team and he wanted to compliment the dashboard touch-screen technology with hand gestures integration in the infotainment experience. This was the primary motivation behind the choice of location of the webcam (behind the center mirror looking down on the dashboard/hand) and the orientation of the hand gestures that can be recognized. Furthermore, all team members wanted to learn about image processing and gesture recognition which made this project a perfect opportunity to build something fun.

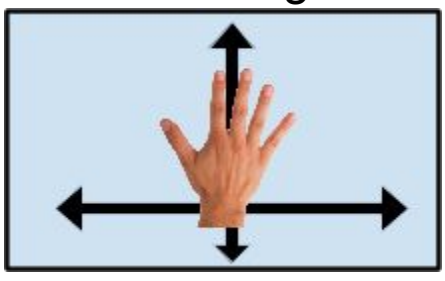
1.2 Objectives and Requirements

At a high level, our objective/requirement for a successful project is to create an API that can be used by other programs to instantly be able to recognize hand gestures. Furthermore, the code must be easily ported to any major desktop OS as well as android and embedded linux. The API should provide the ability to recognize 4 basic gestures: swiping, panning, zooming, and rotating.

Swiping

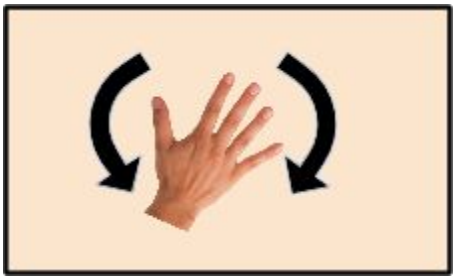


Panning



Rotating

Zooming



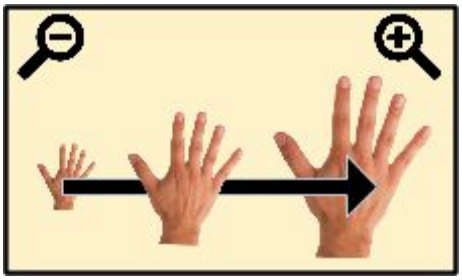


Figure 1: Gestures

This will be accomplished by enabling other processes/threads to listen for events generated by the gesture detector and notify them when one of the above gestures has been recognized. This means the program must be highly concurrent with separate threads for frame collection, data processing, UI interaction, and event notification. Furthermore, it should expose hand position and orientation information as part of the API if requested.

2 Technologies

The technologies used in this project conformed to the overall objective of the project - to create a system which can run on multiple OS's and platforms, with cheap hardware components. Rather than using a product packaged for computer vision such as the Microsoft Kinect or LeapMotion, which have constraints on usage and are costly, hardware was used which is readily available to most people.

2.1 Webcam

The gesture recognition project was designed to work with a regular USB webcam. This allows for both the hardware and the software to be portable to multiple platforms, with minimal expense. The webcam used for the project development and demonstration was the Microsoft HD-3000 Lifecam which is ~$30, but cheaper cameras or the webcam included in the hardware of most laptops/cell phones could be used. While the HD-3000 camera is capable of capturing 720p HD, the development was done at 320 by 240 pixels as this allowed for faster frame capture. This allows for the gesture detection to be ported to lower resolution cameras and slower hardware which may not be able to process 720p video, without seeing a difference in performance of the program.

2.2 Qt Library

Qt is an open source application and user interface framework which can operate on multiple platforms including Windows, Linux, Mac, Embedded Linux, and Android. Qt is developed in C++, and includes a design language called QML which is a Javascript-based markup language that allows for easy creation of graphical user interfaces which are cross-platform to all major OSs. Qt also provides a thread-safe signalling system that allows for easy communication between threads. It is through this mechanism that communication between the gesture recognition and user can occur through an API. The inclusion of Qt in the project allows for the creation of a cross-platform API, and a user interface for project demonstration.

In the project, the Qt Library was initially used to capture the frames from the webcam, as well as handle the application threads and user interface. It was discovered that despite the cross-platform advertisement of Qt it does have slightly different capabilities on different platforms due to the usage of different backends for various components. Unfortunately the current support for webcams on Windows platforms is limited, and since two of the three developers primarily use linux this was not acceptable. To enable Windows as a development platform, the process of capturing video from a webcam done with OpenCV library instead, which was initially going to be used only for image filtering.

2.3 OpenCV Graphics Library

OpenCV (Open Source Computer Vision), as the title suggests, is an open source library for computer vision applications, with an emphasis on real-time applications. OpenCV supports interfaces through C, C++, Python, Java and MATLAB, though it operates natively in C++. The library is officially supported on Windows, Linux, and Mac x86/x64 operating systems, as well as android and iOS on ARM. It can also be compiled for most embedded linux systems. Usage of this library allows for development on a PC running Windows or Linux, and then porting the application to an embedded system running Linux or Android.

The OpenCV library is used to capture frames from the video camera and on supported platforms setting up the resolution, brightness, exposure, and other related properties of the camera. The library also has tools to aid in the filtering of those frames after they have been received by the camera. OpenCV provides low-level operations for use in image manipulation and pattern detection, which were combined and used to perform the detection of the hand.

2.4 i.mx6q Sabre-Lite Development Board

The initial goal of the project was to run on the i.mx6q Sabre Auto board in the Ecocar. Due to the infeasibility of doing all development in the actual vehicle, another similar board called the i.mx6q Sabre Lite was used. Extensive work was done to set up the environment for running the gesture detection software on this board; however many difficulties ensued resulting in the board not working as planned. See the implementation section for a more thorough description.

2.5 Google/LG Nexus 4 Cell Phone

Showcasing the cross-platform and performant aspect of the program was important to the team so when the development board became infeasible different options for running on a relatively low-powered device were investigated. Android was determined to be a platform which could fairly easily support the continued development of the project and was therefore used to explore the result of running the program on a smaller processor.

3 Comparison of Alternative Implementations

During the design process various choices were made for the development platform. The main decision revolved around usage of the Qt toolkit as it provided a lot of value in terms of coding simplicity but requires a build or compiling from source for any potential target hardware.

3.1 Qt User Interface vs OpenGL

In order to create a simple user interface without Qt, the team would have used GLUT (OpenGL Utility Toolkit) as it supports the cross-platform nature of the program. GLUT provides functions for creating a window and very performant drawing of objects in that window on various platforms. It is less portable than Qt so multiple versions of code may have been needed for platforms with full OpenGL vs platforms supporting only OpenGL ES. Furthermore, with this library the code required to implement a simple button in the application window would be much larger than with Qt. Qt's QML modeling language allows for the easy creation of user interface elements using javascript handlers and json data objects. In the end, it was decided to use Qt as development of the actual interface was not at all a focus of the project but rather just for demonstration purposes, so ease of development was preferable to performance.

3.2 Qt Application vs Vanilla C++ Application

The main benefit to using Qt to handle the application code is the signal and slot functionality. The signal functionality is designed to replace and improve on situations where a callback would be used, such as in the notification that a certain gesture has been recognized. A callback is when a pointer to a function is passed to another function, essentially specifying the first function as a variable. A signal is a variable that a calling function will *emit* in order to *signal* the listening *slots* that an event has occurred, or that data is available. In this paradigm, multiple slots can be connected to the same signal. Slots can be within one object, or spread out amongst multiple objects. This abstraction enables notification to the calling program of events without having the program poll for results or rely on interrupts.

The biggest advantage of the signal/slot paradigm is the multiple thread application implementation, and the simplicity of the API connections. In the gesture recognition project, the video processing, filtering, parsing, and recognition is all done by a one thread. A second thread handles the graphical user interface and the API connections. The results of the gesture recognition done on video frames needs to be passed from the video processing thread to the interface/API thread. The signal/slot paradigm allows for a thread safe way of passing this information when it is available. Creating a second set of signals and slots that connects our gesture recognition application to the user-facing API allows for secure and simple method of alerting other applications of the gesture output through the API.

Lastly, the resulting API since written with Qt can be easily exported as a plugin which allows usage from any QML program. This fits well with the end goal of the project, which is to create a simple interface which can be used by different applications to do gesture control with minimal work involved.

4 Implementation of Prototype

The implementation of the prototype involved the creation of a user interface which allowed for the configuration and display of the various parts of the project to be demonstrated. This interface showcased the various filtering techniques, including blur, colour filtering, and background subtraction. The prototype also had the capabilities of displaying each of the gestures being recognized in real-time.

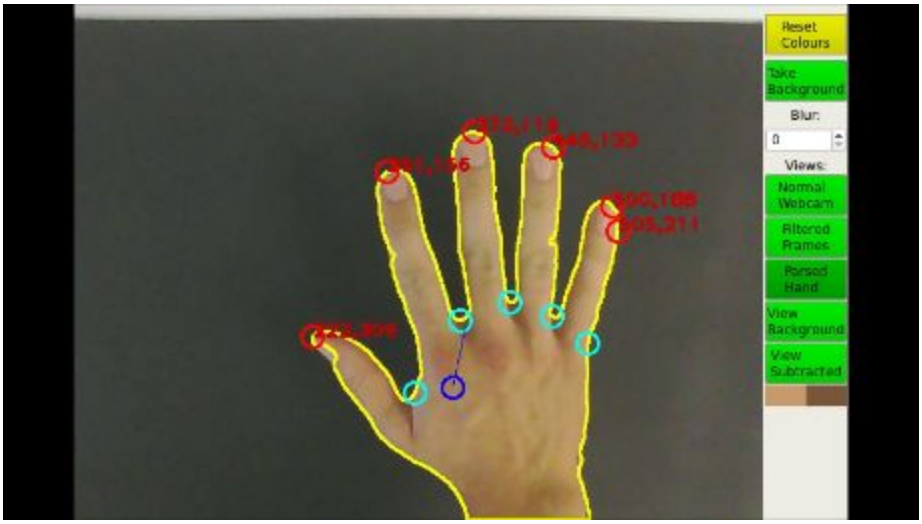


Figure 2: User Interface for Prototype

4.1 Processing Sequence

See Appendix B - Sequence Diagram for a diagram of the process used. This shows the UML representation but does leave out some of the more pertinent information. The flow-chart below is a simplified version, which shows the major image processing elements and techniques.

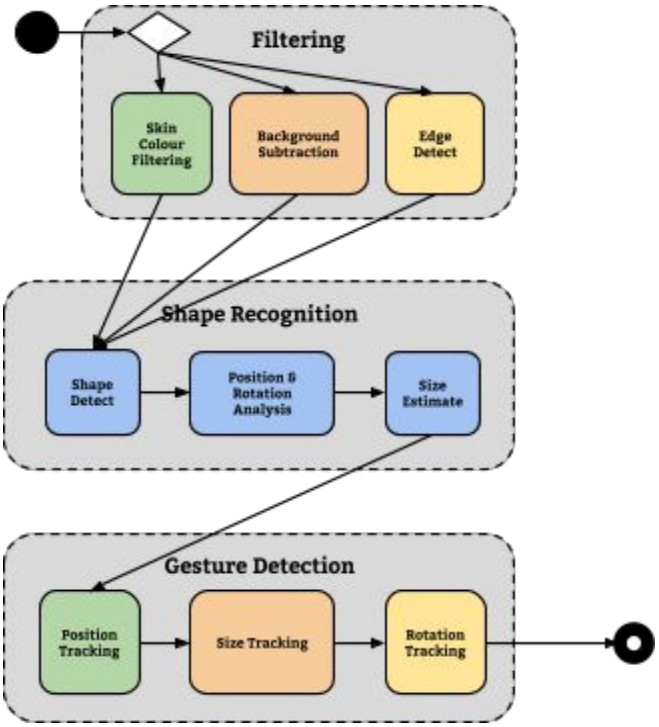


Figure 3: Sequence of Operations

The processing starts with a frame being received by the camera, as shown in the sequence diagram. After the frame is received, the process function calls the Filtering class's function filter(), which is responsible for the actions shown in the above chart. It also performs operations such as smoothing the image before it is processed. The three actions are currently separate; skin colour filtering can be used for the image, or background subtraction, or edge detection, but not all three.

Currently, skin colour filtering is complete and working. Background subtraction works but not to a degree where it is useful as most of the time the cameras being used moved enough to create false positives, unless they were pointed at an object of just one colour.
After filtering has been completed, a black and white image is produced which contains black where the hand is believed to be and white where it is not. This image is passed to the shape recognition class. First the shape recognition program finds all contours in the scene using an opencv chain approximation algorithm. Once these have been found the one of largest size is chosen to be further analyzed as the general use case for the program is that the hand will be the largest object of its colour in the scene.

4.2 Class Architecture

See Appendix A - Class Diagram for the full diagram showing members of the class. This class diagram shows the full system with all members; however it does not convey all the information about the project as the slots/signals are not shown explicitly. For a better diagram of the signal and slot interaction, see below. Closed point arrow indicate function calls, and open point arrows indicate signals.

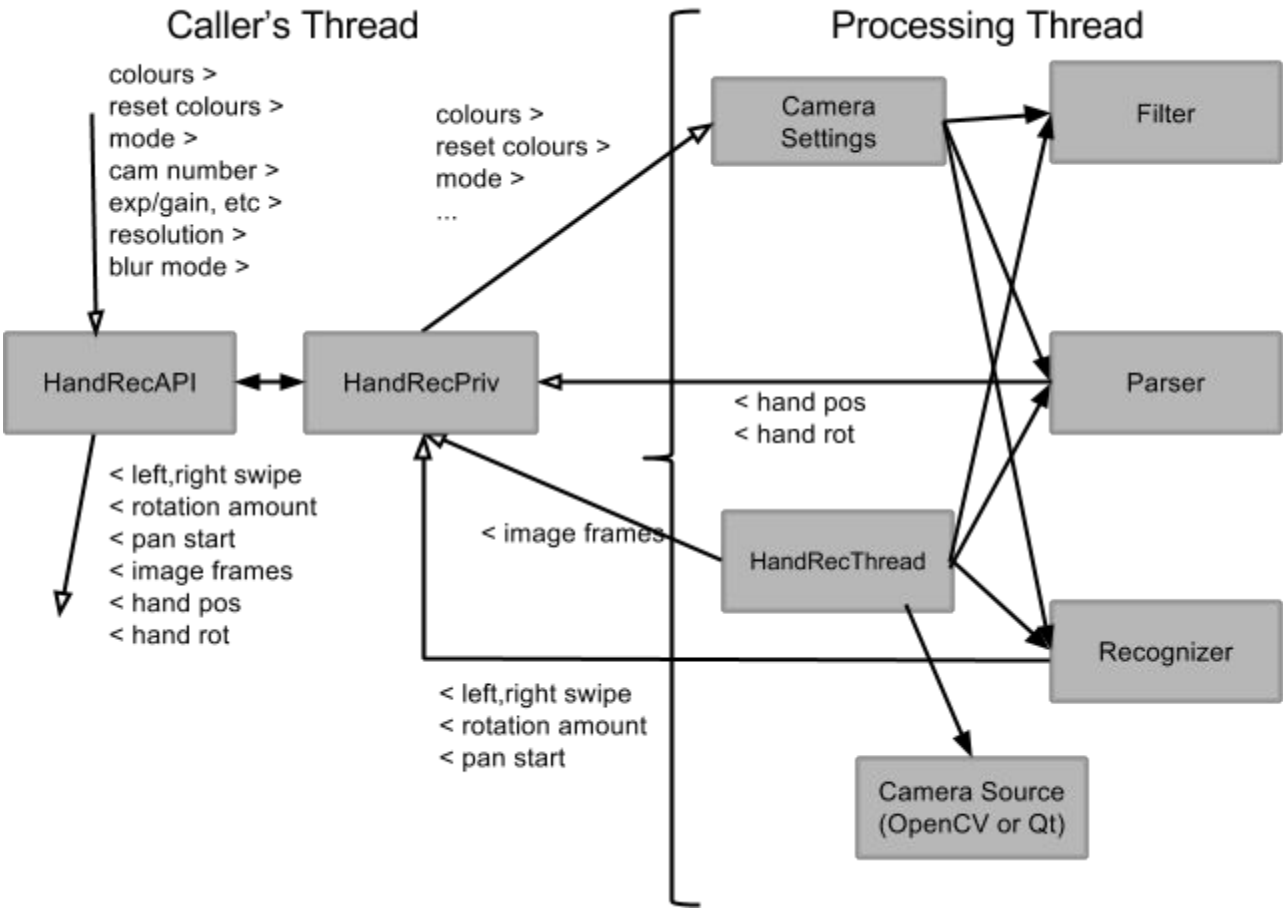


Figure 4: Signal Diagram

On the left hand side of the diagram, the objects which operate in the same thread as the caller are shown. The primary class the developer of a program using the api would see is the HandRecAPI class, which handles all of the interaction between the outside and inside of this system.

The HandRecPrivate class handles the actual interaction between the threads, and most functions in HandRecAPI are directly mapped to another function in HandRecPrivate which actually performs the operation. This follows the PIMPL design pattern (private implementation) by exposing only what is needed in the outside class and keeping the operation of the system hidden. It also simplifies the header for the HandRecAPI class which already contains many functions.

4.3 API

The project is designed to handle filtering and gesture recognition internally, and only expose the recognition of gestures to the user. A user facing API is used to send the Qt signals when a gesture is recognized. Users using the API would hook up Qt slots to the signals, and would receive notifications when a gesture is performed, and can translate that into another program, such as infotainment controls. Currently, the API can output ten different gestures to the user, as well as the filtered video frames.

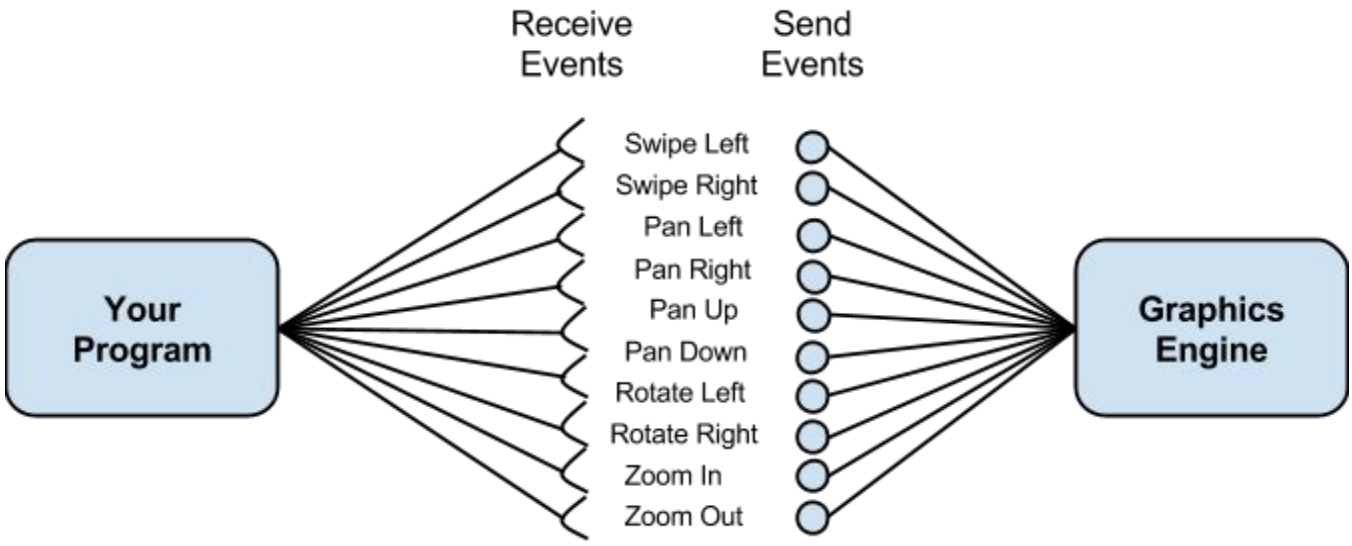


Figure 5: API Signals

5 Embedded System Installation

A part of the project was running the code on a non-desktop system and seeing the result. This started with an i.MX6 board because it has a reasonably fast processor (1GHz quad-core armv7) and was available to the team. The difficulty is that with an embedded system the hardware support is often less robust and much of the software needed had to be build from source. Qt, OpenCV and video4linux needed to be compiled and installed in order to be able to create a graphical user interface on the board's screen, do the image processing, and capture frames from the webcam.

The i.mx6q Sabre Lite board has a reasonably recent Ubuntu build image available for it, so this was chosen as a base point. The board uses an unusual Vivante GPU, so while there were Qt binaries available for download for ARM, they did not properly support it. Therefore, Qt needed to be built from source. This process took several tries to configure properly but eventually started; however, because it needed to be built on the actual board it took several hours to compile a few of the necessary modules. When it started compiling the EGLFS plugin (a linux interface to the framebuffer that Qt uses), it failed due to incompatible header files. Unfortunately this meant that the compilation could not continue and a different method of running was needed.

The next effort involved a custom build of linux using the manufacturer's build tool, Yocto. This build system cross-compiles linux in it's entirety, and had multiple issues which needed to be overcome including compatibility issues with gstreamer for the Qt plugins, the boot image not being found due to an updated version of the boot firmware which expected a different type of image (which could be solved by loading it manually rather than using the automated tooling from the manufacturer), and touch screen configuration. 30GB of compilation later, linux was up and running on the board. Unfortunately, one of the issues with the build of linux is that it has compatibility issues between the video4linux package and the GPU. After several failed attempts at rectifying this, it also turned out to be a dead end.

The final attempt at using the i.mx board involved using a 4.3 build of Android from the manufacturer. OpenCV and Qt binaries exist for the armv7/android combination, so the build process was significantly shorter. Video playback with VPU acceleration worked, as did Qt programs. Unfortunately, the android build did not support USB webcams though the android service and the build of OpenCV did not provide root access to the underlying linux structures providing the video capabilities to Android.

Overall, this was a frustrating experience as each of the different operating systems on the same system could perform parts of what was needed - if all three could be put together the system would work beautifully. Unfortunately the support for linux on this board is not quite at the level where it can be depended on for such uses as video capture.

The level at which android worked, however, was encouraging as all that was not functioning was the bridge between webcam hardware and software. Since a normal cell phone has a similar armv7 processor to the board being used, it was guessed that it would support at least as much as the board had. A build was made and packed into an APK file, installed on the device, and it worked!

The program was not optimized for touch, but the actual video processing worked exceptionally well on the phone. After 10 minutes of operating, the battery usage was analyzed and and found that it used more than not running the program, but not prohibitively so. Many applications cause the phone to heat up significantly with extended usage, but after half an hour of the program running the phone's temperature had not changed significantly from before running the program.

Therefore, ultimately we ported the gesture recognition software to Android 4.3 running on a Nexus 4 cell phone after being unable to successfully integrate all the required components (Qt, OpenCV, etc) on the mx6q Sabre Lite development board.

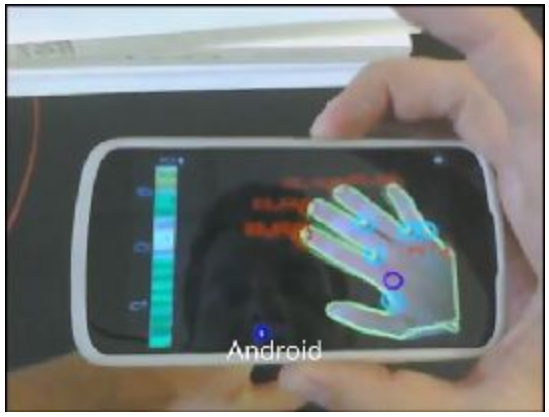


Figure 6: Gesture Recognition on Android

6 Discussion

This application was made to find a hand in a webcam stream based on colour and shape, then use that information to recognize a set of gestures including swiping, panning, and rotating. Initially, we had also planned on implementing automatic colour recognition, background subtraction techniques, and

zooming gesture functionality. Some of these functions were implemented, but not included in the final iteration, and others ended up beyond the scope of this project.

The colour based hand recognition is done by manually selecting several points on the screen and then using the highest and lowest points in that range as the beginning and end of the range of colours accepted. Once this range has been established, the program will use it as the first stage of image filtering. In the ideal situation this process would be automated so that the colour range would not have be reset before the application could work with a different person, but unfortunately it is a manual calibration step at this time.

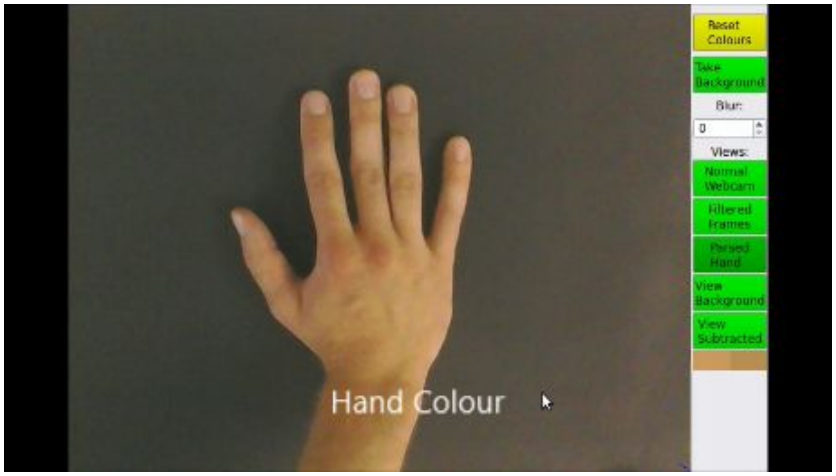


Figure 6: Hand Colour

Figure 7: Background Subtraction

The shape recognition is done after the colour recognition has established what part of the video frame contains the hand's colour. The shape recognizer then selects the largest block of colour as the hand, and finds contours and finger points in that shape. Background subtraction is a technique that could be used to better distinguish the hand from the background. This technique would allow for complex shape recognition to be implemented without worrying that the shape recognition will have false positives from the background of the video frame.

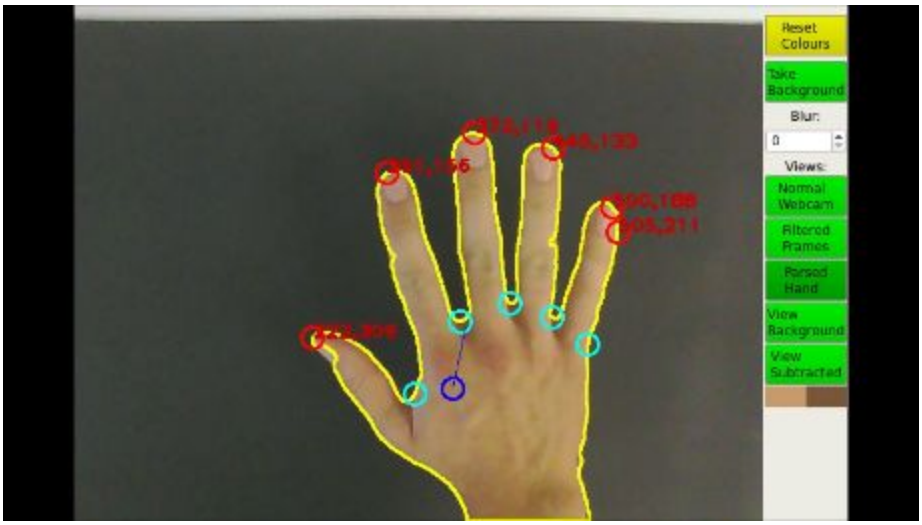


Figure 8: Shape Recognition

The set of gestures recognized by the program include swiping, panning, and rotating. Swiping is a quick motion from off of the screen on one side, to off of the screen on the other side. Panning is the placing of the hand in the middle of the screen, and then moving it left, right, up, or down. Rotating is similar to panning, but the hand remains relatively still, and instead rotates left or right. Zooming is the fourth intended gesture that would rely on detecting hand size to indicate whether the hand was moving towards or away from the camera. Unfortunately time constraints did not allow for zooming to be included in the project. In addition, panning in the y-direction was disabled because of the hand recognition including parts of the arm in the hand shape.

7 Lessons Learned

While working on the project, we discovered that, despite the cross-platform claims, some video based features of Qt did not work on Windows. This meant that the code had to be changed to move webcam handling entirely to OpenCV. We also discovered numerous problems when porting the project to an embedded environment, as outlined in Section 5.

In terms of actual filtering and hand detection, we discovered that environment brightness levels, and camera exposure had a large impact on the colour recognition. These levels had to be controlled in order to get consistency in the colour detection algorithms. This is a problem because even with openCV and Qt, different operating systems have completely different capabilities in terms of feature detection and setting, even with the same camera.

We were also able to do better colour recognition using the RGB colourspace than the HSL colourspace. The finger detection worked well when the fingers were spread out and the defects could be easily detected. When the hand was closed, the hand could still be recognized by colour, but more detailed information regarding the fingers was hard to determine in HSL.

8 Future Plans

In order to turn the project into a fully functional api that could be deployed into an environment such as a car's infotainment system, there would be a few features to be added or improved upon. The features requiring improvement include the colour selection, webcam controls, finger detection, and background subtraction. In addition, the porting and integration of the project into other environments would be critical to moving forward with the project.

The colour selection is currently done manually before the recognition can occur. Ideally the hand colour could be calibrated automatically, just by placing the hand in front of the camera. This could be done through a static background subtraction, or automatic colour detection. A static background subtraction would rely on one or more background images to subtract out and leave the hand. Automatic colour detection could be done by looking for a hand shape based on a range of likely hand colours, and then using the hand shape to detect the actual hand colours. Further colour optimization could be done as the program runs, where it would redetect the colours of the hand with each gesture, and finetune the stored hand colours. This would improve on the colour detection, as the colours may differ slightly based on environment variables, such as lighting.

Enhanced webcam setup would ensure that the program would work correctly using a range of hardware. Controlling the exposure, saturation, brightness and other webcam settings from the program, would ensure that the frames returned from each webcam would not differ greatly, and that the same algorithms could be applied to any frame from any webcam. Different forms of automatic detection would have to be used, as different webcams use different values to indicate the same settings.

The finger detection could be expanded, such as with Canny edge detection. The openCV canny detection would run on the image, producing a binary image showing the contours of all objects in the scene. This would be run through a contour detection scheme, and then all contours big enough to be a hand could be checked against the original image to see if they are in the correct colour range. By doing this, all contours which were part of the hand could be detected and recognized by proximity, and added together to form the total shape of the hand but with more accuracy and precision than simply filtering on a broad range of colours. With this better detection, more parts of the fingers could be recognized, such as the individual joints. The range of gestures that can be recognized would increase, and more actions could be performed using the API.

Work on the background subtraction could be expanded to work beyond a static setting. This would require the background to update to handle camera movement, and then recognize objects in the background. These objects may match the colour of the hand, but once they are recognized as background objects, they can be added to the background and subtracted out of any gesture. This would allow for better gesture recognition and removal a large amount of false positives from the gesture recognition engine.

Once the project has been improved upon, it can be ported to multiple environments. One of the environments that the project was ported during the demonstration was mobile Android. In order to function best on a mobile platform, the improvements that handle camera movement would be required. To this end, the program would be built into a native Android app, with a mobile user interface to properly facilitate the controls. Another environment that the project would be ported to is an embedded environment for a car's infotainment system. The EcoCAR infotainment system, the primary motivation behind the project, is built on top of embedded Linux built using the Yocto build system, and for now requires work to be done by the manufacturer to fully support usb webcam capture natively.

9 Conclusion

This project was mostly a software-based project, and therefore a software design process was utilized as part of the development. Due to being busy students a full agile process was not utilized, but the team did set weekly targets and would communicate about progress whenever any was made. For parts of the project methods such as pair programming were used, and a class diagram was used throughout the development to keep track of the various components in the system. This process enabled the team to work effectively on different parts of the program and still easily be able to integrate the work of all members.

The design focus of the code is to provide an api which was is as easy to use as possible; this came at the cost of simplicity as the api does not currently support advanced tracking of data such as individual finger positions. The design emphasizes flexibility and extensibility, by enabling parts of the algorithm to be swapped with other parts while still compiling, and in some cases even running properly.

As mentioned in the previous section, the development of this project will not stop with the end of the 499 project. The developers have active interest in finishing the project to the point of usability in multiple systems. The extensibility and flexibility of the program as well as the cross-platform nature will enable it to be integrated with different programs easily and possibly integrated as an app for use with mobile devices. Furthermore, the low calibre of hardware required will enable the program to be used with many devices without additional hardware, or simply a cheap webcam the potential user might already own.

10 References

This report was written without the usage of documentation, however several sources of information were used extensively throughout the development process for the project.

This is the page on building qt5 from source:
http://qt-project.org/wiki/Building_Qt_5_from_Git

The qt-project documentation was used in development:
http://qt-project.org/doc

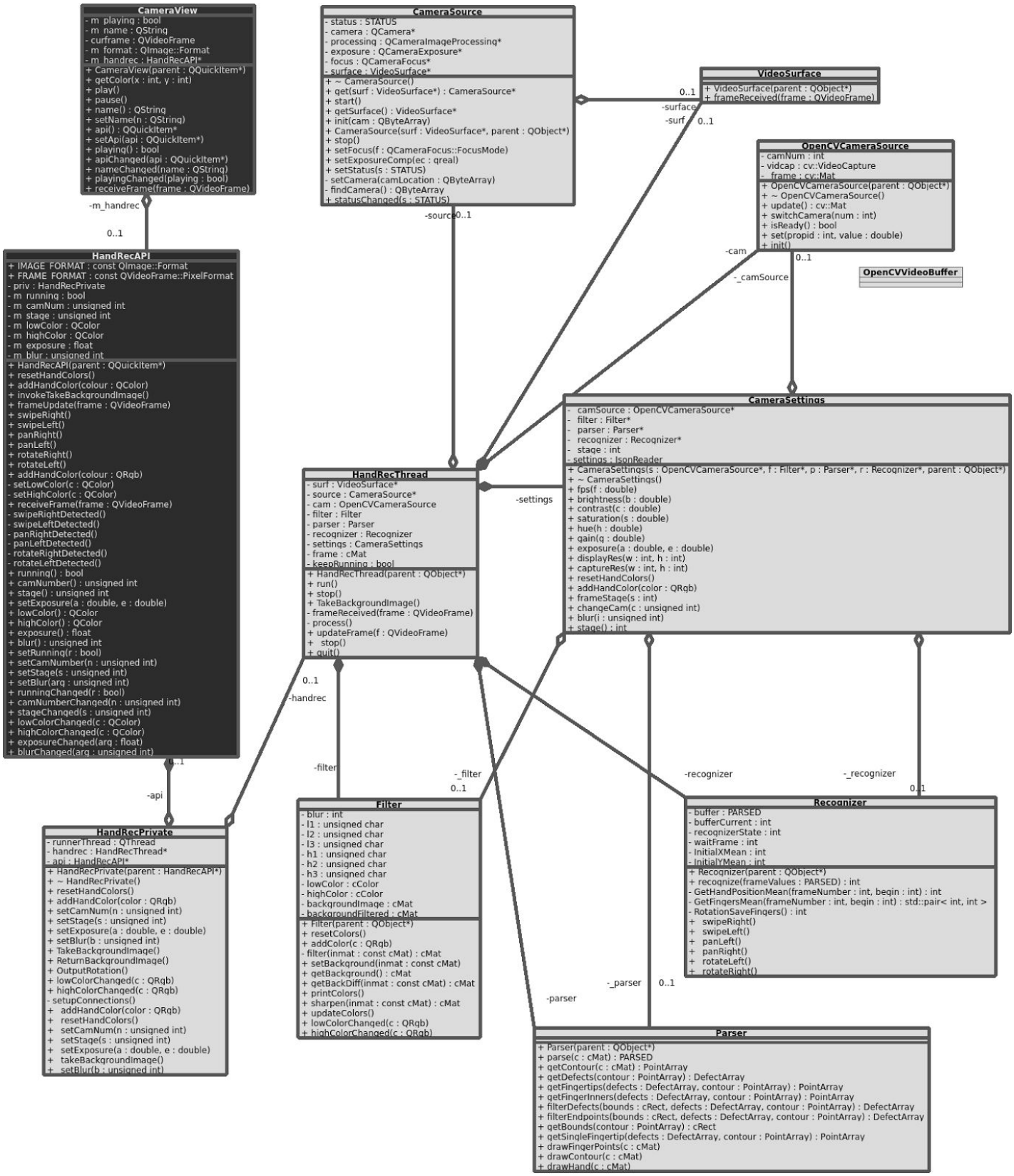
Information about Qt's new version of signals and slots was found here:
http://woboq.com/blog/how-qt-signals-slots-work-part2-qt5.html

Open-CV documentation was also used:
http://docs.opencv.org

In particular, the image processing and convex hull pages:
http://docs.opencv.org/modules/imgproc/doc/imgproc.html
http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html

Finally, the android platform documentation for OpenCV was consulted for usage with android.
http://opencv.org/platforms/android.html

11 Appendix A - Class Diagram



12 Appendix B - Sequence Diagram