Lab 1 Multimedia Systems

Group
Marc Dahlem
Jon Martin Mikalsen

1 Problem Specification

The multimedia course has three labs as hand-ins, where each lab together form a bigger system. The overall idea of this project is to create a video streaming client, for example a computer with a web-cam, that send a stream of video to a server. This server shall then have the ability to detect motion, which then send a notification to a connected Android phone with the detected movement as a recorded video, as well as a message that there has been a movement on the video. See illustration 1.

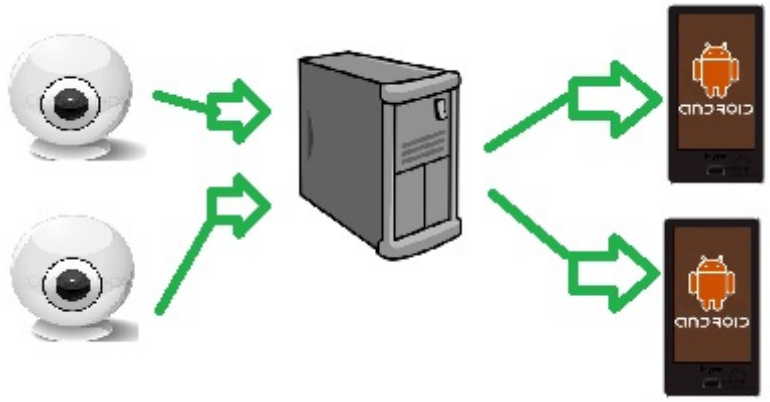

Illustration 1. System architecture

The idea then is to split this into three labs, where the first lab focus on watching and recording a video stream from a web-camera. The second lab focus on transferring the video stream to a server, and then implementing motion detection on this stream,and when a motion is detected, it is recorded on the server. The third and last lab will then focus on getting the recorded motion, sent to a Android phone along with a notification about the motion detected. The android app should also be able to watch the stream in realtime.

1.1 Description

In lab1 there has to be created an application that could view video streams from a web-camera and record it. It had to use the gstreamer network and the shown solution is build on gestreamer-java. The first idea was, to start the recording automatically when a motion is detected, but since the motion detection is a hard task, the first application was build as manually starting video recorder. The motion detection will be addressed in lab 2.

1.2 Assumptions

In the start of lab 1 there were three things that were assumed.

1. That all the plug-ins wanted could be used.
2. That it would be easy to find a plug-in that solved motion detection.
3. That it would be easy to stream and record video at the same time.

But the last two has turned into problems and are discussed in chapter 4.

1.3 Extra features

When creating the application there was a equal feeling in the group that only viewing a video stream would be to little for the first lab. It was decided that different extra features would be implemented to make the application more interesting, and the following things are added as extra features:

- Fullscreen on/off
- Record X video(s) that will be stored with different names
- View video-stream while recording
- File-browsing to choose where recorded files are stored.

Motion detection was also something that was tried out in this lab, but with no success.

2 Usage and user's guide

Here the instructions on how to get the application running is stated.

2.1 Requirements

To get this application up and running there are several requirements that as to be in place.

- Have the knowledge on how to run a given java Eclipse project/program
- Have GStreamer installed on computer
- Have the ability to run other GStreamer applications developed in Java
- Have Eclipse running and working on computer

2.2 How to get it running

This section describes three diffferent ways to get the program running.
For just running the application a runnable jar-file was created and attached to the fronter submission. The second way is to download the source files from the setup git reposity.
If you want to create an own project with the source file, than you have to follow the manual instruction to setup the eclipse project.

2.2.1 Runnable .jar file

The optional way to run this application is to run the .jar file following the hand-in called “Multsys_lab1.jar”, to do this simply open a terminal and run:
java -jar Multsys_lab1.jar

2.2.2 Git repository with code

A already predifened eclipse project is available on the git repository

- https://github.com/MarcDahlem/ANSUR

It includes a fully setup eclipse project and can be imported into the eclipse workplace.

2.2.3 Manual instructions to install

To run the program it can be manually set up in Eclipse, or a single jar file can be run in the console, see “2.2.3 runnable .jar file”.

Following this document is all the code for the first lab, and the main folder is named “ANSUR”. Inside this folder there is another folder called “Multimedia_Lab1” that is the first lab, and “org.eclipse.swt” that is a support project that gives the possibility to run the application in SWT[1].

One way to get these up and running is to create two new java projects in eclipse, called “Multimedia_lab1” and “org.eclipse.swt” and then copy / paste the contents into these project folders.

To make sure everything is ok, the gstreamer-java.jar and jna.jar has to be added in the project. to check this, right click on the mulitmedia_lab1 project, and go to properties. See illustration 2

[1] http://www.eclipse.org/swt/

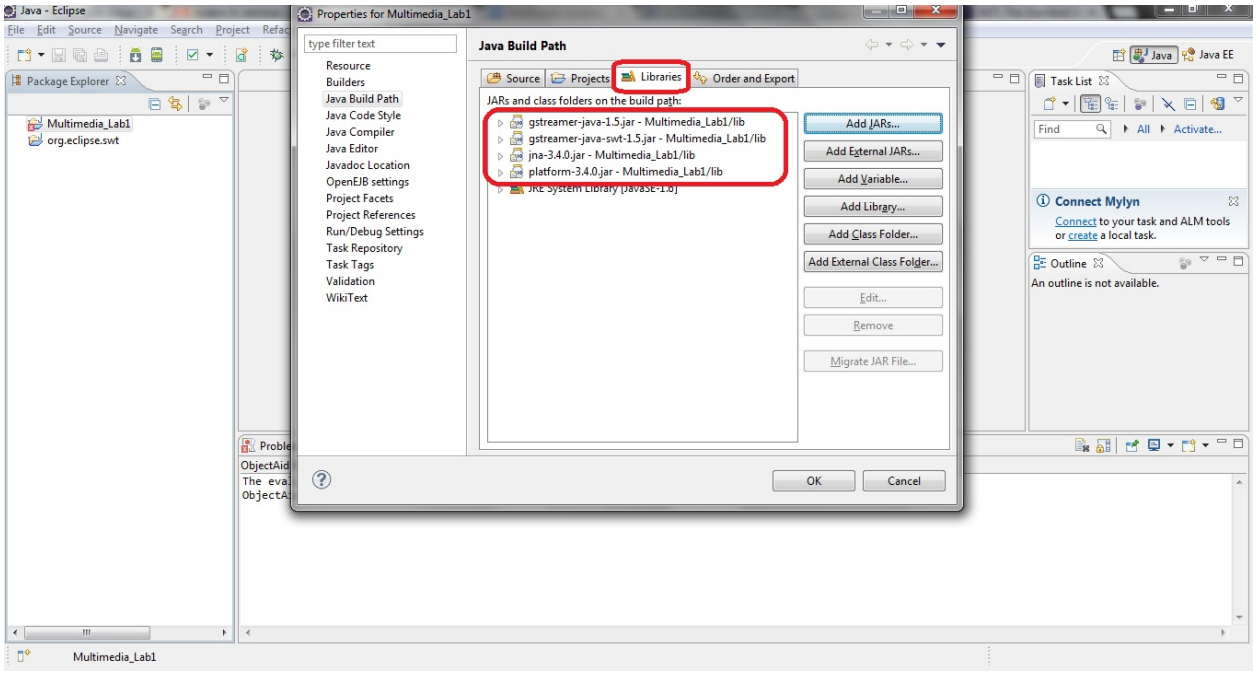

Illustration 2: libraries that has to be in the project java build path

When the project is opened in Eclipse, the “org.eclipse.swt” has also to be added to “Java Build Path” in the projects properties. To do this, right click on the project and go to “properties”. Then on the left the “Java Build Path” should be visible. See illustration 3

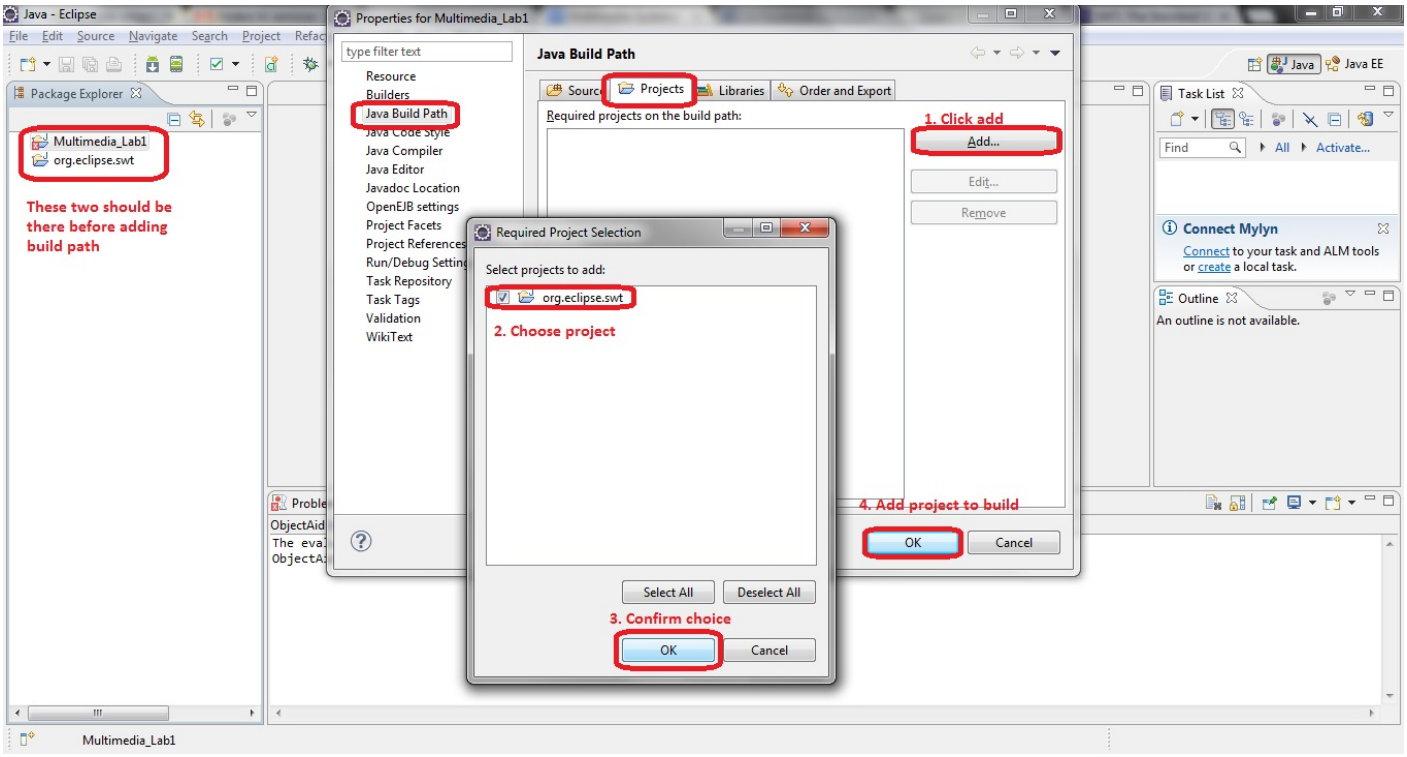

Illustration 3: How to add org.eclipse.swt to project build path

When these things are in place, the program should be ready to run, and to do this just go into the folder “Multimedia_lab1/src/APP/MyRecorderApp.java” in Eclipse, right click and choose “Run as -> Java application”.

2.2 Result

After starting the application from eclipse one can see possible error messages from gstreamer in the eclipse console. If so make sure, that gstreamer and all required plugins[2] are installed correctly.

The default view of the application consists of a (empty) video part and a menu bar. The menu bar has buttons to start and stop the webcam, capture the video and some other preferences. Illustration 4 shows the running application. The webcam is started (Play button is green), and the application is also recording (record button is red). In the preferences in the right bottom corner is the show fps feature set.

Illustration 4: Running application with webcam and recording started. FPS are also turned on.

The application can easily be switched into fullscreen with the fullscreen button at the right corner or pressing ‘f’ on the keyboard when the video part is selected. The default fullscreen view has no menu connected, but if one moves the mouse towards the edges (see Illustration 5), the menu bar will appear and stay until no mouse movement was detected for 3 seconds. Then it will disappear again. Pressing ‘f’ in the fullscreen mode will switch the application back to the windowed-mode.

[2] including https://gitorious.org/gstreamer-motion-plugin for test purposes with the motion. This plugin can be easily installed with automake after setting the correct path in the config file of the plugin subfolder.

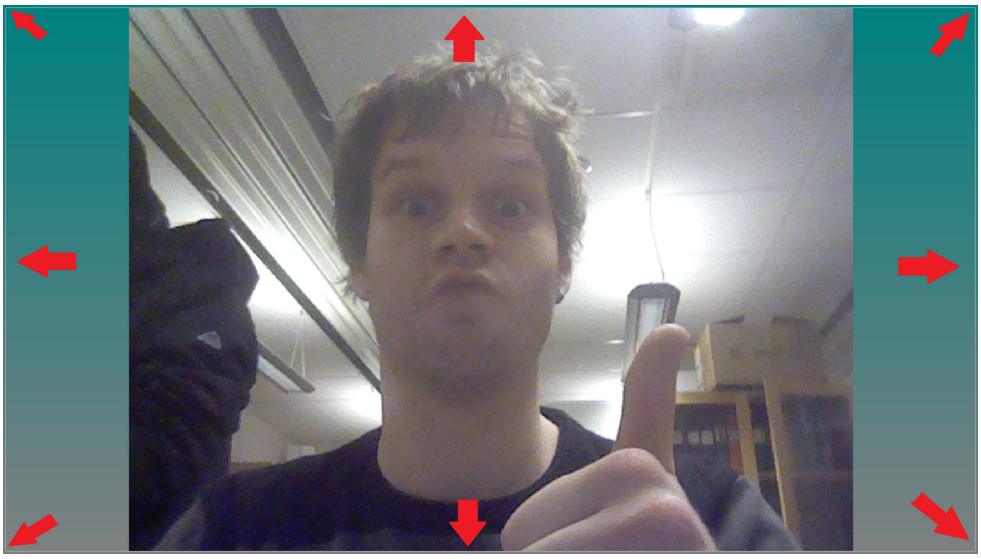

Illustration 5: Fullscreen with arrows indication where menu will pop up

3 System description

3.1 Internal design and structure

The application is split into three classes, that each have their task.

The three classes are called:

- MyRecorderApp.java
- Gui.java
- Recorder.java

The first file “MyRecorderApp” is the main program that run the application, and here the display, video stream and the recorder is set up.

In “Gui.java” is where the user interface is defined, and it is here all the buttons get a connection to an event/method and where all the actual features are set to happen. For example it is here that the applications window size is set, and the feature where the menu disappears after 3 seconds in fullscreen mode.

In the last class “Recorder.java” the actual magic with getting a videostream and recording the videostream is set up by using camerabin with manually set encoders, muxer and sink. For more information on how it is built up, see source code.

3.2 Classes

Here you can see the class diagrams that gives an overview on all the methods that each class consists of. For more information on the different methods, look in the javadoc delivered with the report in fronter.

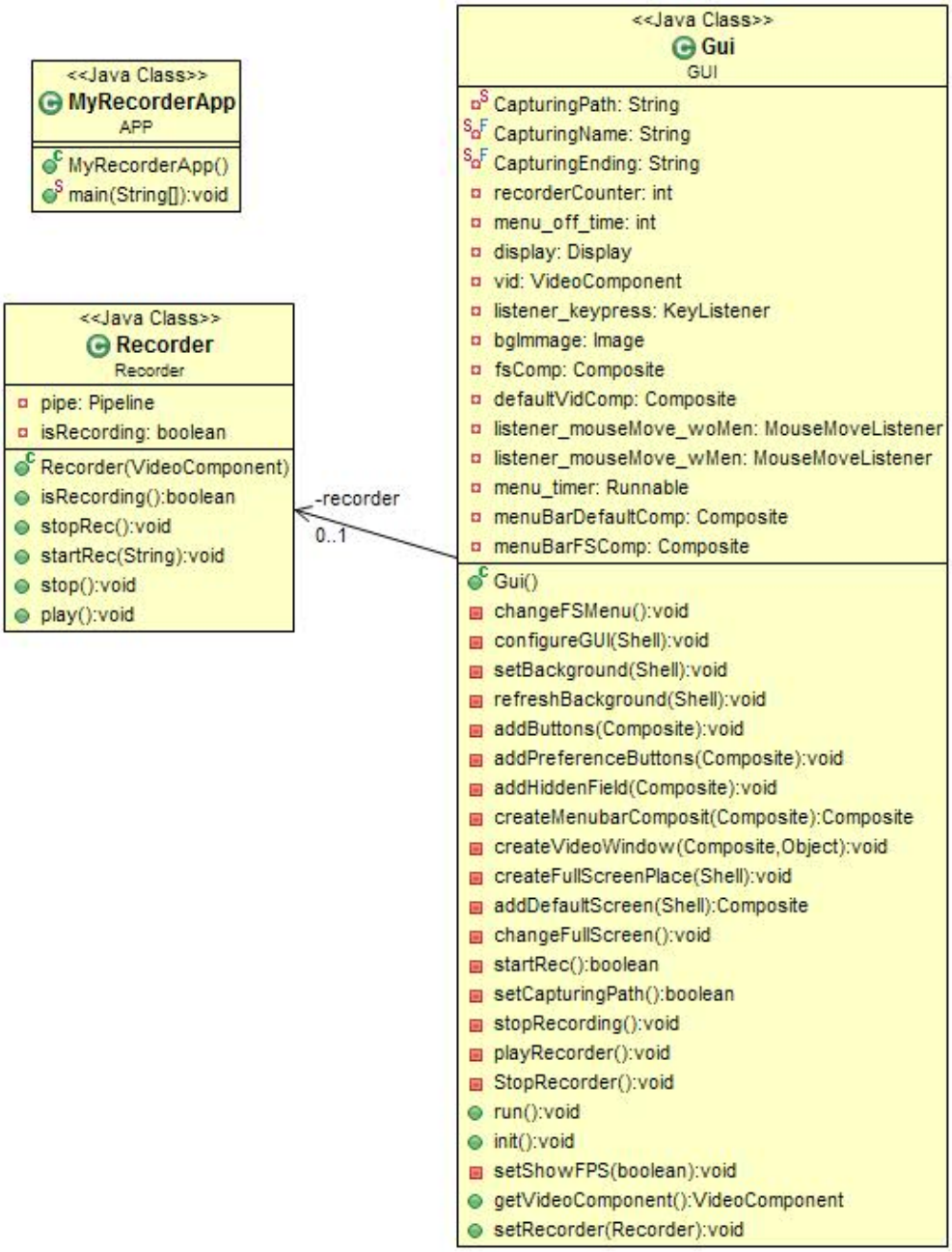

Illustration 6: Classdiagrams

4 Problems and reflections

4.1 Limitations

Since we used camerabin as recorder and player, the used pipeline is not very flexible and extensible. It is an single bin and it can not easy be adapted to needs. So it is for example very difficult to set a normal element out of an pipeline as source for the video
Furthermore it was figured out, that there seems to be an bug, if the pipeline is stopped and restarted when there was something recorded. After the restart the files will only be 6.7kB and will not contain any video content.
Another limitation regarding camerabin is, that the encoding has to be fast enough to be handled in realtime. The reason seems to be the buffers that are used in camerabin.
So the program performance is very dependent on the computer it is running on. Another fact is, that the frame rate (fps) goes on the testcomputer down to 5-10 frames per second, when the video is switched to fullscreen. The fps canges with the resolution used on the VideoComponent of the GUI. If the window is small enough, the framerate will go to 30 fps, what seems to be the maximum possible with the connected webcamera. This fps changes makes also the video capturing hanging in some point in times.

4.2 How to avoid limitations

In order to solve the problems with camerabin, we want to switch to the camerabin2[3] module for the next labs. Or even better would be, if we can get the pipeline described in 4.3.2 running, because it is totally independent of any bins and easily self configurable.
To avoid hanging video in capturing it is very useful to use a small window as possible or to adapt the used resolution received from the webcam. Within the next version we will try to check if the problem is maybe related to the used SWT -Videocomponent and it will be compared with other possibilities.

4.3 Problems during lab

4.3.1 Motion detection

When we started on the lab we wanted to implement motion detection as the key function, and after googling on different solutions on how to do this we though that this would not be that hard. However this turned out to be a little more complicated, and after a hours we gave up the motion detection part for the first lab. But we want to take a closer look to it in the next lab.

4.3.2 Simultaneous video playback and recording

[3] http://www.freedesktop.org/software/gstreamer-sdk/data/docs/2012.5/gst-plugins-bad-plugins-0.10/gst-plugins-bad-plugins-camerabin2.html

As the new idea for the first lab emerged, we wanted to stream and record video from the web-camera at the same time. But in the start we could only do one of these at one time, and not both as we wanted. At that time we used a own build pipeline like shown in Illustration 7.

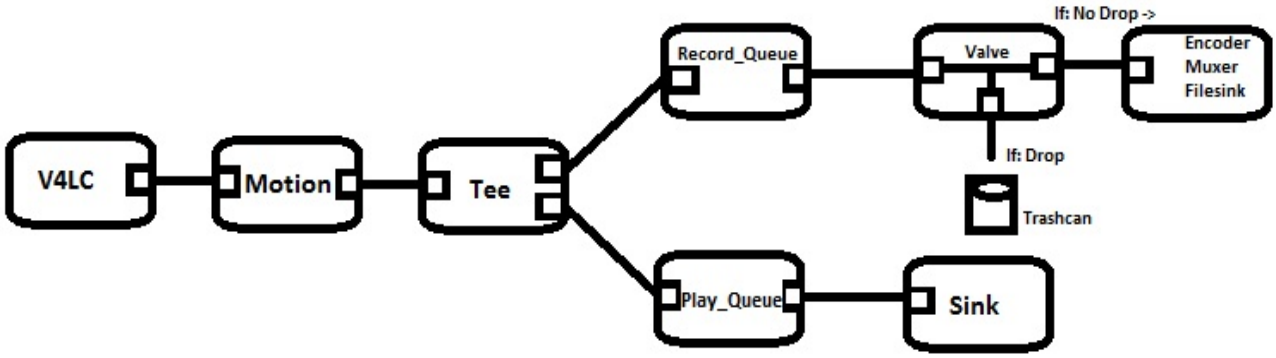

Illustration 7: Pipeline before camera-bin was implemented, Motion is not implemented but is planned

A tee was used to split the data to the recorder and the video playback part. Buffers have to be used after a tee to create different gstreamer threads for both parts[4].
In the playback part the VideoComponent of our GUI was used as sink to playback the video.
The leaky-variable of the play_queue has to be set to 1 or 2 for making simultaneous recordings possible[5] with the avi encoder x264enc we used at this moment.
The recording part was controlled by a valve element, that either forwards the buffer or drop it, depending on a drop variable. In order to change the filename after each stopped record, one has to send an EOS message to the parts following the valve-element and stop all the following elements and then restart them when a recording should be started.

This shown solution worked really fine if one starts to record directly on application startup. But we had a lot of problems when we wanted to start only with a video playback without recording. While searching for solutions for the occurred problems, a module called ‘camerabin’ was found, that hides all the complexity in one element. So we changed after this our recorder to use this plugin.

4.3.3 Gstreamer in Windows 7 64bit

Since Jon Martin has Windows running on his computer, there was a wish on getting Gstreamer to work even here. Following the guide [6] to install it, and after failing and trying for a while, Gstreamer worked in the end. Though, Jon Martin never got any application to run in Eclipse, so the first lab had to be developed on Marc’s computer during the lab hours. The reason for this was an error message in Eclipse about not beeing able to load Gstreamer.

---

[4] http://gstreamer.freedesktop.org/data/doc/gstreamer/head/gstreamer-plugins/html/gstreamer-plugins-tee.html
[5] http://www.oz9aec.net/index.php/gstreamer/410-x264enc-problem-in-gstreamer-video-switcher-solved
[6] http://www.sjkingston.com/blog/gstreamer-on-windows/

Illustration 8: Error message in Eclipse on Windows

5 Contribution

Since Gstreamer in eclipse did not work in Windows, and on Jon Martins computer, all of the code was written on Marc's computer. This resulted in that much of the final code was written by Marc when he sat at home in the evenings programming. But over day both Marc and Jon Martin sat together in the labs programming together on the same computer, similar to Pair Programming [7]. It was also in the start that both came up with the idea for the complete system. When basic functionalities was up and running, like streaming video, recording video and saving files, Marc did the rest of the work with designing the Gui, getting the final functions like saving files using file browsing, and also setting up the program to support showing a videostream while recording.

When the program was complete, Jon Martin worked on the diagrams, illustrations, descriptions and on building the Lab 1 document that you are now reading.

We want to thank the gstreamer community for creating and sharing Gstreamer to the public. Also want to thank all of the people out there who have published their working projects for hints and guidance on how things work when using Gstreamer.

[7] http://en.wikipedia.org/wiki/Pair_programming