TLAP
A time-lapse movie generation project

Vlad Caia

Contents

Chapter 1

Pre-project

1.1 Project idea and background

In 2011, The City of Gjøvik celebrates its 150th birthday, and we are looking to create a range of projects that participates and add to this celebration. The goal of this project is to use open source command-line tools to construct a workflow that could help in the collection, description, transformation, transportation and publishing of timelapse movies from the city of Gjøvik.

> Time-lapse[1] photography is a cinematography technique whereby each film frame is captured at a rate much slower than it will be played back. When replayed at normal speed, time appears to be moving faster and thus lapsing. Processes that would normally appear subtle to the human eye, such as the motion of the sun and stars in the sky, become very pronounced. Time-lapse is the extreme version of the cinematography technique of undercranking, and can be confused with stop motion animation.

1.2 Problem statement

The real problem that we are facing here is how to implement such an automation process with the tools that we have at our disposal. To make it all happen, a lot of these tools have to work togheter to become integrated into a single tool. We will use and/or develop open source software across different platforms, generaly on a Linux based environment.

1.3 Project goal

Our mission is to create an automatic workflow that helps the user create and integrate effortlesly, in every type of digital medium, a timelapse video shot at a location of our choice. The way we are going to achieve this, is to create our own scripts/programs making use of open source software.

[1] `http://en.wikipedia.org/wiki/Time-lapse`

1.4 Research

The solution to the problem will start with research. First we will identify the best open source tools suited for our needs. Secondly all the documentation on these tools have to be studied in order to have a better understanding on how these tools work and how we can use it in our project. The internet offers a vast area of research on different topics. A torough research on time-lapse photography will better educate the researcher on how to do it correctly and efficiently. Further research on how we can achieve a workflow between the capture camera and the computer and the end video will be done.

1.5 Project cost and risk analysis

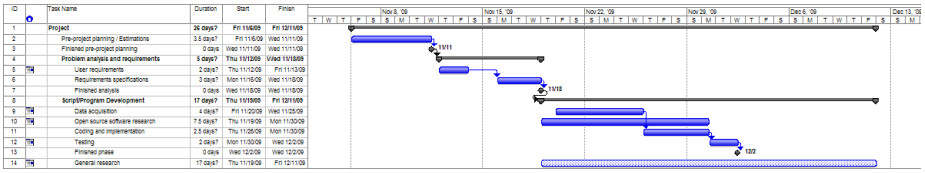
For an estimation, we can say that both data collection (images for later processing into a final time-lapse video), research and code development will take a fair equal amount of working time.

Problem/Risk	Probabilty	Consequence	Action
Poor requirements	2/5	If requirements are unclear, incomplete, too general and not testable there will be problems	Do a deep analysis, reanalyze and estimate if tasks are duable in the time and budget frame
Unrealistic schedule	3/5	If to much work is crammed in too little time, problems will arise	Allow adequate time for planning, design, testing, bug fixing etc. plus allow some slack for use as a backup time
Miscommunication	4/5	If developers don't grasp the need of the client or client has erroneous expectations, problems arise	Re-evaluate, keep the communication tight during the development process; insure that information/documentation is availalbe and up-to-date

1.6 End result

The end result should be a well documented script/program based on open source software in an open source environment. In addition to this, this code should reflect the highest degree possible of the automation workflow.

1.7 Gantt chart



ID	Task Name	Duration	Start	Finish
1	Project	26 days?	Fri 11/6/09	Fri 12/11/09
2	Pre-project planning / Estimations	3.5 days?	Fri 11/6/09	Wed 11/11/09
3	Finished pre-project planning	0 days	Wed 11/11/09	Wed 11/11/09
4	Problem analysis and requirements	5 days?	Thu 11/12/09	Wed 11/18/09
5	User requirements	2 days?	Thu 11/12/09	Fri 11/13/09
6	Requirements specifications	3 days?	Mon 11/16/09	Wed 11/18/09
7	Finished analysis	0 days	Wed 11/18/09	Wed 11/18/09
8	Script/Program Development	17 days?	Thu 11/19/09	Fri 12/11/09
9	Data acquisition	4 days?	Fri 11/20/09	Wed 11/25/09
10	Open source software research	7.5 days?	Thu 11/19/09	Mon 11/30/09
11	Coding and implementation	2.5 days?	Thu 11/26/09	Mon 11/30/09
12	Testing	2 days?	Mon 11/30/09	Wed 12/2/09
13	Finished phase	0 days	Wed 12/2/09	Wed 12/2/09
14	General research	17 days?	Thu 11/19/09	Fri 12/11/09

Chapter 2

Development and implementation

2.1 License

This code provided in this project is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Norway license.
`http://creativecommons.org/licenses/by-nc-sa/3.0/no/deed.en`

2.2 Dependencies

gPhoto - gPhoto is a set of software applications and libraries for use in digital photography. gPhoto supports not just retrieving of images from camera devices, but also upload and remote controlled configuration and capture, depending on whether the camera supports those features.
ImageMagick - ImageMagick is an open source software suite for displaying, converting, and editing raster image files. It can read and write over 100 image file formats.
FFmpeg - FFmpeg is a computer program that can record, convert and stream digital audio and video in numerous formats. FFmpeg is a command line tool that is composed of a collection of free software/open source libraries.

2.3 Collecting data

Two types of data are used in this project: image and GPS data. Using a DSLR camera togheter with a GPS device is crucial for our project goal. For 'in-home' testing and coding I've used an NIKON D40 DSLR camera that doesn't have GPS taging capabilities and neither do I posses any GPS device for the position data. I've coded a GPS location manual input (more on this in the next chapters); for the field we will use the NIKON D5000 kit that comes equiped with an external GPS GP-1 device that can automaticaly tag the GPS data in each picture. Important note in the collection of images is the settings and the lens used in capturing the shots. An automatic lens is preferable for a consistent

shutter speed and aperture setting. Also the focus should be set on manual, the subject has to be focused before hand. Having these things in place guarantees a balanced quality on all the captured images, no sharp cotrasts or blured focus, suitable for a time-lapse movie.
Another important fact that has to be accounted is the way the shots are taken. We need a timer that takes shots at different interval times. Some cameras come with this feature built-in or with extra software run on a computer which the cameras are interfaced with. I had another idea in this project. The types of model of digital photo cameras is huge thus I've planned to use an open source application that can be used with all these models and not be dependent on the brand or model. One of the most essential programs in the development of this project is gPhoto.[1]
gPhoto is a simple program that interfaces with your digital photo camera (over 1100 camera models supported) and gives you the ability to take shots between different time intervals. It's a fast and a reliable application giving you the posibility of modifying different camera paramteres as well (such as shutter speed, flash settings etc.)
Another aspect of collection of data is the quality of the images taken. The FINE quality setting has been used on the camera, as well with the M(edium) 2256x1496 resolution. One thing to note is how gPhoto works. The capturing sequence starts, the image is taken and downloaded directly to your specified folder on the computer. In this way we don't have to worry about running out of storage space on the camera's SD card. For a better visual experience I focused on using the 720p HD final movie format togheter with a flash video file suitable for web integration.
The system is designed to work in a client-server achitecture, where the client (a notebook and a camera) generates all the files need for upload and hosting on a server.
Here is the code I've used for the gPhoto capturing step and the moving of files.

```
echo "\033[1mcapture frames\033[0m"
gphoto2 --interval $INTERVAL --frames $FRAMES --capture-image
echo "\033[1mmoving files to $CAPTUREPATH\033[0m"
mv *.jpg $CAPTUREPATH
```

$INTERVAL variable holds the timer interval between shots (for example 10 seconds)
$FRAMES variable holds how many shots to be taken (for example 500 shots)
–capture-image runs the capturing sequence and dumps the image files on the hard-drive to the same location from where the script is being run.

2.4 Transformation of data

After the collection of data certain transformation steps have to be done: Encoding, transcoding and resizing. In simpler terms, creating a time-lapse video from a sequence of shots and converting the time-lapse video to a flash video intended for web use as well with a preview resized image file. After research

[1] `http://gphoto.sourceforge.net/`

I've decided to use the open source FFmpeg[2] software that is best suitable for the video encoding/transcoding process. ImageMagick[3] will be used to resize the first frame of the image sequence used as a preview image for later web use. It will not be absolutely necessary to do this resizing procedure on all the image files as FFmpeg can do it automaticly without any extra commands. A streamlined and simple workflow is what is best to achieve. Here is the code I've used for the FFmpeg encoding and transcoding.

```
echo "\033[1mcreating preview image $PREVIEWFILE\033[0m"
convert $CAPTUREPATH/capt0000.jpg -resize 640x320 $PREVIEWFILE
echo "\033[1mtime-lapsing MP4 file to $MP4NAME\033[0m"
ffmpeg -y -f image2 -i $CAPTUREPATH/capt%04d.jpg -vcodec mpeg4 -vb $BR -r $FPS -s $SI
echo "\033[1mtime-lapsing FLV file to $FLVNAME\033[0m"
ffmpeg -y -f image2 -i $CAPTUREPATH/capt%04d.jpg -vb $BRFLV -r $FPS -s $SIZEFLV $FLVN
```

-f image2 specifies that the input is a sequence of files. $MP4NAME variable contains the name of the file the movie is being saved to.
$CAPTUREPATH has the path where the sequence of images have been moved to.
$BR variable holds the bit rate used for the high quality file (in our case 4000kb)
$FPS how may frames per second are saved in the video.
$SIZE here happens the scalling from 2256x1496 pixels to 1280x720 pixels.
$FLVNAME holds the name of the flash video file that is being transcoded from the original video.
$BRFLV the bit rate of the flash video file (in our case 800k)
$SIZEFLV the size of the flash video file (in our case 640x360 to maintain the 16:9 aspect ratio)

2.5 KML mashup

Mashing up the GPS location with the appropiate time-lapse video and maintaing a high level of automation workflow was the focus of this project. An important part of KML is the GPS data. This data exists in our images as meta-data. Also the user is prompted with an input field where he/she can input the single line coordinates found in Google maps. This way even if you don't own a GPS device, you can still aproximate your location manualy. For extracting the relevant GPS data from the image file I've used Exiftool[4]. Also an important note about the KML generation, after several attempts in trying to embed the flash video player I've used on the project's website, I decided to post a link with the preview image file. Seems like Google has deliberately disabled flash player embeding on their maps, Youtube works fine but any extra player will just not work. Here is the code I've used for Exiftool and KML dynamic generation.

```
echo "\033[1mwriting to $KMLFILENAME\033[0m"
rm tl.kml
```

[2] http://ffmpeg.org/
[3] http://www.imagemagick.org/
[4] http://www.sno.phy.queensu.ca/~phil/exiftool/

```
printf '<?kml version="1.0" encoding="UTF-8"?>\n' >> $KMLFILENAME
printf '<kml xmlns="http://www.opengis.net/kml/2.2">\n' >> $KMLFILENAME
printf '\t<Placemark>\n' >> $KMLFILENAME
printf '\t\t<name>Time-lapse project</name>\n' >> $KMLFILENAME
printf '\t\t<description>\n' >> $KMLFILENAME
printf '\t\t<![CDATA[\n' >> $KMLFILENAME
printf '\t\t<a href="http://www.stud.hig.no/~050704/"><img src="http://www.stud.hig.no/
~050704/preview.jpg">Time-lapse project</a>\n' >> $KMLFILENAME
printf '\t\t ]]>\n' >> $KMLFILENAME
printf '\t\t</description>\n' >> $KMLFILENAME
printf '\t\t\t<Point>\n' >> $KMLFILENAME
echo "\033[1mcheckin for gps data in $CAPTUREPATH/capt0000.jpg\033[0m"
COORDLAT=$(exiftool -GPS:GPSLatitude -n -b $CAPTUREPATH/capt0000.jpg)
COORDLONG=$(exiftool -GPS:GPSLongitude -n -b $CAPTUREPATH/capt0000.jpg)

if [ "$COORDLAT" != "" ] && [ "$COORDLONG" != "" ] ; then
COORD=$COORDLONG,$COORDLAT
echo "found GPS Position:" $COORD
printf '\t\t\t\t<coordinates>'$COORD'</coordinates>\n' >> $KMLFILENAME

else
printf "\nInput coordinates data (LAT,LONG): "
read COORD
printf '\t\t\t\t<coordinates>'$COORD'</coordinates>\n' >> $KMLFILENAME
fi

printf '\t\t\t</Point>\n' >> $KMLFILENAME
printf '\t</Placemark>\n' >> $KMLFILENAME
printf '</kml>\n' >> $KMLFILENAME
```

We remove any old KML files and begin with a clean slate. The KML structure used is a basic one with only a link and name. We exctract the GPS location to two variables $COORDLAT and $COORDLONG. We then use an IF statement where these two variables are checked and if they are not empty (the GPS tags are present in the image file and contains coordinates) we concatenate them in a single string and save it to another variable $COORD that is also inserted in the <coordinates> tags in our KML file. If the GPS data is not found in the image file, the user is prompted for manual coordinates input. This workflow ensures a dynamic KML generation keeping the user involvment at mininum.

2.6 Distribution

After all of our files have been generated (the high quality time-lapse video, the flash video and the KML file) an uploading process starts to a designated host. The user is prompted with the password.
Here is the code I've used for moving the data.

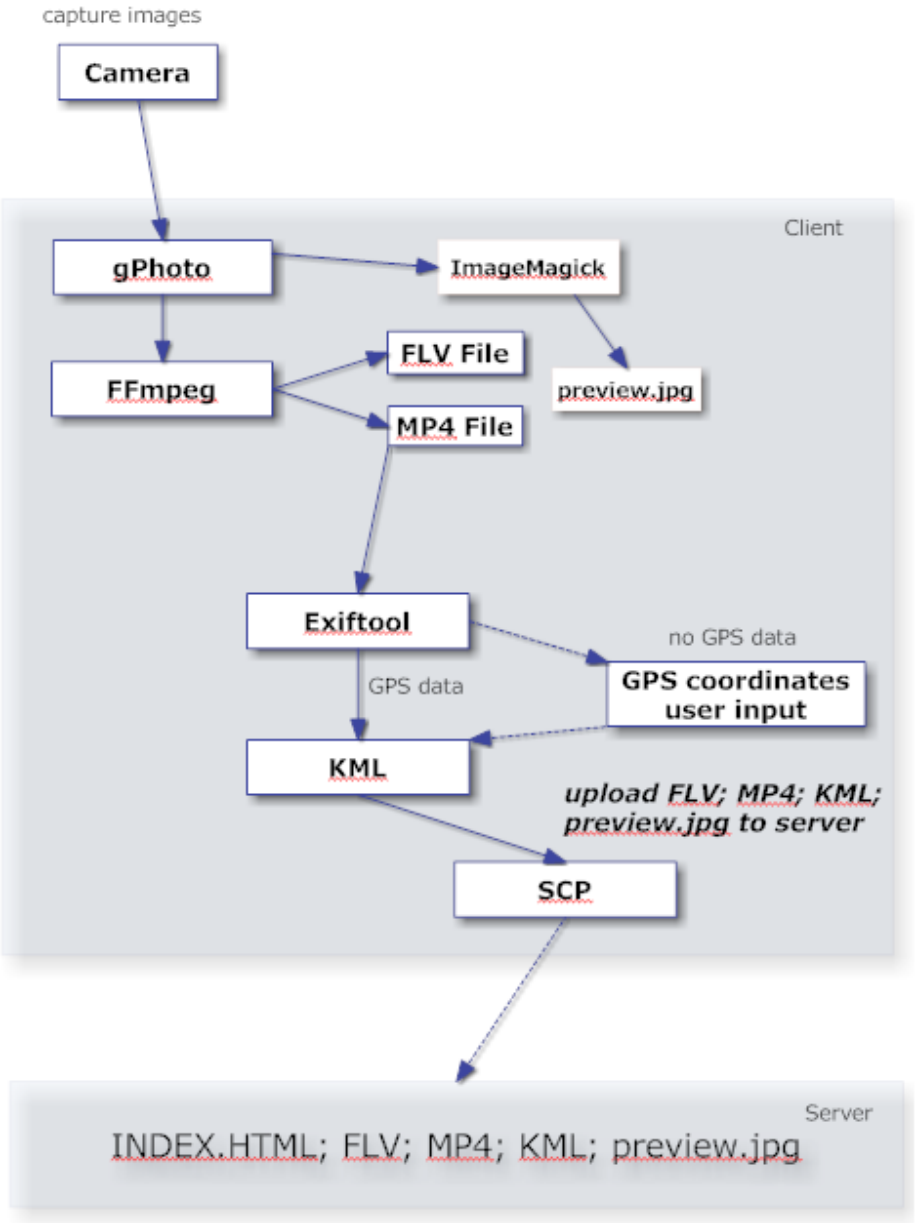
```
echo "\033[1muploading to $REMOTEHOST\033[0m"
scp $MP4NAME $FLVNAME $KMLFILENAME $REMOTEHOSTUNAME@$REMOTEHOST:$REMOTEHOSTDIR
```

We use the SCP[5] protocol for uploading our data to the server. $MP4NAME $FLVNAME $KMLFILENAME represents our files that are being uploaded.
$REMOTEHOSTUNAME our username.
$REMOTEHOST the server that hosts the files.
$REMOTEHOSTDIR and the directory we upload our data in.

2.7 Automation

The automation is an important fact in this project. Certain things can be automated while some user input might be needed (like in GPS location input). It's an ideal thing to let the script do all the work for you and this is what the project has been focused on. Below I have included a flow chart that represents or the automated processes in this project. The only phases that require user input are the phase where a camera doesn't contain a GPS device so coordinate input has to be done manual and when the user is prompted for password for uploading the files to the host server.

[5] `http://en.wikipedia.org/wiki/Secure_copy`


capture images
Camera
Client
gPhoto
ImageMagick
FFmpeg
FLV File
MP4 File
preview.jpg
Exiftool
no GPS data
GPS data
GPS coordinates
user input
KML
upload FLV; MP4; KML;
preview.jpg to server
SCP
Server
INDEX.HTML; FLV; MP4; KML; preview.jpg

Chapter 3

Documentation

3.1 Work log

1. Week 46-47
Setting up a pre-project
Identifying main milestones
Identifying research topics
Problem analysis
Software requirements

2. Week 48-49
Coding the script
Capture shots
Testing different case scenarios
Writing report

3.2 References

http://amwhalen.com/projects/time-lapse-photography http://www.munz.li/?p=48 http://programmer-art.org/articles/tutorials/ffmpeg-time-lapse http://www.flickr.com/photos/armk/3338573106/ http://www.flickr.com/photos/armk/3338573106/ http://www.machinegrid.com/2009/05/controlling-the-nikon-d40-v http://www.moreno.marzolla.name/software/time_lapse_movies/ http://www.imagemagick.org/discourse-server/viewtopic.php?f=1&t=12937 http://justinsomnia.org/2009/08/how-to-make-a-time-lapse-video-with-ffmpeg/ http://rob.opendot.cl/index.php/useful-stuff/ffmpeg-x264-encoding-guide/ http://marc.info/?l=gphoto-devel&m=95576523725934&w=2 http://old.nabble.com/Remote-capture-with-Canon-Digital-Rebel---300D:-success-td23379934.html http://electron.mit.edu/~gsteele/ffmpeg/ https://code.goto10.org/projects/puredyne/wiki/photo-timelapse http://www.sno.phy.queensu.ca/~phil/exiftool/faq.html http://highearthorbit.com/exiftool-is-easy-to-use/

3.3 Code

3.3.1 The shell script

```
#!/bin/sh
##########################################
#IMT4951 Applied Digital Workflow
#TLAP - A time-lapse movie generation project
#Vlad Caia
#CC-BY-NC-SA
##########################################

CAPTUREPATH="/home/vasquez/timelapse/capture"
INTERVAL=10
FRAMES=150
BR=4000k
BRFLV=800k
FPS=25
SIZE=1280x720
SIZEFLV=640x360
MP4NAME="tl.mp4"
FLVNAME="video.flv"
KMLFILENAME="tl.kml"
PREVIEWFILE="preview.jpg"
REMOTEHOST="loke.hig.no"
REMOTEHOSTUNAME="050704"
REMOTEHOSTDIR="public_html"

echo "\033[1mcapture frames\033[0m"
gphoto2 --interval $INTERVAL --frames $FRAMES --capture-image

echo "\033[1mmoving files to $CAPTUREPATH\033[0m"
mv *.jpg $CAPTUREPATH

echo "\033[1mcreating preview image $PREVIEWFILE\033[0m"
convert $CAPTUREPATH/capt0000.jpg -resize 640x320 $PREVIEWFILE

echo "\033[1mtime-lapsing MP4 file to $MP4NAME\033[0m"
ffmpeg -y -f image2 -i $CAPTUREPATH/capt%04d.jpg -vcodec mpeg4 -vb $BR -r $FPS -s $SIZE $MP4NAM

echo "\033[1mtime-lapsing FLV file to $FLVNAME\033[0m"
ffmpeg -y -f image2 -i $CAPTUREPATH/capt%04d.jpg -vb $BRFLV -r $FPS -s $SIZEFLV $FLVNAME

echo "\033[1mwriting to $KMLFILENAME\033[0m"
rm tl.kml

printf '<?kml version="1.0" encoding="UTF-8"?>\n' >> $KMLFILENAME
printf '<kml xmlns="http://www.opengis.net/kml/2.2">\n' >> $KMLFILENAME
printf '\t<Placemark>\n' >> $KMLFILENAME
printf '\t\t<name>Time-lapse project</name>\n' >> $KMLFILENAME
```

```
printf '\t\t<description>\n' >> $KMLFILENAME
printf '\t\t<![CDATA[\n' >> $KMLFILENAME
printf '\t\t<a href="http://www.stud.hig.no/~050704/">
<img src="http://www.stud.hig.no/~050704/preview.jpg">Time-lapse project</a>\n' >> $K
printf '\t\t ]]>\n' >> $KMLFILENAME
printf '\t\t</description>\n' >> $KMLFILENAME
printf '\t\t\t<Point>\n' >> $KMLFILENAME

echo "\033[1mcheckin for gps data in $CAPTUREPATH/capt0000.jpg\033[0m"
COORDLAT=$(exiftool -GPS:GPSLatitude -n -b $CAPTUREPATH/capt0000.jpg)
COORDLONG=$(exiftool -GPS:GPSLongitude -n -b $CAPTUREPATH/capt0000.jpg)

if [ "$COORDLAT" != "" ] && [ "$COORDLONG" != "" ] ; then
COORD=$COORDLONG,$COORDLAT
echo "found GPS Position:" $COORD
printf '\t\t\t\t<coordinates>'$COORD'</coordinates>\n' >> $KMLFILENAME

else
printf "\nInput coordinates data (LAT,LONG): "
read COORD
printf '\t\t\t\t<coordinates>'$COORD'</coordinates>\n' >> $KMLFILENAME
fi

printf '\t\t\t</Point>\n' >> $KMLFILENAME
printf '\t</Placemark>\n' >> $KMLFILENAME
printf '</kml>\n' >> $KMLFILENAME

echo "\033[1muploading to $REMOTEHOST\033[0m"
scp $MP4NAME $FLVNAME $KMLFILENAME $PREVIEWFILE $REMOTEHOSTUNAME@$REMOTEHOST:$REMOTEH
echo "done"
```

3.3.2 The KML file

```
<?kml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
<Placemark>
<name>Time-lapse project</name>
<description>
<![CDATA[
<a href="http://www.stud.hig.no/~050704/">
<img src="http://www.stud.hig.no/~050704/preview.jpg">Time-lapse project</a>
 ]]>
</description>
<Point>
<coordinates>10.677658,60.805263</coordinates>
</Point>
</Placemark>
</kml>
```

3.3.3 The web page source

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" lang="en">
<head>
<title>TLAP - Time-lapse movie generation</title>
</head>
<body>
<p>TLAP - A time-lapse movie generation project</p>
<p>This basic page was setup for the "IMT4951 Applied Digital Workflow" course.</p>
<p>Below you can find a flash player with the time-lapse movie that has been generated by the p

<object id="player" classid="clsid:D27CDB6E-AE6D-11cf-96B8-444553540000" name="player" width="6

<param name="movie" value="player.swf" />
<param name="allowfullscreen" value="true" />
<param name="allowscriptaccess" value="always" />

<param name="flashvars" value="file=video.flv&image=preview.jpg" />
<embed
type="application/x-shockwave-flash"
id="player2"
name="player2"
src="player.swf"
width="640"
height="360"
allowscriptaccess="always"
allowfullscreen="true"
flashvars="file=video.flv&image=preview.jpg"
/>
</object>
<br>
<a href="tl.mp4">Download 720p timelapse</a>
<a href="http://maps.google.com/maps?q=http://www.stud.hig.no/~050704/tl.kml">KML</a>

<br><br><br>
<a rel="license" href="http://creativecommons.org/licenses/by-nc-sa/3.0/no/">
<img alt="Creative Commons License" style="border-width:0"
src="http://i.creativecommons.org/l/by-nc-sa/3.0/no/80x15.png" /></a>.
</body>
</html>
```

Chapter 4

Conclusion and future work

TLAP is a proof of concept for further development that automation can be implemented very easy without any user involvment in its operation. Future work can be derived from this starting point. Some ideas I have would be to create time-lapse video simlutaneously from different locations. Communication from the cameras and the client computer could be done via wifi link and the code itself could be adapted to take care of simultaneous running tasks. One of the ideas I came upon would be taking shots for several days and used in later weather forecast comparison and analysis. The web page could as well be dynamically generated and the entire project build with a DBMS in mind where time-lapse videos are easily organised and accesed.
This project and all the code provided is a proof at what can be achieved with open source software.