

ECE480: Design Team #9 Final Report Video Recording Tool for a Vehicle's Infotainment and Navigation System

Team Members:

Shengzhe Gao
Radhika Somayya
Xinye Ji
Kun Zhang
Yan Gong
Jianying Tang



BOSCH
Invented for life

Table of Contents

Executive Summary	3
Acknowledgments.....	3
Chapter 1: Introduction and Background	4
Chapter 2: Exploring the Solution Space and Selecting a Specific Approach	6
FAST Diagram:	6
Disk Monitor System.....	7
Disk Usage Analyzer	8
Shell Code and c-script.....	8
FFMPEG.....	9
Modifying the Complex Filter	10
Projected Budget:	12
Chapter 3: Technical Description of Work Performed	16
Hardware:	16
Software:	16
Design Efforts	16
Version 1: Getting Started.....	16
Version 2: Improving the Recording.....	17
Chapter 4: Test Data with Proof of Functional Design.....	19
Chapter 5: Final Cost, Schedule, Summary, and Conclusion	21
Appendix 1: Technical roles, Responsibilities, and Work Accomplished.....	23
Radhika Somayya:.....	23
Jianying Tang:	24
Yan Gong:	24
Shengzhe Gao:.....	24
Xinye Ji:	25
Kun Zhang:	25
Appendix 2: Literature and Website References.....	26
Appendix 3: Technical Attachments.....	27

Executive Summary

This tool was developed for Robert Bosch's Car Multimedia Division. Like most companies, this division performs various types of on-road testing of their products. One of the on-road testing that needs to be conducted is to detect bugs in their products. The bugs can be detected just by viewing them. However, while driving, the driver cannot (and should not) record bugs that are occurring in their products.

In order to make their testing easier and more user-friendly, Bosch gave us the project to design a monitoring system with four cameras that had video recording capabilities. The tool that was designed has the ability to record four different video camera outputs and store them into a USB flash drive, solely at the push of a button by the user.

This device also has the ability to concatenate all four video streams into one video file and display that to the user when the flash drive is connected to a computer.

This way, the tool allowed the driver to quickly push the record button when he or she observed a bug. The push of the record button allows the last two minutes of the recording to be stored onto the USB flash drive. When the user presses the stop recording button, the recording is stopped and the file is saved onto the flash drive completely.

Bosch's main requirements were to:

- Have a 6-7 inch display with HDMI input connected to car
- Record two minute chunks of video
- Concatenate all four video streams into one video
- Have a time stamp along with the GPS coordinates on the display

Acknowledgments

We would like to acknowledge Mr. Clark Smalley and Mr. Martin Beeker, at Robert Bosch LLC, for taking time out of every week to speak to us on the progress of our product. With your help, we were able to deliver to you the best working product that you could begin using as soon as possible.

We would also like to thank Dr. Robert McGough for meeting with us every week for the whole semester to give us feedback on how we were going about this project. Your input helped us give a better finished product to our sponsors.

Chapter 1: Introduction and Background

Project 9 for ECE 480 Spring 2014 is sponsored by Robert Bosch LLC. It requires the design group to build a video recording tool for a vehicle's infotainment and navigation system.

This project was originally assigned to a group in fall 2013. The specifications were met; however Bosch wants some improvements done on the tool to make it more user-friendly.

The main problem we are trying to solve is to provide our sponsor with a tool that has all the specifications described by them so that their infotainment/navigation center testing is more accurate and reliable. A better tool than what they use now would allow them to deliver higher quality products to *their* customers in a shorter amount of time.

The design objectives we were given are as follows:

- Have the ability to connect four video cameras to the device and record the video.
- Add a better record and stop trigger. The previous group used a key pad; however Bosch wants something smaller and simpler.
- Add a display approximately 6-7 inches wide with an HDMI port. This display would show the screens of all four video cameras cascaded together. This display would mainly be used to make sure the cameras are aligned properly. It should have high enough video quality to read what is written on the display as well. It should also show time stamp and GPS position logging.
- The four cameras should be positioned to show:
 - Center stacked display
 - Instrument cluster
 - Front view of road
 - Right side view of road
- Start recording the video with the push of a button from (t-2) minutes, where t is the time at which the button is pressed.
- Add a true DVR feature so that it is possible to live record and be able to rewind a certain number of seconds. Previous group only recorded 30 second blocks.
- Notification on screen telling user when to change flash drive.
- A push button to save the video recording into long term memory (USB flash drive).

The past group has a device with many different USB ports, an HDMI output, and a micro SD card with Linux. Our customer at Bosch, Marty, said that we can use this device and just improve it (to be more cost effective).

We believe that this new approach to detect bugs is a lot more efficient than the previous approach. Especially since it can show four video streams at once. It is more beneficial to see all four streams at once; because detecting bugs becomes a lot less gruesome than it was before.

All of the objectives listed above will help Bosch improve their testing process for observing a vehicle's infotainment and navigation system. Currently, the method that Bosch uses to conduct their testing is not cost efficient. The solution we have come up with is a lot less expensive and way more user friendly. This product is also not as much of a safety hazard compared to the product currently being used by Bosch. It is also safer than the design given by the previous team; because we added certain features (explained in Chapter 2) that make recording a lot quicker and more efficient.

Chapter 2: Exploring the Solution Space and Selecting a Specific Approach

FAST Diagram:

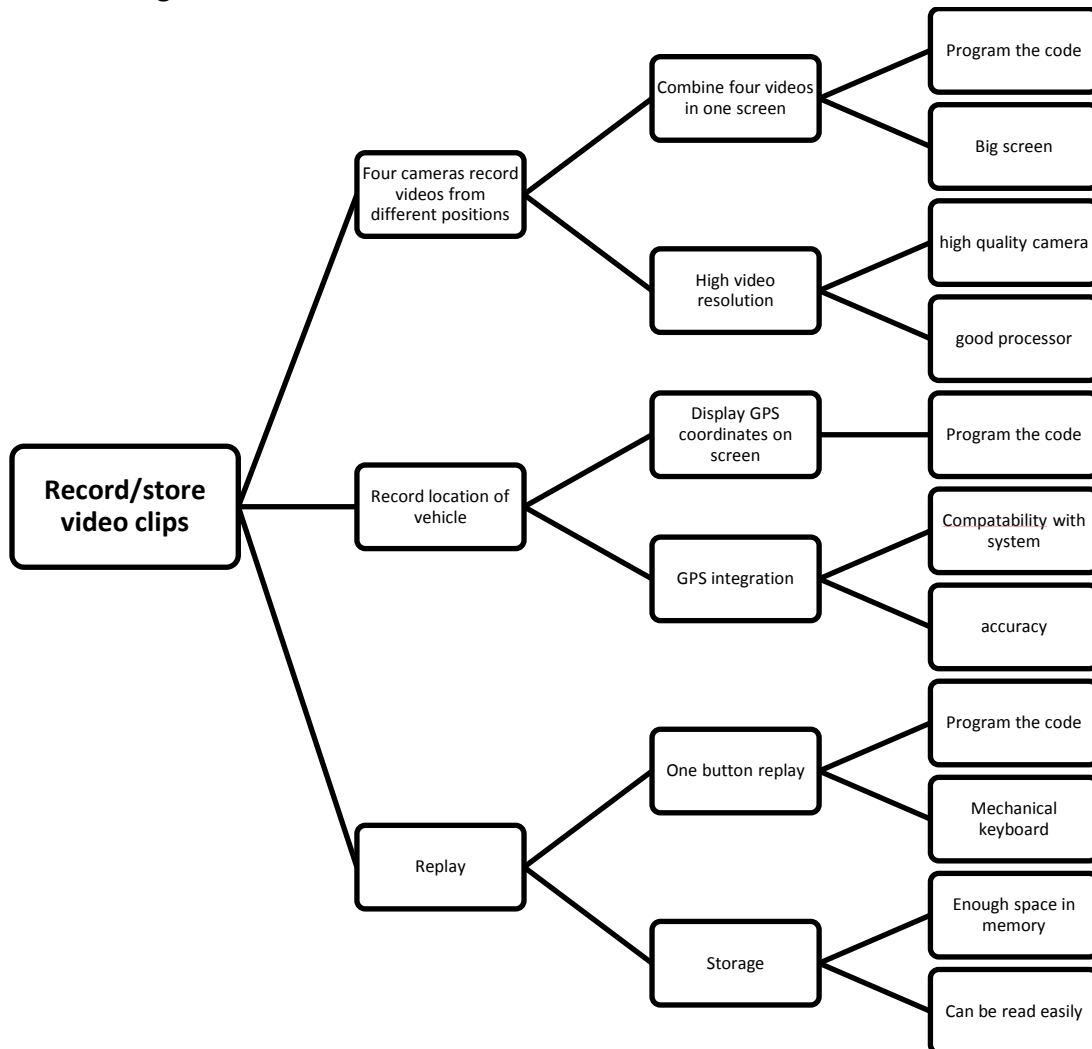


Figure 1

To begin, we decided to achieve our objectives by starting with the display. We attempted to look for the right size display with an HDMI port. After finding one with good reviews, we approved it with our customer and subsequently placed an order for it. We also found a DC/DC step down converter in order to connect the device to the car battery. The converter we selected, which steps down from 12 V to 5 V, was also approved by our customer. With the above all set in place, we placed an order.

The HDMI screen has been used in the system and it turns out it fits all the requirements. The power system which involves the DC to DC converter has been done and can power up both Odroid-XU and screen.

We selected a camera with quality higher than 720P, and also found a switch that would work to turn the recording on and off. The switch would replace the key pad that was used by the previous design team. The camera that we used is from the previous team. However after testing the cameras with different frame rates, (FPS i.e. frames per second), we found a way to improve the video quality that meets the sponsor's requirements. The key switches have been assembled with the Teensy 2.0 control module. The whole design has 8 different buttons that control the system, instead of using the whole keyboard like was used previously.

Our design also has a DVR like function and a memory indicator. The DVR function allows the user to record any time chunk they would like to, with the press of a button. The memory indicator will tell the user how much memory is left on the flash drive where the recordings are being stored. It is also an indicator on when to replace a full flash drive with an empty one to continue the debugging process.

We also came up with an effective and efficient way of storing all the video recordings in a USB flash drive. The design needs to have sufficient capacity to record and store simultaneously. One option, suggested by the sponsor, is to use multiple flash drives. The user can simply plug in a flash drive into our design. Once it is close to running out of memory, the user can simply replace the full flash drive with an empty one.

Also, solely for aesthetics, we made a plastic enclosure with the key switches on the top of the enclosure. This makes the design easier to use in a moving vehicle because the user will not be distracted by bundles of wires lying around. Everything is being stored in the enclosure except one power line and the four cameras.

Most of these features are new to the original testing system. There was no display with HDMI input connected to the car battery. The old system was very slow and not as robust as the new system. These new features will be more successful than the old system because it will make the whole product a lot faster and user-friendly.

Before starting this project, we came up with a projected budget for our design. Knowing that the previous team had already bought most of the components for this project, we knew that we would not be using up a lot of our budget.

Disk Monitor System

One of the requirements is the disk monitor function. Setting up a Disk Usage Monitor can be easy. Using the -df command can fulfill every need, but it is very abstract. The most advanced way is using a shell code, but it is very hard to use and also needs high authority in the system.

```
gong@gong-virtual-machine:~$ df -h
文件系统      容量  已用  可用  已用% 挂载点
/dev/sda1      18G   3.6G   14G   22% /
udev          1000M   4.0K 1000M    1% /dev
tmpfs          403M   768K  402M    1% /run
none           5.0M    0    5.0M    0% /run/lock
none          1007M   152K 1007M    1% /run/shm
/dev/sr0       683M   683M    0  100% /media/Ubuntu precise 20120426-1
```

Figure 2

This command can show all the disk usage, but it is very abstract. You cannot easily see how many disks are left. So we have another command as "df -h." "h" means human readable. However, it still has too much unusable information. So we can put the location behind the "df" command. We will type "df -h /dev/sda1"

```
gong@gong-virtual-machine:~$ df -h /dev/sda1
文件系统      容量  已用  可用  已用% 挂载点
/dev/sda1      18G   3.6G   14G   22% /
```

Figure 3

After all, this is everything that the "df" command can do. If we want more functionality, we need more advanced code.

Disk Usage Analyzer

There is an app that we can download from Ubuntu. It provides real-time usage monitor and analysis. However, it is an app, so we cannot apply it into our own program that easily. This app cannot provide the most important function we need; it cannot provide a warning to the user when disk space is full. So we looked for another solution to our problem.

Shell Code and c-script

The C language is supported by Linux. However, it has low authority level. We ran into the problem of having to type a password each time the program was run. It was definitely a safety hazard and not acceptable. The solution was to upgrade to a high authority version. This can be provided by shell script.

One of the simplest ways to back up a system is using a shell script. For example, a script can be used to configure which directories to backup, and use those directories as arguments to the tar utility, creating an archive file. The archive file can then be moved or copied to another location. The archive can also be created on a remote file system such as an NFS mount.

The tar utility creates one archive file out of many files or directories. *Tar* can also filter the files through compression utilities reducing the size of the archive file.

Shell script can allow the system to run in the background, automatically.

After testing those three solutions, the answer seems very clear; shell script is the best answer. However, after a long discussion, we decided to use the Disk Usage Analyzer as the final solution. Although the shell script could provide the most accurate results, it would be the most gruesome to use. Considering the complexity and stability of the shell script, we decided to use the disk usage option instead.

FFMPEG

The solution of the DVR function was very clear, FFMPEG. There was already some code on FFMPEG from the previous team, so our main focus was improvement. To improve the settings we used the shell code. In order to change certain settings while recording a live stream with FFMPEG, this base shell script must be changed (shown in figure 4).

```
17 ▼ ffmpeg \
18     -f video4linux2 -r 10 -vcodec mjpeg -i /dev/video0 \
19     -f video4linux2 -r 10 -vcodec mjpeg -i /dev/video1 \
20     -f video4linux2 -r 10 -vcodec mjpeg -i /dev/video2 \
21     -f video4linux2 -r 10 -vcodec mjpeg -i /dev/video3 \
22     -filter_complex \
23     "nullsrc=size=640x480 [base];
24     [0:v] setpts=PTS-STARTPTS, scale=320x240 [upperleft];
25     [1:v] setpts=PTS-STARTPTS, scale=320x240 [upperright];
26     [2:v] setpts=PTS-STARTPTS, scale=320x240 [lowerleft];
27     [3:v] setpts=PTS-STARTPTS, scale=320x240 [lowerright];
28     [base][upperleft] overlay=shortest=1 [tmp1];
29     [tmp1][upperright] overlay=shortest=1:x=320[tmp2];
30     [tmp2][lowerleft] overlay=shortest=1:y=240[tmp3];
31     [tmp3][lowerright] overlay=shortest=1:x=320:y=240" \
32     -q:v 4 -y -t 5 temp.mp4
```

Figure 4

Line 17: ffmpeg – tells the terminal that ffmpeg is being used.

Lines 18-21: Defines the input feeds and the settings of each feed below.

- Video4linux2 is an open source package that allows the machine to have a live feed of the webcam
- -r 10 refers to the frame rate (in frames per second)
- -vcodec refers to the video codec that the stream is encoded in
- -i refers to the input source

Lines 22-31: Complex Filters – This allows the user to add either color filters or overlays.

- filter_complex: to implement filters. This is formatted and constructed in the quoted text (highlighted yellow)
- Line 23: This refers to the base resolution of the whole video. [base] can be any name.
- Lines 24-27: Instantiating the input feeds
 - [n:v] : The numbering of the inputs, in this case 0-3
 - setpts : sets the time point. In this case, PTS-STARTPTS refers to 0, ensuring that all video streams are synchronized.
 - Scale= is the individual resolution of each respective feed.
 - The last part refers to the name of this input. It can be labeled whatever is needed. In this case, it labels the four corners.
- Lines 28-31: Calls the base layer and immediately stacks the input feeds in between temporary layers.

- Overlay=shortest=1 enables the video to end as soon as the shortest video feed ends.
- The X and Y calls simply refer to the upper left corner of each feed. If x = 320 and y = 240, that means that the upper left corner of a video feed is located at 320 by 240.

Line 32: Settings for the output video

- -q:v – refers to the video quality. This number can range from 1-32; one (1) being the highest quality, thirty-two (32) being the lowest quality.
 - The higher the quality, the larger the file size.
- -y – This enables the script to overwrite a file.
- -t – Sets the time in seconds. Additionally, 00:00:00.00 can also be inputted.

temp.mp4 – The output file name, this can be named everything.

Modifying the Complex Filter

Being able to modify the filter is pivotal to custom tailoring the live feeds to your needs. At first, the feed had a layout similar to the one shown in figure 5.

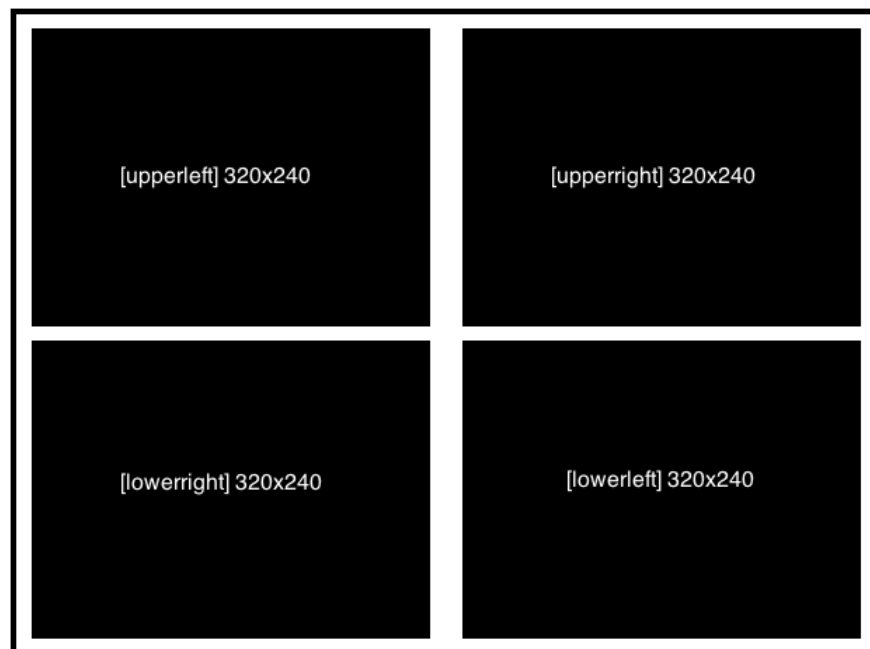


Figure 5

When we were improving the solution, a problem came up: What if one camera feed needs more detail than the rest? For a successful product,, every situation must be considered. So we came up with an answer - expanding that view and cropping the others respectively.

One way to do this is is shown in figure 6.

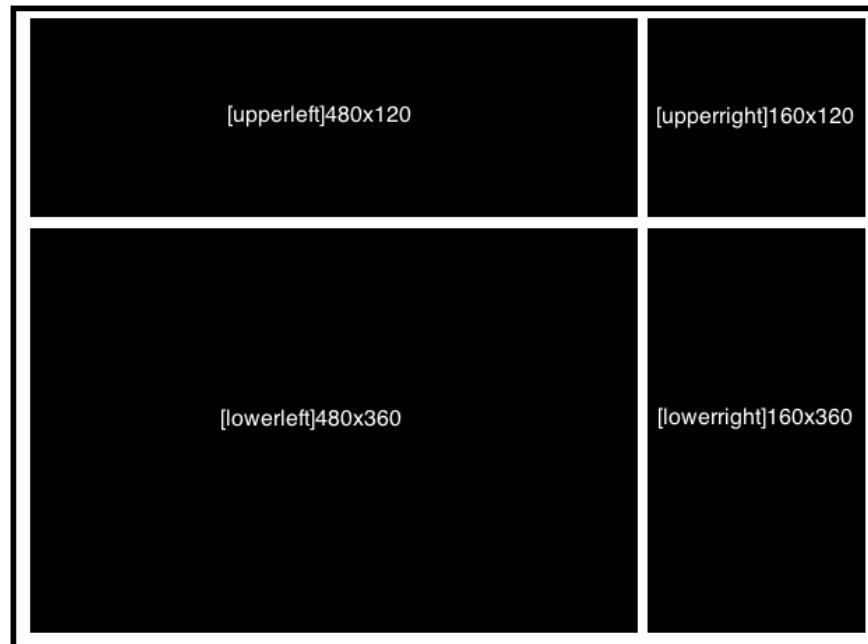


Figure 6

While this isn't the most efficient method, it serves the purposes of the tutorial well.

Step 1: Setting the base resolution

The total resolution of the video can be changed here. In this situation, do not modify the highlighted value.

```
"nullsrc=size=640x480 [base];
[0:v] setpts=PTS-STARTPTS, scale=320x240 [upperleft];
[1:v] setpts=PTS-STARTPTS, scale=320x240 [upperright];
[2:v] setpts=PTS-STARTPTS, scale=320x240 [lowerleft];
[3:v] setpts=PTS-STARTPTS, scale=320x240 [lowerright];
[base][upperleft] overlay=shortest=1 [tmp1];
[tmp1][upperright] overlay=shortest=1:x=320[tmp2];
[tmp2][lowerleft] overlay=shortest=1:y=240[tmp3];
[tmp3][lowerright] overlay=shortest=1:x=320:y=240" \
```

Figure 7

Step 2: Resizing the individual feeds.

These values need to be changed accordingly:

```
"nullsrc=size=640x480 [base];
[0:v] setpts=PTS-STARTPTS, scale=320x240 [upperleft];
[1:v] setpts=PTS-STARTPTS, scale=320x240 [upperright];
[2:v] setpts=PTS-STARTPTS, scale=320x240 [lowerleft];
[3:v] setpts=PTS-STARTPTS, scale=320x240 [lowerright];
[base][upperleft] overlay=shortest=1 [tmp1];
[tmp1][upperright] overlay=shortest=1:x=320[tmp2];
[tmp2][lowerleft] overlay=shortest=1:y=240[tmp3];
[tmp3][lowerright] overlay=shortest=1:x=320:y=240" \
```

Figure 8

Upper left: 480x120

Upper right: 160x120

Lower left: 480x360

Lower right: 160x360

Step 3: As this script stands, this output will not be positioned tightly, there will be odd black space between the feeds, as such we need to change the x and y coordinates of each feed.

Upper left: No changes needed, begins at (0, 0)

Upper right: (480, 0), x must be set to 480 instead of 320.

Lower left: (0, 160), y must be set to 160 instead of 240.

Lower right: (480, 360), x must be 480, and y must be 360.

By using FFMPEG, we can provide a reliable solution for the DVR function. The film will record using four cameras and the view will input to Odroid. The Odroid will process the view input in post-production, and then output to screen. The view will be a integration of the four cameras' input images.

Another improvement we decided to do was the packaging of the product. The main reason to do this was because of the safety hazard caused by having multiple wires hanging loosely in a moving vehicle. This could pose a number of safety threats. Exposed electrical components are never okay in a moving vehicle.

We decided to design an enclosure for our whole design. Another improvement was adding button switches. This allows the user to quickly start and stop recording in a moving vehicle while spending the least amount of time with their hands off the steering wheel.

Projected Budget:

Ordered Parts	Quantity	Cost	Total
Koolertron® 7 inch 16:9 HD TFT LCD	1	159.99	159.99
DROK SR Waterproof DC Buck Converter	1	15.62	15.62
Button	8	1.18	9.44
Packaging box	2	4.99	9.98
Total Cost		195.03	

Table 1

The parts we received from the previous team are as follows:

PART	Quantity	Cost	Total
Odroid-Xu	1	\$169.00	\$169.00
Odroid RTC Battery	1	\$8.00	\$8.00
8GB Micro SD w/ Android Installed	1	\$13.00	\$13.00
Transcend 32 GB microSD HC Class 10 UHS-1	1	\$22.99	\$22.99
Kingston 32 GB microSD card HC class 4	1	\$26.99	\$26.99
Logitech C270 Camera	1	\$24.99	\$24.99
Logitech C270 Camera	1	\$32.23	\$32.23
Logitech C310 Camera	2	\$29.99	\$59.98
Microsoft USB GPS	1	\$47.04	\$47.04
USB Number Pad	1	\$12.56	\$12.56
MicroHDMI - HDMI Cable	1	\$9.00	\$9.00
Belkin USB 2.0 4-Port Hub	1	\$6.99	\$6.99
Total Shipping		\$30.00	\$30.00
Total Cost			\$462.77

Table 2

Given the critical customer requirements (CCRs) of this project, we as a team had to decipher all the requirements and determine the most feasible method of creating a successful project. The device has to be capable of recording, compiling, and saving four video streams into one. We ranked the importance of each of the specifications given to us and tackled the whole project in the order of priority. Table 3 shows the ranks we gave to each component of our design. We also had to determine how our approach to each requirement was different from the past, and why.

We realized that each requirement was a little different from what Bosch used previously. We were positive that the methods we were using to tackle each requirement would be successful. Especially considering that we knew what the result of having unsuccessful objectives was (from the previous version of this design).

A House of Quality table ranking all of our requirements and its respective importance is shown in table 3 below.

Level Critical customer requirements	Very Important	Important	Neutral	Technical importance level/Reasoning
DVR Function	☆			Very basic function of the system.
Disk monitor	☆			The one of main requirements
GPS		☆		Show's the location of the user
Screen	☆			Monitor the system's working situation
Power supply		☆		Basic structure of the system
Packaging			☆	Protection
Button	☆			Simplify user operate
Camera quality		☆		
Interface Design			☆	
Storage		☆		

Table 3

Further, we have included our GANTT chart on the following page, showing our design schedule from week 4 onwards until design day.

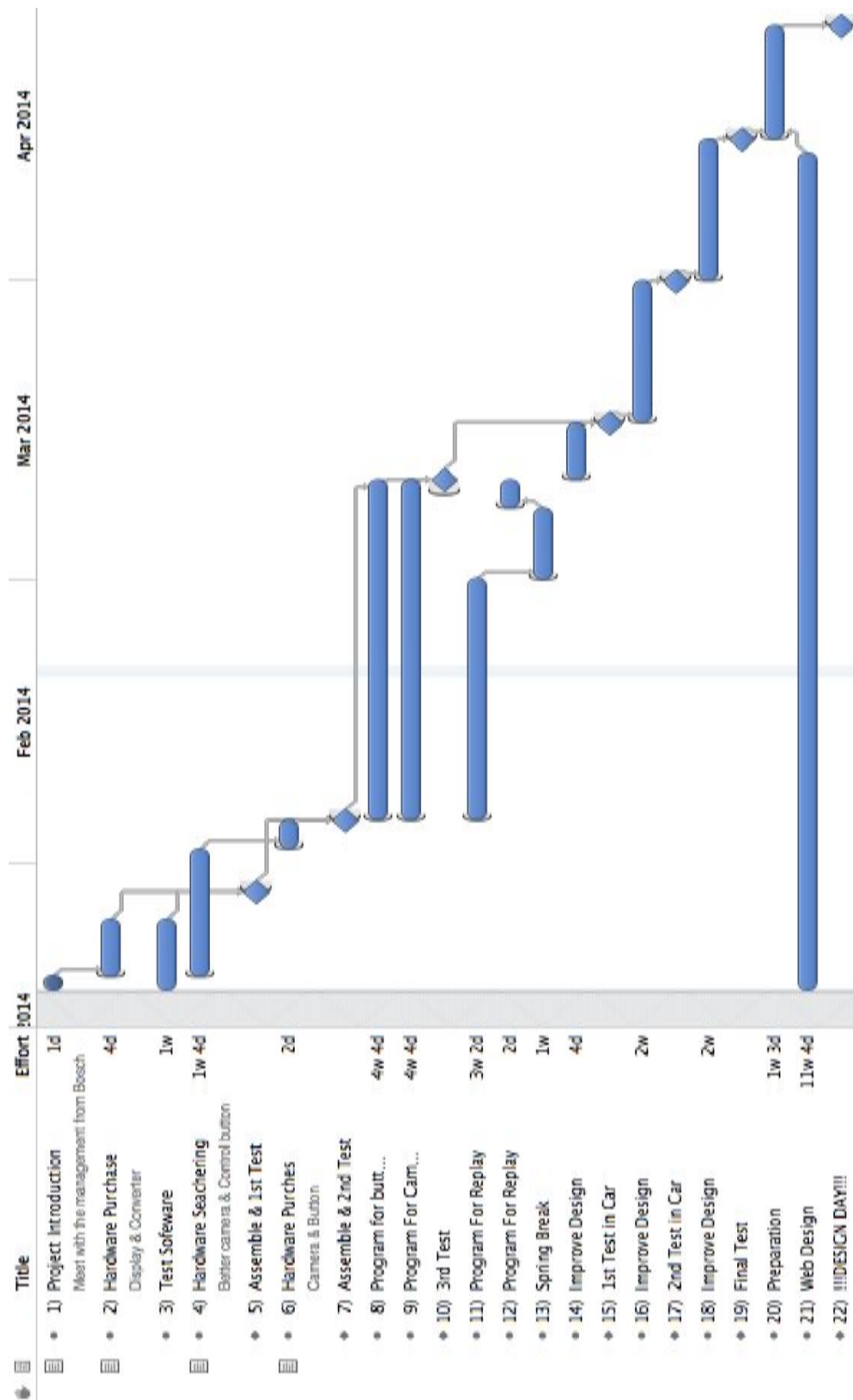


Figure 9

Chapter 3: Technical Description of Work Performed

Due to the fact that we were continuing a project from last semester, certain implementations were determined from the previous team. As such, much of what we did involved improving the design and functionality of the current product.

Hardware:

Design Efforts and Implementation

Certain components of the design, such as the microprocessor (Odroid) and video cameras (Logitech C310 and C270), were received from the previous team's implementation of the design. Hardware design decisions the team had to make revolved mostly around setting up a user interface for the end user.

Components we added to the design include:

- **Teensy 2.0 Microcontroller:** this was used to design a control panel for the design. The Teensy 2.0 was chosen due to the large amount of documentation surrounding the platform. (Arduino)
- **Cherry MX Switches:** these switches were used to implement the various controls needed to program; which include, start recording, stop recording, 2 minute DVR, 5 minute DVR, DVR output. These switches were chosen due to their high actuation rate and tactile response.
- **12V/5V converter:** this component was needed to convert power from the car power supply to the microcontroller and monitor.
- **Casing for the recording module:** with the addition of components, a case to contain all the components was necessary for the aesthetics of the module.

Software:

Much of what was left to improve on the project involved software design.

Design Efforts

Video quality and DVR functionality were the primary concerns of the project. However, due to lack of documentation in the code written by the previous team, we ran into certain issues that could have been avoided. These issues include: building programs needed in ARM versions of Linux, version compatibility issues with FFMPEG, consistency of camera start up at the beginning of the FFMPEG script, and the design of the FFMPEG code.

Version 1: Getting Started

Initially, the team came up with an approach to record with the four cameras in parallel. (Using four separate FFMPEG commands) Additionally, as FFMPEG does not natively support any DVR functionality, this was the main concern of the team. Initially, the DVR function was planned to capture the last 5 minutes of recorded video.

The initial application worked by recording 1 minute intervals of video using four separate commands. At the end of each interval, stitch all four videos into one FFMPEG command. When the user wants to generate a DVR video, concatenate the previous five 1-minute videos together and save it to the flash drive.

As one may imagine, this was not the most efficient method to approach the DVR function. At this point in time, the user controls were not yet implemented.

Version 2: Improving the Recording

In this version of the code, one of the main issues was streamlining the recording process. Recording each video feed individually and then stitching the individual feeds into one video added unnecessary computational complexity. Additionally, this caused issues with synchronization of the four videos. Each video could stray up to 10 seconds apart.

Instead, the FFMPEG script was changed to stitch together the four video feeds while they were still recording. While this does add some more stress to the recording thread, this is less computationally complex overall, as it circumvents an unnecessary write and then read to and from the disk.

The DVR function and user controls had not underwent any change.

Implementation

The final version of the software worked as such:

- DVR Functionality – between version 2 and the final version of the software portion of the design, some changes in the functionality was made. DVR would work by first generating a time stamp when the end user would press a button. The button would either generate a time stamp from two minutes ago or five minutes ago depending on which option was chosen. After a certain amount of time a second button would be pressed. This button will generate the video $x + n$ minutes long, where x is either two or five minutes, and n is the time in between the first and second button press.
- Recording – the video settings are recorded as such:
 - 10 Frames per second – this low frame rate can be attributed to the sheer computational complexity of recording and encoding video at the same time.
 - Mjpeg encoding format – mjpeg was chose due to bandwidth limits in USB ports. Each set of USB ports can only support up to 900 Mb/s. With two cameras inputting raw video requiring roughly 960Mb/s, recording raw video was not possible in the design.
 - Multi-threading – due to how computationally expensive video recording and encoding is, it seemed natural to implement multi-threading in the recording algorithm. While this didn't necessarily improve performance, it helped with recording consistency. (The recording tended to crash occasionally on only one thread.)
 - Time stamp subtitles are included in the video.

- User Controls – This was done on the Arduino based teensy 2.0 microcontroller.
The controls were done as such:
 1. Start Recording: `./rec.sh`
 2. Stop Recording: `q`
 3. 2 minute DVR: `Alt + Shift + T, python 2minutes.py, exit`
 4. 5 minute DVR: `Alt + Shift + T, python 2minutes.py, exit`
 5. Finish DVR: `q \n ./dvr.sh`

Chapter 4: Test Data with Proof of Functional Design

Our team has tested our prototype in the ECE480 design lab using the 7" display (shown in figure 10), the most crucial part of our design. Since we were not testing in a vehicle at this point in time, we used a 12V/5V power converter to supply the power for the display.

The procedure to connect our whole design is pretty simple (shown in figure 11). Connect the four C310 Logitech cameras to the Odroid chip through USB hubs. The next thing is to connect the eight button switches that have been welded together with the microcontroller to Odroid chip. The GPS and USB memory stick have to connect to the chip as well. The 5V/4A DC converter, on the other hand, has to be connected to the cigarette lighter 12V/10A in order to step down the voltage from 12V to 5V to satisfy the power requirement. Moreover, the 7 inch screen with 12V/2A will be connected to the Odroid chip directly and can be powered up with 12V/10A from cigarette lighter. This is the overall layout of how the team is build up for testing the product at the end.

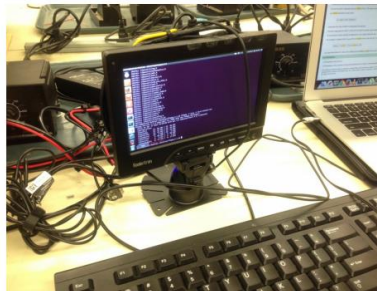


Figure 10



Figure 11

The power converter is plugged into the cigarette lighter. When the power is turned on, the light on the Odroid chip will turn green, which represents that the computing unit is fully functioning. After you connect the cigarette lighter with the 7 inch screen, the display screen will boot up within less than 30 seconds and the Ubuntu terminal will be called in the background to run the DVR function.

As you type out the command script, you have to ask the program to implement the recording process. The terminal will check on the status whether or not the four cameras have been connected. If everything has been setup correctly, the FFMPEG will be running continuously for 2 minute chunks or 5 minutes chunks of video shown in figure 1. Then, the output video file will be stored into the USB memory sticks. You can see the .mkv video file on a computer. The quality of the output video file is clearer than before. As you can see in figure 3, the video is recorded as four streams merged concurrently. The background processing stops in less than ten seconds after recording has stopped. In order to test the resolution of the camera, we placed a stopwatch in front of one of camera, to see how clear the video turns out after compiling. The result turned out to be that the number on the watch can be clearly seen (as shown in left upper corner of figure 12), which satisfies the requirement from the sponsor. On the other hand, the monitor system will always be recording in the background until you press the stop button.



Figure 12

Almost all of the requirements given from the sponsor were met. The sponsor wanted the display to show GPS coordinates. This was a function of the product that was working from last semester so we were told was working correctly. After carefully observing the data, we realized that the GPS code was not working as well as it should. However, at that point in time, it was not an important requirement of the project. So we focused on the rest of the design and left the GPS as it was working before.

Chapter 5: Final Cost, Schedule, Summary, and Conclusion

Our final prototype had many improvements in hardware and software compared to the previous prototype delivered to Bosch by the previous team. We met almost all of the requirements given to us by our Sponsor; however one to two functions were not accomplished due to time constraints.

For hardware, we added a 7" TFT LCD Touch VGA Monitor with HDMI input. This display has a resolution range from 800x480 to 1920x1440. A new power supply was also designed. This power supply has the ability to be powered by the 12V cigarette lighter in a vehicle.

Further, in order to control the display, we have added eight switches by using the Cherry MX mechanical key switches. All the switches are connected to a programmed teensy board. We also designed an enclosure for our whole design so that it is more presentable than just a bundle of wires.

For the software, one of the main improvements was the operating system. We replaced the previous system with a lighter and faster system. A true DVR function is also added to our design.

Our system is always recording in the background, and allows the user to save up to five minutes of video by pressing one of the key switches we added. Also, the design can record up to four streams of video and merge them concurrently into one video. We added a feature to allow the user to view the image of each video camera separately, at the push of a button. This helps the user position the cameras to their liking. Further, when the battery is on, the recording device is ready to use. The quality of the video was also improved from the previous prototype.

There are, however, a few functions that need to be tweaked and fixed in the future. A user HMI needs to be designed for the screen to show the video file name, current state, and present state of the memory remaining. Some minor bugs may need to be fixed in the future. This project does not need another team to work on it; however it may need a computer science engineer to work on the minor improvements. It would be beneficial if that computer science engineer was good at interface design. The minor tweaks and improvements could easily be done within one month's time.

The final costs of the design are as follows:

PART	QUANTITY	COST
Kollertron 7inch screen	1	\$154.99
Quick splice connector	1	\$12.04
Coleman cable 14-gauge	1	\$6.59
12V/5A lighted On/off Plug	1	\$5.93
Male Jack DC power adapter	1	\$4.88
Ac to dc adapter car cigarette lighter	1	\$14.99
Cherry MX key switch	12	\$18.00
Key Cap	8	\$9.99

3M Double Side Tape	1	\$4.99
Super Glue	2	\$4.99
Plastic Box	2	\$10.00
Total Cost		\$247.39

Table 4

Since the last team has already bought the main components for this project. Our final prototype cost was a lot less than what our budget allowed for. Our total cost is \$247.39.

Appendix 1: Technical roles, Responsibilities, and Work Accomplished

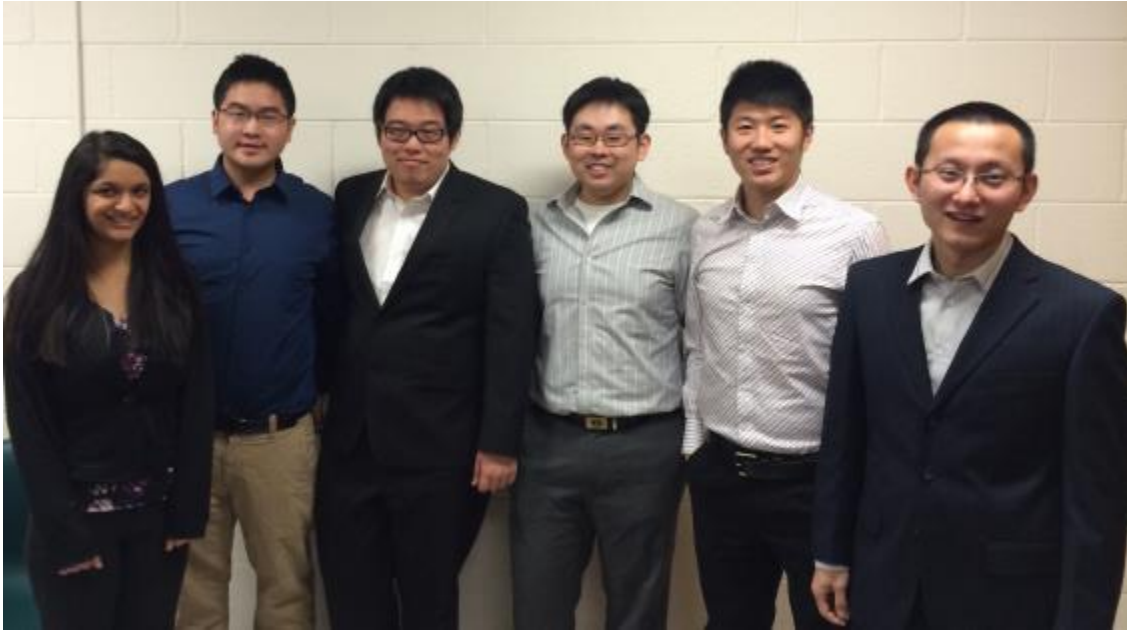


Figure 13 - Left to Right: Radhika Somayya, Jianying Tang, Yan Gong, Xinye Ji, Shengzhe Gao, Kun Zhang

Radhika Somayya:

Radhika's focus in the technical aspect of this project was to understand how to design an enclosure for our whole design. At first, the best solution was to design the whole enclosure in computer-aided design (CAD) software. This required a lot of research on how to construct basic faces on the Siemens software, NX 8.5. Once she learned the basics, she had to measure out our whole design and come up with a drawing of how to enclose our whole project into a 'box' of some sort. She had to make sure the design was reliable and had vents for the fan on our microcontroller. After designing this whole enclosure on NX 8.5, Radhika came to realize that the whole enclosure was not a feasible design for the 3D printers available. It would have taken way too long to construct. After considering various different options, we as a group decided to go ahead and look for enclosures sold somewhere else and alter them to fit our design. Radhika travelled to a few different hardware stores. She settled on one and analyzed what kind of enclosures they had. Liking what was available, the team placed an order. The team as a whole worked together to alter the enclosures that were bought to fit the design dimensions we had.

Jianying Tang:

Jianying Tang has contributed to the hardware design, team website, and certain aspects of software in this project. He helped the team in researching on buttons switches components and determining which kind of microcontroller and button switches would be most compatible with the Odroid chip. He also researched on how to implement them with online instructions. He was active in ordering parts for the team. The most important components such as the Teensy ++ USB development board with pins and five Cherry MX button switches were all ordered by him. He also helped the team in researching in the library on material that would be best to program the Teensy Microcontroller. He found out about some useful software such as Arduino modules and teensy loader applications, etc. Moreover, with respect to development of software, he helped the team in researching which media player is best compatible with .mkv output files generated by FFMPEG. He compared a series of media players and came up with VLC media player as the first choice to play .mkv video files. In addition, he also spent most of his time researching the programming with Ubuntu to figure out the user HMI issues we had. Such as showing the current state of recording and the function that would generate a video file name after the recording process is finished. Lastly, he created the team website by using adobe muse and the URL address is located at <http://www.egr.msu.edu/classes/ece480/capstone/spring14/group09/>

The website contains all the information about what our team project is, the resources the team provided, and the team member profiles as well as some pictures and videos of our presentations.

Yan Gong:

The main area of study for Yan is the disk monitor system. He used a three-way approach to obtain to the goal - command, script, and application. Although the shell script could give the team more advanced functions, using a mature application is the most suitable option for the project. He also helped the hardware design and was involved in the hardware testing and package assembling. He also focused on the GPS portion of the design. Working with Xinye, they overthrew the old GPS design from the last team, and came up with a new idea for GPS. Yan also helped with the button design. Although his idea was not used in the final product, it was still a valuable contribution in obtaining the final finished product we have.

Shengzhe Gao:

Shengzhe's technical role is related to both hardware and software. He completed three jobs, programming of the teensy micro controller, building the switch control, and designing the whole circuit of the project. He built two circuits related to the switches. The first circuit is built for the SPDT switches and the second one is for Cherry MX key switches. Then he chose the teensy board as a micro controller to control the eight switches. He did a lot of research on how to learn how to write the program on teensy board. After learning about the features of the teensy, he realized there are two

programs have been created to control the SPDT switches and Cherry MX key switches. When he tested the two switch controls in person, he decided to use the Cherry MX key switches instead of the SPDT switches. Then he welded the eight switches with the teensy board. He also designed the whole circuit of the project. Since Kun designed the power supply circuit. He combined the power supply circuit, switch control circuit, and the four cameras USB wires to one circuit so that it can be attached on the back side of the screen. He designed the whole circuit to be as small as possible so it can properly fit in the plastic case, which Radhika has built.

Xinye Ji:

Xinye Ji worked primarily on the software development of the project. Most of what Xinye has done has revolved around researching and familiarizing the group with FFMPEG, doing research on bash scripts, and brushing up on python. Additionally, he implemented the DVR function (Done in combination with shell scripts and python), as well as rewrote the whole recording function (done with FFMPEG based shell scripts). Much of his time was spent around testing and modifying the recording function. Over the course of the semester, not only has the computational complexity of this function has gone down, but the quality of the output has also increased quite significantly.

Kun Zhang:

Kun Zhang is the lab coordinator in the team. He contributed in both the hardware and software part of the project. On the hardware side, the power supply has been constructed by Kun. The original power supply drew power from an 110V outlet which did not meet the design requirements of the sponsor (Robert Bosch LLC). They preferred the power to come from a car cigarette power. Kun was in charge of selecting, ordering, and assembling the connection wire with appropriate gauge, T cap connector, 12V cigarette power simulator, 2.1x5.5 mm male adapter, 12V cigarette plug, and 12V to 5V step down converter. The power system was designed and tested by Kun to make sure it functioned as it was designed to function. All the power is drawn with one single power line from the car cigarette power 12V and powers up the Odroid-XU and 7 inch screen. On the software side, Kun was in charge of burning the Ubuntu system into the Micro SD card and testing different versions of Ubuntu to find out which one meets the requirements of our cameras. He also looked out for the best boot up time version with the full camera driver function. He decided on the 12.04 version instead of 13.04 version of Ubuntu, which has a boot time 70 seconds faster than the 13.04 version. He also made sure the version was compatible with the four Logitech cameras. On the system setting part, he did research on how to enable auto login and shutdown system without a password. He replaced the system file that related to these functions with new code and root permission for 12.04 version Ubuntu system. The pseudo code was used by Kun to gain the root permission to replace and change the code from the system file.

Appendix 2: Literature and Website References

"Andrewault.net." *Andrewaultnet*. N.p., n.d. Web. Mar. 2014.

"How Can I Make Shutdown Not Require Admin Password?" *Administration*. N.p., n.d. Web. Apr. 2014.

"How to Tweak Autologin in Ubuntu and Linux Mint." *How to Tweak Autologin in Ubuntu and Linux Mint*. N.p., n.d. Web. Feb. 2014.

"Linux: Trigger a Real-time Alarm on a Low Disk Space Condition." *Filesystems*. N.p., n.d. Web. Mar. 2014.

"Logitech - HD Webcam C310." *Logitech - HD Webcam C310*. N.p., n.d. Web. Jan. 2014.

"MX1A-E1NW." *Cherry*. N.p., n.d. Web. Mar. 2014.

"Step-by-step Ubuntu SD Card Setup -- for Windows Users." - *Odroid US*. N.p., n.d. Web. Feb. 2014.

"Teensy USB Development Board." *Teensy USB Development Board*. N.p., n.d. Web. Jan. 2014.

"Ubuntu (operating System)." *Wikipedia*. Wikimedia Foundation, Mar. 2014. Web. Mar. 2014.

"UNIX Get An Alert When Disk Is Full." *Linux Unix Tutorial for Beginners and Advanced Users NixCraft RSS*. N.p., n.d. Web. Mar. 2014.

"Virtual Machine Setup." N.p., n.d. Web. Feb. 2014.

"VLC Media Player." *VideoLAN*. N.p., n.d. Web. Feb. 2014.

"VLC Media Player Shortcuts ." *51 Useful Keyboard Shortcuts for VLC Media Player* . N.p., n.d. Web. Mar. 2014.

"VMware Workstation." *Wikipedia*. Wikimedia Foundation, Feb. 2014. Web. Apr. 2014.

Appendix 3: Technical Attachments

The C-script for the disk monitor is shown below:

```
#include <stdio.h>
#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include <unistd.h>
#include <errno.h>
#include <limits.h>
#include <linux/fanotify.h>
#include <sys/stat.h>
#include <sys/statvfs.h>
#include <fcntl.h>

int main(const int argc, const char **argv) {
    if (argc < 4) {
        fprintf(stderr, "Supply a path to a file on the mountpoint to
listen to, a monitor file and a free %% threshold..\n");
        exit(1);
    }

    if (access(argv[1], R_OK) < 0) {
        fprintf(stderr, "Unable to read file: %s\n", strerror(errno));
        exit(1);
    }

    int len, rc;
    unsigned char donestat = 0, alerted = 0;
    const char *path = argv[1];
    const char *monpath = argv[2];
    int threshold = atoi(argv[3]);
    char buf[4096];
    struct fanotify_event_metadata *fem = NULL;
    int fan_fd = -1;
    uint64_t mask = FAN_CLOSE_WRITE;
    struct statvfs sfs;
    float bfree;

    memset(&sfs, 0, sizeof(sfs));
    unlink(monpath);

    if (threshold <= 0 || threshold >= 100) {
```

```

        fprintf(stderr, "Incorrect threshold provided");
        rc = 1;
        goto end;
    }

    fan_fd = fanotify_init(FAN_CLASS_NOTIF, FAN_CLOEXEC);
    if (fan_fd < 0) {
        perror("fanotify_init");
        rc = 1;
        goto end;
    }

    rc = fanotify_mark(fan_fd, FAN_MARK_ADD|FAN_MARK_MOUNT, mask,
AT_FDCWD, path);
    if (rc < 0) {
        perror("fanotify_mark");
        rc = 1;
        goto end;
    }

    while ((len = read(fan_fd, buf, sizeof(buf))) > 0) {
        fem = (void *)buf;
        donestat = 0;

        while (FAN_EVENT_OK(fem, len)) {
            if (fem->vers < 2) {
                fprintf(stderr, "fanotify is too old\n");
                goto end;
            }

            if (!donestat) {
                rc = fstatvfs(fem->fd, &sfs);
                if (rc < 0) {
                    perror("fstatvfs");
                    rc = 1;
                    goto end;
                }
                bfree = 100 - (((float)(sfs.f_blocks - ((sfs.f_blocks
- sfs.f_bfree)))) / (float)(sfs.f_blocks)) * 100);
                if ((bfree < (float)threshold)) {
                    if (!alerted) {
                        creat(monpath, S_IRUSR|S_IWUSR);
                        alerted = 1;
                    }
                }
            }
            else {

```

```

        if (alerted) {
            unlink(monpath);
            alerted = 0;
        }
    }
}
donestat = 1;
close(fem->fd);
fem = FAN_EVENT_NEXT(fem, len);
}
}
if (len < 0) {
    perror("Read fan_fd");
    rc = 1;
    goto end;
}
end:
    close(fan_fd);
    exit(rc);
}

```

The Shell code for the disk monitor is shown below:

```

#!/bin/bash
# Tested Under FreeBSD and OS X
FS="/usr"
THRESHOLD=90
OUTPUT=$(LC_ALL=C df -P ${FS})
CURRENT=$(echo ${OUTPUT[11]} | sed 's/%//')
[ $CURRENT -gt $THRESHOLD ] && /usr/bin/zenity --warning --text="The disk $FS
($CURRENT% used) is almost full. Delete some files or add a new disk." --
title="df Warning"

```