**Advanced Digital Smart**

**Mirror Interface**

**CONTENTS**

**Chapter Title Page**

Table of Contents i

Abstract 7

1. Introduction 8
   1. Theoretical Background 8
   2. Aim of the proposed Work 9
   3. Methodology Adopted 10
   4. Expected Results 10
2. Literature Survey 10
   1. Survey of the Existing Models/Work 10
3. Overview of the Proposed System 11
   1. Introduction 11
   2. System Architecture 11
   3. System Diagrams 11
4. Proposed System Analysis and Design 12
   1. Introduction 12
   2. Requirement Analysis 12
      1. Functional Requirements 13
         1. Product Perspective 13
         2. Product features 13
         3. User characteristics 14
         4. Assumption & Dependencies 14
         5. Domain Requirements 14
         6. User Requirements 14
      2. Non Functional Requirements 15
         1. Product Requirements 15
            1. Efficiency 15
            2. Reliability 15
            3. Portability 15
            4. Usability 15
      3. Engineering Standard Requirements 16
         1. Economic 16
         2. Environmental 16
         3. Social 16
         4. Health and Safety 16

4.2.3.5 Legality 16

* + 1. System Requirements 17
       1. H/W Requirements 17
       2. S/W Requirements 18

1. Results and Discussion 19
   1. Sample Test Cases 20
   2. Summary of the Result 23
2. Conclusion, Limitations and Scope for future Work 24

Appendix 28

References 40

**Abstract**

Using hardware components, the project would take shape with the help of programming to combine multiple components into a single unit. A smart mirror will assist in simplifying personal day-to-day tasks as well as commercial ones, including but not limited to, the fashion industry, textile industry, home-automation, hairstyling etc. Using a smart mirror would extensively reduce the time spent on smartphones and PCs to check for the information the user needs. It reduces human efforts as it automatically updates the users’ calendar schedule to-do lists, twitter feeds, weather, news etc. With the help of the mentioned features, multi-tasking in today’s world becomes easier for the user as all the information can be accessed at a single glance. A mirror is among one of the most used household equipment, it is something which almost everyone looks at once a day at least. Digitizing an everyday-use household good such as a mirror will not only increase the technological support of a person but reduce the dependency on smart phones. One no longer needs to have an app for every single task, the smart mirror will take care of all such details in one place. In fact, it can be synced with a smart phone to make it even better. In an era where technology is creeping into every aspect of our lives, aiding wherever possible, helping with whatever means possible, why should a mirror be excluded.

Keywords: Smart mirror, embedded systems, Raspberry pi, Magic mirror, advanced mirror interface

**1. Introduction**

Due to the exponential growth in technology and development of newer components, mobile phones then became smart phones, mirrors became smart mirrors. This project will help us understand various concepts of Raspberry-pi and its programming and how a smart mirror is built using it from scratch. The luxury applications of smart mirrors are endless, ranging from personal to commercial, both appealing to the user or the customer aesthetically. At a time when everything is connected to everything else, when IoT is taking over our digital and personal lives, why should a mirror, something which every household possesses, stay behind. Efficient to assemble, a smart mirror can change the way how people look at themselves in a glass.

**1.1 Aim**

The aim of our project is to develop an advanced smart digital mirror interface capable of handling various inputs from the user such as voice recognition, PIR, and displaying a variety of information such as weather, time, date, day, news, social media feed.

**1.2 Theoretical Background**

A Smart Mirror is a hardware interface consisting of a two-way mirror screen, hardware equipment, installed software and a frame. A two-way mirror is the basic component needed. It has one side covered in a thin reflective coating, which is the side the user will face, the other side will have the screen and the other hardware components, which will be displayed on the user-facing side, however the user will not be able to see what is behind the screen except the .contents of the screen itself. Smart mirrors are used at multiple places. From luxury changing-rooms, to criminal investigation rooms, to decorative pieces in hotels. The applications are endless. The hardware side of the smart mirror should always be faintly illuminated whereas the user-side should have enough light so that the user can see a reflection as well as the contents of the screen.

**1.3 Methodology Adapted**

For our project we have adapted a waterfall model to proceed with considering various factors and the required consequences with the necessary time constraints as well as resources. In a waterfall model, each phase must be completed fully before the next phase can begin. This type of software development model is basically used for the project which is small and there are no uncertain requirements. At the end of each phase, a review takes place to determine if the project is on the right path and whether or not to continue or discard the project. In this model software testing starts only after the development is complete. In waterfall model phases do not overlap. We have chosen waterfall model because the development of our project will be in phases. Also, we will be testing the final product only after every previous phase is completed and we have a final product. After every phase we will review the work. Our requirements are understood beforehand and therefore it is easier to proceed under waterfall model.

**1.4 Expected Results**

By the end of the project we expect to have a full working model of an advanced digital mirror interface along with its various components and modules. We expect to also have accomplished our goals of successfully understanding embedded systems in detail as well as the usage of Raspberry Pi, its multiple components, sensors and the knowledge of how to plan, implement and present a project. We will also be learning in a detailed way the usage of programming languages such as python, jss etc with hardware components helping us understand real world scenarios and aiding is in the future when working in the industry.

**2. Literature Review**

Instead of the usual literature review of research papers, our product requires an in-depth analysis of existing systems and products developed by organizations. Such a study was undertaken to better understand the dynamics and workings of a smart mirror so as to develop a better solution. The following existing systems were analyzed:

**2.1 Microsoft's Magic Mirror**

Microsoft unleashed a prototype of its Magic Mirror in 2016.It uses a Raspberry Pi 3B with Windows 10 as its OS. It is powered by web applications and IOT. It has advanced features such as facial recognition, so based on the person standing in front of the mirror the content is displayed. The mirror helps people get ready in the morning and prepares them for the to-do tasks and schedules throughout the day. It also informs about the current traffic, the best route to take and weather forecast throughout the day.

**2.2 Apple Mirror by Rafael Dymek**

Mr. Dymek created a touchscreen mirror running on Apple's iOS 10. It has the same interface and functionality as an iPhone. It has all the features available on an iPhone. Although, the product has not been out in the market it is a platform for developers to work on and improve.

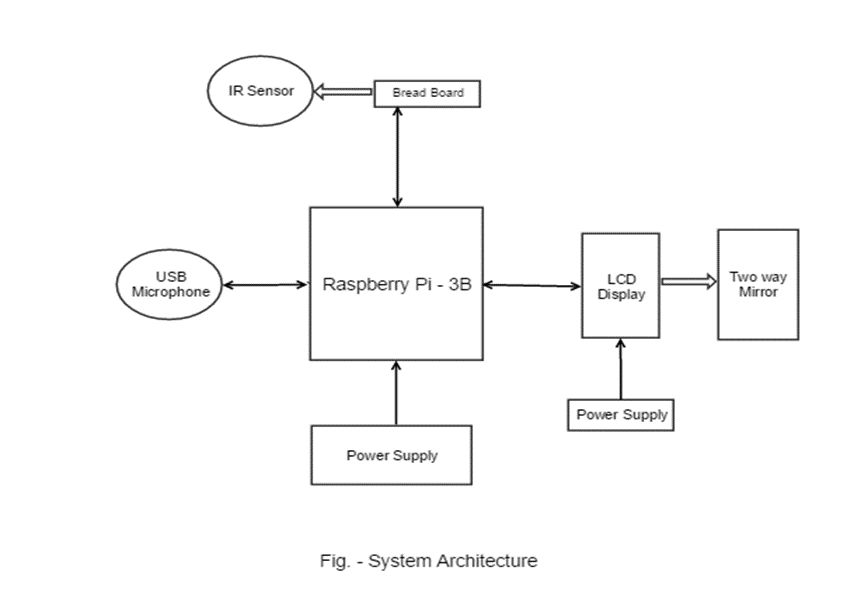
**2.3 The Naked Fitness tracker**

It captures the frame of the body using various sensors to scan the structure of the body. Based on the data it analyzes how exercises are affecting his/her body and required improvements. It visualizes the progress and provides feedbacks like body fat percentage, muscle mass percentage, muscle isolation etc.

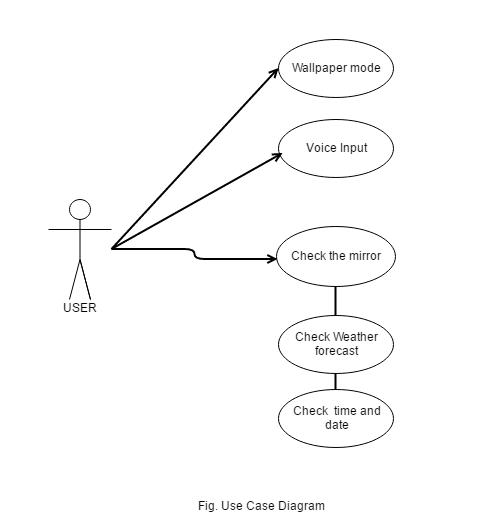
**3. Overview of the Proposed System**

**3.1 Introduction**

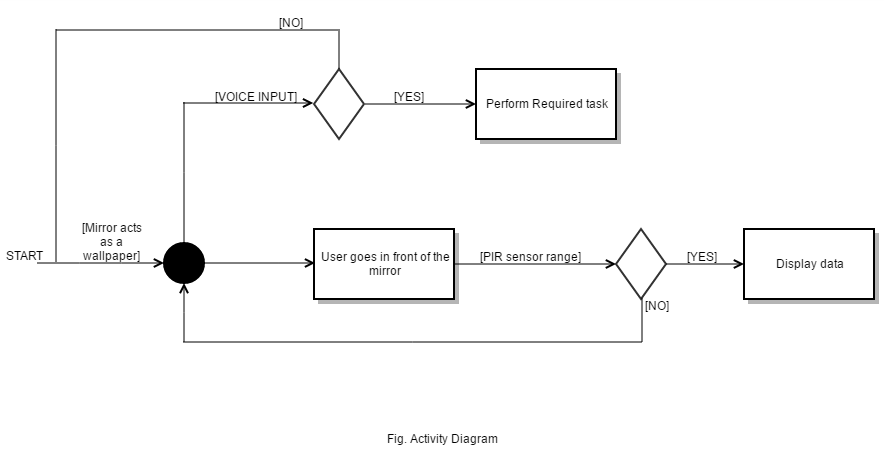
**3.1 System Architecture**



**3.2 Use Case Diagram**

****

**3.3 Activity Diagram**

****

**4. Proposed System Analysis and Design**

**4.1 Introduction**

Our proposed advanced digital mirror interface consists of multiple hardware components. Like any project, a complex blend of hardware partnered with appropriate software is required to make it a success. A framed box to hold the 2-way mirror, behind which we will have the RaspberryPi along with the LCD screen, sensors, and camera. This would require a delicate working of wood and glass to shape the framed container, so as to not damage neither the working components nor the stationary ones.

**4.2 Requirement Analysis**

4.2.1 Functional Requirements

Functional requirements define the specific functions of a system and their properties. It is used to judge the various objectives the project will undertake to complete.

*4.2.1.1 Product Perspective*

The aim of this product is to create a luxury item for everyday use. It simplifies and eases certain tasks, while looking at a mirror which a majority of the people do. The simplicity of a mirror combined with the complexity of the ever-changing everyday technology. Waking up and looking at a mirror is a daily-task and if it is combined with technology and imparted into a connected system, the technology is revolutionary, futuristic and aesthetically pleasing.

*4.2.1.2 Product features*

* A painting/picture collage when in inactive mode i.e. user is not in front of the mirror.
* A voice recognition system to command the interface
* A real-time weather display system
* Updated daily news from the internet
* A clock with day, date and time
* Social media feed based on the user’s choices
* PIR sensor to detect if user is present, then go into active mode

*4.2.1.3 User Characteristics*

The user is expected to be technology literate. Even though the system is basic on the outside with easy-to-use features, the user should be able to operate basic technological systems with multiple features. He should have knowledge about the internet, interface-changing and should be able to distinguish between active and inactive mode.

*4.2.1.4 Assumption and dependencies*

* An internet connection for displaying weather, social media feed
* An appropriate power source to power the equipment
* An unblocked view in front of the PIR sensor so that it can distinguish.
* A technology-friendly user
* The correct storage environment so as to not damage the mirror or the equipment.

*4.2.1.5 Domain Requirements*

The smart mirror works in any home environment. It only requires appropriate placement, a power source, and voice recognition/PIR sensing to function. It can also be implemented at a work environment in an advanced role with various added features, such as banners with employee login, changing advertisements with interface, a decorative piece with multiple paintings etc.

4.2.2 Non-Functional Requirements

These are required to judge how the system will operate, instead of specific behavioral characteristics of it.

*4.2.2.1 Product Requirements*

Our product would fulfill and cater to the following requirements in specific ways.

*4.2.2.1.1 Efficiency*

The efficiency of any project and system is of paramount importance. The smart mirror system would be highly efficient in detecting and displaying its functions. The components used are of the highest quality. It will use minimal power and minimal bandwidth to do its job, while occupying a nominal amount of space. It will be quick to display information and change between inactive and active mode.

*4.2.2.1.2 Reliability*

The reliability of the proposed system will be thoroughly tested so as to display accurate information while having negligible faults. A variety of development tests would be carried out such as the performance under various circumstances and inputs to determine the output and the reactions of the system.

*4.2.2.1.3 Portability*

A smart mirror is not portable as the proposed system work on a fixed system with a container attached. It can be mounted or fixed on a wall or on various other surfaces. To move the system, one has to disconnect the components from the power source, unmount the mirror and move it to a new location.

*4.2.2.1.4 Usability*

The proposed system is easy-to-use with a user friendly interface so as to make it an efficient system for all age groups. It can be made to display custom pictures if programmed correctly and the user can even alter the details with voice commands if required.

*4.2.2.2 Organizational Requirements*

Organizational requirements give us an idea of how to implement the project and what all requirements should be met for the successful development of it.

*4.2.2.2.1 Implementation Requirement*

The system requires a stable location to be mounted or fixed onto. It is suitable to have ample amount of light to act as a mirror and as a smart mirror (both active and inactive). The wooden frame of the smart mirror needs to be dry at all times. The mirror and the components inside must not be exposed to any amount of water.

*4.2.2.2.2 Engineering Standard Requirements*

All parts and components used in the system are certified by the necessary certifying agencies, such as IEEE, ISO etc.

4.2.3 Engineering Standard Requirements

Engineering standards are essential for a product so as to give us an idea of how it will impact across a wide spectrum of areas. The following standards have been taken care of.

*4.2.3.1 Economic*

The economic requirements of this project are the costs incurred while designing, testing and developing it. The majority of costs incurred is from the RaspberryPi, the hardware frame and the connectors and wires. The designing which requires cutting of wood and glass also takes up a part of the budget.

*4.2.3.2 Environmental*

The environmental affect is minimal. The product is based on a frame of plywood and not wood. The electricity costs are also minimal as it hardly takes up power. The screen and the RaspberryPi run directly from a plugged socket.

*4.2.3.3 Social*

From a social standpoint, the product offers a luxury for any user. The product meets social requirements and standards as it does not pose any threat and will be helpful to society.

*4.2.3.4 Health and Safety*

The product is safe to use. It has no health hazards associated with it. However caution must be exercised as the product does use electricity form the main lines. The safety rests on the user to carefully use the electricity sockets.

*4.2.3.5 Legality*

The product is legal and all components of it are legal as well. It is a product designed by students as an academic endeavor. It does not break any laws in India as long as it used within the laws laid down. It should not be used to cause mischief or destruction. The functionality provided by the developers intends no legal tussle or harm in any way to any individual, group, organization or government.

4.3.3 System Requirements

The system requires both hardware and software components to be implemented successfully. A proper connection and integration of both the components on a single platform is the objective so as to make the project work as per requirements.

*4.3.3.1 Hardware Requirements*

Raspberry Pi 3-B - Raspberry Pi is a mini sized computer used to promote and encourage Computer Science Projects in real time. It has a wide range of applications which include music, weather stations, sports etc.

Raspberry Pi 3B is the 3rd Generation Raspberry Pi. Some of its features include:

* Quadcore 64 bit ARM processor,
* 1GB SDRAM
* 4 USB ports
* Wireless LAN
* Bluetooth
* HDMI
* Ethernet port

Two way mirror - This kind of mirror is a normal mirror at one side and is see through at the other side. It is placed on top of the LCD display with reflecting mirror surface facing towards us and the see through channeling the LCD display.

LCD Display - We are using a 14.65 inches LCD display as the screen for the Smart Mirror. It will be placed behind a two way mirror. It would a draw a power source separate to the Raspberry Pi.

IR Sensor - Infrared sensor is an electronic device use to sense certain characteristics (like heat, motion etc.) in the surrounding. It senses by emitting or detecting infrared radiation.

In our project the IR sensor is connected to the Raspberry Pi via a Bread board. IR sensor will detect whether anyone is in front of a mirror or not. Based on the input certain planned tasks are performed.

USB Microphone - Since, Raspberry Pi does not have a sound card, it is only possible to enable voice input module using a USB microphone (we can also use a Web camera consisting a mic.)

Frame and Support - The encasing of the entire project would be done inside a wooden frame, built specifically for the mirror of the required size. After the mirror is fit inside the frame, the other hardware components will be fixed behind it in a compartment.

*4.3.3.2 Software Requirements*

Raspbian Jessie - Raspbian is a free Operation System based on Linux for Raspberry Pi hardware. Raspbian Jessie with PIXEL is the latest version. It is highly flexible and the inner workings of it are virtually invisible to the user, allowing the user to make effective programs on the hardware..

Jasper - Jasper is an open source Voice Control platform in a Raspberry Pi. Several voice control applications can be implemented using Jasper. Other open source Voice Control Platform include Amazon's Alexa.

Jasper requires a Wifi Card and a USB microphone to operate.

Python - Python is used for general purpose programming. It is a high-level programming language and currently one of the widest used ones in the world. It is very popular in the Raspbian Community as it supports various modules and library and as well as the availability of a Python interpreter for the RaspberryPi. It makes coding modules easier along with readily available, preprogrammed packages.

**5. Results and Discussion**

**5.1 Sample Test Cases**

For our project we have carried out thorough testing based on numerous test cases. Both hardware and software testing was done to ensure the efficiency and reliability of the system is optimal. All sample cases were performed diligently and thoroughly to ensure there is no lapse in the system and errors and failures are kept to a minimum throughout the life cycle of the product.

|  |  |
| --- | --- |
| Sl. No. | 1 |
| Purpose | Inactive mode testing |
| Assumption | System not in working mode |
| Preconditions | Inactive commands |
| Steps to reproduce | Change to active mode |
| Expected Results | System goes into active mode |
| Actual Outcome | System goes into active mode |
| Post conditions | System starts up |

|  |  |
| --- | --- |
| Sl. No. | 2 |
| Purpose | Testing of voice recognition |
| Assumption | Voice recognition software installed |
| Preconditions | Voice command by user |
| Steps to reproduce | Input voice commands preprogrammed into the system |
| Expected Results | System responds to specific commands |
| Actual Outcome | System responds to specific commands |
| Post conditions | Commands executed, results displayed |

|  |  |
| --- | --- |
| Sl. No. | 3 |
| Purpose | Weather testing |
| Assumption | Weather package installed |
| Preconditions | System active, looking for weather |
| Steps to reproduce | Command system to activate |
| Expected Results | Weather details displayed correctly |
| Actual Outcome | Weather details displayed correctly |
| Post conditions | Screen shows current weather |

|  |  |
| --- | --- |
| Sl. No. | 4 |
| Purpose | News Display testing |
| Assumption | News library installed |
| Preconditions | System active, looking to display news |
| Steps to reproduce | Command to display news |
| Expected Results | Display latest news |
| Actual Outcome | Latest news displayed |
| Post conditions | News displayed correctly |

|  |  |
| --- | --- |
| Sl. No. | 5 |
| Purpose | Clock and Time command check |
| Assumption | System active, system has correct time |
| Preconditions | System active |
| Steps to reproduce | Instantly display time and clock after system becomes active |
| Expected Results | Time and clock displayed correctly |
| Actual Outcome | Time and clock displayed correctly |
| Post conditions | System active with correct time |

|  |  |
| --- | --- |
| Sl. No. | 6 |
| Purpose | Microphone testing |
| Assumption | USB Microphone connected |
| Preconditions | USB Microphone works |
| Steps to reproduce | Check microphone, if reading input then display an output |
| Expected Results | Necessary output displayed |
| Actual Outcome | Necessary output displayed |
| Post conditions | System still active |

|  |  |
| --- | --- |
| Sl. No. | 7 |
| Purpose | Social Media feed test |
| Assumption | Social Media package installed  Social Media logged in |
| Preconditions | Social Media not showing |
| Steps to reproduce | Command to display social media feed |
| Expected Results | Social Media feed displayed |
| Actual Outcome | Social Media feed displayed |
| Post conditions | Social Media feed is showing, system runs successfully |

|  |  |
| --- | --- |
| Sl. No. | 8 |
| Purpose | Testing switch off command |
| Assumption | Voice command installed and preprogrammed |
| Preconditions | System active |
| Steps to reproduce | Command to switch off system |
| Expected Results | System reads input  System switches off |
| Actual Outcome | System reads input  System switches off |
| Post conditions | System inactive |

|  |  |
| --- | --- |
| Sl. No. | 9 |
| Purpose | Power testing |
| Assumption | Correct adapters and converters |
| Preconditions | System connected |
| Steps to reproduce | Connect the power adapters, cables and converters |
| Expected Results | System switches on |
| Actual Outcome | System switches on |
| Post conditions | System runs correctly |

|  |  |
| --- | --- |
| Sl. No. | 10 |
| Purpose | Proximity testing |
| Assumption | Proximity sensor installed |
| Preconditions | Proximity program running |
| Steps to reproduce | Put an object in front of the proximity sensor |
| Expected Results | Screen takes input and displays what is required |
| Actual Outcome | Proximity sensor input is read and result displayed |
| Post conditions | System runs succesfully |

|  |  |
| --- | --- |
| Sl. No. | 11 |
| Purpose | Sensor continuous input test |
| Assumption | All sensors working |
| Preconditions | Sensors connected |
| Steps to reproduce | Continuously give input and feedback to all sensors |
| Expected Results | Sensors still work |
| Actual Outcome | Sensors still work |
| Post conditions | System running |

**5.2 Summary of the result**

All the test cases were simulated successfully with the necessary outcomes. The recorded outputs of each case is as what was needed. The simulations show that the system modules have been programmed correctly and handle their inputs efficiently with utmost reliability and proficiency. These basic test cases are derived from the conditions required to make this project a success.

**6. Conclusions**

Over the course of the last 6 months, we have successfully designed, programmed, tested and delivered the necessary outputs and outcomes for the project. All the modules have worked as expected. The advanced mirror interface displays the results as needed. It takes input from voice commands as needed and it can be reprogrammed and preprogrammed to function with flexibility as well as reliability.

**6.1 Limitations**

The limitations of the product are restricted to the portability of it and the user perception of the product as a whole. The product is hardly portable and needs a fixed environment with the right lighting conditions to function (which is the same lighting for a normal mirror). It is also limited by how a user may perceive the need for such a product to be.

**6.2 Future Scope**

The scope for this product is endless. A smart mirror can be programmed and reprogrammed with necessary updates to display and detail whatever the user may require. The hardware will always remain the same, however the displays and functions can be advanced over time as in when required. The aesthetic appeal with futuristic design allows the smart mirror to be constant for a long period of time. A lot of commercial, corporate and industrial functionalities can be added as further modules to increase the scope of the product.

**Appendix**

**Code**

from Tkinter import \*

import locale

import threading

import time

import requests

import json

import traceback

import feedparser

from PIL import Image, ImageTk

from contextlib import contextmanager

LOCALE\_LOCK = threading.Lock()

ui\_locale = ''

time\_format = 12

date\_format = "%b %d, %Y"

news\_country\_code = 'in'

weather\_api\_token = '01b5f464b8caa5e158b2baa92c9c1' # from(username Rahul Nagarkoti ) https://darksky.net/dev/

weather\_lang = 'en'

weather\_unit = 'us'

lat = ''#'43.534'

lon = ''#'120.87'

xlarge\_text\_size = 94

large\_text\_size = 48

medium\_text\_size = 28

small\_text\_size = 14

@contextmanager

def setlocale(name):

with LOCALE\_LOCK:

saved = locale.setlocale(locale.LC\_ALL)

try:

yield locale.setlocale(locale.LC\_ALL, name)

finally:

locale.setlocale(locale.LC\_ALL, saved)

icon\_lookup = {

'clear-day': "assets/Sun.png", # clear sky day

'wind': "assets/Wind.png", #wind

'cloudy': "assets/Cloud.png", # cloudy day

'partly-cloudy-day': "assets/PartlySunny.png", # partly cloudy day

'rain': "assets/Rain.png", # rain day

'snow': "assets/Snow.png", # snow day

'fog': "assets/Haze.png", # fog day

'clear-night': "assets/Moon.png", # clear sky night

'hail': "assests/Hail.png" # hail

}

class Clock(Frame):

def \_\_init\_\_(self, parent, \*args, \*\*kwargs):

Frame.\_\_init\_\_(self, parent, bg='black')

# initialize time label

self.time1 = ''

self.timeLbl = Label(self, font=('Helvetica', large\_text\_size), fg="white", bg="black")

self.timeLbl.pack(side=TOP, anchor=E)

# initialize day of week

self.day\_of\_week1 = ''

self.dayOWLbl = Label(self, text=self.day\_of\_week1, font=('Helvetica', small\_text\_size), fg="white", bg="black")

self.dayOWLbl.pack(side=TOP, anchor=E)

# initialize date label

self.date1 = ''

self.dateLbl = Label(self, text=self.date1, font=('Helvetica', small\_text\_size), fg="white", bg="black")

self.dateLbl.pack(side=TOP, anchor=E)

self.tick()

def tick(self):

with setlocale(ui\_locale):

if time\_format == 12:

time2 = time.strftime('%I:%M %p')

else:

time2 = time.strftime('%H:%M') #hour in 24h

day\_of\_week2 = time.strftime('%A')

date2 = time.strftime(date\_format)

if time2 != self.time1:

self.time1 = time2

self.timeLbl.config(text=time2)

if day\_of\_week2 != self.day\_of\_week1:

self.day\_of\_week1 = day\_of\_week2

self.dayOWLbl.config(text=day\_of\_week2)

if date2 != self.date1:

self.date1 = date2

self.dateLbl.config(text=date2)

self.timeLbl.after(200, self.tick

class FullscreenWindow:

def \_\_init\_\_(self):

self.tk = Tk()

self.tk.configure(background='black')

self.topFrame = Frame(self.tk, background = 'black')

self.bottomFrame = Frame(self.tk, background = 'black')

self.topFrame.pack(side = TOP, fill=BOTH, expand = YES)

self.bottomFrame.pack(side = BOTTOM, fill=BOTH, expand = YES)

self.state = False

self.tk.bind("<Return>", self.toggle\_fullscreen)

self.tk.bind("<Escape>", self.end\_fullscreen)

# clock

self.clock = Clock(self.topFrame)

self.clock.pack(side=LEFT, anchor=N, padx=150, pady=60)

# weather

self.weather = Weather(self.topFrame)

self.weather.pack(side=RIGHT, anchor=N, padx=100, pady=60)

# news

self.news = News(self.bottomFrame)

self.news.pack(side=LEFT, anchor=S, padx=90, pady=30)

def toggle\_fullscreen(self, event=None):

self.state = not self.state # toggling the boolean

self.tk.attributes("-fullscreen", self.state)

return "break"

def end\_fullscreen(self, event=None):

self.state = False

self.tk.attributes("-fullscreen", False)

return "break"

if \_\_name\_\_ == '\_\_main\_\_':

w = FullscreenWindow()

w.tk.mainloop()

class Weather(Frame):

def \_\_init\_\_(self, parent, \*args, \*\*kwargs):

Frame.\_\_init\_\_(self, parent, bg='black')

self.temperature = ''

self.forecast = ''

self.location = ''

self.currently = ''

self.icon = ''

self.degreeFrm = Frame(self, bg="black")

self.degreeFrm.pack(side=TOP, anchor=W)

self.temperatureLbl = Label(self.degreeFrm, font=('Helvetica', xlarge\_text\_size), fg="white", bg="black")

self.temperatureLbl.pack(side=LEFT, anchor=N)

self.iconLbl = Label(self.degreeFrm, bg="black")

self.iconLbl.pack(side=LEFT, anchor=N, padx=20)

self.currentlyLbl = Label(self, font=('Helvetica', medium\_text\_size), fg="white", bg="black")

self.currentlyLbl.pack(side=TOP, anchor=W)

self.forecastLbl = Label(self, font=('Helvetica', small\_text\_size), fg="white", bg="black")

self.forecastLbl.pack(side=TOP, anchor=W)

self.locationLbl = Label(self, font=('Helvetica', small\_text\_size), fg="white", bg="black")

self.locationLbl.pack(side=TOP, anchor=W)

self.get\_weather()

def get\_ip(self):

try:

ip\_url = "http://jsonip.com/"

req = requests.get(ip\_url)

ip\_json = json.loads(req.text)

return ip\_json['ip']

except Exception as e:

traceback.print\_exc()

return "Error: %s. Cannot get ip." % e

def get\_weather(self):

try:

#location2 = ""

# get weather

weather\_req\_url = "https://api.darksky.net/forecast/01b5f464b8caa5e158b18b2baa92c9c1/12.934968,79.146881"#"https://api.darksky.net/forecast/%s/%s,%s?lang=%s&units=%s" % (weather\_api\_token, lat, lon, weather\_lang, weather\_unit)

r = requests.get(weather\_req\_url)

weather\_obj = json.loads(r.text)#added json.loads() instead of json.load()

degree\_sign= u'\N{DEGREE SIGN}'

temperature2 = "%s%s" % (str(int(weather\_obj['currently']['temperature'])), degree\_sign)

currently2 = weather\_obj['currently']['summary']

forecast2 = weather\_obj["hourly"]["summary"]

icon\_id = weather\_obj['currently']['icon']

icon2 = None

if icon\_id in icon\_lookup:

icon2 = icon\_lookup[icon\_id]

if icon2 is not None:

if self.icon != icon2:

self.icon = icon2

image = Image.open(icon2)

image = image.resize((100, 100), Image.ANTIALIAS)

image = image.convert('RGB')

photo = ImageTk.PhotoImage(image)

self.iconLbl.config(image=photo)

self.iconLbl.image = photo

else:

# remove image

self.iconLbl.config(image='')

if self.currently != currently2:

self.currently = currently2

self.currentlyLbl.config(text=currently2)

if self.forecast != forecast2:

self.forecast = forecast2

self.forecastLbl.config(text=forecast2)

if self.temperature != temperature2:

self.temperature = temperature2

self.temperatureLbl.config(text=temperature2)

'''if self.location != location2:

if location2 == ", ":

self.location = "Cannot Pinpoint Location"

self.locationLbl.config(text="Cannot Pinpoint Location")

else:

self.location = location2

self.locationLbl.config(text=location2)'''

except Exception as e:

traceback.print\_exc()

print "Error: %s. Cannot get weather." % e

self.after(600000, self.get\_weather)

@staticmethod

def convert\_kelvin\_to\_fahrenheit(kelvin\_temp):#kelvin to celsius

return (kelvin\_temp - 273)

class News(Frame):

def \_\_init\_\_(self, parent, \*args, \*\*kwargs):

Frame.\_\_init\_\_(self, parent, \*args, \*\*kwargs)

self.config(bg='black')

self.title = 'News' # 'News' is more internationally generic

self.newsLbl = Label(self, text=self.title, font=('Helvetica', small\_text\_size), fg="white", bg="black")

self.newsLbl.pack(side=TOP, anchor=W)

self.headlinesContainer = Frame(self, bg="black")

self.headlinesContainer.pack(side=TOP)

self.get\_headlines()

def get\_headlines(self):

try:

# remove all children

for widget in self.headlinesContainer.winfo\_children():

widget.destroy()

if news\_country\_code == None:

headlines\_url = "https://news.google.com/news?ned=us&output=rss"

else:

headlines\_url = "https://news.google.com/news?ned=%s&output=rss" % news\_country\_code

feed = feedparser.parse(headlines\_url)

for post in feed.entries[0:5]:

headline = NewsHeadline(self.headlinesContainer, post.title)

headline.pack(side=TOP, anchor=W)

except Exception as e:

traceback.print\_exc()

print "Error: %s. Cannot get news." % e

self.after(600000, self.get\_headlines)

class NewsHeadline(Frame):

def \_\_init\_\_(self, parent, event\_name=""):

Frame.\_\_init\_\_(self, parent, bg='black')

image = Image.open("assets/Newspaper.png")

image = image.resize((25, 25), Image.ANTIALIAS)

image = image.convert('RGB')

photo = ImageTk.PhotoImage(image)

self.iconLbl = Label(self, bg='black', image=photo)

self.iconLbl.image = photo

self.iconLbl.pack(side=LEFT, anchor=N)

self.eventName = event\_name

self.eventNameLbl = Label(self, text=self.eventName, font=('Helvetica', small\_text\_size), fg="white", bg="black")

self.eventNameLbl.pack(side=LEFT, anchor=N)

class FullscreenWindow:

def \_\_init\_\_(self):

self.tk = Tk()

self.tk.configure(background='black')

self.topFrame = Frame(self.tk, background = 'black')

self.bottomFrame = Frame(self.tk, background = 'black')

self.topFrame.pack(side = TOP, fill=BOTH, expand = YES)

self.bottomFrame.pack(side = BOTTOM, fill=BOTH, expand = YES)

self.state = False

self.tk.bind("<Return>", self.toggle\_fullscreen)

self.tk.bind("<Escape>", self.end\_fullscreen)

# clock

self.clock = Clock(self.topFrame)

self.clock.pack(side=LEFT, anchor=N, padx=150, pady=60)

# weather

self.weather = Weather(self.topFrame)

self.weather.pack(side=RIGHT, anchor=N, padx=100, pady=60)

# news

self.news = News(self.bottomFrame)

self.news.pack(side=LEFT, anchor=S, padx=90, pady=30)

def toggle\_fullscreen(self, event=None):

self.state = not self.state # toggling the boolean

self.tk.attributes("-fullscreen", self.state)

return "break"

def end\_fullscreen(self, event=None):

self.state = False

self.tk.attributes("-fullscreen", False)

return "break"

if \_\_name\_\_ == '\_\_main\_\_':

w = FullscreenWindow()

w.tk.mainloop()

import pygame

import time

import RPi.GPIO as GPIO

sensor=GPIO.input(11

pygame.init()

pygame.mouse.set\_visible(False)

screen = pygame.display.set\_mode((0, 0), pygame.FULLSCREEN)

screen.fill((0, 0, 0))

done = False

while not done:

for event in pygame.event.get():

if == pygame.QUIT:

done = True

if (i ==0): pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

done = True

time.sleep(0.01)

pygame.quit()

import pygame

import time

import RPi.GPIO as GPIO

sensor=GPIO.input(11

pygame.init()

pygame.mouse.set\_visible(False)

screen = pygame.display.set\_mode((0, 0), pygame.FULLSCREEN)

screen.fill((0, 0, 0))

done = False

while not done:

for event in pygame.event.get():

if == pygame.QUIT:

done = True

if (i ==0): pygame.KEYDOWN:

if event.key == pygame.K\_ESCAPE:

done = True

time.sleep(0.01)

pygame.quit()

**Algortihm**

Step 1: • Initialize required variables, weather\_api

, lattitude,longitude,news\_country\_code,

icon\_lookup<-array of all the images

required

Step 2: • Create class clock

• Create time label

timeLbl1< - Label(self,..)

timeLbl1.pack()

• Create day label- dayOWlbl<- Label(self,..)

dayOWlbl.pack()

• Create date label

datelbl<-Label(self,...)

datelbl.pack()

• Create a function tick()

• Update and display time

timeLbl1.after(200,tick) to call self function

every 200 ms to update time etc.

Step3: • Create class weather

• Initialize all the required variables, labels

temperature<-' '

location<- ' '

icon<- ' '

forecast <- ' '

• Create a function get\_weather

get\_weather():

weather\_req\_url<- "darksky.net/forecast..."

r<-requests.get(weather\_req\_url)

//r receives the contents from the website in

//JSON format

weather\_obj <- json.loads(r.text)

//JSON.loads decodes the data in JSON

//format to txt format.

• Select the icon from the icon{} array based

on the weather data received from the

website

• Create a function tick()

• After (60000,tick) to call tick function every

6000 s to update time etc.

Step 4: • Create class NewsHeadline

Pack newspaper image in the Tkinter

window

Image<Image.open("assets/Newspaper.png")

Photo <-ImageTk.PhotoImage(image)

iconLbl<-Label(self,image=photo)

iconLbl.pack()

Step 5: • Create class news

• Initialize required variables

news\_country\_code<- ' '

headlines\_url<

"https://news.google.com/news?ned=

%s&output=rss"

//feedparser is a module to parse feeds

feed <- feedparser.parse(headlines\_url)

• Display the first five headlines in the

Tkinter window.

for post in feed.entries[0:5]:

headline <-

NewsHeadline(self.headlinesContainer,

post.title)

headline.pack()

self.after(600000, self.get\_headlines) calls

get\_headlines function every 600000 ms

Step 6: • Create class Fullscreen

• Divide the Tkinter window into Top frame

and Bottom Frame

• Create object of class news and use the

object to pack the Tkinter window

news.pack()

• Create object of class weather and use the

object to pack the Tkinter window

weather.pack()

• Create object of class clock and use the

object to pack the TKinter window

clock.pack()

Step 7: Inside main():

• Create an object of class Fullscreen as w

• Run Tkinter mainloop function with the

Fullscreen object.

**Screenshots**



Fig. 4 The Hardware components

In the above image we can see the following components:

* The screen which will be behind the mirror
* The USB webcamera and microphone
* The proximity sensor
* Necessary cables
* VGA to HDMI converter
* Power adapter

This setup is used for the project to run successfully. The setup requires exact connections and layout for it to function. Once a device has been connected to a port it should be changed as the RaspberryPi would have registered that device to that port.

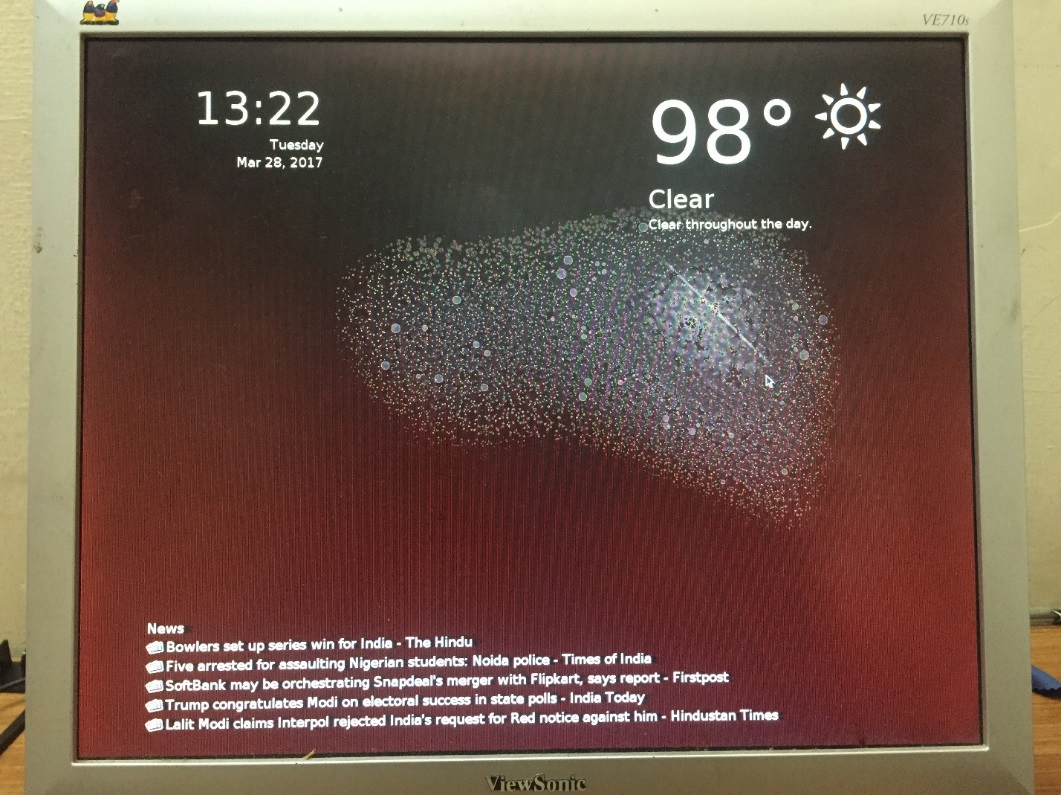


Fig. 5 Running our project on Raspberry Pi

The screenshot above shows the homepage of our project. It has the following features visible-

* On the top left is the date and time module
* On the top right is the weather module
* The bottom shows us a live newsfeed for the day

As we can see all the modules are working successfully for the homepage section of the project. The system inputs all the data from the internet, using a wifi connection. The preprogrammed data for the links and websites from which the system takes the data is working as required. The format for the weather is Fahrenheit whereas the format for the time is a 24h clock and date is MM: DD: YYYY. The news feed takes news only for India and certain world news websites.

****

Fig. 6. Wooden frame and mirror

****

Fig. 7 Behind the mirror.

In the above picture we can see how the screen is fixed behind the mirror. It is encased in a box for extra protection and ease of use. The sensors are attached to the sides of the mirror and to the bottom. The power supply is taken from the bottom of the mirror whereas the RaspberryPi and other hardware equipment is attached in a separate box/case. The final apparatus would require lighting from the front but not the back as the screen should be the sole source of light from behind the mirror.

**List of Figures**

**Title Page No.**

Fig 1. System Architecture 10

Fig. 2 Use Case Diagram 11

Fig. 3 Activity Diagram 11

Fig. 4 Screenshot 1 31

Fig. 5 Screenshot 2 32

Fig. 6 Screenshot 3 33

Fig. 7 Screenshot 4 34

**References**

Weblinks:

* <http://lifehacker.com/build-a-magic-mirror-with-a-raspberry-pi-and-an-old-mon>itor
* http://www.twowaymirrors.com/smart-mirror/

Journal:

* Vijayaraghavan, Induhumathi T.A, J. Chopra A.R.N, K. Miracline R, “*A REAL Time Virtual Dressing Room Application Using OpenCV*”, Anna University: Chennai 600 025, April 2014.
* G. Yolcu, S. Kazan, and C. Oz, “*Real Time Virtual Mirror Using Kinect*,” Baikan journal of Electrical & Computer Engineering, vol. 2, no. 2, pp. 75-78, (2014)
* U. Cheema, M. Rizwan, R. Jalal, F. Durrani, N. Sohail, “*The Trend of online shopping in 21st century: Impact of enjoyment in tam model*” Asian Journal of Empirical Research, vol. 3, no. 2, pp. 131-141,(2014)
* M Anwar Hussain, P.K. Atrey, Abdulmotaleb El Saddik, “*Smart mirror for ambient home environment*”, DOI: 10.1049/cp:20070431 · IEEE Xplore, Conference: Intelligent Environments, 2007. IE 07. 3rd IET International Conference

**References**

Weblinks:

* <http://lifehacker.com/build-a-magic-mirror-with-a-raspberry-pi-and-an-old-mon>
* http://www.twowaymirrors.com/smart-mirror/

Journal:

* Vijayaraghavan, Induhumathi T.A, J. Chopra A.R.N, K. Miracline R, “A REAL Time Virtual Dressing Room Application Using OpenCV”, Anna University: Chennai 600 025, April 2014.
* G. Yolcu, S. Kazan, and C. Oz, “Real Time Virtual Mirror Using Kinect,” Baikan journal of Electrical & Computer Engineering, vol. 2, no. 2, pp. 75-78, (2014)
* U. Cheema, M. Rizwan, R. Jalal, F. Durrani, N. Sohail, “The Trend of online shopping in 21st century: Impact of enjoyment in tam model” Asian Journal of Empirical Research, vol. 3, no. 2, pp. 131-141
* Smart mirror for ambient home environment, M Anwar Hussain, P.K. Atrey, Abdulmotaleb El Saddik, DOI: 10.1049/cp:20070431 · Source: IEEE Xplore, Conference: Intelligent Environments, 2007. IE 07. 3rd IET International Conference

Font – Times New Roman

Font Size – 12

Line Spacing – 1.5

* Abstract: (300 words) (with Key words)
* System Architecture / System Design / System Model with description
* Methodology Adapted
* Expected Results with discussion
* Details of Hardware and Software
* Reference : (IEEE format)