Smart Pharmacy System Based On Iot And Raspberry Pi

Vishal. T, Bharath Kumar. R, Vinoth. K, S.R. Rajeswari
Student, Student, Student, A.P(O.G)
Computer Science and Engineering,
SRM Institute of Science and Technology, Chennai, India

Abstract: ***The retailers in the pharmacy usually face the problem of selling an expired product to the consumers which is really unsafe as it is done manually which can be replaced by barcode system. In this project, we propose a system which makes less error and time taken to verify the stock availability of the medicines and its expiry date has been created by the use of IOT and raspberry pi with the help of barcodes***

I. INTRODUCTION

In general, a patient would give their prescription to pharmacy, then a pharmacist get this prescription, seeks for the medicine and gives to the patient. While giving the medicine to the patient, the pharmacist will check for expiry date manually and delivers it to the patient which can cause error at times by selling expired products to the patient. During this process, it takes lots of time on checking drug and it's also unsafe for the patients So in this paper, we have introduced a system to check the details of the medicine such as the expiry date, stock availability, cost and name of the medicine by normal web camera and display the result in LCD screen and also by speech for output with the help of raspberry pi. The library which contains the barcode details has been first created by reading the barcode from the web camera which converts the image into grey image and by converting the barcodes into binary digits which forms an identity for the barcode. Using the barcode number obtained by conversion is used for storing the details of the medicine such as name of the medicine, stock availability, cost of the tablet and expiry date of the medicine using python language in raspberry pi that stores the details in the SD card which has been inserted in the raspberry pi kit. The purpose of this project is to make sure the safety for the patients by not selling the expired products to the patients and also to take a note of the stock availability by reducing the errors and time taken.

II. LITERATURE SURVEY

2.1 EXISTING SYSTEM

In the existing system, to know the stock availability of a medicine the pharmacist has to check it manually from a database and also to know to the details of the medicine such as its expiry date, cost and name of the medicine would be checked manually by reading the details which is on the medicine.

2.2 Issues in Existing System

The time taken is more when it's performed manually to check the stock availability of the medicine form a database, as we have to keep updating the stock availability for each time after selling a product. Since the process of checking is done manually by the pharmacist there are chances of errors to take place such as by selling the expired product to the patient which is very unsafe.

2.3 Details of Literature Survey

In the proposed system we are going to find the details of the medicine such as its stock availability, expiry date, cost and name of the medicine by use of barcodes which reads the detail of the medicine from the database that has been created by openCV to store in the SDcard of the raspberry pi kit and displays the result on the LCD screen, by this process the time taken for checking the stock availability and also there is no need for rechecking on expiry, cost and stock availability by single process. Since the process is done by machines there are lesser chances of making errors.

2.4 Problem Statement

At pharmacy there are chances of giving expired products to a patient while selling the medicine or also by not checking the expiration of the products which are in stock. In responsible to this problem we propose a system by using barcodes to check the stock availability and expiration of the medicine.

III. SYSTEM ANALYSIS AND DESIGN

3.1 Analysis of the Problem

To reduce the chances of selling an expired product to a patient can be solved by using machines to read the details from the database and display the result on screen

3.2 System Architecture

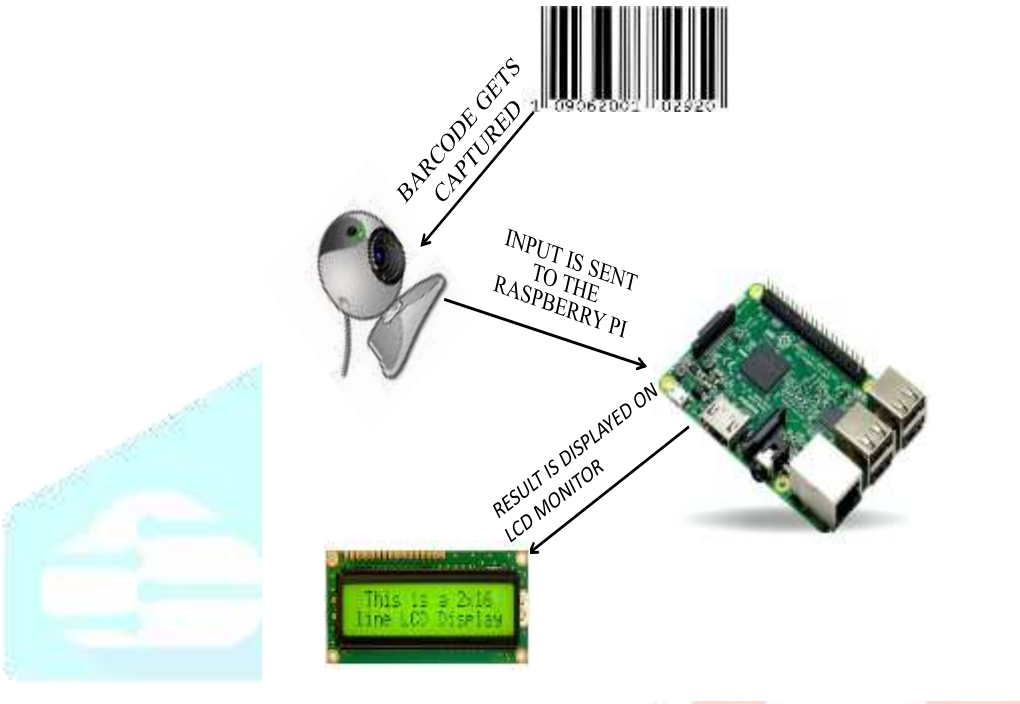

Figure 1 Overall System Architecture

3.3 Description

At first the barcode which is over the medicine is scanned by the web camera that is connected to the raspberry pi by the USB port reads the image and converts the image into grey image for conversion of the codes into binary codes to decrypt the code in which the details of the medicine is stored on SDcard of the raspberry pi which acts as the database containing the information of medicine such as its name, expiry date and cost, then it checks for expiry by comparing with the current date to the date of expiration, if expired it display the result on LCD screen that it has been expired if else it will start to display the name of the medicine, its date of expiry, cost and the stock availability will be displayed on the LCD screen after completion of scanning it will start to scan for next image by web camera.

3.4 Implementation Methodology

In our design, we use the barcode system as the database maintenance system by first reading the barcode through a web camera which converts the image into grey image so that it would be easy to convert the image into binary digits as it would be of black and white in colour and with the help of the barcode library which has the value for each code, the binary digit is formed setting an threshold value which identifies the dark and light shades for conversion to display the unique identity number for the barcode. By using that unique identity number of the barcode, we store the information of the medicine such as its name, expiry date, cost and its stock availability of a particular medicine is stored with the help of python in the SDcard of the raspberry pi which acts as the database and when the barcode is scanned it checks for expiry by comparing the current date with the expiry date of the medicine, if its expired it displays the result on LCD screen that it has been expired and if it's not expired it will display its name, cost, expiry date and stock availability of the medicine on the LCD screen.

3.5 System Requirements

3.5.1 Hardware tools

- Raspberry pi
- Camera
- LCD screen
- Computer

3.5.2 Software tools

- Raspbian Jessie OS
- OpenCV
- Language: Python

IV. MODULE DESCRIPTION

4.1 Barcode Conversion

The conversion of barcode into an unique identity number is done in this module by first capturing the barcode through the web camera which captures the image in the form of grey image so that it would be easy to convert the grey image into binary digits as it would be of black and white in colour and with the help of the barcode library that is in openCV which has the value for each code, the binary digit is formed setting an threshold value which identifies the dark and light shades for conversion to display the unique identity number of the barcode and stores the information of the barcode in SDcard by openCV.

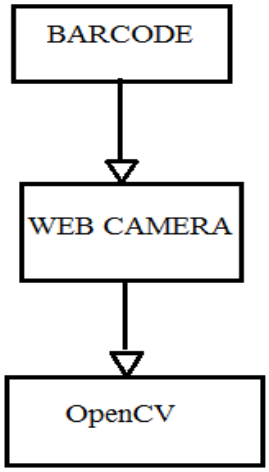

Figure 2 Barcode Conversion

4.2 OpenCV to Store Information

By knowing the unique identity number of the barcode which has been found using the web camera that has been stored in the SDcard using openCV will store the information of a medicine such as the name of the medicine, its cost, expiry date and stock availability will be stored in the SDcard which has been inserted in the raspberry pi kit so that it will be used as database for information.

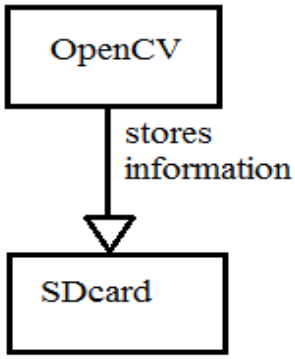

Figure 3 OpenCV to Store Information

4.3 Detection by LCD Screen

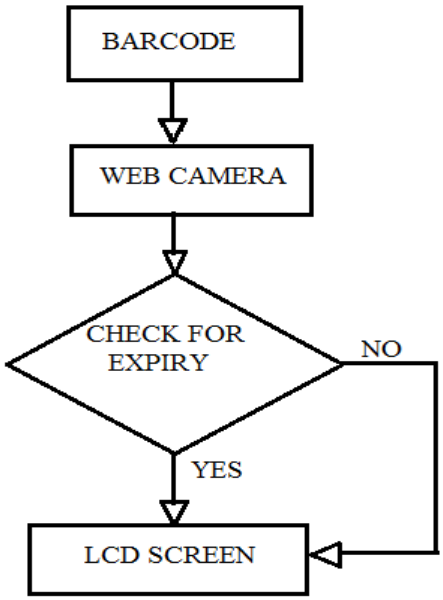

Figure 4 Detection by LCD Screen

After capturing the barcode by using web camera which converts the code into unique identity number to check for the details stored using the unique identity number on SDcard using OpenCV will have the information of the medicine such as its name, cost , expiry date and stock availability of the medicine by checking the expiry date of the medicine with the current date and displays the result on LCD screen. If the medicine is expired, it displays the message that it has been "expired" and if it's not expired it will display the name of the medicine, expiry date, cost and stock availability of the medicine on the LCD screen.

V. SYSTEM IMPLEMENTATION

5.1 Overview of the Platform

5.1.1 Raspberry PI – ARM vs. x86

The processor at the heart of the Raspberry Pi system is a Broadcom BCM2837 system-on-chip (SoC) multimedia processor. This means that the vast majority of the system's components, including its central and graphics processing units along with the audio and communications hardware, are built onto that single component hidden beneath the 256 MB memory chip at the centre of the board It also uses a different instruction set architecture (ISA), known as ARM. The BCM2837 SoC, located beneath a Hynix memory chip Developed by Acorn Computers back in the late 1980s, the ARM architecture is a relatively uncommon sight in the desktop world. Where it excels, however, is in mobile devices: the phone in your pocket almost certainly has at least one ARM-based processing core hidden away inside. Its combination of a simple reduced instruction set (RISC) architecture and low power draw make it the perfect choice over desktop chips with high power demands and complex instruction set (CISC) architectures. The ARM-based BCM2837 is the secret of how the Raspberry Pi is able to operate on just the 5V 1A power supply provided by the onboard micro-USB port. It's also the reason why you won't *fi*nd any heat-sinks on the device: the chip's low power draw directly translates into very little waste heat, even during complicated processing tasks. It does, however, mean that the Raspberry Pi isn't compatible with traditional PC software. The majority of software for desktops and laptops is built with the x86 instruction set architecture in mind, as found in processors from the likes of AMD, Intel and VIA. As a result, it won't run on the ARM-based Raspberry Pi.The BCM2837 uses a generation of ARM's processor design known as ARM11, which in turn is designed around a version of the instruction set architecture known as ARMv6. This is worth remembering: ARMv6 is a lightweight and powerful architecture, but has a rival in the more advanced ARMv7 architecture used by the ARM Cortex family of processors. Software developed for ARMv7, like software developed for x86, is sadly not compatible with the Raspberry Pi's BCM2837—although developers can usually convert the software to make it suitable

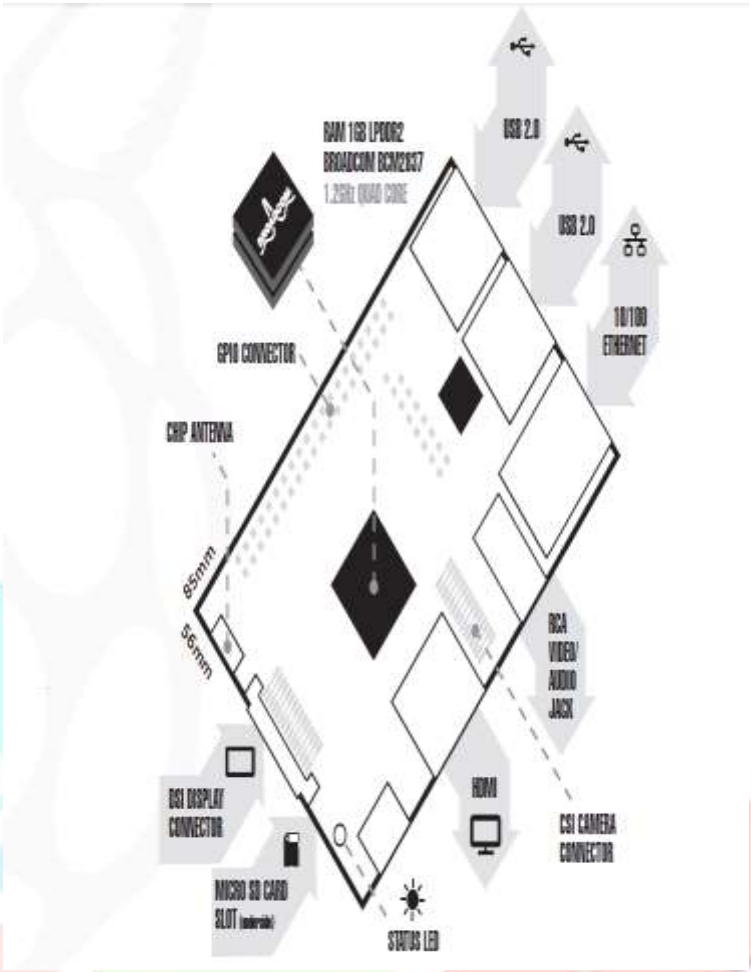

Figure 5 Raspberry pi ARM vs. x86

5.2 Sample Coding

```
from __future__ import print_function
from collections import namedtuple
from contextlib import contextmanager
from ctypes import cast, c_void_p, string_at
from operator import itemgetter

from .pyzbar_error import PyZbarError
from .wrapper import (
    zbar_image_scanner_set_config,
    zbar_image_scanner_create, zbar_image_scanner_destroy,
    zbar_image_create, zbar_image_destroy, zbar_image_set_format,
    zbar_image_set_size, zbar_image_set_data, zbar_scan_image,
    zbar_image_first_symbol, zbar_symbol_get_data,
    zbar_symbol_get_loc_size, zbar_symbol_get_loc_x, zbar_symbol_get_loc_y,
    zbar_symbol_next, ZBarConfig, ZBarSymbol, EXTERNAL_DEPENDENCIES
)

__all__ = ['decode', 'EXTERNAL_DEPENDENCIES']


Rect = namedtuple('Rect', ['left', 'top', 'width', 'height'])
Decoded = namedtuple('Decoded', ['data', 'type', 'rect'])
```

```
_FOURCC = {
    'L800': 808466521,
    'GRAY': 1497715271
}

_RANGEFN = getattr(globals(), 'xrange', range)


@contextmanager

def _image():
    """A context manager for `zbar_image`, created and destoyed by
    `zbar_image_create` and `zbar_image_destroy`.

    Yields:
        POINTER(zbar_image): The created image

    Raises:
        PyZbarError: If the image could not be created.
    """
    image = zbar_image_create()
    if not image:
        raise PyZbarError('Could not create zbar image')
    else:
        try:
            yield image
        finally:
            zbar_image_destroy(image)

@contextmanager

def _image_scanner():
    """A context manager for `zbar_image_scanner`, created and destroyed by
    `zbar_image_scanner_create` and `zbar_image_scanner_destroy`.

    Yields:
        POINTER(zbar_image_scanner): The created scanner

    Raises:
        PyZbarError: If the decoder could not be created.
    """
    scanner = zbar_image_scanner_create()
    if not scanner:
        raise PyZbarError('Could not create image scanner')
    else:
        try:
            yield scanner
        finally:
            zbar_image_scanner_destroy(scanner)


def _bounding_box_of_locations(locations):
    """Computes a bounding box from scan locations.

    Args:
        locations: iterable of tuples of ints (x, y).

    Returns:
        `Rect`: Coordinates of the bounding box.
```

```
    """
    x_values = list(map(itemgetter(0), locations))
    x_min, x_max = min(x_values), max(x_values)
    y_values = list(map(itemgetter(1), locations))
    y_min, y_max = min(y_values), max(y_values)
    return Rect(x_min, y_min, x_max - x_min,  y_max - y_min)


def _symbols_for_image(image):
    """Generator of symbols.

    Args:
        image: `zbar_image`

    Yields:
        POINTER(zbar_symbol): Symbol
    """
    symbol = zbar_image_first_symbol(image)
    while symbol:
        yield symbol
        symbol = zbar_symbol_next(symbol)


def _decode_symbols(symbols):
    """Generator of decoded symbol information.

    Args:
        image: iterable of instances of `POINTER(zbar_symbol)`

    Yields:
        Decoded: decoded symbol
    """
    for symbol in symbols:

  data = string_at(zbar_symbol_get_data(symbol))
       symbol_type = ZBarSymbol(symbol.contents.type).name
        locations = [
            (
                zbar_symbol_get_loc_x(symbol, index),
                zbar_symbol_get_loc_y(symbol, index)
            )
            for index in _RANGEFN(zbar_symbol_get_loc_size(symbol))
        ]

        yield Decoded(
            data=data,
            type=symbol_type,
            rect=_bounding_box_of_locations(locations),
        )


def _pixel_data(image):
    """Returns (pixels, width, height)

    Returns:
        :obj: `tuple` (pixels, width, height)
    """
        if 'PIL.' in str(type(image)):
        if 'L' != image.mode:
```

```
            image = image.convert('L')
        pixels = image.tobytes()
        width, height = image.size
    elif 'numpy.ndarray' in str(type(image)):

        if 3 == len(image.shape):
            image = image[:, :, 0]
        if 'uint8' != str(image.dtype):
            image = image.astype('uint8')

try:
            pixels = image.tobytes()
        except AttributeError:
          pixels = image.tostring()
        height, width = image.shape[:2]
    else:
        pixels, width, height = image
        if 0 != len(pixels) % (width * height):
            raise PyZbarError((
                'Inconsistent dimensions: image data of {0} bytes is not '
                'divisible by (width x height = {1})'
            ).format(len(pixels), (width * height))
            )
    bpp = 8 * len(pixels) // (width * height)
    if 8 != bpp:
        raise PyZbarError(
            'Unsupported bits-per-pixel [{0}]. Only [8] is supported.'.format(
                bpp
            )
        )

    return pixels, width, height


def decode(image, symbols=None, scan_locations=False):
    """Decodes datamatrix barcodes in `image`.

    Args:
        image: `numpy.ndarray`, `PIL.Image` or tuple (pixels, width, height)
        symbols (ZBarSymbol): the symbol types to decode; if `None`, uses
            `zbar`'s default behaviour, which is to decode all symbol types.
        scan_locations (bool): If `True`, results will include scan
            locations.

    Returns:
        :obj:`list` of :obj:`Decoded`: The values decoded from barcodes.
    """
    pixels, width, height = _pixel_data(image)

    results = []
    with _image_scanner() as scanner:
        if symbols:
            disable = set(ZBarSymbol).difference(symbols)
            for symbol in disable:
                zbar_image_scanner_set_config(
                    scanner, symbol, ZBarConfig.CFG_ENABLE, 0
                )
            for symbol in symbols:
                zbar_image_scanner_set_config(
```

```
            scanner, symbol, ZBarConfig.CFG_ENABLE, 1
        )
    with _image() as img:
        zbar_image_set_format(img, _FOURCC['L800'])
        zbar_image_set_size(img, width, height)
        zbar_image_set_data(img, cast(pixels, c_void_p), len(pixels), None)
        decoded = zbar_scan_image(scanner, img)
        if decoded < 0:
            raise PyZbarError('Unsupported image format')
        else:
            results.extend(_decode_symbols(_symbols_for_image(img)))

return results
```

5.3 Screen Shots

5.3.1 Capturing Barcode

Figure 6 Capturing Barcode

5.3.2 Details of the Medicine

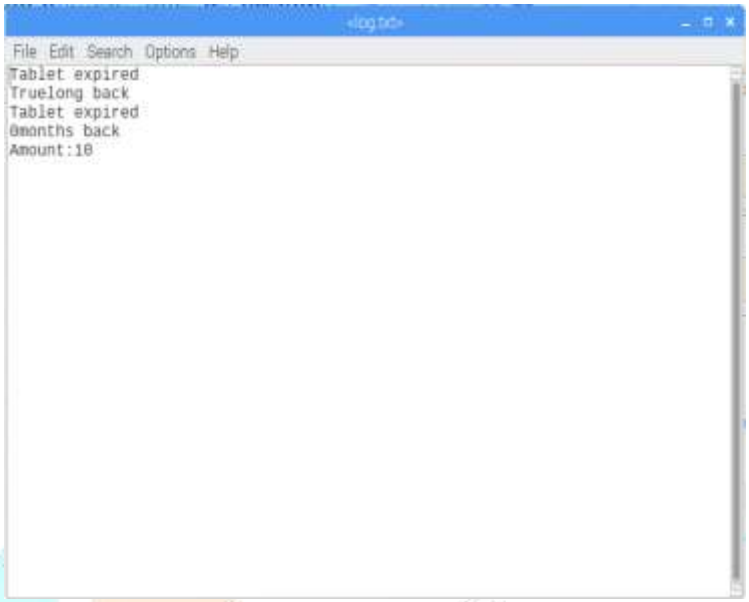

Figure 7 Details of the Medicine

VI. CONCLUSION

As for conclusion, the details within the card can be read out and be able to call out the details from the SDcard module showing necessary information regarding the products. This will help to identify the expired products in the supply chain and to know the stock availability of the inventory. In future work, will try to do this as a smart reader system and to add it for billing.

REFERENCES

[1] T. M. Lehmann, C. Gonner, and K. Spitzer, "Survey: Interpolation methods in medical image processing," *IEEE Transactions on Medical Imaging*, vol. 18, no. 11, pp. 1049-1075, Nov. 1999.

[2] T. Huang and R. Tsai, "Multi-frame image restoration and registration," *Advances in computer vision and Image Processing*, vol. 1, no. 2, pp. 317-339, 1984.

[3] S. Farsiu, M. D. Robinson, M. Elad, and P. Milanfar, "Fast and robust multi-frame super resolution," *IEEE Transactions on Image Processing*, vol. 13, no. 10, pp. 1327-1344, Oct. 2004.

[4] W. T. Freeman, T. R. Jones, and E. C. Pasztor, "Examplebased superresolution," *IEEE Comput. Graph. Appl.*, vol. 22, no. 2, pp. 56-65, Apr. 2002.

[5] H. Chang, D. Yeung, and Y. Xiong, "Super-resolution through neighbor embedding," in Proc. *CVPR*, 2004, pp. 275-282.

[6] D. Glasner, S. Bagon, and M. Irani, "Super-resolution from a single image," in Proc. *ICCV*, 2009, pp. 349-356.

[7] J. Yang, J. Wright, T. Huang, and Y. Ma, "Image superresolution via sparse representation, *IEEE Transactions on Image Processing*, vol.19, no.11, pp. 2861-2873, 2010.

[8] D.H. Trinh, M. Luong, F. Dibos, J.M. Rocchisani,C.D. Pham, and T. Q. Nguyen, "Novel Example-Based Method for Super-Resolution and Denoising of Medical images," *IEEE Transactions on Image Processing*, vol. 23, no. 4, pp.1882-1895, 2014.

[9] S Tang, L. Xiao, P. Liu, et al., "Edge and color preserving single image superresolution," *Journal of Electronic Imaging*, vol. 23, no. 3, 033002, 2014.

[10] Lakshman H, Lim W Q, Schwarz H, et al. , "Image interpolation using shearlet based sparsity priors," *ICIP*, 2013, pp. 655-659.

[11] David R. Hardoon , Sandor Szedmak and John ShaweTaylor, Canonical correlation analysis: An overview with application to learning methods," *Neural Computation*, vol. 16, no. 12, pp. 2639-2664, 2004.

[12] Mairal J, Bach F, Ponce J, Sapiro G, "Online dictionary learning for sparse coding," *ACM International Conference on Machine Learning*, 2009, pp. 689-696.