RASPBERRY PI BASED
IMAGE ACQUISITION SYSTEM

By

Savan Anadani (ID No. 16CEUES065)

Ravi Dayani (ID No. 16CEUES043)

A project submitted

In

partial fulfilment of the requirements

for the degree of

BACHELOR OF TECHNOLOGY

In

Computer Engineering

Internal Guide

Prof. Bhavika Gambhava

Assistant Professor

Dept. of Comp. Engg.

External Guide

Mr. Vishnu Patel

Project Manager

Inst. of Plasma Research

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

April 2020

CERTIFICATE

This is to certify that the project work titled

"**RASPBERRY PI BASED IMAGE ACQUISITION SYSTEM**"

is the bonafide work of

Savan Anadani (16CEUES065)

Ravi Dayani (16CEUES043)

carried out in the partial fulfillment of the degree of Bachelor of Technology in Computer Engineering at Dharmsinh Desai University in the academic session

December 2019 to April 2020.

Prof. Bhavika Gambhava	Dr. C. K. Bhensdadia
Assistant Professor	Head,
Dept. of Computer Engg.	Dept. of Computer Engg.

Faculty of Technology

Department of Computer Engineering

Dharmsinh Desai University

April 2020

Company Certificate

Acknowledgements

We would like to express the deepest appreciation to our external guide Mr. Vishnu Patel, who has the attitude and substance of a genius: he continually and convincingly conveyed a spirit of adventure in regard to research and development of system, and excitement in regard to teaching. Without his guidance and persistent help this system would not have been possible.

Also we would like to thank our internal guide Prof. Bhavika Gambhava, who guides us, comment on our work, give remarks and also take our reporting regularly throughout the project work.

Table of Contents

List of Figures

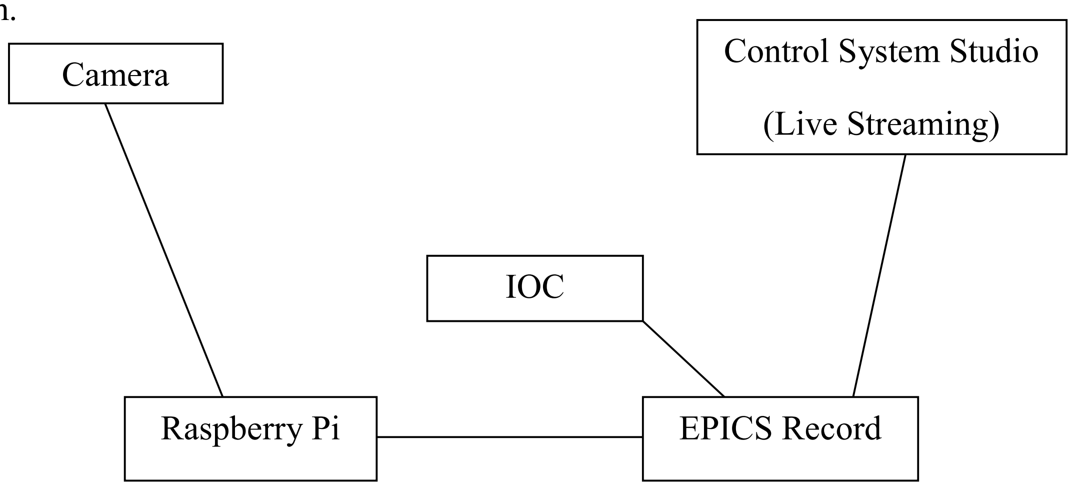
Chapter 1

Introduction

1. Introduction to the system

1.1. System Description:

Raspberry Pi is small and multi-use computer. It is developed by the Raspberry Pi Foundation, and it might be the most versatile tech ever created. Raspberry Pi is used as home security cameras, server monitoring devices, voice-enabled IOT devices and it is also used do robotics. EPICS is set of Open Source software tools, libraries and applications developed collaboratively and used worldwide to create distributed soft real-time control systems for scientific instruments. Raspberry Pi based Image Acquisition System is capturing images using a raspberry pi camera and put that image into EPICS record and this system will be used by the Institute of Plasma Research premises in the area of high radiation to capture images and analyze it. For the implementation of this system, we have use python packages to perform different operation of raspberry pi camera. We have used Control System Studio(CS-Studio) to display image, histogram of that image. CS-Studio is a collection of tools used to monitor and operate large scale control system.



1.2. Purpose:

The purpose of this document is to provide complete description of Image Acquisition System and EPICS. It will explain the purpose and features of the system, what the system will do, where the system will use.

1.3. Document Conventions:

All headings in this document are in bold. And the font used in this document is Times New Roman.

1.4. Assumptions:

Here we assumed that EPICS is installed in Raspberry Pi and operator PC contains Control System Studio through which we can display image.

1.5. Intended Audience and Reading Suggestions:

Developers working on Raspberry Pi and EPICS are advised to read this document.

1.6. Product Scope:

This system will be used by Institute for Plasma Research(IPR) premises to do live streaming of particular areas where humans can not go due to high radiation and various diagnostics will be applied on live stream data. User can do analysis and monitoring of that area where the camera is located.

1.7. Platform/Technology/Tools:

1.7.1. Platform :- Raspbian, Python language, EPICS

1.7.2. Tools/Hardware :- Raspberry Pi

1.8. Acronyms:

EPICS - Experimental Physics and Industrial Control System

IOC - Input/Output Controller

OPI - Operator Interface

CCD - Charged Coupled Device

CMOS - Complementary Metal Oxide Semiconductor

CA - Channel Access

PV - Process Variable

Chapter 2

About the system

2. About the System

2.1. Component of System

2.1.1. EPICS:

The Experimental Physics and Industrial Control Systems (EPICS) is an extensible set of software components and tools with which application developers can create a control system. This control system can be used to control accelerators, detectors, telescopes, or other scientific experimental equipment. EPICS base allows an arbitrary number of target systems, IOCs (input/output controllers), and host systems, OPIs (operator interfaces) of various types. IOCs collect experiment and control data in real time, using the measurement instruments attached to them. This information is then provided to clients, using the high-bandwidth Channel Access(CA), or pvAccess networking protocol. IOCs hold and run a database of records which represent either devices or aspects of the devices to be controlled. There are two type of IOC.

2.1.1.1. Soft IOC:

It runs in the same environment as which it was compiled. It is also called Host-based IOC. It is also possible to have many IOCs on single machine.

2.1.1.2. Target IOC:

It runs in the different environment that it was compiled. It is also called Hard IOC.

Data is held in the database of records are represented by unique identifiers, known as Process Variables(PVs). These PVs are accessible over the channels provided by the Channel Access network protocol.

Database records are available for different types of input and output or to provide different behaviour such as calculation record. It is also possible to create custom record types. Each record consists of a set of fields, which hold its data and specify its behaviour.

2.1.2. areaDetector:

The areaDetector module provides a general-purpose interface for area (2-D) detectors in EPICS. It is intended to be used with a wide variety of detectors and cameras, ranging from high frame rate CCD and CMOS cameras, pixel-array detectors such as the Pilatus, and large format detectors like the Perkin Elmer flat panels. It minimizes the amount of code that needs to be written to implement a new detector. It provides a standard interface defining the functions and parameters that a detector driver must support. It provides a set of base EPICS records that will be present for every detector using this module. This allows the use of generic EPICS clients for displaying images and controlling cameras and detectors. It also Provide a mechanism for device-independent real-time data analysis such as regions-of-interest and statistics.

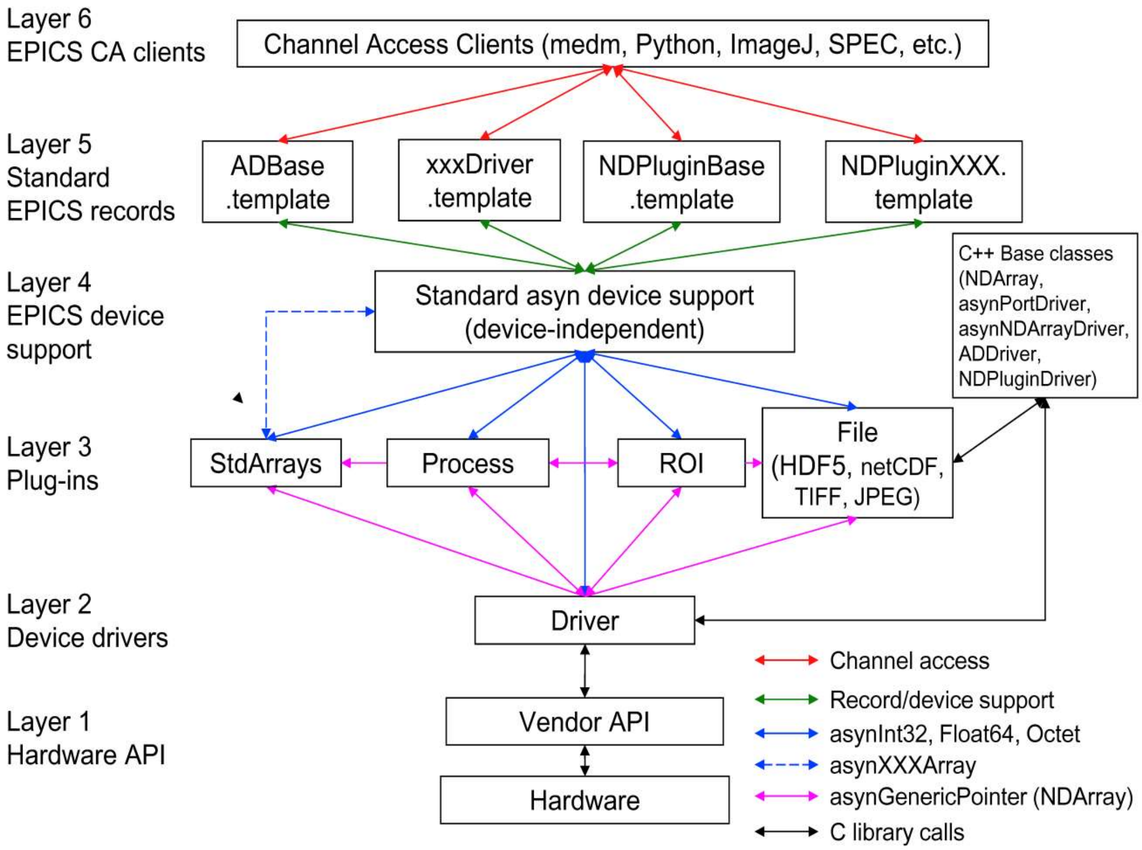


Fig 1. EPICS areaDetector Architecture

There are different drivers of areaDetector:

2.1.2.1. ADCore:

ADCore plugin is core driver of EPICS areaDetector. It includes base classes for drivers and code for all of the standard plugins.

2.1.2.2. NDPluginStdArrays:

NDPluginStdArrays plugin is the tool for converting the NDArray data produced by asynNDArrayDriver drivers into a form that can be accessed by EPICS Channel Access clients. NDPluginStdArrays inherits from NDPluginDriver. NDPluginStdArrays converts the

NDArray data from a callback into the 1-dimensional arrays supported by the standard asyn array interfaces.

2.1.2.3. ADPICam:

It is EPICS areaDetector driver for Cameras from Princeton Instruments that support the PICAM library. PICam This driver is based on the PICAM Virtual Camera Library. It only runs on Microsoft Windows (Vista, 7 & 8) and supports only 64-bit versions of Windows.

2.1.2.4. ADSimDetector:

This is an EPICS areaDetector driver for a simulated area detector. The simulation detector is useful as a model for writing real detector driver. It is also very useful for testing plugins and channel access clients.

2.1.2.5. ADProsilica:

This is areaDetector driver for Gigabit Ethernet and Firewire cameras from Allied Vision Technologies, who purchased Prosilica. The driver is supported under Windows, Linux and Mac OS X using the vendor library provided for those operating systems. This driver inherits from ADDriver. It implements nearly all of the parameters in asynNDArrayDriver.h and in ADArrayDriver.h. It also implements a number of parameters that are specific to the Prosilica cameras.

2.1.3. Control System Studio:

Control System Studio is an Eclipse-based collection of tools to monitor and operate large scale control systems, such as the ones in the accelerator community. CS-Studio contains many tools: Alarm handler, archive engine, as well as several operator interface and control system diagnostic tools. Most of them deal with Process Variable(PV) i.e. named control system data points that have a value, time stamp, alarm state, maybe units and display ranges, but they do this in different ways. One tool displays the value of a PV, one displays details of the PV configuration, while another concentrates on the alarm state of a PV. Each individual tool deserves some attention, and the Experimental Physics and Industrial Control System toolkit, EPICS, indeed offers each functionality as a separate tool. A key point of CSS is the integration of such functionalities.

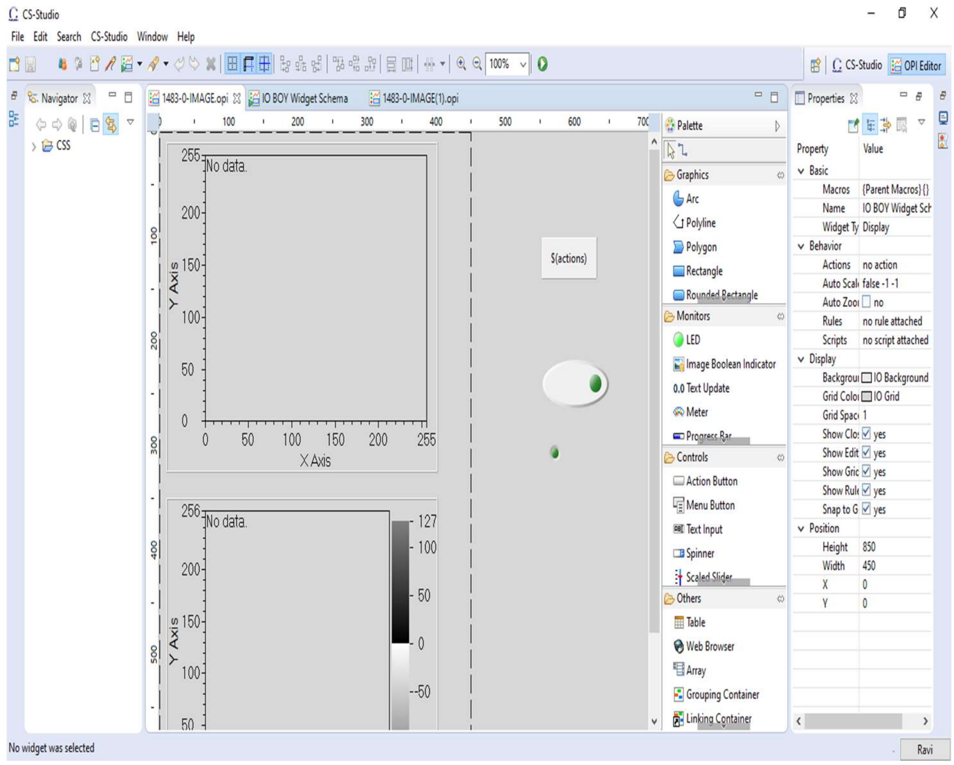


Fig 2. Default view of CS-Studio

2.1.3.1. CSS BOY:

BOY is a one of the component of Control System Studio. CSS BOY stands for Control System Studio Best OPI, Yet. CSS BOY is an Operator Interface (OPI) development and runtime environment. An OPI is a general GUI but with extra facilities to connect to your live data directly. CSS BOY allows building your GUI with drag and drop and connecting to your data instantly. It also allows using JavaScript or Jython to manipulate the GUI in a very similar way as using JavaScript in HTML. We have used CSS BOY widgets to display live streaming of captured image.

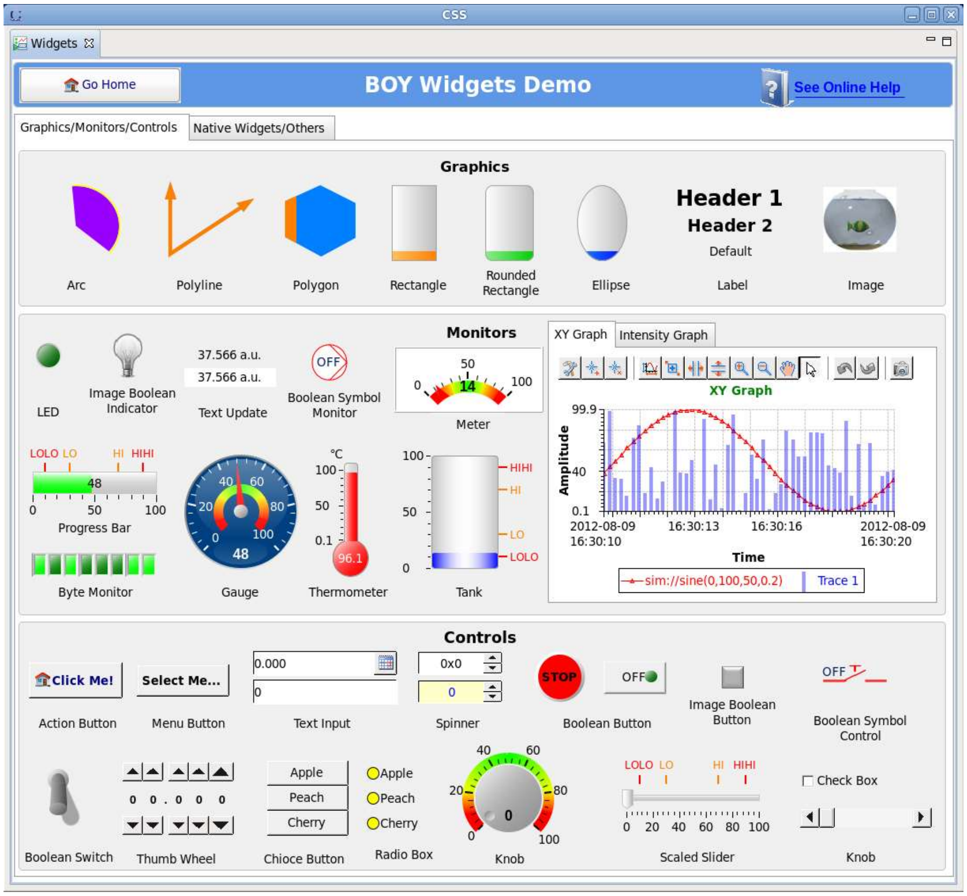


Fig 3. Different widgets of CS-Studio

Chapter 3

Design

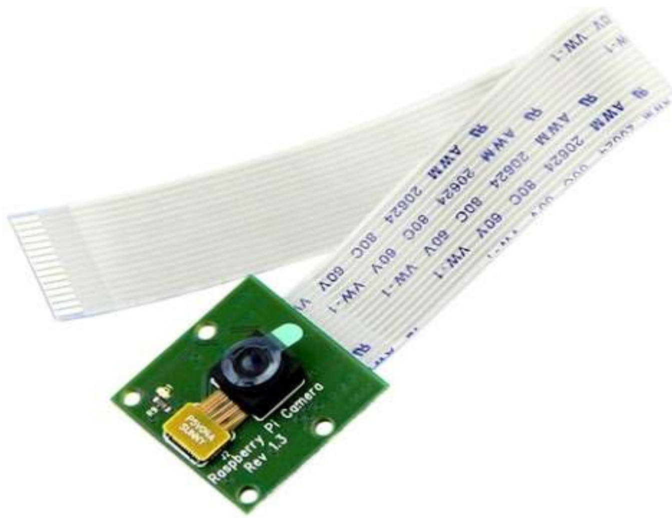


Fig 4. Raspberry Pi Camera

3. Design

3.1. Raspberry Pi:

3.1.1. Camera Module:

The Raspberry Pi camera module is capable of taking full HD 1080p photo and video and can be controlled programmatically.

3.1.1.1. Connecting the camera:

The flex cable inserts into the connector situated between the Ethernet and HDMI ports, with the silver connectors facing the HDMI port. The flex cable connector should be opened by pulling the tabs on the top of the connector upwards then towards the Ethernet port. The flex cable should be inserted firmly into the connector, with care taken not to bend the flex at too acute an angle. The top part of the connector should then be pushed towards the HDMI connector and down, while the flex cable is held in place.

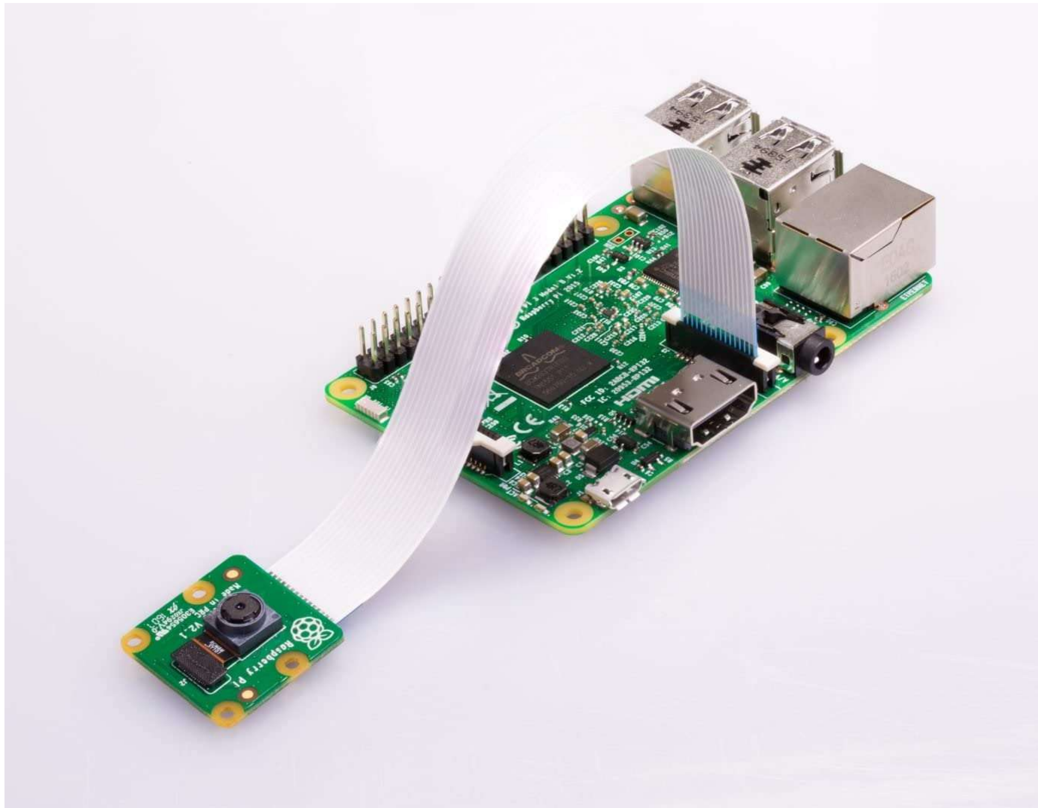


Fig 5. Raspberry Pi with Camera Module

3.1.1.2. Enabling the camera:

Open the raspi-config tool from the Terminal:

sudo raspi-config

Select 'Enable Camera' and hit 'Enter', then go to 'Finish' and we will be prompted to reboot.

3.1.2. Raspbian:

Raspbian is the recommended operating system for normal use on a Raspberry Pi. Raspbian is a free operating system based on Debian, optimised for the Raspberry Pi hardware. Raspbian comes with over 35,000 packages: precompiled software bundled in a nice format for easy installation on your Raspberry Pi. Raspbian is a community project under active development, with an emphasis on improving the stability and performance of as many Debian packages as possible.

3.2. Image Acquisition System:

We can capture image through Raspberry Pi Camera in two ways:

3.2.1. Linux command line:

‘raspistill’, ‘raspivid’ and ‘raspiyuv’ are command line tools for using the camera module.

3.2.1.1. raspistill:

raspistill is command line tool for capturing still photographs with the camera module. We can capture image using following command:

Fig 6. Demo image captured through raspistill

raspistill -o test.jpg

raspistill command provide many option to improve resolution of image, change file size of captured image, flip image in vertical or horizontal direction, change quality of captured image.

3.2.1.2. raspivid:

raspivid is command line tool for capturing video with the camera module. For video capturing enter following command in Terminal:

raspivid -o demo.h264 -t 10000

Here -t specify time in millisecond.

3.2.1.3. raspiyuv:

raspiyuv has the same set of features as raspistill but instead of outputting standard image files such as .jpg, it generates raw unprocessed image files from the camera.

3.2.2. Python:

Python provide picamera package to capture image using raspberry pi camera module. Demo code for capturing image is given as below:

```python
from picamera import PiCamera
from time import sleep
from PIL import Image
import sys
import glob

camera = PiCamera()
camera.start_preview()
sleep(10)
camera.capture('/home/pi/Desktop/Trainee/demo.jpg')
camera.stop_preview()

list_file=glob.glob('/home/pi/Desktop/Trainee/onlyImages/*')
latest_file = max(list_file,key=os.path.getctime)
temp = latest_file[:-4]
count = int(temp[-4:]) + 1

filename = '/home/pi/Desktop/Trainee/onlyImages/final_demo' + '%0004d'%count + '.jpg'
img = Image.open("/home/pi/Desktop/Trainee/demo.jpg")
img = img.rotate(180)
img.save(filename)
```

Python package picamera is also used to capture video using Raspberry Pi Camera module. In video capturing we have to specify time limit. Note that we use wait_recording() in the example above instead of time.sleep() which we've been using in the image capture recipes above. The wait_recording() method is similar in that it will pause for the number of seconds specified, but unlike time.sleep() it will continually check for recording errors (e.g. an out of disk space condition) while it is waiting. Demo code for video capturing through picamera python package is given as below:

```
from picamera import PiCamera
from time import sleep
from PIL import Image

import subprocess
import sys
import numpy as np
import os
import glob

list_file = glob.glob('/home/pi/Desktop/Trainee/video/onlyVideos/*')
latest_file = max(list_file, key = os.path.getctime)
temp = latest_file[:-5]
count = int(temp[-4:]) + 1

dura=int(input("Enter Duration:-"))
camera=PiCamera()
camera.start_preview()
pathname='/home/pi/Desktop/Trainee/video/onlyVideos/demo_video'+ '%0004d'%count + '.h264'
camera.start_recording(pathname)

sleep(dura)
camera.stop_recording()
camera.stop_preview()
```

Chapter 4

Implementation

4. Implementation

4.1. Capturing Image:

With picamera python package, we can get a camera object to control Raspberry Pi Camera module.

```
import picamera

camera = picamera.PiCamera()
```

To generate a camera snapshot, we use capture() method of camera object.

```
camera.capture('snapshot.jpg')
```

4.1.1. picamera.array module:

The picamera.array module provides a set of classes which aid in constructing n-dimensional numpy arrays from the camera output. In order to avoid adding a hard dependency on numpy to picamera, the module is not automatically imported by the main picamera package and must be explicitly imported. The following class is defined in the module.

4.1.1.1. picamera.array.PiRGBArray:

It produces a 3-dimensional RGB array from an RGB capture. This custom output class can be used to easily obtain a 3-dimensional numpy array, organized (rows, columns, colors), from an unencoded RGB capture. The array is accessed via the array attribute. Example code of this is given as below:

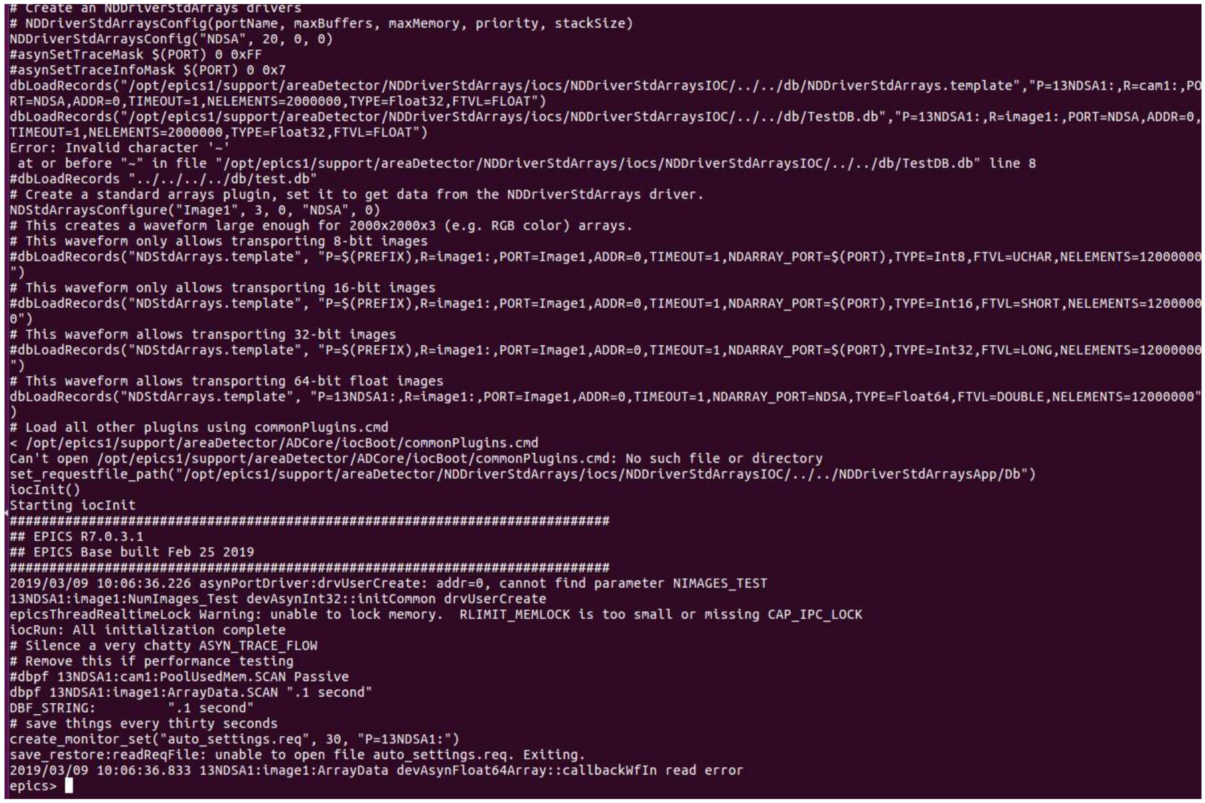
```
import picamera
import picamera.array

with picamera.PiCamera() as camera:
    with picamera.array.PiRGBArray(camera) as output:
        camera.capture(output, 'rgb')
        print('Captured %dx%d image' % (
                output.array.shape[1], output.array.shape[0]))
```

```python
from picamera.array import PiRGBArray
from picamera import PiCamera
from PIL import Image
import epics
import time

camera=PiCamera()
raw=PiRGBArray(camera)
count=0

start=time.time()

while (1):
    camera.capture(raw, format="rgb",use_video_port=True)
    img=raw.array
    img=img.flat
    raw.seek(0)
    raw.truncate()
    count=count+1
    print(count)

end=time.time()

print("Starting time(sec) : "+str(start))
print("Ending time(sec) : "+str(end))
diff_time=int(end)-int(start)
print("total time: "+str(diff_time))
```

4.2. Running IOC:

As its name implies, an IOC often performs input/output operations to attached hardware devices. IOC associates the values of EPICS Process Variables with the results of these input/output operations. IOC can perform sequencing operations, closed-loop control and other computations. Here we have to take input from

Raspberry Pi and store it in Process Variable. Therefore before moving further, we have to start IOC of ADProsilica driver. For that we have to do following steps:



```
# Create an NDDriverStdArrays drivers
# NDDriverStdArraysConfig(portName, maxBuffers, maxMemory, priority, stackSize)
NDDriverStdArraysConfig("NDSA", 20, 0, 0)
#asynSetTraceMask $(PORT) 0 0xFF
#asynSetTraceInfoMask $(PORT) 0 0x7
dbLoadRecords("/opt/epics1/support/areaDetector/NDDriverStdArrays/iocs/NDDriverStdArraysIOC/../../db/NDDriverStdArrays.template","P=13NDSA1:,R=cam1:,PO
RT=NDSA,ADDR=0,TIMEOUT=1,NELEMENTS=2000000,TYPE=Float32,FTVL=FLOAT")
dbLoadRecords("/opt/epics1/support/areaDetector/NDDriverStdArrays/iocs/NDDriverStdArraysIOC/../../db/TestDB.db","P=13NDSA1:,R=image1:,PORT=NDSA,ADDR=0,
TIMEOUT=1,NELEMENTS=2000000,TYPE=Float32,FTVL=FLOAT")
Error: Invalid character '~'
 at or before "~" in file "/opt/epics1/support/areaDetector/NDDriverStdArrays/iocs/NDDriverStdArraysIOC/../../db/TestDB.db" line 8
#dbLoadRecords "../../../../db/test.db"
# Create a standard arrays plugin, set it to get data from the NDDriverStdArrays driver.
NDStdArraysConfigure("Image1", 3, 0, "NDSA", 0)
# This creates a waveform large enough for 2000x2000x3 (e.g. RGB color) arrays.
# This waveform only allows transporting 8-bit images
#dbLoadRecords("NDStdArrays.template", "P=$(PREFIX),R=image1:,PORT=Image1,ADDR=0,TIMEOUT=1,NDARRAY_PORT=$(PORT),TYPE=Int8,FTVL=UCHAR,NELEMENTS=12000000
")
# This waveform only allows transporting 16-bit images
#dbLoadRecords("NDStdArrays.template", "P=$(PREFIX),R=image1:,PORT=Image1,ADDR=0,TIMEOUT=1,NDARRAY_PORT=$(PORT),TYPE=Int16,FTVL=SHORT,NELEMENTS=1200000
0")
# This waveform allows transporting 32-bit images
#dbLoadRecords("NDStdArrays.template", "P=$(PREFIX),R=image1:,PORT=Image1,ADDR=0,TIMEOUT=1,NDARRAY_PORT=$(PORT),TYPE=Int32,FTVL=LONG,NELEMENTS=12000000
")
# This waveform allows transporting 64-bit float images
dbLoadRecords("NDStdArrays.template", "P=13NDSA1:,R=image1:,PORT=Image1,ADDR=0,TIMEOUT=1,NDARRAY_PORT=NDSA,TYPE=Float64,FTVL=DOUBLE,NELEMENTS=12000000"
)
# Load all other plugins using commonPlugins.cmd
< /opt/epics1/support/areaDetector/ADCore/iocBoot/commonPlugins.cmd
Can't open /opt/epics1/support/areaDetector/ADCore/iocBoot/commonPlugins.cmd: No such file or directory
set_requestfile_path("/opt/epics1/support/areaDetector/NDDriverStdArrays/iocs/NDDriverStdArraysIOC/../../NDDriverStdArraysApp/Db")
iocInit()
Starting iocInit
############################################################################
## EPICS R7.0.3.1
## EPICS Base built Feb 25 2019
############################################################################
2019/03/09 10:06:36.226 asynPortDriver:drvUserCreate: addr=0, cannot find parameter NIMAGES_TEST
13NDSA1:image1:NumImages_Test devAsynInt32::initCommon drvUserCreate
epicsThreadRealtimeLock Warning: unable to lock memory.  RLIMIT_MEMLOCK is too small or missing CAP_IPC_LOCK
iocRun: All initialization complete
# Silence a very chatty ASYN_TRACE_FLOW
# Remove this if performance testing
#dbpf 13NDSA1:cam1:PoolUsedMem.SCAN Passive
dbpf 13NDSA1:image1:ArrayData.SCAN ".1 second"
DBF_STRING:          ".1 second"
# save things every thirty seconds
create_monitor_set("auto_settings.req", 30, "P=13NDSA1:")
save_restore:readReqFile: unable to open file auto_settings.req. Exiting.
2019/03/09 10:06:36.833 13NDSA1:image1:ArrayData devAsynFloat64Array::callbackWfIn read error
epics>
```

Fig 7. EPICS IOC Running

4.3. Putting Image in EPICS record:

We have to put the image which is captured using PiRGBArray module of picamera.array in the waveform record(13PS1:image1:ArrayData) of ADProsilica driver of EPICS areaDetector. For that we have to use PyEpics python package to put image in waveform record.

4.3.1. PyEpics:

The python epics package provides several function, modules, and classes to interact with EPICS Channel Access. PyEpics provides many modules: ca, PV, Motor, Device, Alarm. PV module provide object of Process Variable that has methods to read and change the value of PV. The simplest approach uses the functions caget(), caput(), and cainfo() within the top-level EPICS module to get and put values of Epics Process Variables. These functions are similar to the standard command line utilities.

4.3.1.1. caget():

We can get value of Process Variable using caget() function. caget() function has many arguments:

Parameters:

- **pvname** – name of Epics Process Variable.
- **as_string** (True/False) – whether to return string representation of the PV value.
- **count** (integer or None) – number of elements to return for array data.
- **as_numpy** (True/False) – whether to return the Numerical Python representation for array data.
- **timeout** (float or None) – maximum time to wait (in seconds) for value before returning None.
- **use_monitor** (True/False) – whether to rely on monitor callbacks or explicitly get value now.

The count and as_numpy options apply only to array or waveform data. The default behavior is to return the full data array and convert to a numpy array if available. The count option can be used to explicitly limit the number of array elements returned, and as_numpy can turn on or off conversion to a numpy array. The timeout argument sets the maximum time to wait for a value to be fetched over the network. If the timeout is exceeded, caget() will return None. This might imply that the PV is not actually available, but it might also mean that the data is large or network slow enough that the data just hasn't been received yet, but may show up later. The use_monitor argument sets whether to rely on the monitors from the underlying PV. The default is False, so that each caget() will explicitly ask the value to be sent instead of relying on the automatic monitoring normally used for persistent PVs. This makes caget() act more like command-line tools, and slightly less efficient than creating a PV and getting values with it. If performance is a concern, using monitors is recommended. The as_string argument tells the function to return the string representation of the value. Example code of caget() is given as below:

```
>>> from epics import caget, caput, cainfo
>>> m1 = caget('XXX:m1.VAL')
>>> print(m1)
1.2001
```

4.3.1.2. caput():

We can set value of Process Variable using caput() function. caput() function has many arguments:

Parameters:	• **pvname** – name of Epics Process Variable • **value** – value to send. • **wait** (True/False) – whether to wait until the processing has completed. • **timeout** (*double*) – how long to wait (in seconds) for put to complete before giving up.
Return type:	integer

The optional wait argument tells the function to wait until the processing completes. This can be useful for PVs which take significant time to complete, either because it causes a physical device (motor, valve, etc) to move or because it triggers a complex calculation or data processing sequence. The timeout argument gives the maximum time to wait, in seconds. The function will return after this (approximate) time even if the caput() has not completed. This function returns 1 on success, and a negative number if the timeout has been exceeded. Example code of caput() is given as below:

```
>>> caput('XXX:m1.VAL', 1.90)
>>> print(caget('XXX:m1.VAL'))
1.9000
```

4.3.1.3. camonitor():

This function sets a monitor on the named PV, which will cause something to be done each time the value changes. By default the PV name, time, and value will be printed out (to standard output) when the value changes, but the action that actually happens can be customized.

```
>>> from epics import camonitor
>>> camonitor('XXX.m1.VAL')
XXX.m1.VAL 2010-08-01 10:34:15.822452 1.3
XXX.m1.VAL 2010-08-01 10:34:16.823233 1.2
XXX.m1.VAL 2010-08-01 10:34:17.823233 1.1
XXX.m1.VAL 2010-08-01 10:34:18.823233 1.0
```

4.4. Displaying Image in CS-Studio:

We have to run opi file in CS-Studio editor and then using action button we have to set ADAcquire record to one. After that graphical interface of opi file will continuously display image. We can also stop image acquisition using action button.

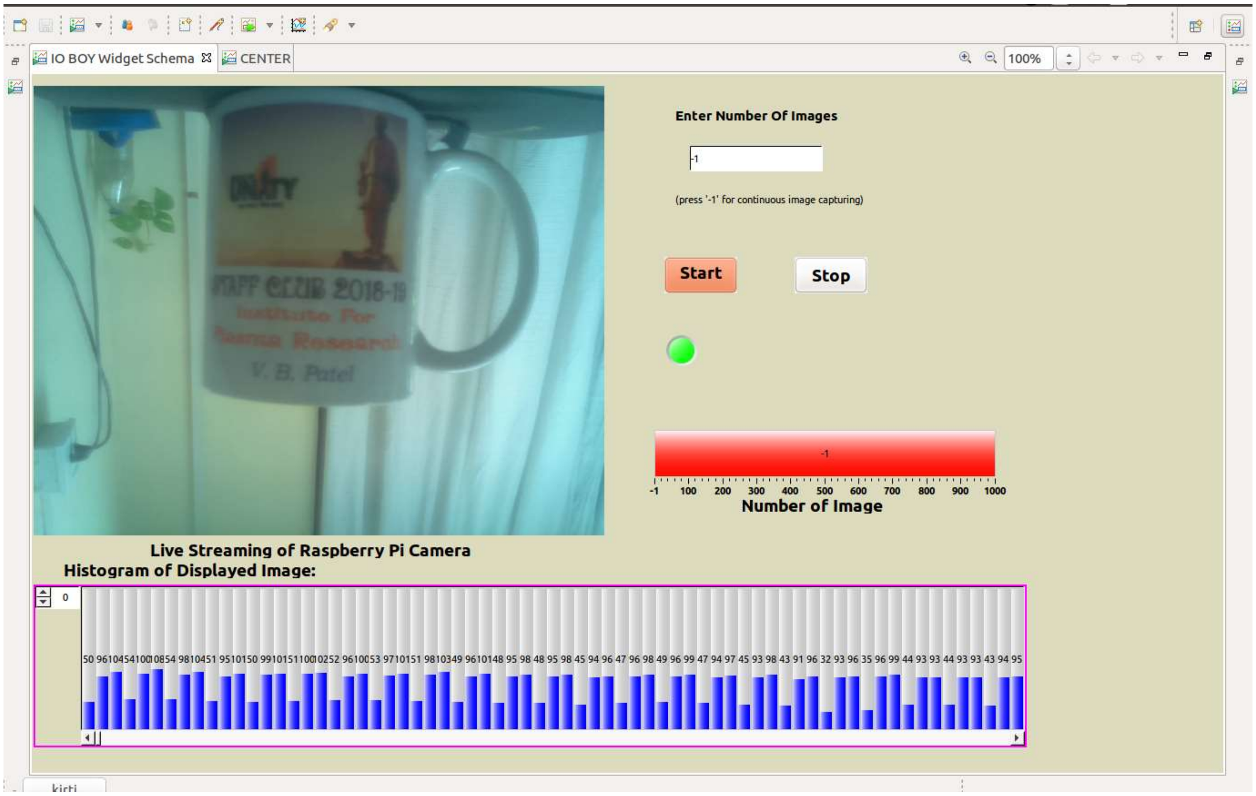


Fig 8. Display Image in CS-Studio

Chapter 5

Testing

5. Testing

Test Case ID	Test Scenario	Test Steps	Test Data	Expected Result(s)	Pass or Fail
T01	Initialising camera object	Assign the picamera.PiCamera() object to local variable	Raspberry Pi Camera Information	Camera handler	Pass
T02	Capturing Image	Call capture() function of camera object	Empty Numpy array	Image is captured as numpy array(3-dimensional array)	Pass
T03	Putting Image in EPICS waveform record	Call caput() function of pyepics	Name of Waveform record and numpy array	numpy array is put in waveform record	Pass
T04	Displaying Image in CS-Studio	Set ADAcquire record to one through action button of opi file	ADAcquire record	Image is displayed in opi file	Pass
T05	Stop image displaying in CS-Studio	Set ADAcquire record to zero through action button of opi file	ADAcquire record	Image displaying is stopped	Pass
T06	Run IOC(Input/ Output Controller)	Compile ADProsilica driver and run IOC using st.cmd file	Running IOC and EPICS records	IOC is running and EPICS records are in ADProsilica driver are ready to use	Pass

T07	Continuous streaming of captured image	Enter -1 in number of image field and start acquisition	Image Streaming in Intensity graph widget of control system studio and LED	Continuous streaming is started and LED is ON	Pass
T08	Streaming of limited number of images	Enter positive number in number of image field	Image Streaming in Intensity graph widget of control system studio and LED	Number of images are streamed and while streaming LED is ON and after that it will be OFF.	Pass

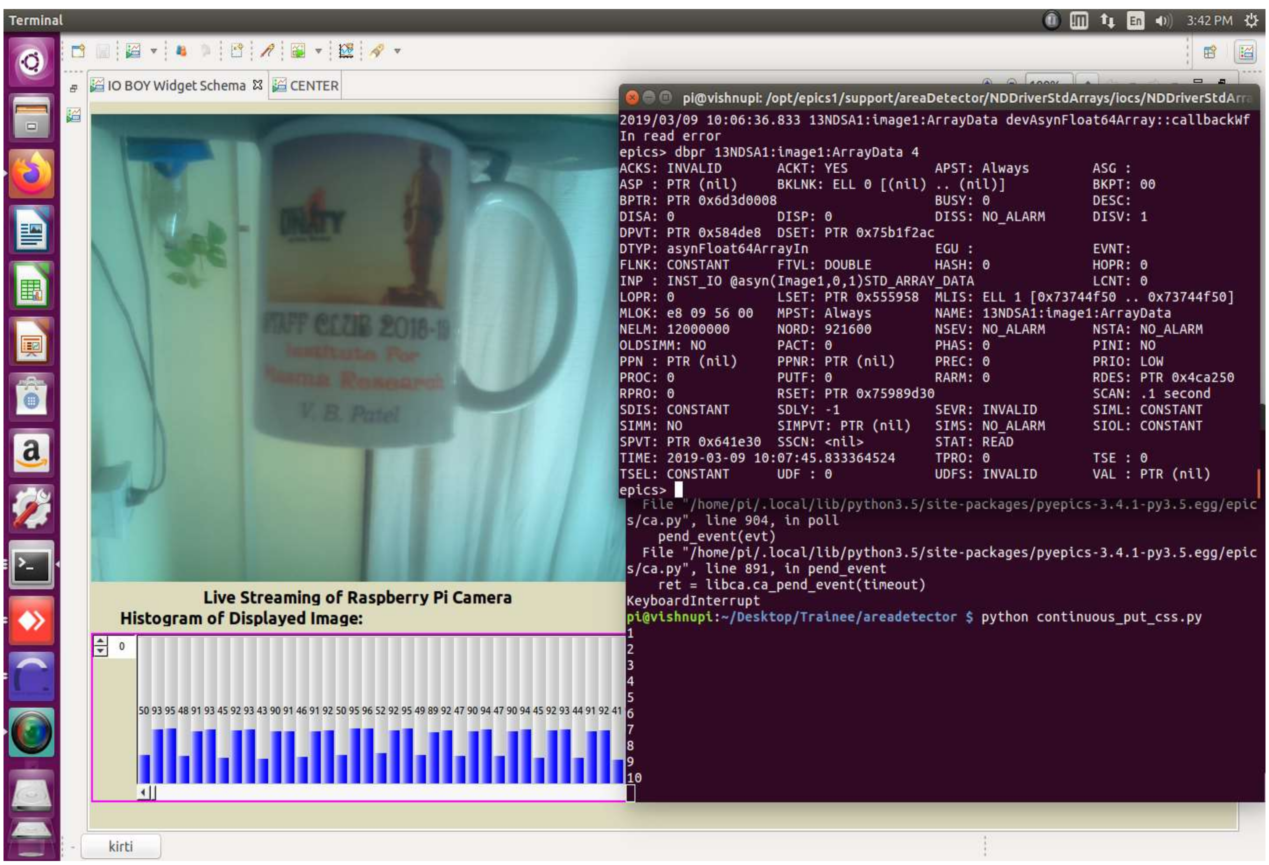


Fig 9. Test Case 2. Capturing Images

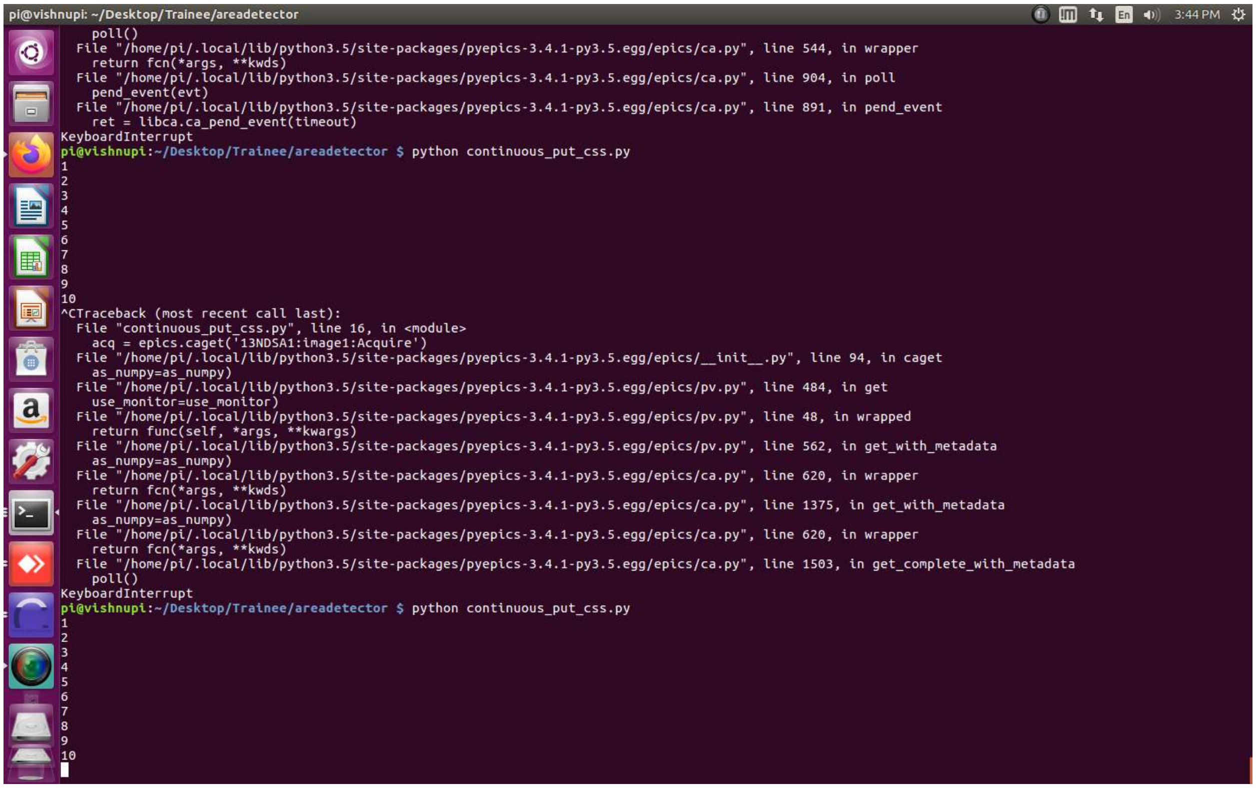


Fig 10. Test Case 3. Putting Images in EPICS waveform record

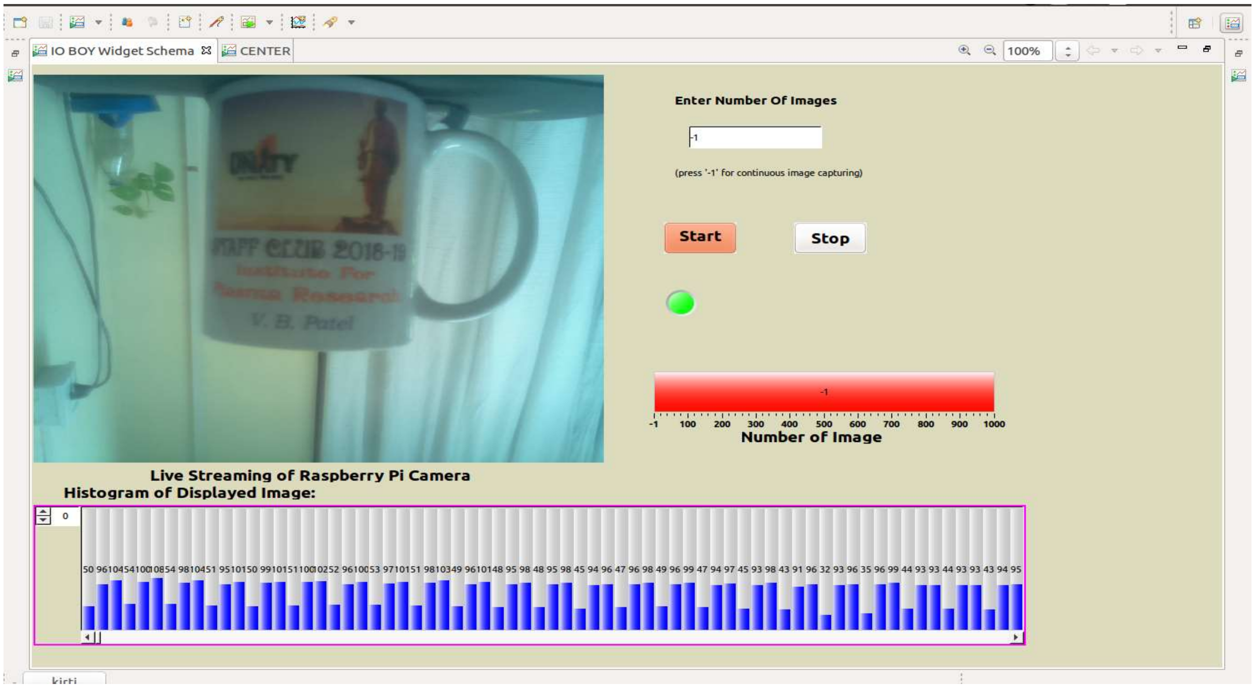


Fig 11. Test Case 4. Displaying Images

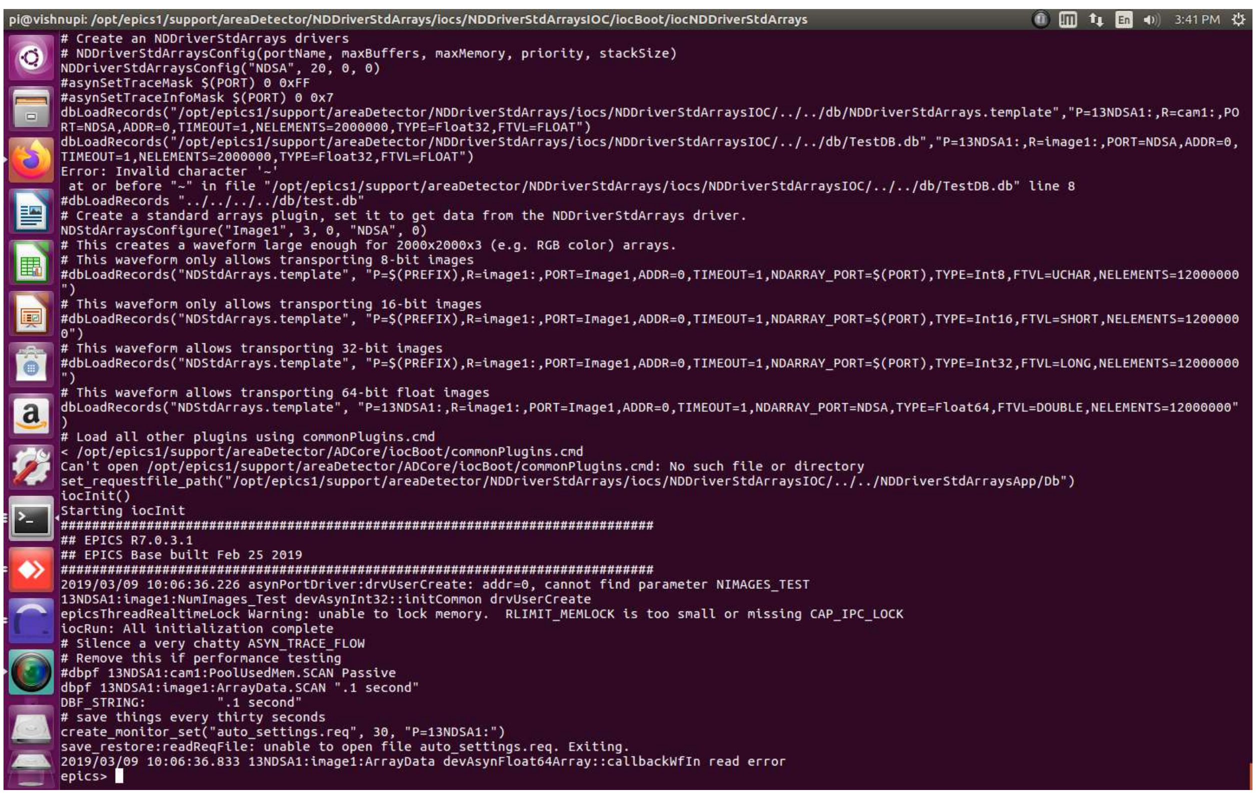


Fig 12. Test Case 5.Run IOC

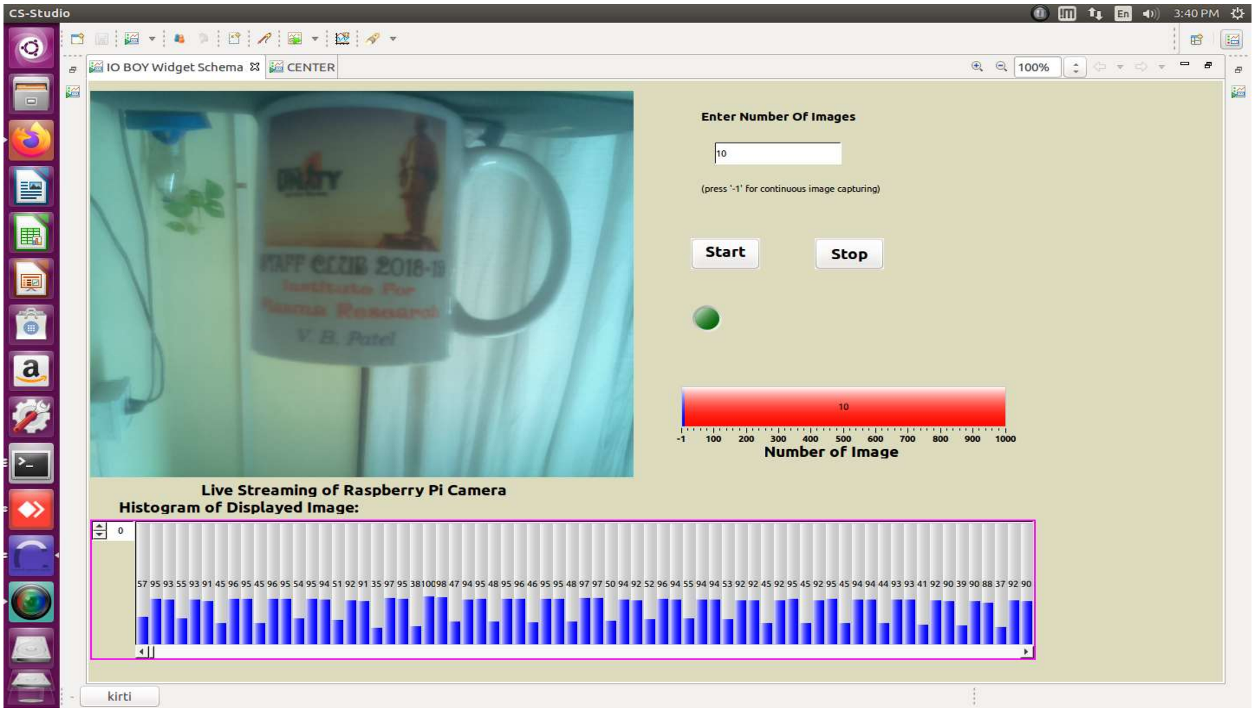


Fig 13. Test Case 8. Streaming of limited number of Images

Chapter 6

Conclusion and future extension

6. Conclusion and future extension

6.1. Conclusion:

Raspberry Pi is a capable little device that enables people of all ages to explore computing, and to learn how to program in languages like Scratch and Python. It's capable of doing everything we would expect a desktop computer to do, from browsing the internet and playing high-definition video, to making spreadsheets, word-processing, and playing games. Raspberry Pi has the ability to interact with the outside world, and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras. EPICS has been successfully applied to many application: data acquisition, supervisory control, closed-loop control, sequential control and data archiving.

To conclude, Raspberry Pi based Image Acquisition System is useful for monitoring and analyzing in high radiation area where human can not go easily. It has same mechanism as many applications which are used for live streaming nowadays but through this system client can not only monitor but control the system without moving from his/her position.

6.2. Future Extension:

Future work in this project involves object recognition in captured image, process capture image for scientific use and improve speed of image putting in EPICS areaDetector record. Speed of live streaming can be improved by storing image into areaDetector record from ADProsilica driver.

Bibliography

1) Control System Studio Guide by Kay Kasemir and Gabriele Carcassi (Oak Ridge National Laboratory).

2) EPICS Architecture by L.R. Dalesio, M.R. Kraimer, A.J. Kozubal (Argonne National Laboratory, Los Alamos National Laboratory).

3) Raspberry Pi (www.raspberrypi.org)

4) EPICS areaDetector by Mark River