



TECHNISCHE UNIVERSITÄT  
CHEMNITZ

# **Development of Streaming Solution for Live-Video from Mobile Robot**

## **Automotive Software Engineering Research Internship**

Fakultät Informatik  
Professur Technische Informatik

Fakultät für Elektrotechnik und Informationstechnik  
Professur Prozessautomatisierung

Masters Automotive Software Engineering

Supervised by: Prof. Dr. Wolfram Hardt  
Prof. Dr.-Ing. Peter Protzel  
Dr.-Ing. Sven Lange

Submitted by: Aditya Gollapinni Manjunath, Matrikel-Nr. 329543  
Email: [maadi@hrz.tu-chemnitz.de](mailto:maadi@hrz.tu-chemnitz.de)

## **Abstract**

With the advance in technology, comes various constraints on systems and security of systems. Also, it feels good to have a host of applications that make tasks easier for the organisation and the person responsible for the system and its maintenance.

Once such technology that has advanced over the past few years is the transmission of data from a source to another receiver. From achieving this over short distances using a transmission cables and wire, to doing the same over long distances and wirelessly, the complexity and efficiency of such systems has increased manifold. There are two kinds of transmissions of data, digital and analog. While digital transmission is the communication of discreet messages, analog is the transmission of data is a transfer of continuously varying analog signals. Conversion of data from digital to analog and vice-a-versa happens and is used quite often. Analog transmission is mostly used for wireless transmission of images, video, voice, data and signal. It uses a continuous signal with varying amplitude or any other property that differentiates the variable. Other methods of transmission include wired, fibre-optic cable and water.

The Chair of Automation Technology at the Technische Universität Chemnitz in Chemnitz, is developing a technology to enable a real time transmission of images/video from an Autonomous Mobile Robot to a stationary computer. In this project, the system is developed using a Raspberry Pi embedded computer, Mobile Robot and an Ubuntu PC. Also, H264 encoding is used for the encoding of the video. The capture-frames per second can be set to the User's liking, but in this project a standard capture speed is used. The video frames or images go through WiFi and reach the computer.

There is a need for wireless transmission of images and videos, which constraints on the network infrastructure, and also costs. The technology designed in this academic project fits the constraints and is affordable. Typical embedded computers are either large, or unaffordable or both. The Raspberry Pi embedded computer is the size of a credit card, and is well equipped with standard connection ports, including USB and Ethernet, which has been used to the projects advantage. For the capture of video frames, the camera designed for the considered embedded computer is used. The system was first designed using a Virtual Machine, a basic code was implemented to capture video. The second phase saw the basic code in the first phase implemented on an emulator of the embedded computer i.e. the Raspberry Pi Model-B. In the final phases the complete system was developed to work on the embedded computer.

# Contents

1) Task Analysis.....	5
2) System Concept.....	6
3) System Functionality.....	7
i) Video Frame Capture.....	7
ii) Operating Principle, Transmission and Reception.....	7
iii) Communication Protocol.....	7
iv) Programming Environment.....	8
4) GStreamer.....	9
i) Communication in GStreamer.....	9
5) Multi-Media Abstraction Layer(MMAL) .....	11
i) MMAL Components.....	10
ii) Component Ports.....	10
iii) Buffers.....	11
6) Alternatives and Other Software.....	11
i) MATLAB and Simulink.....	11
ii) Other Libraries and Tools.....	12
7) Hardware and Communication Framework.....	13
i) Hardware Used.....	13
ii) Communication Framework.....	13
8) Design and Implementation of Video Capture.....	15
i) Camera Driver.....	15
ii) Video Capture Functionality.....	15
iii) Encoder Component.....	15
iv) User-Defined Parameters.....	17
9) Initial Setup.....	17
i) Initial System and Results.....	17
ii) Changes Adopted.....	18
10) System Design.....	19
i) System Setup.....	19
ii) Real-Time Protocol.....	20
iii) Server-Side System.....	22
iv) Client-Side System.....	24
11) Analysis and Results.....	26
SUMMARY.....	31
X APPENDIX.....	34
BIBLIOGRAPHY.....	42
NOMENCLATURE.....	43

## List of Diagrams and Figures

Fig 2.1 System Concept.....	6
Fig 4.1 GStreamer Framework.....	9
Fig 6.1 Simulink Video Streaming.....	13
Fig. 8.1 Video Capture Process .....	16
Fig 8.2 User Defined Parameters.....	17
Fig 9.1 Initial Video Stream .....	18
Fig 9.2 Rate vs. Frames Rendered.....	19
Fig 10.1 System Design.....	20
Fig 10.2 Server RTP Session.....	23
Fig 10.3 Server GStreamer Pipeline.....	24
Fig 10.4 Client RTP Session.....	25
Fig 10.5 Client GStreamer Pipeline.....	26
Fig 11.1 Raspberry Pi Processor Usage.....	27
Fig 11.2 PC Network History before Transmission.....	27
Fig 11.3 PC CPU Usage.....	28
Fig 11.4 PC Network History after Network Establishment.....	28
Fig 11.5 PC Network History during Process.....	28
Fig 11.6 CPU Usage during Process.....	29
Fig 11.7 Video Stream.....	29
Fig 11.8 Frame Rate vs Rendered Frames.....	30
Fig 11.9 Frame Rate vs Rendered Frames 720x480.....	30
Fig 11.10 Frame Rate vs Rendered Frames 1280x720.....	31
Table 11.11 System Scenario.....	31
Fig. Edimax Setting 1.....	35
Fig. Edimax Setting 1.....	36
Fig. RTPbin 1.....	40
Fig. RTPbin 2.....	41

# 1

## Task Analysis

The statement for this Research Internship is as follows: The task involves Autonomous Mobile Robot, which is to be equipped with the considered Raspberry Pi Model B embedded computer and its corresponding camera device, and a computer (PC), which is the ground station. Firstly, the video frames must be captured on the embedded computer, compressed into a buffer. This is to take place locally on the Autonomous Mobile Robot. The establishment of a communication network follows, between the embedded computer and the PC. This communication network handles the data (video/image frame) transfer between the Mobile Robot and the PC. Finally, the data is transferred over the channel and displayed on the PC.

This operation is initialised on the PC. The software technology used in building this system is designed such that it is initialised on the PC. The following functions are to be achieved in this project:

- Function to capture video frames from the camera attached to the embedded computer
- Functions to process these video frames: count the number of video frames per second and compress them into any one of the many video buffers created, so that there is no loss of frames.
- Along with, the above the software also includes certain user controlled functions, which can modify the image as required, or continue with default parameters.
- Finally, transmit the frame to the PC to be displayed

At the PC, the following is implemented

- Retrieve the frame and display it
- Manage the wireless connection established with the embedded computer on the Mobile Robot

The above functionalities will be elaborated in the following sections.

## 2

### System Concept

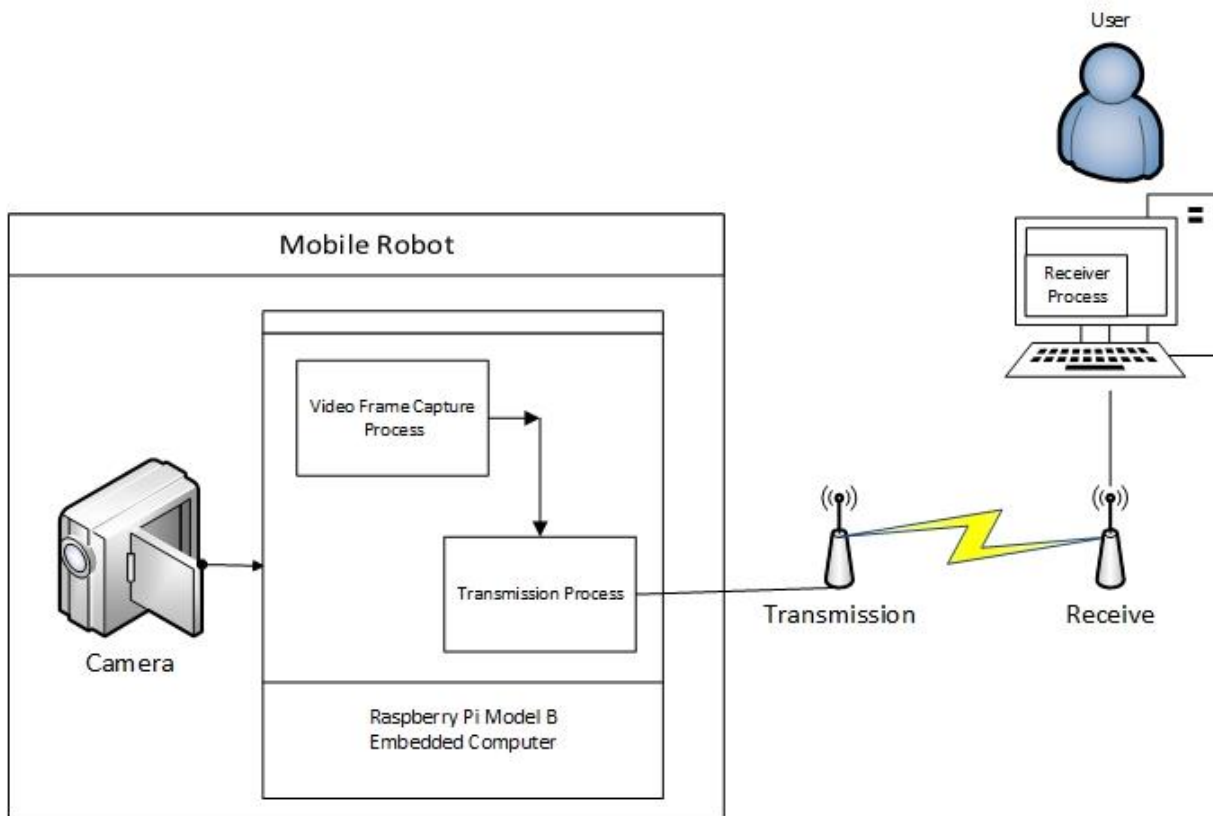


Fig. 2.1 System Concept

The above diagram displays the system setup. The Mobile Robot [10] holds the Raspberry Pi [3] embedded board with the RaspiCamera [7] attached to it. The Raspberry Pi board has the Raspbian Operating Systems [11] installed on it. Attached to the Raspberry Pi is an Edimax EW-7811Un [12], 150Mbps USB adaptor. This USB adaptor is configured to broadcast network, which essentially makes the Raspberry Pi a hotspot [see appendix]. GStreamer1.0 [1] is installed on the Raspberry Pi [see appendix]. The Ubuntu PC itself has GStreamer1.0 installed on it [see appendix].

## 3

### System Functionality

#### 3.1 Video Frame Capture Process

The capture function runs on the embedded computer located on the Mobile Robot. The driver for the video capture is written using the built-in GStreamer and MMAL capture structures. The capture structures are written for any hardware with an image or video capturing ability. The capture structures are configured for Raspberry Pi Camera, which is customised for the Raspberry Pi. The driver includes functionalities for initialisation, capture and buffer actions, among other functions. The system achieves a capture of a stream of video frames at a desired frames per second(fps). It will be stored on the Raspberry Pi and then transmitted to the computer.

#### 3.2 Operating Principle, Transmission and Reception

Streaming technology invariably refers to the technique where data is exchanged between systems. Owing to the large size of data it is difficult to transmit data in one go. Hence, it is cut up into smaller portions and transmitted. A similar technology is used in this project. The principle is to compress the data and then send it over the network. [9]

Transmission of frames depend on the GStreamer design and concept of pipelines. GStreamer contains classes and objects, and an *element* is the most important class of objects. A chain of elements enables dataflow and this is how programming is usually achieved in GStreamer. Each *element* has one specific function, for instance, reading of data. A *pipeline* is a chain of such elements. Therefore, a *pipeline* performs a specific task, such as media playback or media capture. The pipeline architecture of GStreamer streamlines the operation of collection of frames and transmitting them to the receiver at the observer side. The data flowing in a pipeline consists of both buffers and events.

Once the frames are collected and are compressed into the buffer, the transmission from the Mobile Robot achieves the following:

- The frames are sequenced and compressed into the buffers. There are more than one buffer running so as to avoid any loss of frames.
- GStreamer classifies the beginning and the end points as the source and sink. Along with the mention of these in the pipelines, the system can achieve real time transmission of the frames.

The receiver function at the user end or the *sink* of GStreamer performs the following

- Receive the image frames and decompress them and display them for viewing
- Constantly check for connection with the source i.e. the transmitter on the Mobile Robot

### **3.3 Communication Protocol**

Due to the nature of this application and the technologies developed, the Mobile Robot and the PC need to communicate wirelessly. To avoid any loss of frames, the application uses more than one buffer, but for the effective communication of these packets there is a need for strong communication channels. As the communication is wireless, the choices available are WiFi Hotspot, Bluetooth Personal Area Network, Zigbee or GSM. Zigbee and GSM have very low data rate, which is impractical for this project, as it requires high data rate for video transmission. Although Bluetooth Personal Area Network could achieve the data rate desired, its communication distance is restricted and hence undesirable in this project. The only other option, which satisfies the criteria is WiFi Hotspot. With WiFi Hotspot the communication receives a good data rate and communication over required distances.

To handle the transmission of data, implementation of standard transmission procedures or protocols are needed, which are Transmission Control Protocol and User Datagram Protocol. There are many other streaming protocols RTP, RTSP and session description protocols. However this project has used the Real time transport protocol (RTP). RTP provides services to deliver data with real time features. While RTP itself does not provide any technique to transport the data in time, it uses underlying protocols such like UDP, TCP/IP, etc. The favoured protocol is UDP, as data is delivered faster. TCP/IP does not guarantee the timeliness of the data delivery. In the same position, TCP/IP is highly reliable with delivery of data, but UDP does not guarantee reliability. RTP is primarily employed to stream H.264 and MPEG-4 video, which worked as an advantage as a H.264 video is streamed.

### **3.4 Programming Environment**

The programming language used for this project is C language. It was chosen as all the libraries and the techniques are optimised for C programming. Many host libraries were used in this project. Programming was done in Ubuntu Linux environment using Codeblocks and Notepad++ for simplicity and easy access. Kate was used in the initial stages, but owing to increasing complexity of the code, alternatives were used. Programming in Windows 7 environment was done in Visual Studio 2010, when unable to program Ubuntu Environment. The source programs and the libraries were built using GNU gcc (GNU Compiler Collection for C). The libraries used in this project are GStreamer SDK and Multi-Media Abstraction Layer libraries.



## 4

### GStreamer

GStreamer is a framework to create streaming applications. The GStreamer framework is designed to write applications that handle video and audio, but this application will focus only on the video part of it. GStreamer provides APIs for multimedia application, a pipeline architecture, able to handle media type, and also audio and video protocol plugins. Variety of formats like muxers, demuxers, parsers, codecs-good and bad, and sinks for both audio and video.

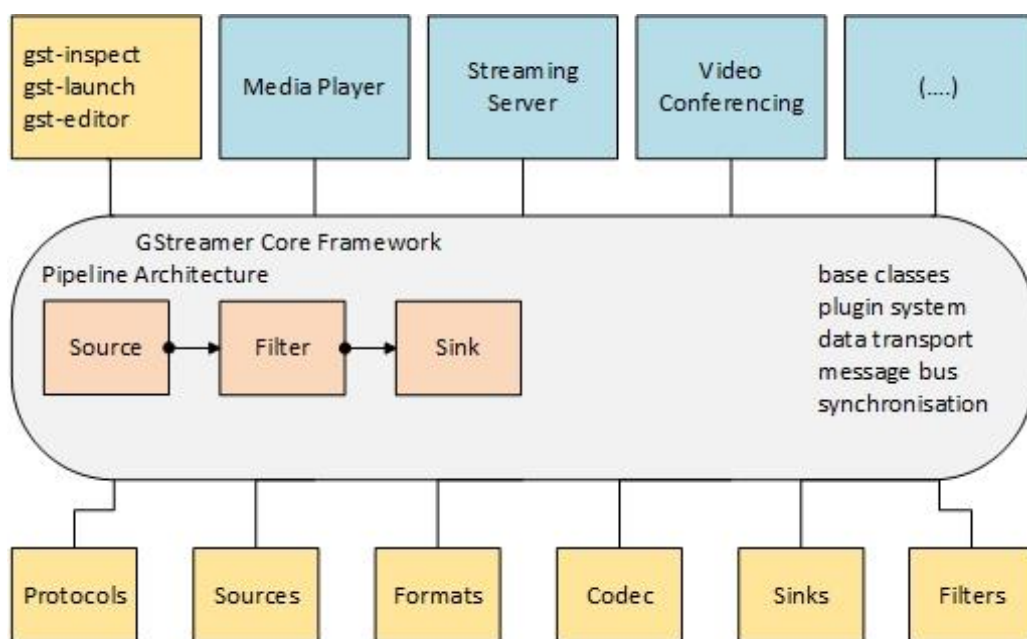


Fig. 4.1 GStreamer Framework

GStreamer is also object oriented and it is extensible using the GObject inheritance methods. Also, while streaming, the threads running on the stream are dedicated threads. For most of the part of GStreamer, data flows in one direction. *Elements* are classes of objects and each element performs one specific function, and *pads* are connections between the elements. Data flows from one element to another through the *pads*. There are different types of elements, *source*, source of stream of data. *Sinks*, the destination of the data stream and *filters*, the manipulators of the stream of data.

#### 4.1 Communication in GStreamer

Communication in GStreamer occurs in many ways. Communication is enabled in the following methods

- *Buffers* enable data exchange between elements within the pipeline. They enable streaming of data, both video and audio, in one direction.

- *Events* are objects that are exchanged between elements or between application and elements. Events can flow in both directions.
- *Messages* are objects exchanged between the elements and application. They are posted on the pipeline bus by the elements. Messages are handled separately by the application's main thread, and typically inform the application regarding errors, changes in state, changes in buffering state, etc. safely.
- *Queries* enable application to receive information about the stream from the pipeline. Queries typically concerned with playback situations and duration of the stream. Also, elements can request queries from the application, for instance, elements could request for the size of file or the duration, etc.

This project uses the GStreamer1.0 library and its plugins. The video is captured at the source and moves to the sink. The Raspberry Pi is the server and the computer connected to it is the client.

## Multi-Media Abstraction Layer

Multi Media Abstraction Layer (MMAL) [13] provides a framework for applications to be built on VideoCore. It provides an interface to the multimedia components running on VideoCore. New components can be added to the existing framework. MMAL is also compatible with non-VideoCore devices. MMAL is used in this project for the main reason, that it simplifies programming and as mentioned before, integrating new components onto the existing framework is not difficult. It provides prebuilt interfaces for buffers, which is very important for this project.

The concept of MMAL is based on components, ports and buffers. The components created by the client have ports attached to them. There is a port for each elementary flow of data, i.e. there are individual ports for audio and video. These ports are also used for input of data and output of data and they are independent of each other. For the processing of data, the components are attached to buffer headers.

### 5.1 MMAL Components

Components are created from MMAL API [13]. Components and clients exchange data using buffers. Buffer header pointers to the payload data are created. Buffer headers are sent to and received from ports attached to the components and client.

In this project, the following Components were created:

```
MMAL_COMPONENT_T *camera;
```

```
MMAL_COMPONENT_T *encoder;
```

The components were created by calling `mmal_component_create` with the name of the component, which will return an `MMAL_COMPONENT_T` context and expose the ports connected to these components.

### 5.2 Component Ports

A port exposes the component to an elementary stream. The port is able to process a stream as an elementary stream. Also buffer headers are attached to the ports. The format of the ports differ component to component. In this project, the following ports to the components were created:

```
MMAL_PORT_T *camera_video_port;
```

```
MMAL_PORT_T *camera_still_port;
```

```
MMAL_PORT_T *encoder_o_port;
```

## 5.3 Buffers

Buffer headers are used to exchange data between components and clients. The buffers point to the data being transmitted and do not contain any data in themselves. The reason for separating the buffers from the data transmitted is to ensure that, even if the data is allotted memory outside of MMAL, i.e. data is received from another library, the data will be handled the way they need to be handled.

While in this project, two buffers were created, and a pool of buffers was also created. Along with the above two buffers, buffers are created whenever needed for *callbacks*.

The pool of buffers was created, so that the buffers once empty will be recycled. There is also a possibility for the buffers to handle metadata, such as the features of the still of video being shot, for example, the exposure time, the brightness and sharpness of the frame.

## 6

## Alternatives and Other Software

### 6.1 MATLAB and Simulink

MATLAB is a computing environment and programming language. It is mainly used for plotting of graphs under various paradigms, also provides interfacing with many well established programming languages- C, C++, Fortran, etc. Its various *toolboxes* provide additional interfacing to design electronic and software systems. Along with Simulink, the simulation platform of MATLAB which allows the programmer to design systems, program graphically and test a design before implementation, and also interface with many hardware boards and processors, it has become a very useful tool for scientific analysis.

In this project, MATLAB and Simulink 2013b [4] tools were used initially access the Raspberry Pi board remotely. For this, MATLAB provides a separate toolbox, called Raspberry Pi toolbox, also executable from Simulink. The configuration of the Raspberry Pi, i.e. installation of the operating systems, the programming environment and mainly the configuration of the Raspberry Pi board as a Hotspot, so that, a network is established between the Ubuntu PC and the Raspberry Pi for the exchange of data.

Apart from the above, Simulink also provides various functions. From within Simulink, a UDP connection is established, to send and receive, with the Raspberry Pi and also capture images and videos. Audio capture and playback functionalities are also available in Simulink. Even simple glowing of LEDs can be configured using Simulink.

This alternative, apart from the basic configuration, was not considered for the completion of this project. The reasons being that, MATLAB versions 2013a and below do not support the hardware in consideration and the Ubuntu PCs do not have the supported versions of 2013a and above installed. Also, MATLAB itself is expensive and not affordable by everyone.

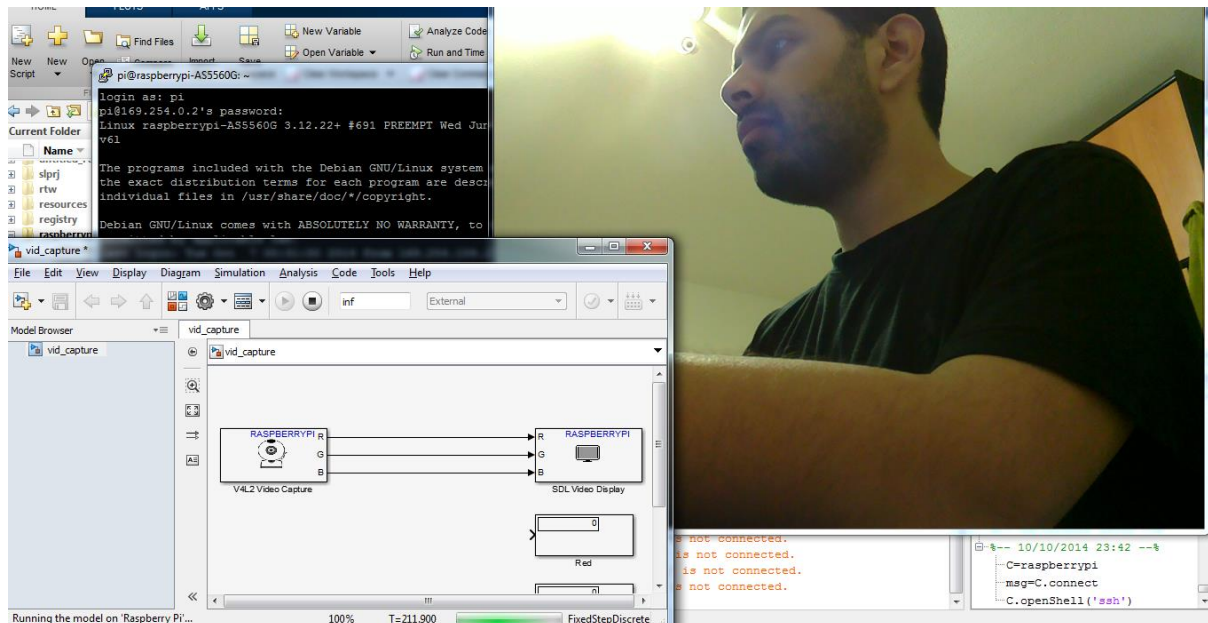


Fig. 6.1 Simulink Video Streaming

The above screenshot depicts the result of streaming from Simulink Raspberry Pi library. The resulting stream of video is clear and the delay is less than 200ms. Apart from the above, the Simulink library also provides an Inversion Algorithm. These functionalities may be used for Image/Video processing.

## 6.2 Other libraries and tools

OpenCV [15] provides libraries to perform computer vision tasks and analysis on images and video frames. While OpenCV could have been used in this project, its need was unnecessary. As this project concentrates on the grabbing on images/video frames and their delivery in a very short amount of time, GStreamer alone accomplishes this task efficiently enough.

LIVE555 Streaming Media [14] is an alternative software library available for multimedia streaming. It implements the standard protocols-RTP/RTCP, RTSP, SIP. It provides numerous C++ APIs. The disadvantage in using LIVE555 streamer on the Raspberry Pi is that it does not support H.264 format and the streamer invariably uses VLC media player which require additional software.

Ubuntu 12.04 provides Kate as a programming environment. Kate was initially used for programming, but owing to the unfriendly nature of the tool, Codeblocks was used. While not working on Ubuntu 12.04, programming was done on Notepad++ running on Windows 7 and as GStreamer is also available for Windows OS, Microsoft Visual Studio 10 was used. To generate documentation, Doxygen and Lyx were used. To generate other diagrams Microsoft Visio was used.

## **7**

### **Hardware and Communication Framework**

#### **7.1 Hardware Used**

The system, described in the previous chapters, is implemented on a Raspberry Pi Model B embedded computer. The camera, used to grab images/frames, is the Raspberry Pi camera module. The hardware is inexpensive and its various ports allow the programmer to achieve the system goals without much difficulty.

#### **7.2 Communication Framework**

The communication framework used in this project is GStreamer. GStreamer is designed on a plugin based architecture. Hence the flexibility for a programmer to implement filters to suit the application. GStreamer is primarily designed for C programming, but for object orientation purposes, it is built on GObject and Glib. The tool used to construct pipelines is gst-launch.

## 8

### Design and Implementation of Video Capture

#### 8.1 Camera Driver

The camera functionality is already implemented for the raspberry pi and need not be custom written. The below sections explain the execution and capturing of video.

The camera used in this project is the Raspberry Pi camera module. The advantage in using the camera module is that, the camera driver need not be custom written for it. Firstly, the camera component is declared using the MMAL [13] component declaration technique. Using the MMAL\_COMPONENT\_T structure, a pointer to the camera is created. This pointer helps us access all the required information pertaining to the image and video. More importantly, access to camera itself is possible, which is declared as a component, like explained previously. Once the camera is created, the camera port by calling the camera callback function. Also, the processor running on the embedded computer supports videocore and its functionalities. The camera used in this project is the Raspberry Pi Camera Module. The camera module is connected to a dedicated CSI on the Raspberry Pi board. Due to the static nature of the camera, it is advised to earth oneself while handling the camera. Concern need not be placed on the interfacing of the camera as the programs are written such that they recognise the camera. This camera is capable of supporting the capture of images up to 5 megapixels. Can record videos in H.264, baseline, main or high-profile formats.

#### 8.2 Video Capture Functionality

In this functionality a pointer to the camera is created and also 2 port pointers, a still port and a video port are created. This gives us the advantage to configure the video frames according to the stills. Using the MMAL\_PARAMETER\_CONFIGURE\_CAMERA, the parameters, the width, the height are initialised. The format of the encoding is set on the created video port and the still port. The port pointers of the video and still were created using the MMAL\_PORT\_T structures. The data stream is considered an elementary stream of data and is passed to the parameters of the video/image set previously. Once the elementary stream and the parameters are configure, the final stream is committed to the port using the mmal\_port\_format\_commit function and the buffer is filled with the stream from the port. The same process is repeated for the still and video viewing port.

#### 8.3 Encoder Component

The encoder component that was declared at the beginning of the program, is now used. There are two ports created to the encoder component- *in* and *out*. The pointer is created to the MMAL\_COMPONENT\_T structure. The result of the encoder component is a H.264 stream of data. Using the declared buffer size, the encoder is assigned the buffer size and the changes are committed to the output port. The rate control parameters are then set. Following few other assignment parameters, the encoder component is enabled and a pool of buffers are created for the output to use.

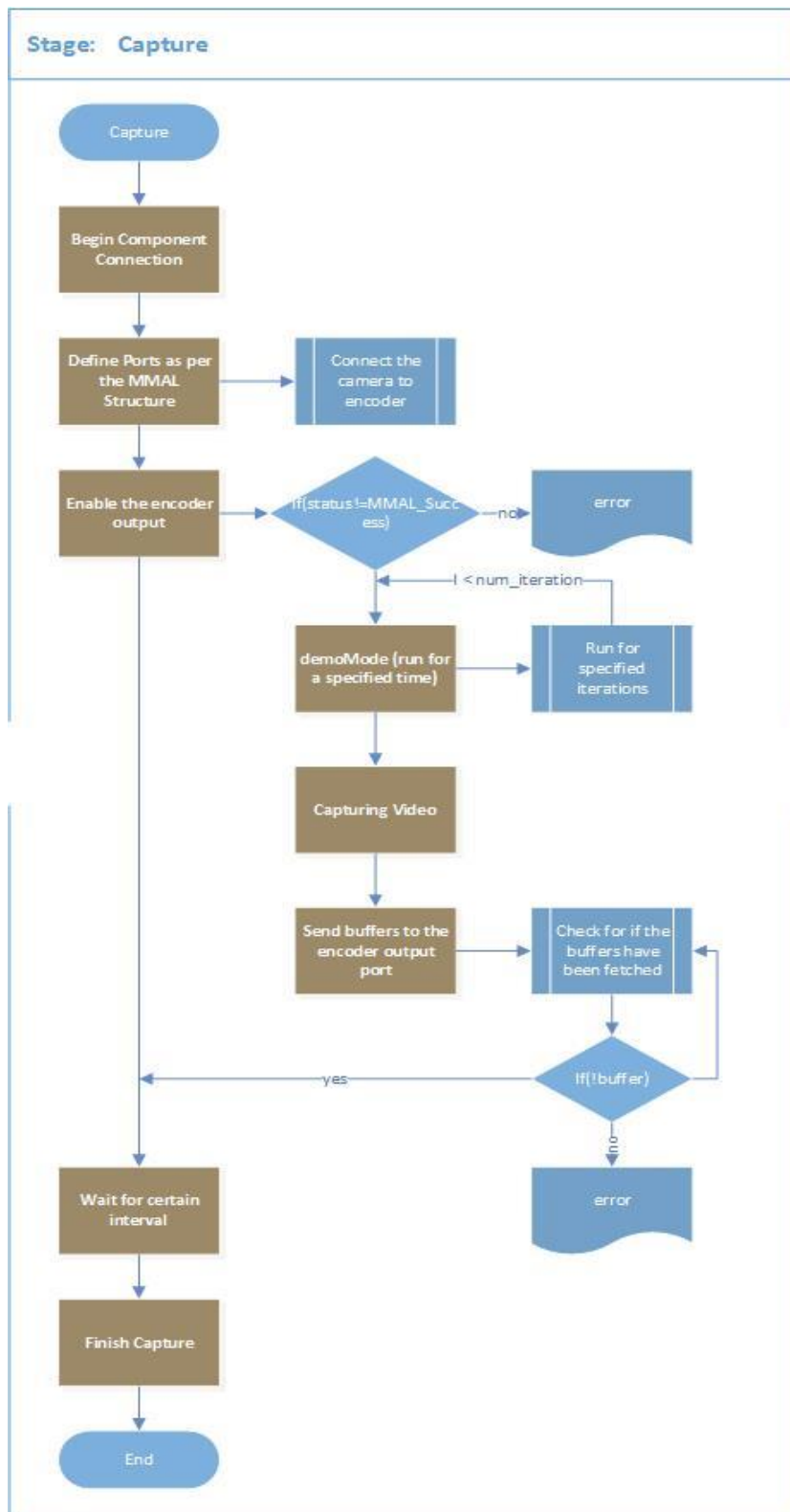


Fig. 8.1

Flowchart showing the Video Capture process



## 8.4 User Defined Parameters

There are certain parameters that are accessible by the user from the final station or the Ubuntu PC. When the user would like to change some parameters to view the final image displayed on the Ubuntu PC, this can be accomplished from the terminal. These parameters are

Assist displays a message about the function

Length which accepts a value to set the length of the video window displayed. Default is 1920.

Height which accepts a value to set the height of the display window. 1080 set as default.

Bitrate is set to 30Mbit/s. But can be changed by the user.

Framerate specifies the rate of frames to record.

Verbose displays the verbose output information.

Format, allows the option to select the format of output H.264.

Refresh Period specifies the intra refresh period.

Demo allows viewing of the output, but the capture is not recorded.

Preview, displays the image after the capture.

Output, outputs the decoded image.

Time Period, mentions the time duration of the capture, set to indefinite capture duration.

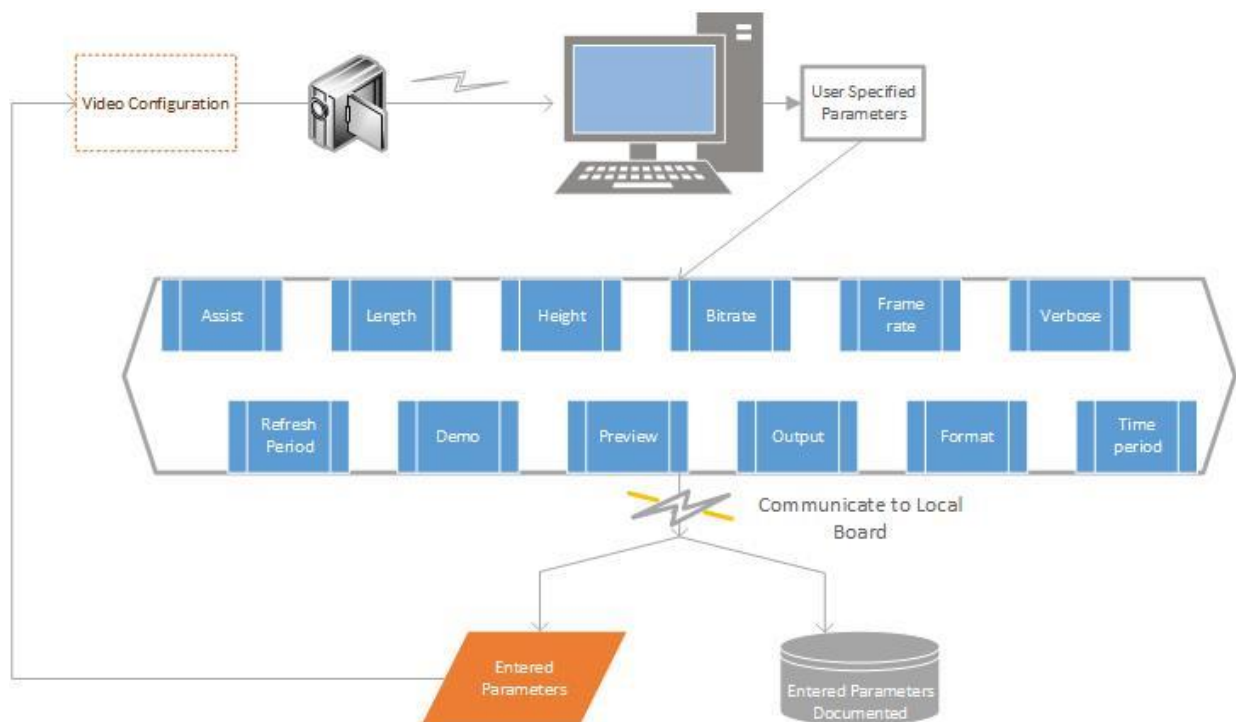


Fig. 8.2 User Defined Parameters

There are also many other parameters, that can be set by the user from the terminal, which have been declared and executed from another program which has been designed to control the other aspects of the images/frames captured. These parameters only have a visual impact on the image and will only be displayed as a demo.

## 9

### System Setup

As explained before, the system consists of a Mobile Robot [10] and Raspberry Pi along with camera module. Also, the Raspberry Pi is attached with a WiFi module- the Edimax Nano USB Module [12]. The Edimax module broadcasts a network, to which is connected the ground station Ubuntu PC via a USB WiFi adaptor stick. The Raspberry Pi also has an SD card running the Raspbian OS with GStreamer and other software installed on it. The Ubuntu PC also runs the same version of GStreamer installed on the Raspbian OS running on the Raspberry Pi.

The programs are loaded onto the Raspberry Pi. Connection to the Edimax router from the Ubuntu PC is established, which helps us keep a control over the program. The programs can be executed and stopped as wished. The video captured will be stored in a file and saved on the Raspberry Pi.

#### 9.1 Initial Results

Initial pipeline constructed to stream video from the raspberry pi gave the following results.

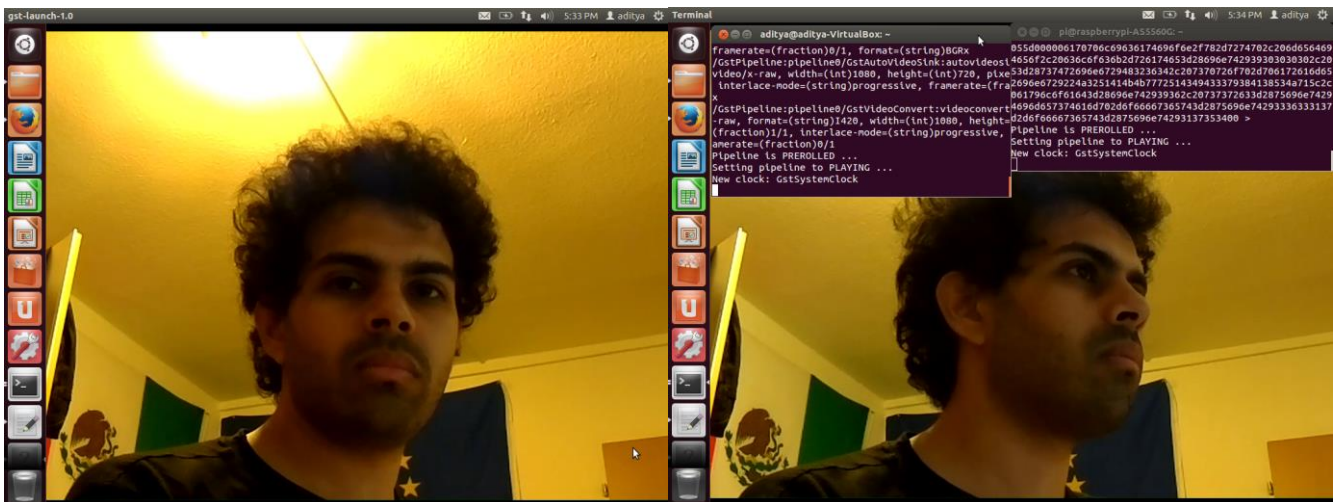
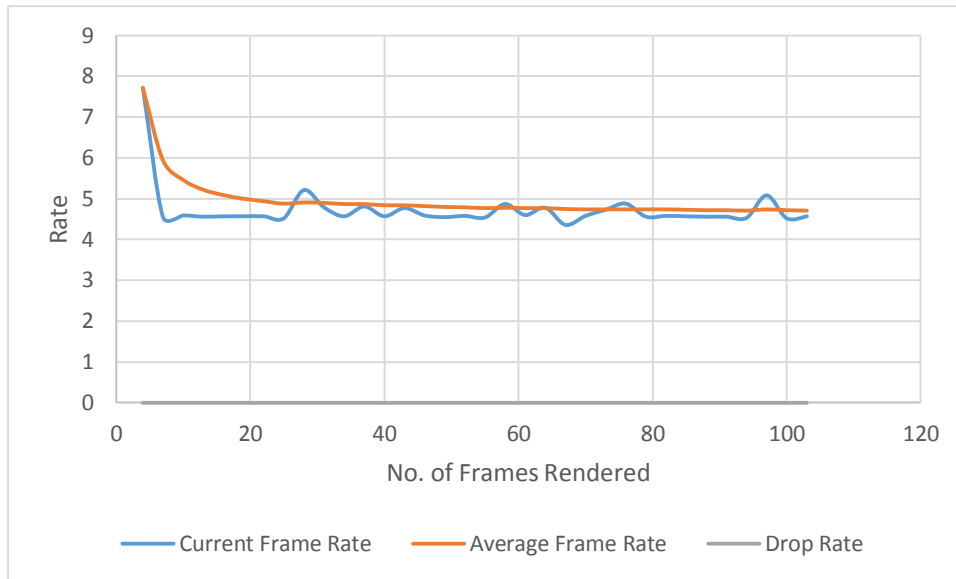


Fig. 9.1 Initial Video Stream

The above following graphic gives the result of the stream taken per 100 frames.



**Fig. 9.2 Rate vs Frames Rendered**

From the graph above, it is observed that the drop rate is NULL, even though the rate of frames rendered is very less. From this the conclusion is derived that there had to be a problem in the execution of the initial pipeline. This initial pipeline, did not include an encoder as it was considered not required, as the format of the video captured from the Raspberry Pi is of H.264 encoded format. But this is the hardware accelerated format and just a standard, hence, this becomes similar to an interface, and thus needed and actual encoding into H264 format. A conversion to software H264 format is needed. Also, it was considered that streaming over the User Datagram Protocol(UDP) would be effective compared to TCP. The resulting stream was slow and broken. There were constant spikes as not all the frames were rendered in time. Hence, a new system was conceptualised.

## 9.2 Changes Adopted

From the above explained initial results, the need for an improved system was imperative for the success of this project. The following changes are adopted,

- The communication protocol was changed to Real Time Protocol (RTP) over UDP, from using only Transport Control Protocol (TCP). The advantage in this change is that UDP is a 'wrapper' that interfaces application to the network, and wireless transmission is possible. It also has very low overhead and a high transmission speed. RTP also transmits data in-time, i.e. in real-time, and does not re-transmit data that is lost or corrupted during transmission.
- Implementation of an encoder on the server side of the system, which is the Raspberry Pi.
- The queueing of frames where required so as to avoid overflow of frames.

## System Design and Implementation

### 10.1 System Design

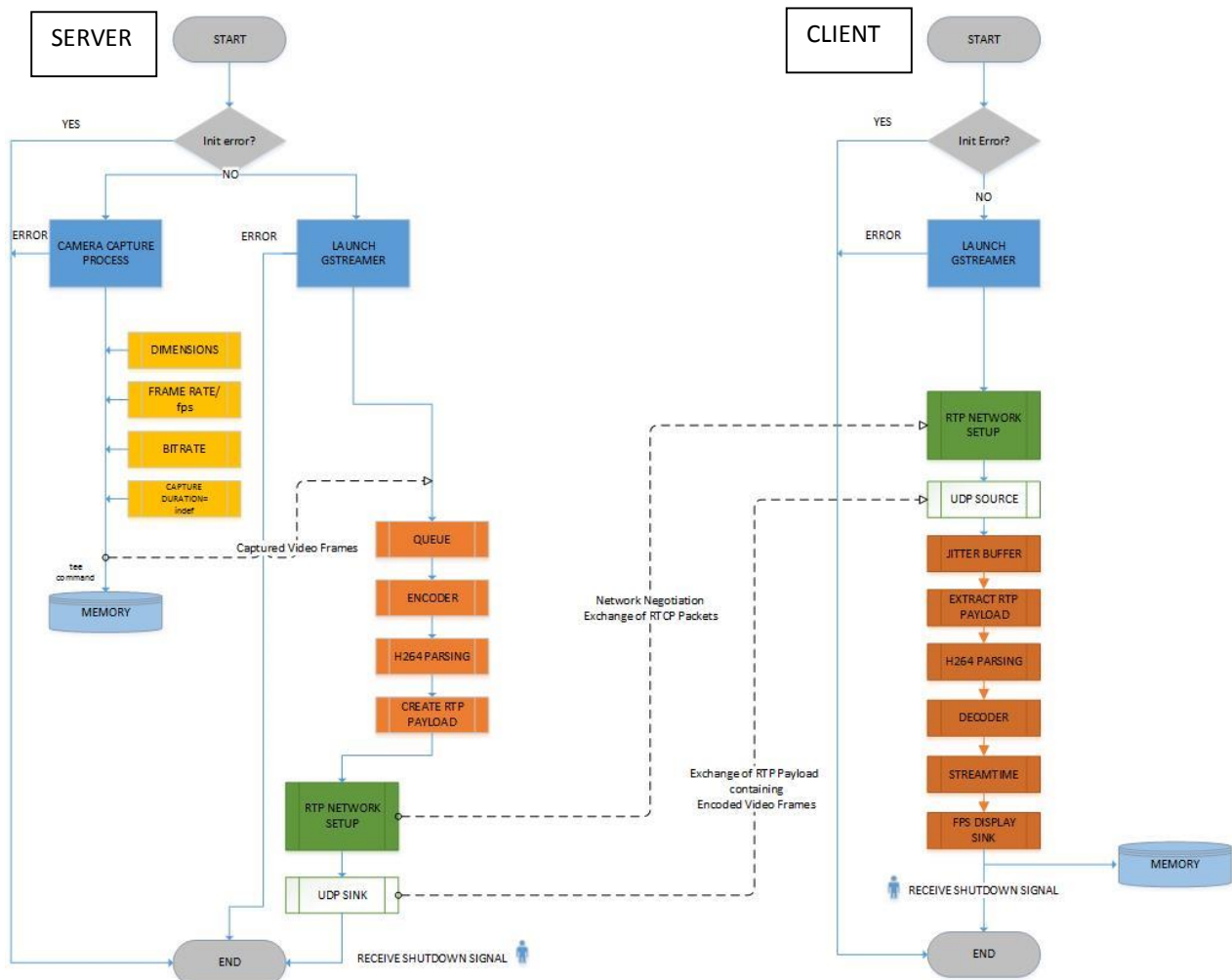


Fig. 10.1 System Design

The above diagram shows the design and breakup of the different parts of the developed system.

The Server consists of the video capture process and encoding of the video into H264 packets before transmission. The Client consists of the Ubuntu PC that connects to the Server, receives the packets from the Server, decodes them and displays the video.

## 10.2 Real-Time Protocol (RTP)

There are 3 main elements in the RTP,

- *rtpbin*
- *rtpjitterbuffer*
- *rtpdemux*
- *rtph264pay*
- *rtph264depay*

*rtpbin* – contains all the required functionalities. It also enables the synchronisation of sessions. This project implements initiates a session between the server and client and the communication is handled within this session. This bin may or may not be combined with Real Time Control Protocol (RTCP) session. This project does not implement the RTCP.

*rtpjitterbuffer* – this element records the packets/data received on the network. Also it removes any duplicate packets on the stream. Any packet/data arriving later than the set latency is dropped. It uses a thread to release packets from the queue.

*rtpdemux* – this element acts as a demuxer for RTP packets. It allows an application to receive and decode an RTP stream, but is most handy when dealing with RTP streams that come with multiple payload types.

*rtph264pay* – this element payload encodes the H264 video into RTP packets before transmission.

*rtph264depay* – element extracts the H264 video from the received RTP packets.

The collaboration diagram of *rtpbin* element can be seen in the appendix.

### 10.3 Server-Side System

The Server side, Raspberry Pi, executes the following design. The video itself is captured according to the Raspberry Pi default commands, which execute as explained previously in the Camera Driver functionality. The captured video frames are immediately collected and transmitted to the destination PC. The following describes the process.

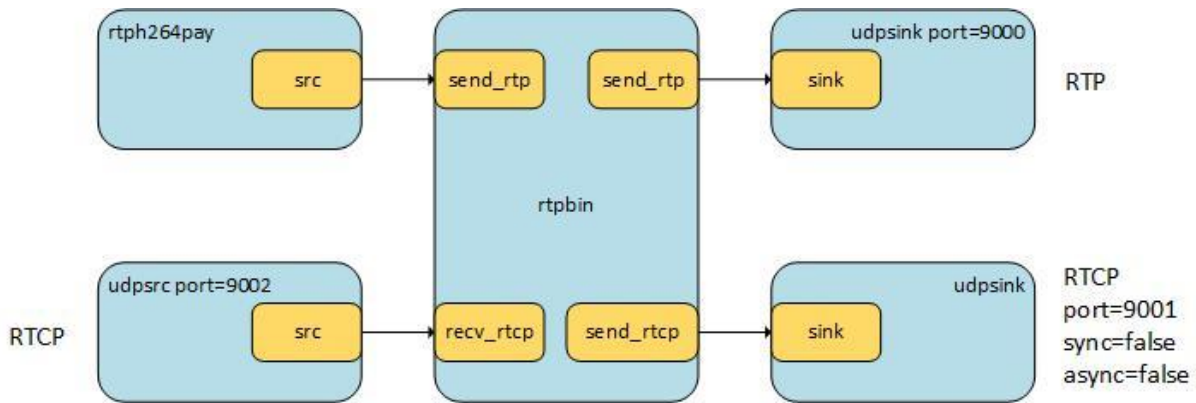


Fig. 10.2 Server RTP Session

The above diagram describes the RTP session in on the server side of the system. Real-time Transport Protocol (RTP) Control Protocol (RTCP) is an additional functionality of the RTP. While it does not transport any media per se, it provides control information for an RTP session. It also provides statistics on jitters, octet transmitted and packet counts-lost and transmitted and delay time.

The encoded video is sent to session 0 in the *rtplib*. The video packets are sent on UDP port 9000. The video RTCP packets for session 0 are sent on port 9001. RTCP packets for session 0 are received on UDP port 9002. Both sync and async are set to false as the packets need to be sent as soon as possible.

The captured video is then simultaneously streamed as *fdsrc*, which is ‘Read from File Descriptor’. This is of advantage to this project as packets and data are dynamically generated. The packets are then time-stamped, i.e. applies current stream time to buffers. This data is then fed to a queue which simply is a data queue. It also creates a new thread on the source pad to decouple processing on sink and source pad. With the *leaky* property set, any old buffers will be dropped, hence producing a flawless stream of packets. *Decodebin* produces raw data so that it may be encoded later. *Videorate* element produces a perfect stream of data by dropping and adjusting timestamps on video frames to make a perfect stream. It makes a stream that matches the framerate on its source. *Videoconvert* converts video to a different colorspace. The stream is then encoded using the *omxh264enc* encoder, with which a H264 video is generated. With *h264parse*, the stream of data is parsed. *Rtph264pay* payloads H264 packets to RTP packets. This is then dispatched over the UDP defined port.

The following diagram show the server side implementation of the GStreamer pipeline as described above.

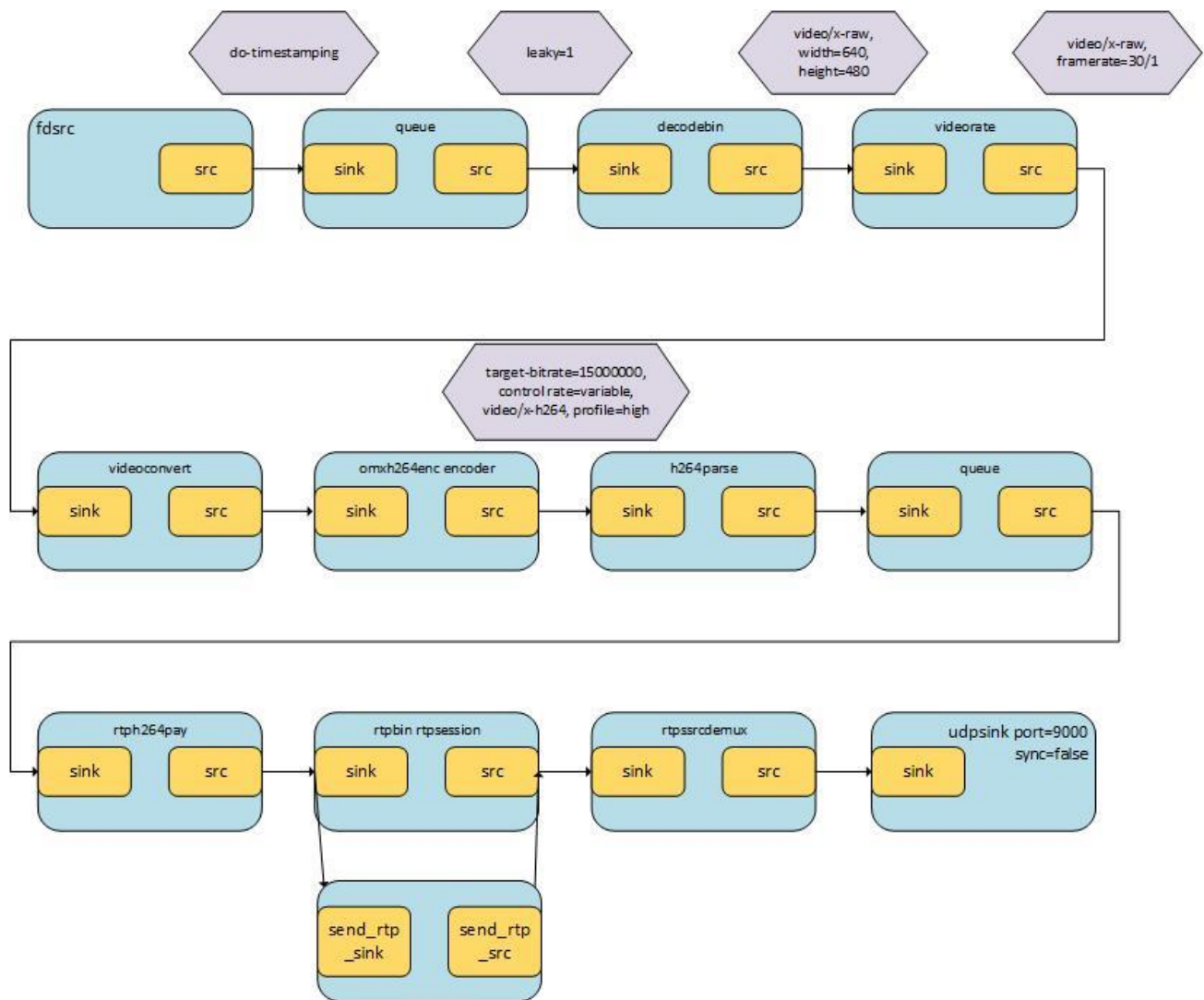


Fig. 10.3 Server GStreamer Pipeline



## 10.4 Client-Side Implementation

*udpsrc* on port 9000, receives data over network sent from the *udpsink*. It is combined with RTP streaming. *Rtpssrcdemux*, *rtpdemux* and *rtpjitterbuffer* perform their functions as explained earlier.

The *rtpjitterbuffer* requires the clock-rate of the RTP payload in order to estimate a delay. This element has a ‘latency’ property which defines a definitive time limit for the packets to arrive. Packets arriving later than this limit are considered to be lost packets. The de-payloader uses the lost packet events to create concealment data. It uses Decode Time Stamp(DTS) of the incoming buffer to create a time stamp on the outgoing buffer.

The jitter buffer attempts to estimate the arrival time of buffers. The initial expected arrival time is calculated as follows, a certain sequence number *seqnum* N arrives at time T, *seqnum* N+1 is expected to arrive at T+packet-spacing+delay, where packet-spacing is calculated from DTS of two consecutive RTP packets. The RTP packets need to have different *rtptime*.

*RtpH264depay* extracts H264 video packets from RTP packets. H264parse provides library for H264 video bitstream parsing. Using *avdec\_h264* decoder, the packets are decoded, and converted to a different colorspace. *Timeoverlay* functionality displays the duration of the streamed video, and it also provides a textoverlay functionality. *Fpsdisplaysink* keeps a track of the streamed frames.

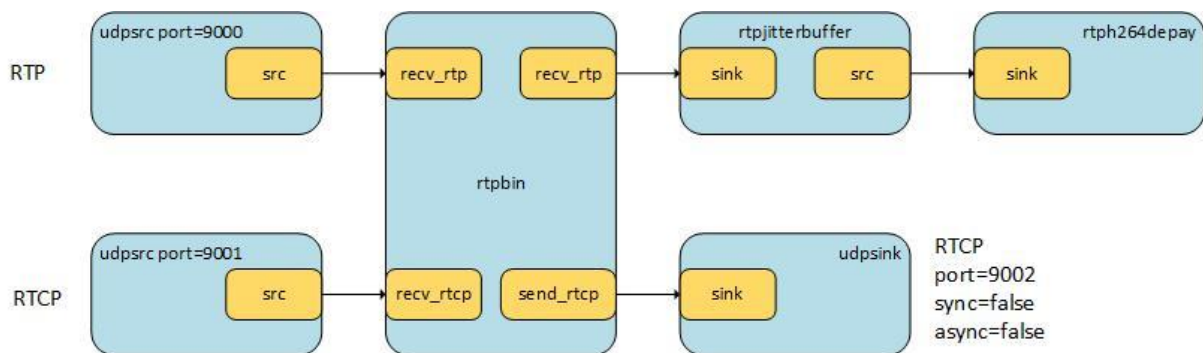


Fig. 10.4 Client RTP Session

Receive H264 on port 9000, send it through *rtpbin* in session 0, the jitter buffer, de-payload, decode and display the video. Receive server RTCP packets for session 0 on port 9001. These packets will be used for session management and synchronisation



The below diagram depicts the client side implementation of the GStreamer pipeline as described above.

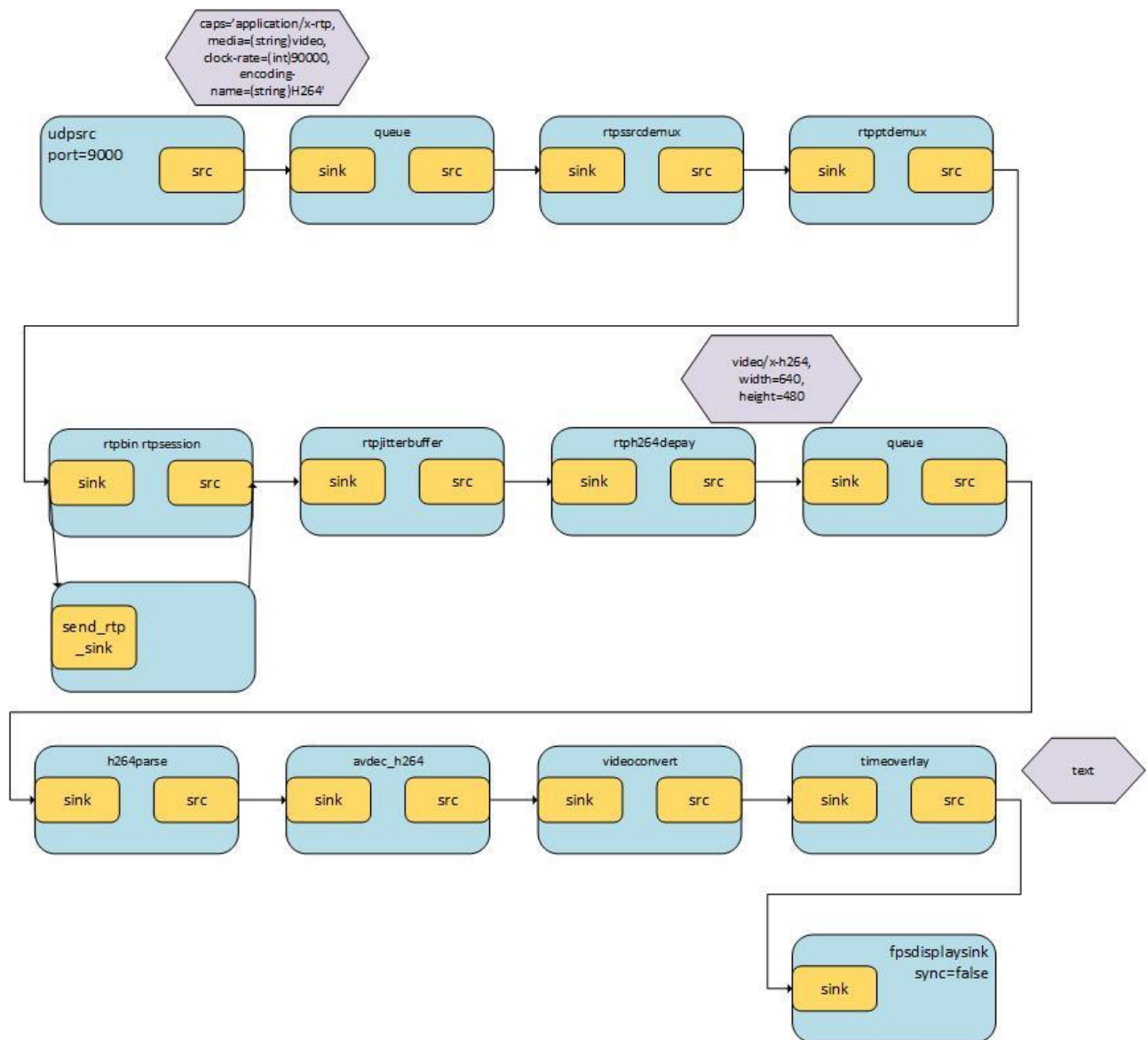


Fig. 10.5 Client GStreamer Pipeline

## Analysis and Discussion of Results

The following graph shows the processor usage on the Raspberry Pi. It is observed that there is a significant rise in the CPU Usage, when executing the server side functionality. It rises to over a 90% usage from a meagre less than 6% usage. This is due to the encoding functionality implemented on the server side.

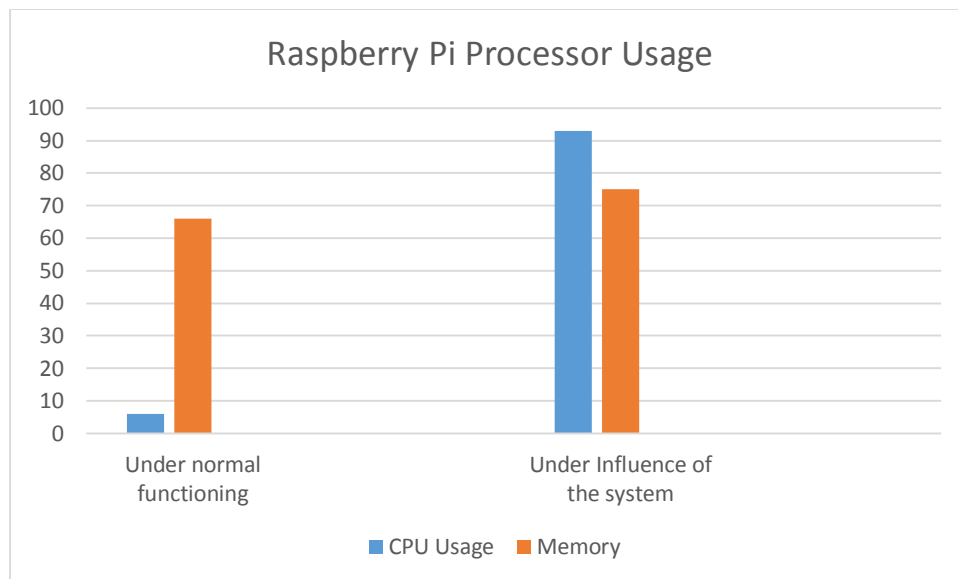


Fig. 11.1

The graph below, shows the network packets received on the client side of the system. It is observed that when there is no data being received, the graph remains idle.

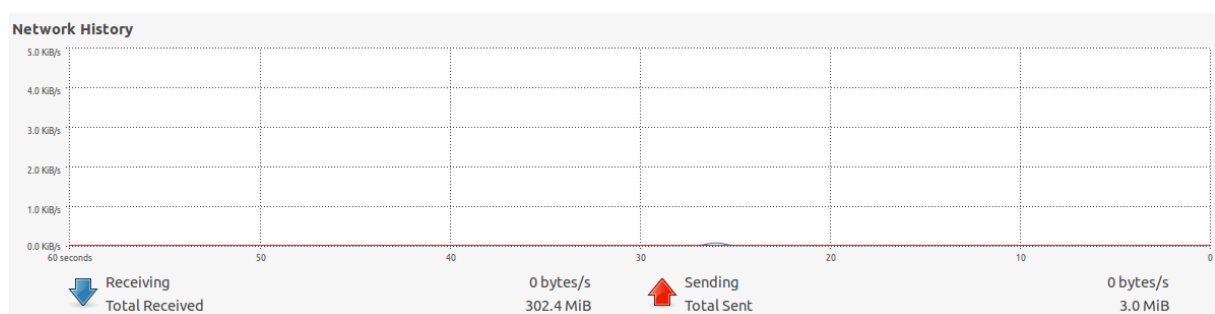


Fig. 11.2 PC Network History before Transmission

As the network is established between the Ubuntu machine and the Raspberry Pi the CPU usage increases.

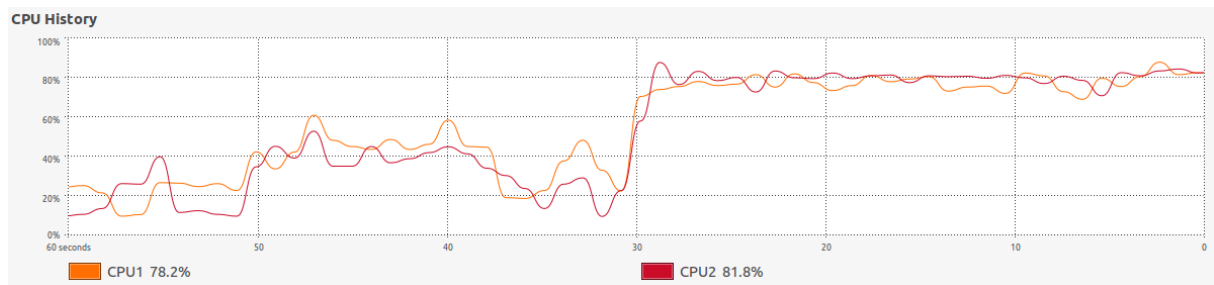


Fig. 11.3 PC CPU Usage

The below graph displays the network status and the total data received. From the being almost nil (as shown in the above graph) to a spike in the network received data. It can also observe that, there is a very steep drop in the network data received, and then immediately rises. This is due to the initial establishment of network and then the streaming of data.

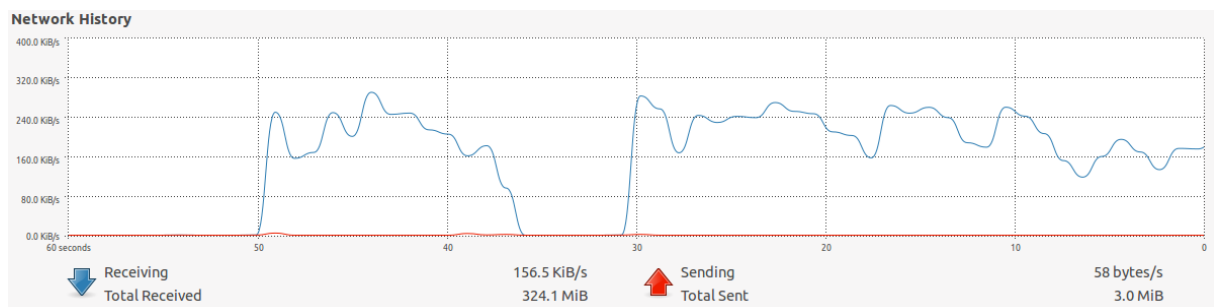


Fig. 11.4 PC Network History after Network Establishment

The below graphs depict the network and CPU history till the 2314<sup>th</sup> rendered frame.

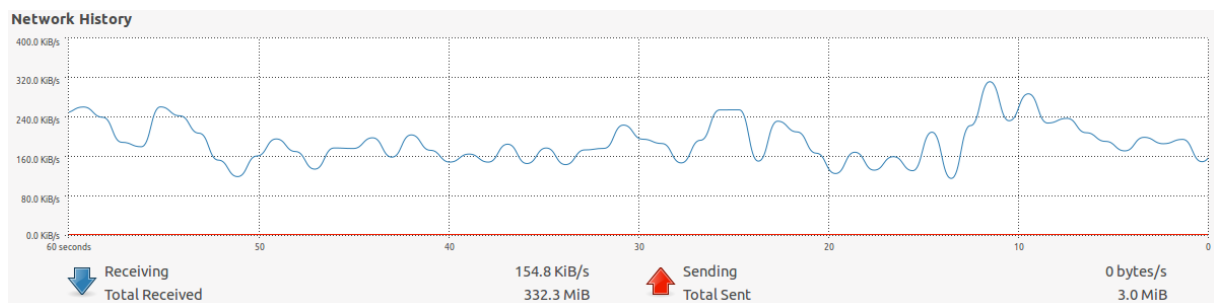


Fig. 11.5 PC Network History during Process

From the above graph- the variation of the rate of data received. The total data received 332.3 MiB and the current rate of data received is 154.8 KiB/s.

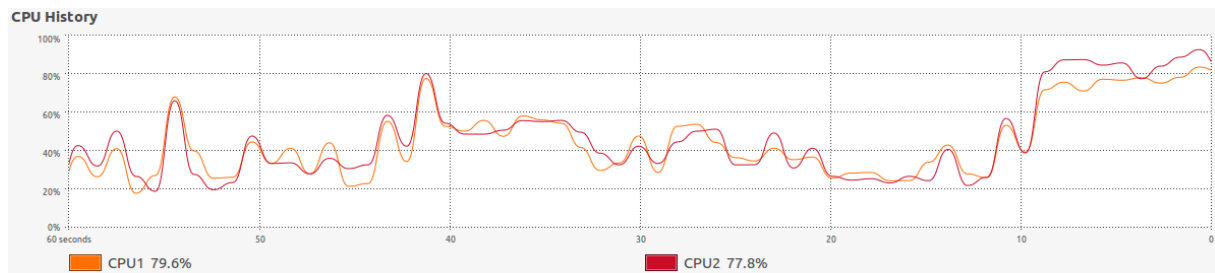


Fig. 11.6 CPU Usage during Process

The above graphs shows the high usage in CPU time during the process.

The streamed video, used in analysis, streamed till 1 minute 24 seconds and 279 milliseconds.



Fig. 11.7 Test Video Stream

With the dropped frames being nil, the system is very stable and all the packets of data are transferred to the destination PC.

The following graph shows the variation in the current frame rate at every rendered frame for the display resolution of 640x480. The dotted line above the bars represent the 2 period moving average.

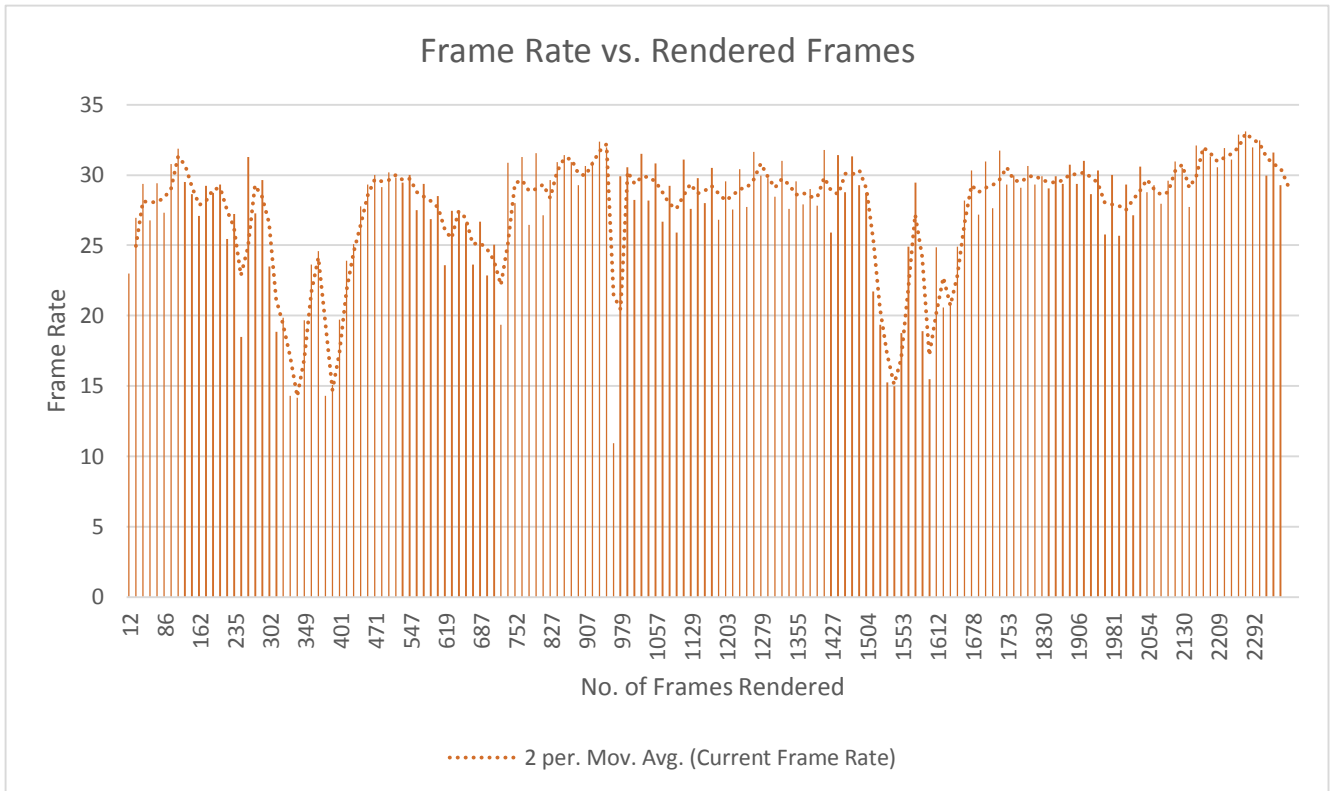


Fig. 11.8 Frame Rate vs Rendered Frames for DR<sup>1</sup> 640x480

DR<sup>1</sup>-Display Resolution

From the above graph, it may be observed that the Frame Rate is not always constant and sometimes may drop to a very low value of 10.95 at the 963<sup>rd</sup> rendered frame. The highest frame rate recorded is 35.79 at the 2314<sup>th</sup> rendered frame. Although the Frame Rate can be very low at certain time, there have been no dropped packets reported.

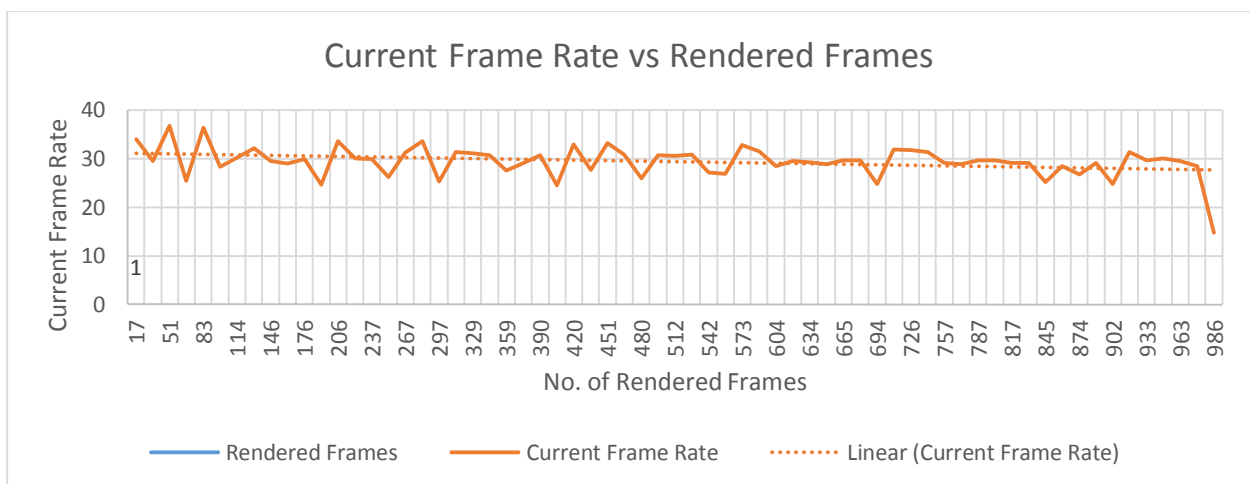


Fig. 11.9 Frame Rate vs Rendered Frames for DR<sup>1</sup> 720x480

The above graph shows the variation in the current frame rate at every rendered frame for the display resolution of 720x480. The dotted line above the bars represent the linear trendline. With the Current Frame Rate staying within the 25-35 limit the condition remains similar to the standard 640x480 display resolution.

The below graph shows the variation in the current frame rate at every rendered frame for the display resolution of 1280x720. The current frame rates are shown above the graph line. With the Current Frame Rate staying within the 16-10 limit the conditions for this resolution is not suitable for streaming as video will be displayed slow and will freeze at different points and very frequently. The video streaming resolution is not suitable for further applications of video/image processing.

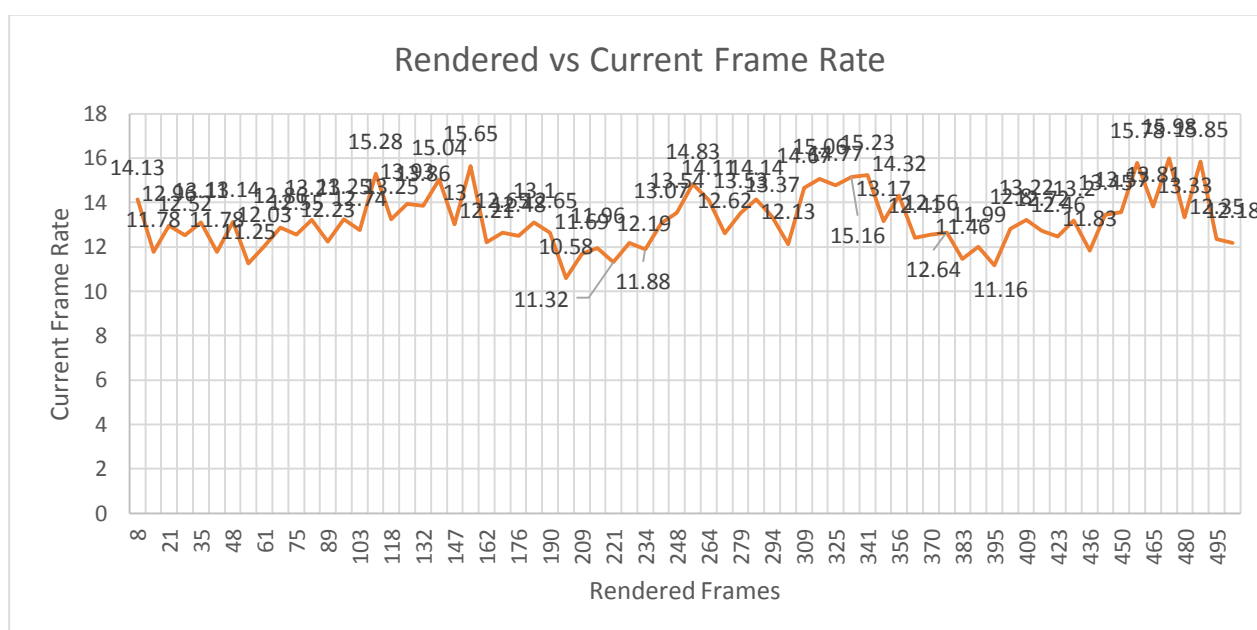


Fig. 11.10 Frame Rate vs Rendered Frames DR<sup>1</sup> 1280x720

In conclusion the display resolutions of 640x480 and 720x480 are the most suitable for applications and further enhancing of this application.

The testing scenario includes the Raspberry Pi board with the following features. The transmission takes place on 150 Mbps network bandwidth.

Platform	Processor	RAM	Frame Size	Avg. FR(fps)
Raspberry Pi	ARM1176JZ-F	512MB (SDR)	640x480	29
Raspberry Pi	ARM1176JZ-F	512MB (SDR)	720x480	29
Raspberry Pi	ARM1176JZ-F	512MB (SDR)	1280x720	12

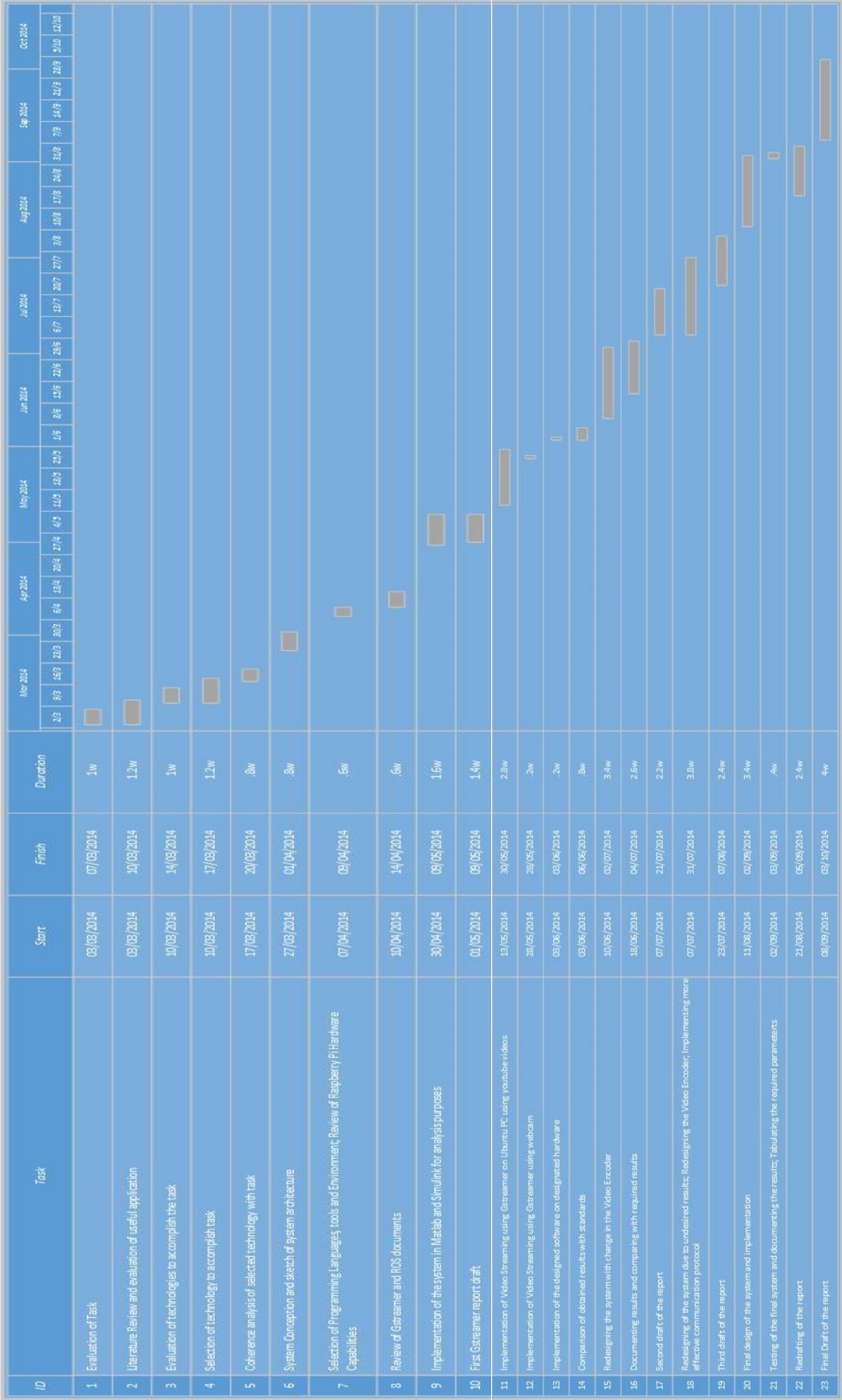
Tab. 11.11 System Scenario

## Summary

During the course of this internship project, a system has been designed to activate a camera to capture video using a Raspberry Pi board off an Autonomous Mobile Robot and wirelessly transmit this to a stationary Ubuntu PC. The system has been developed on the GStreamer programming framework, which also handles the communication network. The video is transmitted and saved on the Ubuntu PC for later evaluation. The Raspberry Pi has been configured to act as a WiFi hotspot and the Ubuntu PC connects to this network, to enable communication.

This system is built for the Mobile Robot [10] designed by the team of the Chair of Automation Technology. Skills in engineering have been learnt and applied or improved in this project. This software may also be improved for further application for audio streaming. The system have been developed according to specifications, task analysis, review of software libraries, design and evaluation, and technical report formulation.





This diagram depicts the Timeline of the Project.



## **Acknowledgment**

My sincere gratitude firstly to Prof. Dr. Wolfram Hardt and Prof. Dr. Peter Protzel for giving me the opportunity to work on this internship topic, and to Dr. Sven Lange for his patience, excellent guidance, providing me with the materials and allowing me to work at my pace, and directing me with regards to the progress of this topic throughout the research internship period.

Secondly I would like to thank Dr. Mirko Caspar and Dr. Ariane Heller for assisting me with useful information that I required to progress with this work.

## X Appendix

### I. Raspberry Pi Configuration [[1] , Chapter 1]

The Raspberry Pi has its operating system written on an SD card, which is mounted on the board. The operating system used in this project is the latest Raspbian, which is a version of Debian Wheezy. The image is downloadable from the following website

<http://www.raspberrypi.org/downloads>

Once downloaded, the image is written onto the SD card the following way.

Mount SD Card onto a card reader and attach to the desktop on a computer. Run ‘df -h’ to check for the devices mounted. If the card is mounted properly, the resulting lines will have /dev/mmb1k0p1 or /dev/sdd1 listed.

The SD card is then unmounted, as it is dangerous to write onto it an image while it is mounted. Care needs to be taken, as the ‘df -h’ command may also result in multiple entries of the SD card. This is due to the number of partitions on the card. All the partitions should be unmounted. Now, the image is written to the card with the execution of the following command,

```
34nmou=4M if=2014-09-09-wheezy-raspbian.img of=/dev/sdd
```

During the process, the *dd* command does not inform on the progress. But it does take more than around 5 minutes for the process to complete. To check for correct writing of the image onto the SD card, it is possible to *dd* from the card to another image on the hard disk and then running diff on the two images. After this, run ‘sync’ command which flushes the write cache and it is safe to unmount the SD card.

### II. Installation of Edimax WiFi EW-7811UN [16]

The Edimax EW-7811UN is a nano wireless adaptor that supports DHCP and is configured to WPA2-PSK.

The adaptor is first mounted on the Raspberry Pi through one of the USB ports. The adaptor can be located using the command ‘lsusb’ which will list all the devices attached. The adaptor will be listed the following way

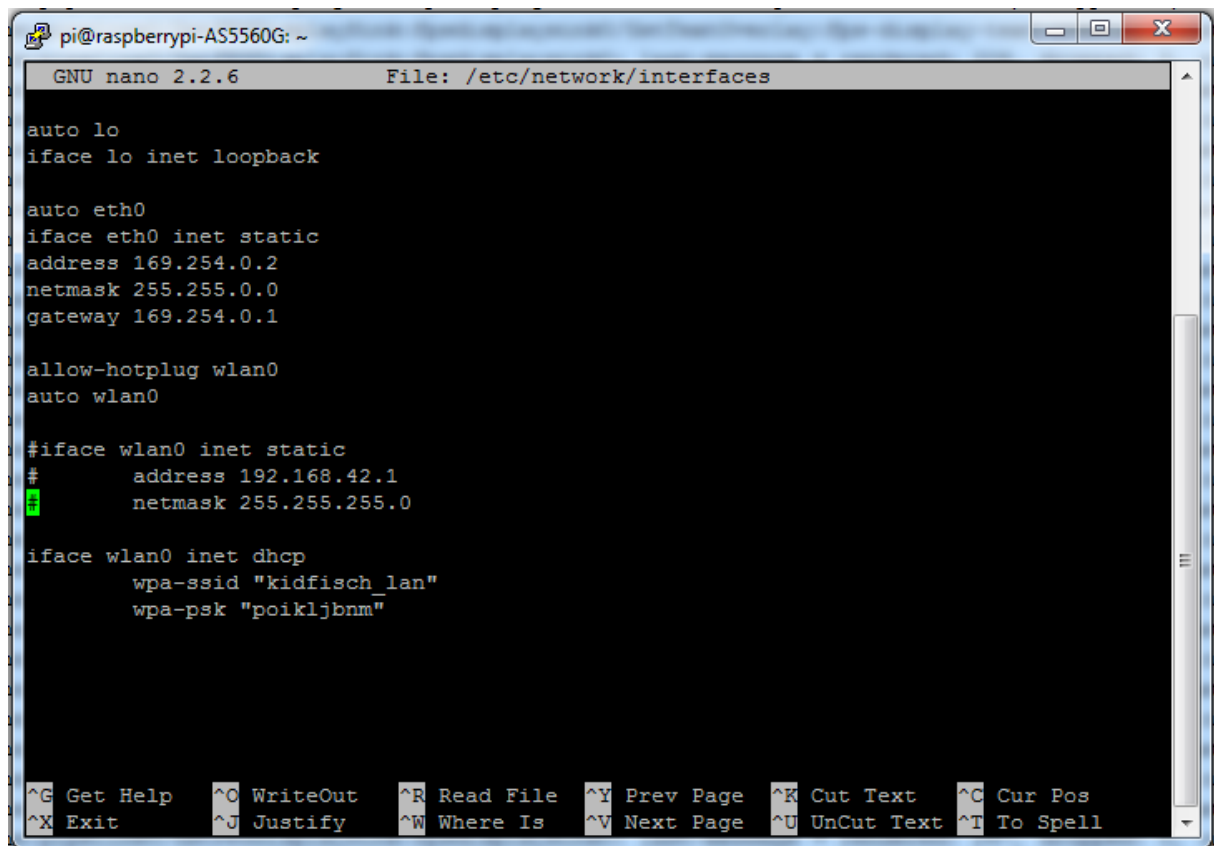
```
BUS 001 Device 004: ID 0bda:8176 Realtek Semiconductor Corp. RTL8188CUS  
802.11n WLAN
```

The Raspberry Pi is then rebooted, to install the necessary drivers. The latest operating systems of the Raspberry Pi come equipped with the necessary drivers, hence, no manual installation is required. Once rebooted, the following commands configure the access to wireless network

```
sudo nano /etc/network/interface
```

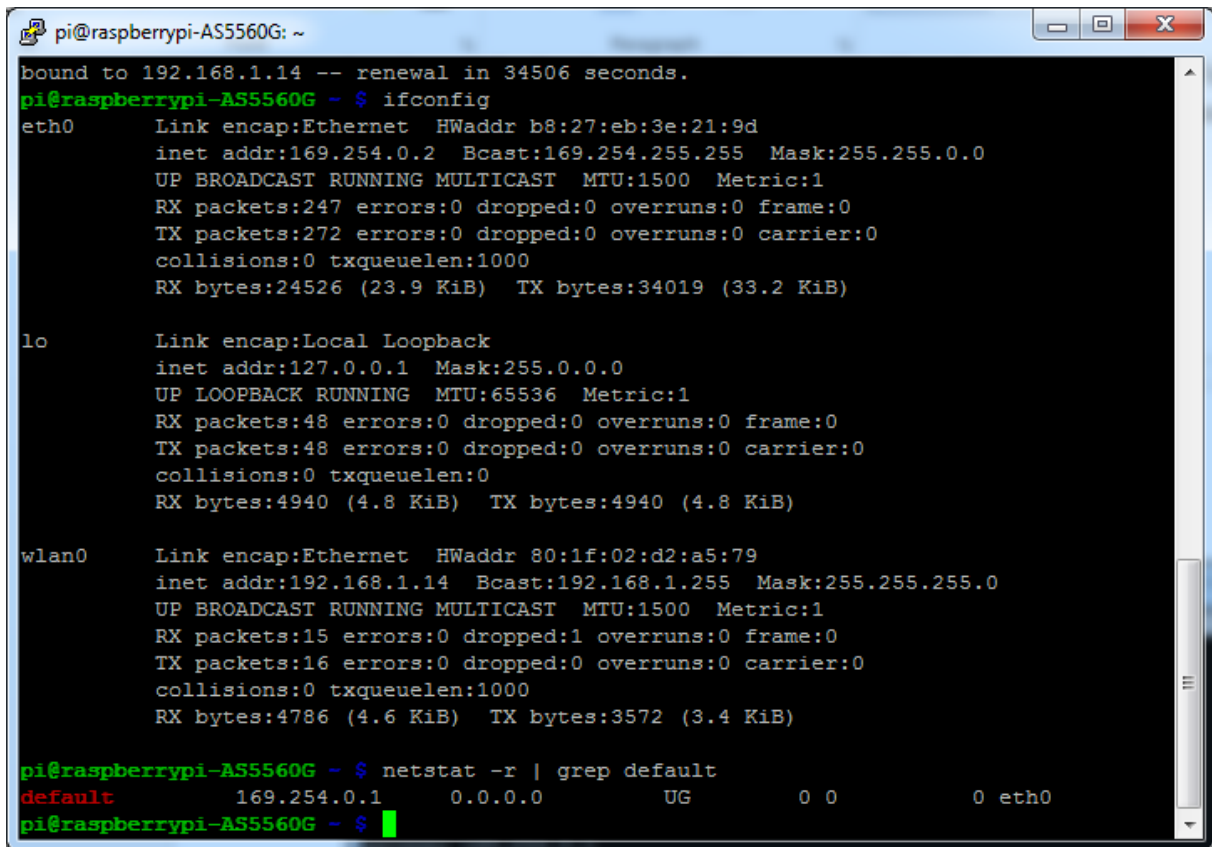
Within this file, the hashtag in front of ‘#auto wlan0’ is edited out and the following lines are added ‘iface wlan0 inet dhcp’, and ‘wpa-ssid “network-name” ’ and ‘wpa-psk “network-

*password*” ’ are entered, and the may resemble the below image. The Raspberry Pi can either be rebooted for the changes to affect, or ‘*sudo ifup wlan0*’ command may be executed. To check if the adaptor is connected to the wireless network, ‘*ifconfig*’ may be executed from the terminal which outputs the network status. If successfully connected, an IP address will be assigned in the ‘*wlan0*’ entry.



```
pi@raspberrypi-AS5560G: ~  
GNU nano 2.2.6 File: /etc/network/interfaces  
  
auto lo  
iface lo inet loopback  
  
auto eth0  
iface eth0 inet static  
address 169.254.0.2  
netmask 255.255.0.0  
gateway 169.254.0.1  
  
allow-hotplug wlan0  
auto wlan0  
  
#iface wlan0 inet static  
#    address 192.168.42.1  
#    netmask 255.255.255.0  
  
iface wlan0 inet dhcp  
    wpa-ssid "kidfisch_lan"  
    wpa-psk "poikljbnm"  
  
^G Get Help  ^O WriteOut  ^R Read File  ^Y Prev Page  ^K Cut Text   ^C Cur Pos  
^X Exit      ^J Justify   ^W Where Is   ^V Next Page  ^U UnCut Text ^T To Spell
```

Fig. Edimax Setting 1

A screenshot of a terminal window titled 'pi@raspberrypi-AS5560G: ~'. The terminal shows the output of the 'ifconfig' command for three network interfaces: eth0, lo, and wlan0. eth0 is an Ethernet interface with IP 169.254.0.2. lo is a local loopback interface with IP 127.0.0.1. wlan0 is an Ethernet interface with IP 192.168.1.14. Below the ifconfig output, the 'netstat -r | grep default' command is executed, showing a default route to 169.254.0.1 via the eth0 interface.

```
pi@raspberrypi-AS5560G: ~  
bound to 192.168.1.14 -- renewal in 34506 seconds.  
pi@raspberrypi-AS5560G ~ $ ifconfig  
eth0      Link encap:Ethernet  HWaddr b8:27:eb:3e:21:9d  
          inet addr:169.254.0.2  Bcast:169.254.255.255  Mask:255.255.0.0  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:247 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:272 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:24526 (23.9 KiB)  TX bytes:34019 (33.2 KiB)  
  
lo        Link encap:Local Loopback  
          inet addr:127.0.0.1  Mask:255.0.0.0  
          UP LOOPBACK RUNNING  MTU:65536  Metric:1  
          RX packets:48 errors:0 dropped:0 overruns:0 frame:0  
          TX packets:48 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:0  
          RX bytes:4940 (4.8 KiB)  TX bytes:4940 (4.8 KiB)  
  
wlan0     Link encap:Ethernet  HWaddr 80:1f:02:d2:a5:79  
          inet addr:192.168.1.14  Bcast:192.168.1.255  Mask:255.255.255.0  
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1  
          RX packets:15 errors:0 dropped:1 overruns:0 frame:0  
          TX packets:16 errors:0 dropped:0 overruns:0 carrier:0  
          collisions:0 txqueuelen:1000  
          RX bytes:4786 (4.6 KiB)  TX bytes:3572 (3.4 KiB)  
  
pi@raspberrypi-AS5560G ~ $ netstat -r | grep default  
default    169.254.0.1    0.0.0.0      UG          0 0          0 eth0  
pi@raspberrypi-AS5560G ~ $
```

Fig. Edimax Setting 2

### III. Camera Setup

This project uses the camera module of the Raspberry Pi. This camera module has a five-megapixel fixed focus. It connects to the hardware board via a CSI port and is accessed through the MMAL libraries. The following commands install the camera module. The board has to be connected to the internet during this section.

Connect the camera ribbon cable to the CSI port located in front of the Ethernet port on the board. The blue marking on the ribbon cable should face the Ethernet port. Once connected, the following command is to be executed from the terminal.

```
sudo apt-get update
```

The above command updates the board with the latest camera software and also installs the camera module.

### IV. GStreamer Installation [8]

It is necessary that the Raspberry Pi is connected to the internet throughout this section. This project requires the availability of GStreamer1.0 on both the server, which is the Raspberry Pi and the client, which is the computer that connects to the Raspberry Pi. The installation of

GStreamer1.0 on the Raspberry Pi is achieved with the execution of the following commands. These commands will obtain the required software and its libraries.

```
apt-get install autoconf automake libtool gtk-doc-tools libglib2.0-dev
git clone git://anongit.freedesktop.org/GStreamer/gst-omx
```

The downloaded libraries are then compiled,

```
cd gst-omx
./autogen.sh
```

An alternative for the above method is to install from the precompiled version.

To /etc/apt/sources.list, add the following line

```
deb http://vontaene.de/raspbian-updates/ . main
```

The above line adds the location of the repository that contains the precompiled version of the required software. Then proceed with the installation with the execution of the following commands,

```
apt-get update
apt-get install libGStreamer1.0-0-dbg GStreamer1.0-tools libGStreamer-
plugins-base1.0-0 GStreamer1.0-plugins-good GStreamer1.0-plugins-bad-dbg
GStreamer1.0-omx GStreamer1.0-alsa
```

## **V. Network Setup[6]**

The Edimax WiFi adaptor connected to the Raspberry Pi embedded computer is configured to so that it acts as the network broadcaster and hence the Raspberry Pi itself becomes the WiFi Access Point. isc-dhcp-server is installed, and the guide in the configuration file helped to setup the server, the lines that setup domain names were commented as they were not required and the line that makes the DHCP server the official DHCP server for the local network was uncommented. Again, as guided by the configuration file, the following lines were added at the end of the file so that it activates the network.

```
subnet 192.168.42.0 netmask 255.255.255.0 {
range 192.168.42.10 192.168.42.50;
option broadcast-address 192.168.42.255;
option routers 192.168.42.1;
default-lease-time 600;
max-lease-time 7200;
option domain-name "local";
option domain-name-servers 8.8.8.8, 8.8.4.4;
}
```

The above lines serve to provide a range to the network broadcasted by the Raspberry Pi. Any computer that recognises the network can connect to it. Continuing with the setup, the dhcp-server, the interface to which the components will connect to is the Edimax WiFi adaptor. If now the WiFi adaptor is connected to an internet, disconnect it and hence the WiFi adaptor was disconnected from the internet router. The adaptor was then set as static the wlan0 in the file interfaces within networks. Basically, all the old wlan0 settings were removed. The following lines are added

```
iface wlan0 inet static
address 192.168.42.1
netmask 255.255.255.0
```

The above lines initiate the static assignment of an IP address to the WiFi adaptor. The following lines then were executed, which assigns the IP address to the WiFi adaptor.

```
sudo ifconfig wlan0 192.168.42.1
```

Continuing with the configuration, next the Access Point is setup. A password sensitive network was setup so that only eligible systems can connect. A new file called **hostapd.conf** was created within **/etc/hostapd** directory. The following lines were added

```
interface=wlan0
driver=rtl871xdrv
ssid=.....
hw_mode=g
channel=6
macaddr_acl=0
auth_algs=1
ignore_broadcast_ssid=0
wpa=2
wpa_passphrase=.....
wpa_key_mgmt=WPA-PSK
wpa_pairwise=TKIP
rsn_pairwise=CCMP
```

The above lines add the settings for the WiFi. If the driver is different, then the details following **driver**, has to be changed accordingly. Also, the name of the **ssid** and the **wpa\_passphrase** can be changed to the persons liking. The Raspberry Pi needed to be directed to the above file, for that the following line had to be executed

```
sudo nano /etc/default/hostapd
```

which opened the file hostapd where line **#DAEMON\_CONF=""** was uncommented and the following was added between the quotes **/etc/hostapd/hostapd.conf**. Next the Network Address Translation(NAT) was configured. It was made to accept multiple clients to connect to the WiFi have all the data ‘tunneled’ through the Ethernet IP. It had to be done even if only one client needs to connect. In the file sysctl.conf under /etc the following line was added

```
net.ipv4.ip_forward=1
```

The above line would activate the IP when the Raspberry Pi is booted. But as the activation was required unconditionally, it was activated immediately by running the command

```
sudo sh -c "echo 1 > /proc/sys/net/ipv4/ip_forward"
```

and the following commands were executed to create the NAT between eth0 and wlan0, i.e. between the Ethernet and the WiFi.

```
sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE
sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state RELATED,ESTABLISHED -j ACCEPT
sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT
```

To avoid creating this every time on boot, the following command was run

```
sudo sh -c "iptables-save > /etc/iptables.ipv4.nat"
```

and, in the file /interfaces under /etc/network the following line was added at the end.

```
up iptables-restore < /etc/iptables.ipv4.nat
```

The hostapd was compiled and run with the following command.

```
sudo /usr/sbin/hostapd /etc/hostapd/hostapd.conf
```

The network on one Ubuntu LTS 12.04 system and one Windows 7 system was checked and the WiFi was recognisable and able to connect to. Then the setup was finished up with the following commands which will set the WiFi to start every time the Raspberry Pi was booted into.

```
sudo service hostapd start
```

```
sudo service isc-dhcp-server start
```

The status can always be checked by replacing the above *start* with *status*. Next the daemon services were started with the following commands

```
sudo update-rc.d hostapd enable
```

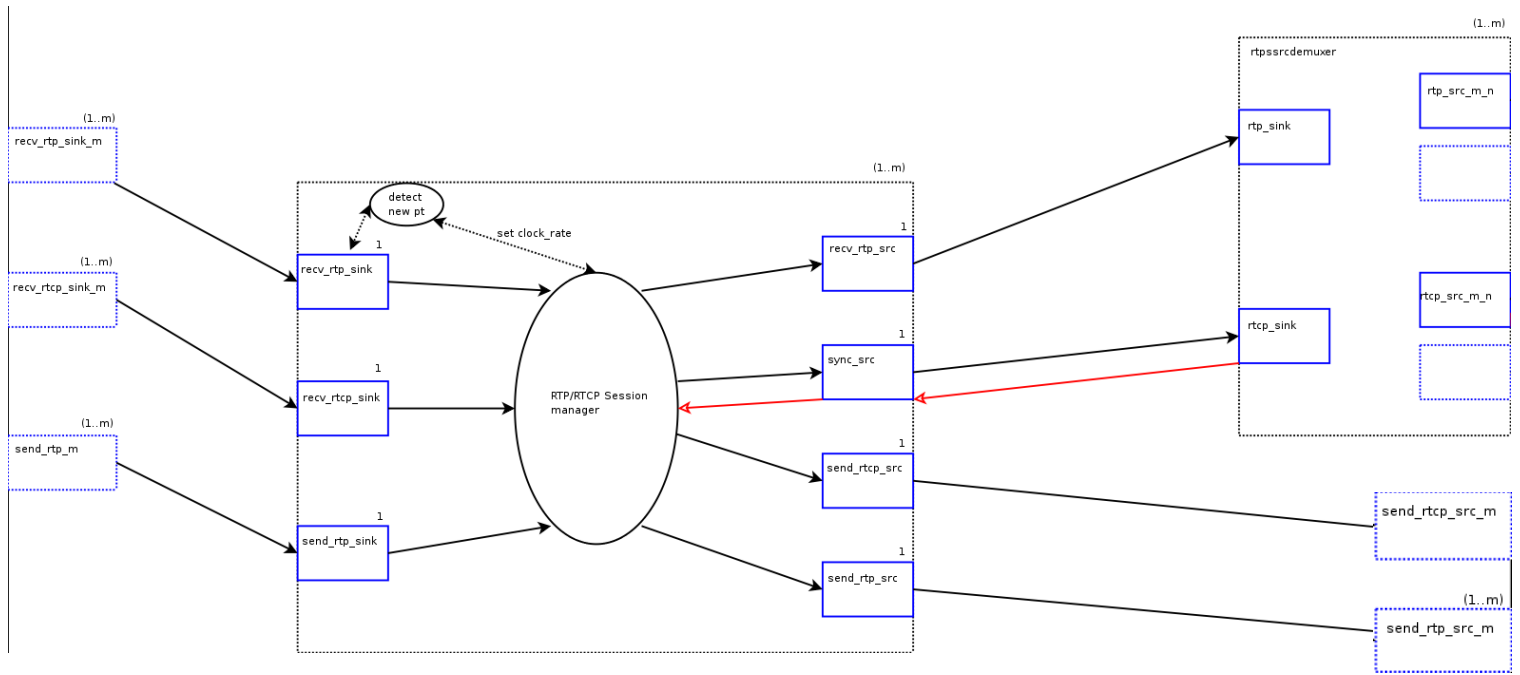
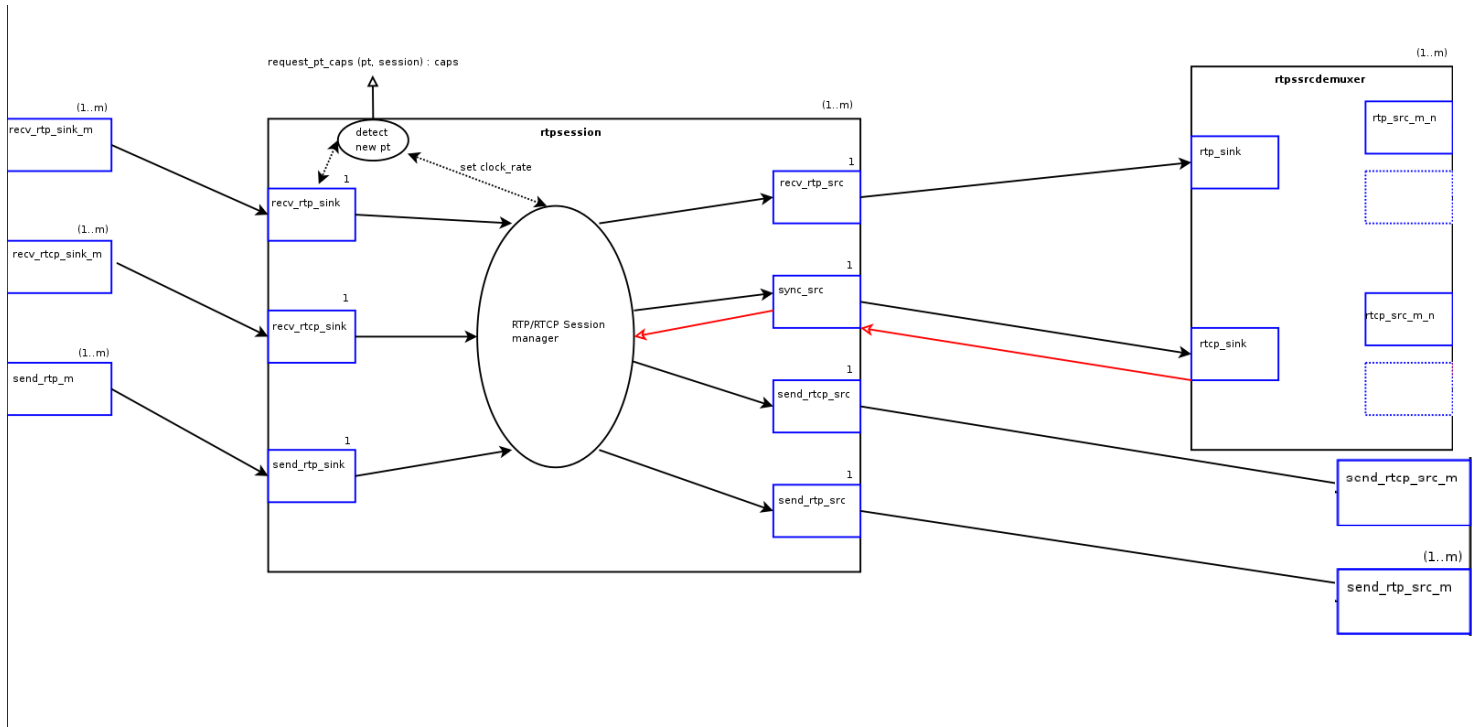
```
sudo update-rc.d isc-dhcp-server enable
```

The connection was again tested and well established. The system log on the Raspberry Pi can be checked with the following command ***tail -f /var/log/syslog***.

## **VI. RTPbin Internal Interaction Structure**

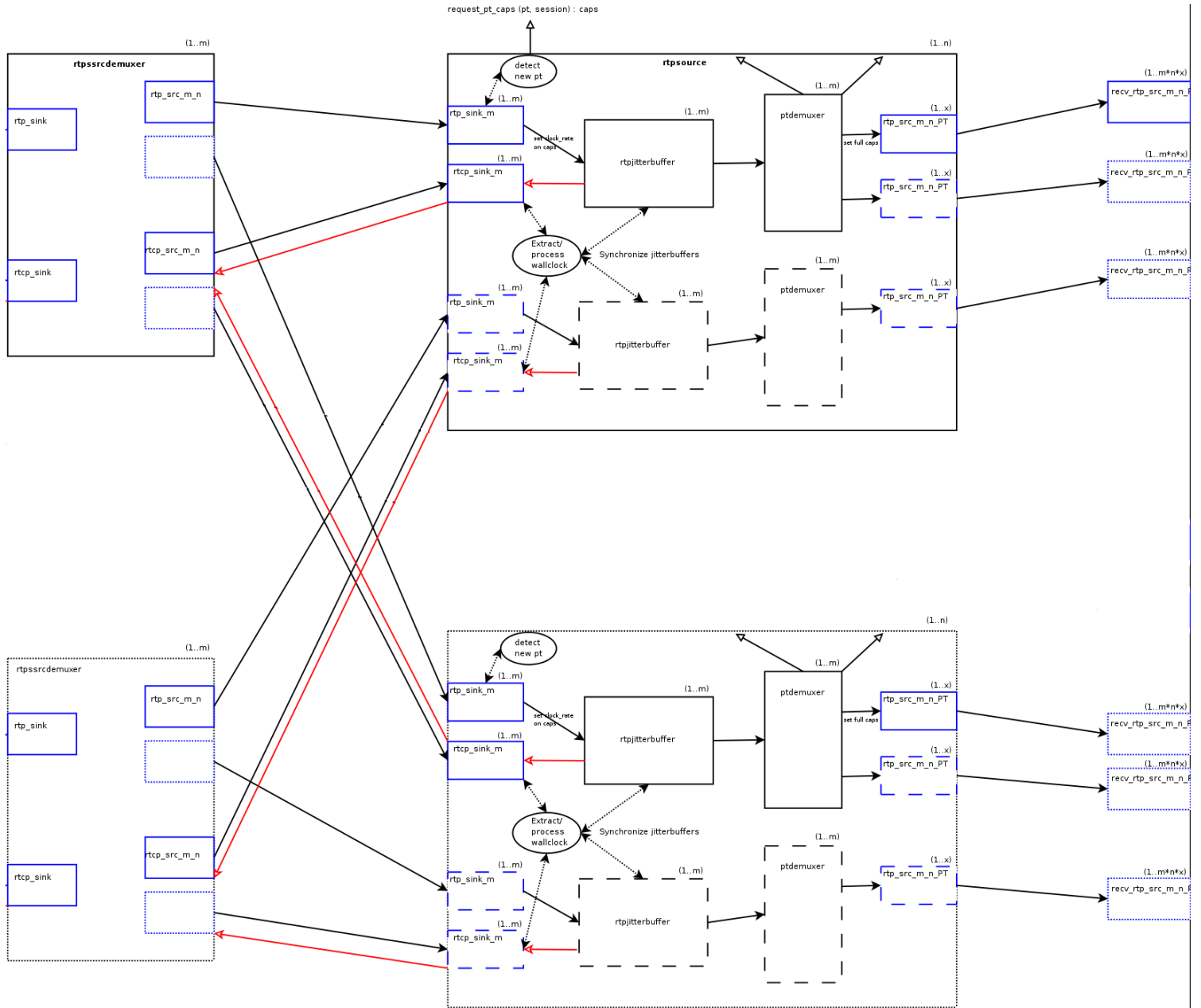
The following collaboration diagram is taken from the following website.

[<http://www.freedesktop.org/software/farstream/rtpdesign/rtpbin.png>]



**Fig. RTPbin 1**





**Fig. RTPbin 2**

## Bibliography

- [1] GStreamer Application Development Manual (1.2.3), by Wim Taymans, Steve Baker, AndyWingo, Ronald S. Bultje, and Stefan Kost
- [2] GStreamer Plugin Writer's Guide (1.2.3), by Richard John Boulton, Erik Walthinsen, Steve Baker, Leif Johnson, Ronald S. Bultje, Stefan Kost, Tim-Philipp Müller, and Wim Taymans
- [3] [Membrey, Peter and Hows, David], *Learn Raspberry Pi with Linux*. Apress Media, ISBN-13: 978-1-4302-4821-7
- [4] [Kurniawan, Agus], *Getting Started with Matlab Simulink and Raspberry Pi*. 1<sup>st</sup> Edition, 2013, ISBN: 978-1-300-95391-3, Chapter 5-Simulink and Video Capture
- [5] [Monk, Simon], *Adafruit's Raspberry Pi Lesson 6. Using SSH*, Adafruit Learning System, Adafruit Industries, Last updated on 2013-07-08 12:00:42 PM EDT
- [6] [Ladyada], *Setting up a Raspberry Pi as a WiFi access point*, Adafruit Learning System, Adafruit Industries, Last updated on 2013-09-11 02:15:35 PM EDT
- [7] RaspiCam Documentation, Raspberry Pi (*online*),  
<http://www.raspberrypi.org/documentation/usage/camera/README.md>
- [8] [Dröge, Sebastian], *GStreamer on Raspberry Pi*, Wed Mar 20 04:29:21 PDT 2013(*online*)  
<http://lists.freedesktop.org/archives/GStreamer-devel/2013-March/040163.html>
- [9] [Loonstra, Arnaud], *Videostreaming with GStreamer*, Leiden University (*online*), 5-6-2014  
[http://www.z25.org/static/\\_rd\\_/videostreaming\\_intro\\_plab/index.html](http://www.z25.org/static/_rd_/videostreaming_intro_plab/index.html)
- [10] Information on Autonomous Mobile Robot (*online*),  
<https://www.tu-chemnitz.de/etit/proaut/lehre/praktikumMobileRoboter/informationen.html>
- [11] Raspbian Operating Systems (*online*),  
<http://www.raspbian.org>
- [12] Edimax EW-7811Un Wireless USB Adaptor  
[http://www.edimax.com/edimax/merchandise/merchandise\\_detail/data/edimax/in/wireless\\_adapters\\_n150/ew-7811un/](http://www.edimax.com/edimax/merchandise/merchandise_detail/data/edimax/in/wireless_adapters_n150/ew-7811un/)
- [13] Multi Media Abstraction Layer, *My Project*, Wed May 15 2013  
<http://www.jvcref.com/files/PI/documentation/html/index.html>
- [14] LIVE555 Streaming Server (*online*),  
<http://www.live555.com/liveMedia/>
- [15] OpenCV Documentation (*online*),  
<http://docs.opencv.org/#>
- [16] [Monk, Simon], *Adafruit's Raspberry Pi Lesson 3. Network Setup*, Adafruit Learning System, Last updated on 2013-11-06 11:45:18 AM EST

## Nomenclature

fps	frames per second
gcc	GNU Compiler Collection for C
CSI	Camera Serial Interface
DTS	Decode Time Stamp
GNU	GNU's Not Unix
GObject	GLib Object System
GLib	GNOME Library
MMAL	Multi-Media Abstraction Layer
NAT	Network Address Translation
RTP	Real-Time Protocol
RTCP	Real-Time (RTP) Control Protocol
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
WiFi	Wireless Network
USB	Universal Serial Bus