IMPLENTATION OF TELEPRESENCE IN PEPPER ROBOT

A PROJECT REPORT

submitted by

Aditya Mohana Sivaraj (118005003)

towards partial fulfillment of the requirements for the award of the degree

of

Bachelor of Technology
in
Electrical & Electronics Engineering

School of Electrical and Electronics Engineering

SASTRA DEEMED TO BE UNIVERSITY
(A University established under section 3 of the UGC Act, 1956)
Tirumalaisamudiram
Thanjavur-613 401

JUNE 2018

BONA FIDE CERTIFICATE

This is to certify that the project work entitled "**IMPLEMENTATION OF TELEPRESENCE IN PEPPER ROBOT"** is a bona fide record of the work carried out by

Aditya Mohana Sivaraj (118005003)

students of final year B.Tech., Electrical and Electronics Engineering, in partial fulfillment of the requirements for the award of the degree of B.Tech in Electrical and Electronics Engineering of the **SASTRA DEEMED TO BE UNIVERSITY, Thirumalaisamudiram, Thanjavur - 613401,** during the year 2017-2018.

NAME OF THE INTERNAL GUIDE:

SIGNATURE:

Project Viva-voce held on _____

Examiner - I **Examiner – II**

ACKNOWLEDGEMENTS

First, to Prof. Amy Loutfi, for providing me with a wonderful topic and a great learning opportunity.

Second, to Mr. Sai Krishna, for guiding me towards the right direction and for showing me what research is.

Third, to Ms. Neziha Akalin, for all your help at the beginning of the project.

Fourth, to Prof. John Bosco, for providing me with the opportunity to go to an exciting place with a great research atmosphere.

Finally, to Prof. R.Sethuraman, Vice-Chancellor, SASTRA DEEMED TO BE UNIVERSITY, Dr. K.Vijayarekha, Associate Dean, EEE and all other faculty members and friends who have been of immense help at various instances of time, we could not have done this without all your contributions.

ABSTRACT

KEY WORDS: Telepresence; Teleoperation; video-call; Mobile Robotic Telepresence

A Mobile Robotic Telepresence system can be defined as a mobile robot, whose spacial movement can be teleoperated remotely, mounted with a video conferencing setup. With this setup, two parties can communicate just as though they are present at the same physical location. One of the main focusses of robotics research, currently, at Örebro University is to equip robots to interact with senior citizens at homes for the elderly. Since human health-care professionals cannot always be present and monitor the health conditions, comfort level, etc., of the residents of the home, robots can be deployed at these places. Through these robots the health-care professionals can keep a tab on the elderly people, they can administer medications, keep them comfortable and it also has the added advantage of them being able to communicate with their loved ones.

There are commercially available robots that exclusively serve the purpose of telepresence like Giraff (now discontinued), Amy A1, Ohmni, etc. However, there is a need for this function to be integrated with a more wholesome robot for projects like the aforementioned. Pepper is one such wholesome robot being looked into. Pepper is a robot developed by SoftBank Robotics for both commercial and research purposes. The aim of this project is to implement the concept of telepresence in Pepper robot. Pepper has a tablet screen at its chest through which the video communication is implemented.

TABLE OF CONTENTS

List of Figures

CHAPTER 1

1.1 INTRODUCTION

Telepresence is a phenomenon where two parties are communicating with each other with the feeling that both of them are present at same spatial location at the same time. Along with the feature of video-communication, it gives the user the ability to remotely maneuver the robot, generally in a 2-D plane, from a distant location. For this very reason, these machines are aptly called Mobile Robotic Telepresence device. These devices are created with the aim to make the pilot user and the person on the other side of the communication channel feel like they are talking to each other in person, to achieve this the pilot user is given a virtual interface or a physical controller like a joystick to remotely drive the robot. The pilot user connects to the robot using a client application which provides the tools to maneuver the robot and an Audio-Video feed of the server.

These devices are helpful in many places like retirement homes, hospitals, schools, work-places, etc. At retirement homes the residents are very likely to be in a fragile state and in need of constant monitoring and the human health-care professionals will not be able to do that. To help them with this problem, these robots can be deployed. In addition to this application there is another area of research in the same environment which is to make the robot behave in accordance to the cultural preferences of the person it assigned to take care of. Similar applications are required in a regular hospital too with a faster decision-making speed and more precision. In work-places these devices can be used to conduct meeting between two parties who are in different places in the world but need the mobility to observe things at different

places of the building. In schools, these devices can be used by teachers who is not present at the same place as the students, so that the teacher has more mobility to observe his/her students. In a similar way the students can use it in case they are not in a position to be physically present in class.

Currently, telepresence is not a completely new topic among researchers. There are many companies building exclusive telepresence robots like PRoP (Personal Roving Presence), Giraff, Amy A1, Ohmni, etc. These robots are manufactured with the exclusive purpose of telepresence. Giraff is a 163 cm tall robot with an LCD screen, speakers, wide-angle camera and a microphone mounted on top and wheels at the bottom. It uses a software called VSee for the conferencing and teleoperation. In this program the pilot user can use his/her mouse to navigate around. They can click on a place on the video feed to move to that place. The speed of its movement depends on the length of the line the pilot draws on the video feed.

Fig 1.1: Amy A1

Fig 1.2: Giraff

Pepper is a humanoid robot built by Aldebaran Robotics, a subsidiary of SoftBank Corp of Japan, headquartered in Paris, France. It was primarily built to display emotions and connect with people.

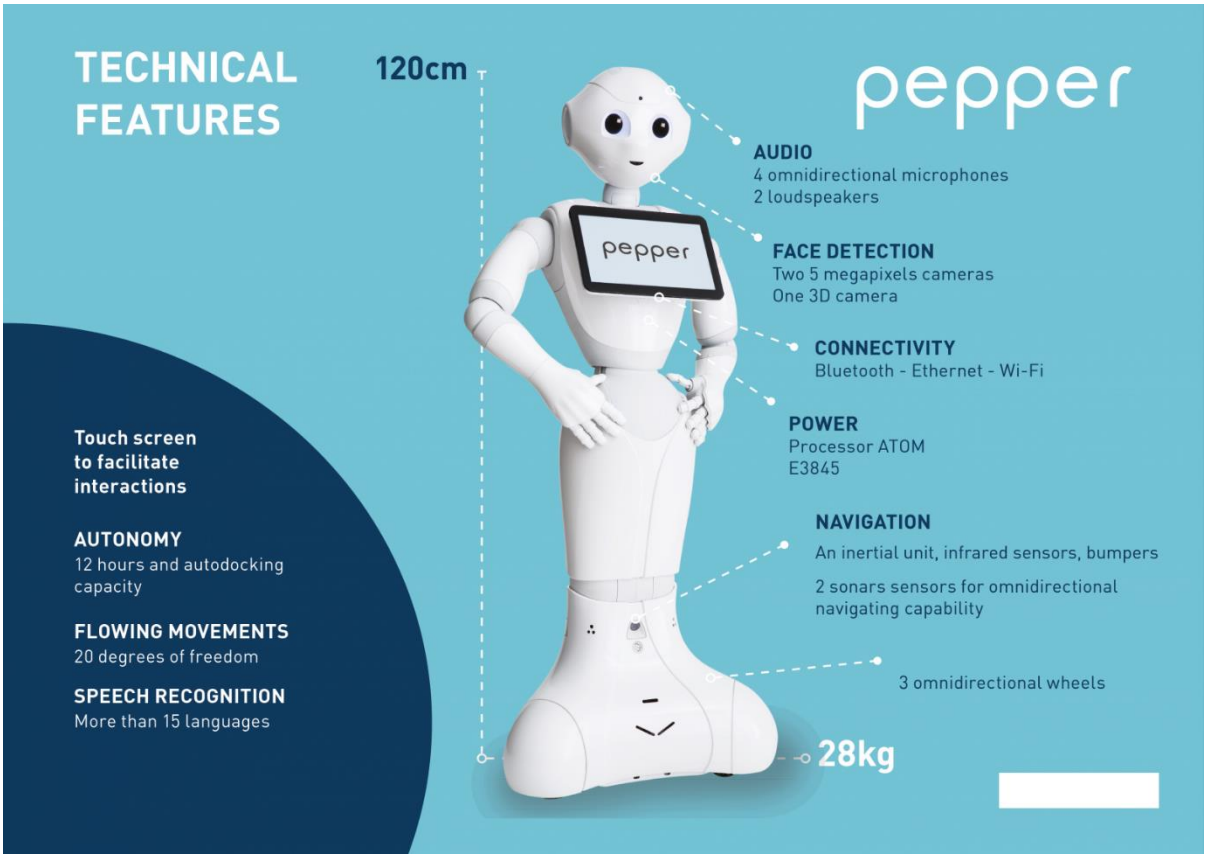

Fig 1.3: Pepper Technical Features

To help with its navigation it has infrared sensors and sonar sensors at the near bottom of its body. It also has bumpers on various parts of its body to protect itself in case of a collision. It has 3 multi-directional wheels that is responsible for its movement on the floor, it also allows the robot to have a 360° rotation. There 20 in-built motors which will control the movement of arms, fingers, head, hip, etc. It keeps a handle on its balance using an inertial unit which houses multiple 3-axis gyrometers with an angular speed of 500°/s (approx.) and a 3-axis accelerometer with an acceleration of 2g (approx.). It is powered by Lithium ion battery.

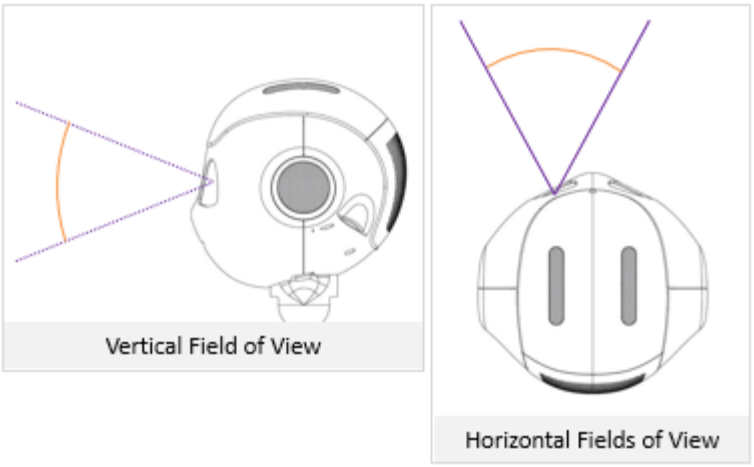

Fig 1.4: 3D Sensor

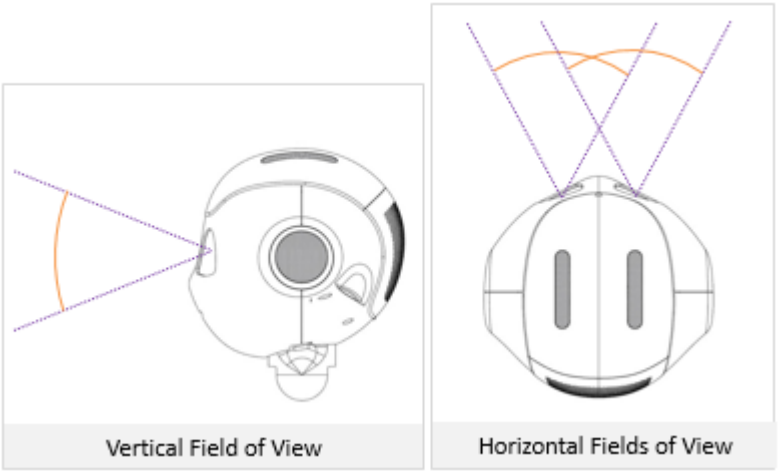

Fig 1.5: Stereo Camera

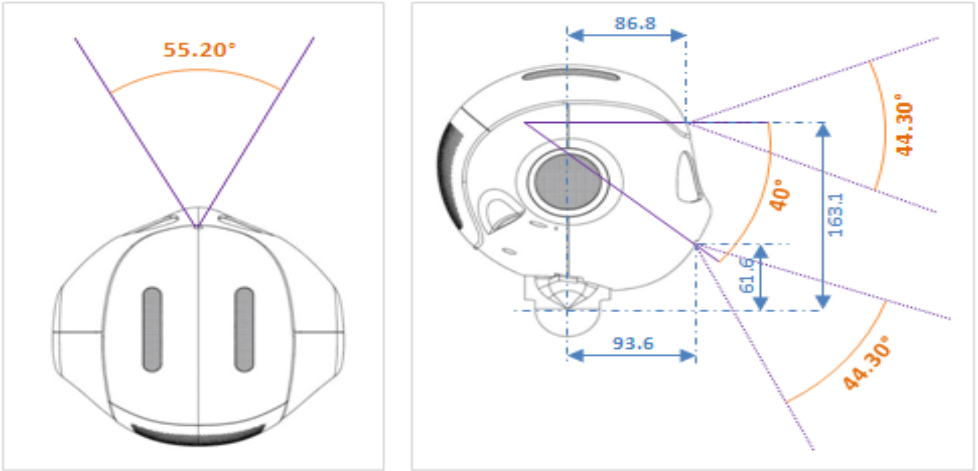

Fig 1.6: 2D Cameras

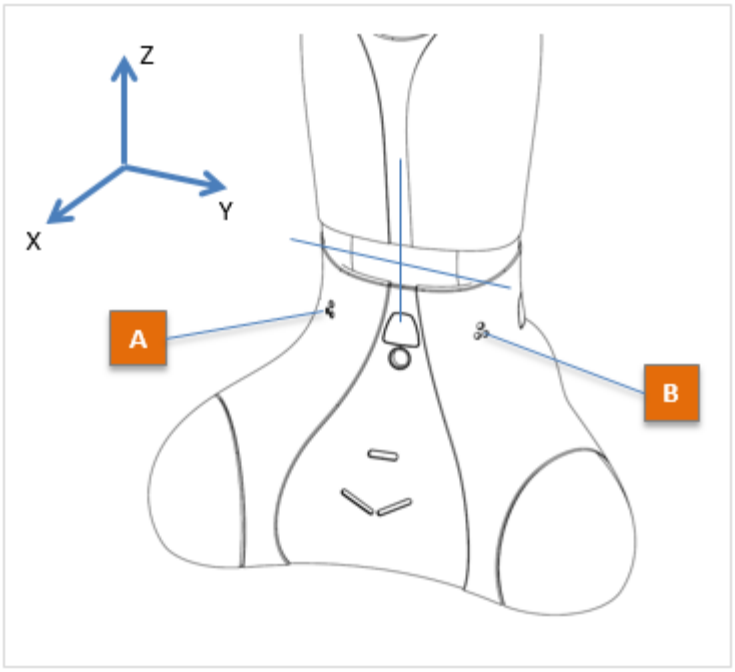

Fig 1.7: Infrared Sensors

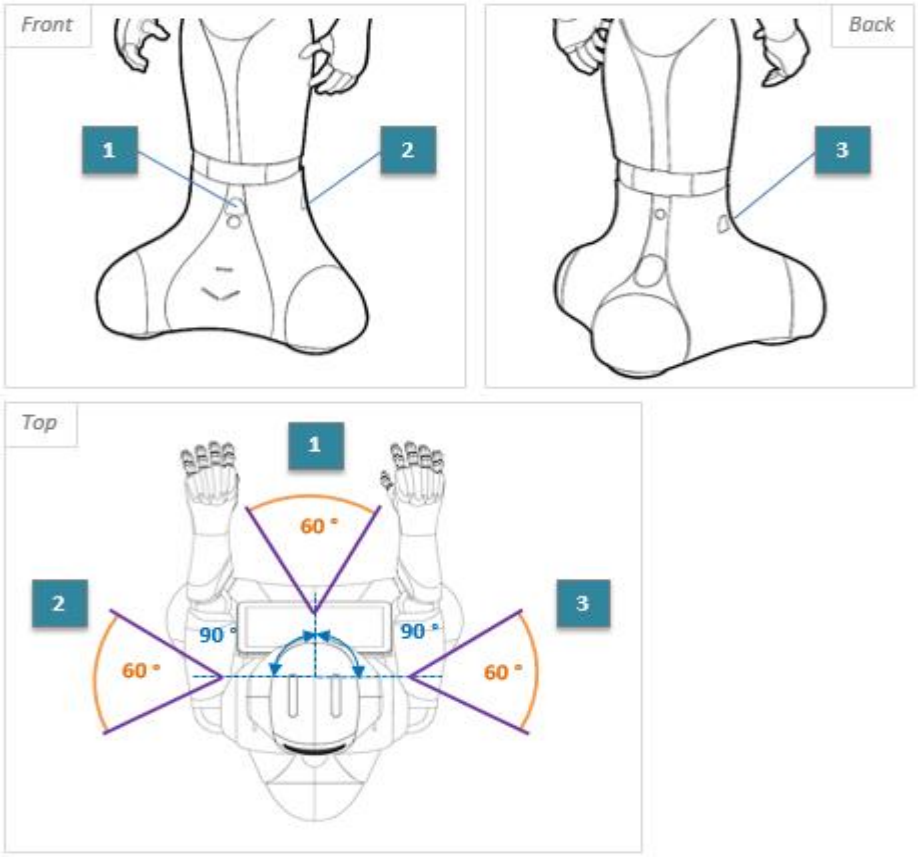

Fig 1.8: LASER Sensors

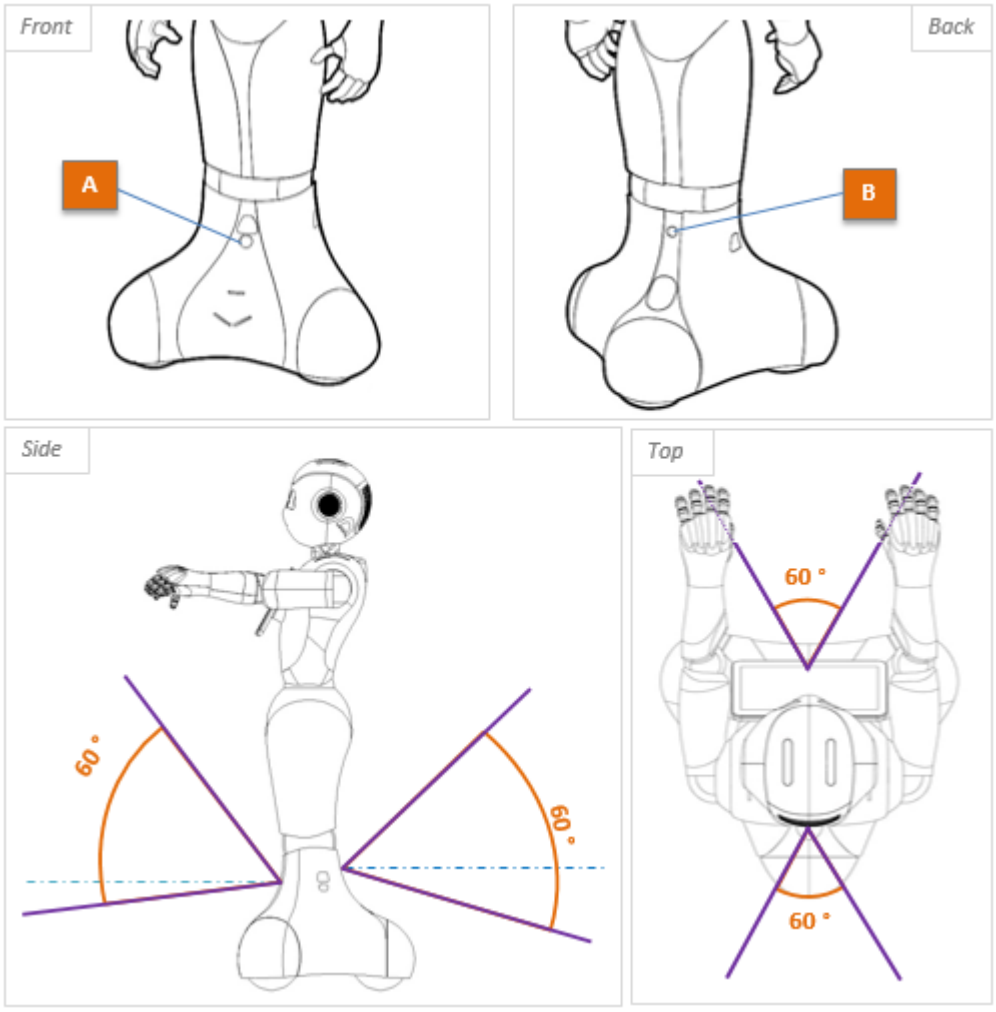

Fig 1.9: SONAR Sensors

Coming to its vision, it is equipped with a high-resolution camera in each of its eyes. When vision feeds of these two cameras are put together, they form a 3-D image, much like our eyes. This is called stereo image. But for a normal use it has two 2-D cameras at its forehead, one of its eyes and at its mouth. It has a laser depth sensor in of its eyes which calculates the distance between an object and itself. Other than its eye, it has a laser sensor in each of its ears and at each side of the virtual triangle at the bottom.

Other features include tactile sensors at the top of its head and on its hands, and nearly 30 Hall effect sensors.

1.2 BASIC PROGRAM STRUCTURE

1.2.1 LIBRARIES AND SOFTWARE REQUIRED

- ROS (Robot Operating System – An open source middleware for robots.)
- Python 2.7 (Language in which it is coded.)
- Qi Framework (Contains functions to handle the Pepper robot.)
- PyQt4 (The library that the GUI is built on.)
- tmux (The Terminal/Command Prompt application embedded in the GUI.)
- Flask (Web development framework for Python.)
- WSGIserver (Library that takes care of the server side implementation.)
- PyGObject (Library to handle media files.)
- Choregraphe (Software for handling Pepper Robot)
- cv2 (OpenCV)
- STK (Toolkit to handle Choregraphe functions outside.)
- pepper_bringup (ROS package)

1.2.1.1 ROS

ROS stands for Robot Operating System. Although the name contains the words *operating system* it is not technically an operating system. It is actually a middleware that takes care of the communication between the user and the controls the sequence of the robot's actions. More specifically, it provides services designed for heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. It allows developers to create supporting applications for handling the hardware of individual robot models. There is one for Pepper too.

1.2.1.2 QI FRAMEWORK

It is a free developers' library that contains the classes and methods to control different features/functionalities of the robot. It has separate packages for Python, Java and C++. It is also packaged differently for Linux, Windows and MacOS platforms. It controls features like webpage display on its tablet, getting a feed from its cameras, control and read data off of different sensors, etc.

1.2.1.3 PYQT4

Qt is an IDE and GUI application development framework. It works on the basis of signals, which are passed when a certain action is done on the GUI. It is basically built for C++. PyQt is a Python binding of the cross-platform GUI toolkit Qt, implemented as a Python plug-in. PyQt4 is the version used to build the GUI in the project.

1.2.1.4 TMUX

tmux is basically a terminal/command prompt multiplexer. It allows multiple separate terminal sessions inside a single window. In this project it is used as an embedded terminal window that will execute a certain set of commands when different buttons are pressed.

1.2.1.5 FLASK

Flask is a web development micro framework that was built exclusively for Python. It is called a microframework because it does not need any particular tools or libraries. In this project it is used to create the webpage in which the stream of images will be displayed on in the tablet on Pepper's chest.

1.2.1.6 WSGISERVER

WSGI stands for Web Server Gateway Interface. WSGIserver is a library that takes care of the server side of the WSGI interface for running Python applications. A WSGI server simply invokes a callable object on the WSGI application as defined by the PEP 3333 standard. In this program it is used to send a stream of images (or frames) embedded in a HTML page over the network.

1.2.1.7 PYGOBJECT

PyGobject is a python package that provides bindings for all GObject based libraries. GObject based libraries include GStreamer, GTK, Glib, etc. These libraries are basically C++ based and PyGobject provides a means to access them using python. We are using GStreamer here to try to create pipelines through which audio data streams.

1.2.1.8 CHOREGRAPHE

Choregraphe is a licensed software (by SoftBank Robotics) that is used access Aldebaran built robots. It is a multi-platform desktop application used to create different complex actions for the robots. It can also create an application package that can be installed inside an Aldebaran robot and can be called whenever we want. The actions we create are called behaviors.

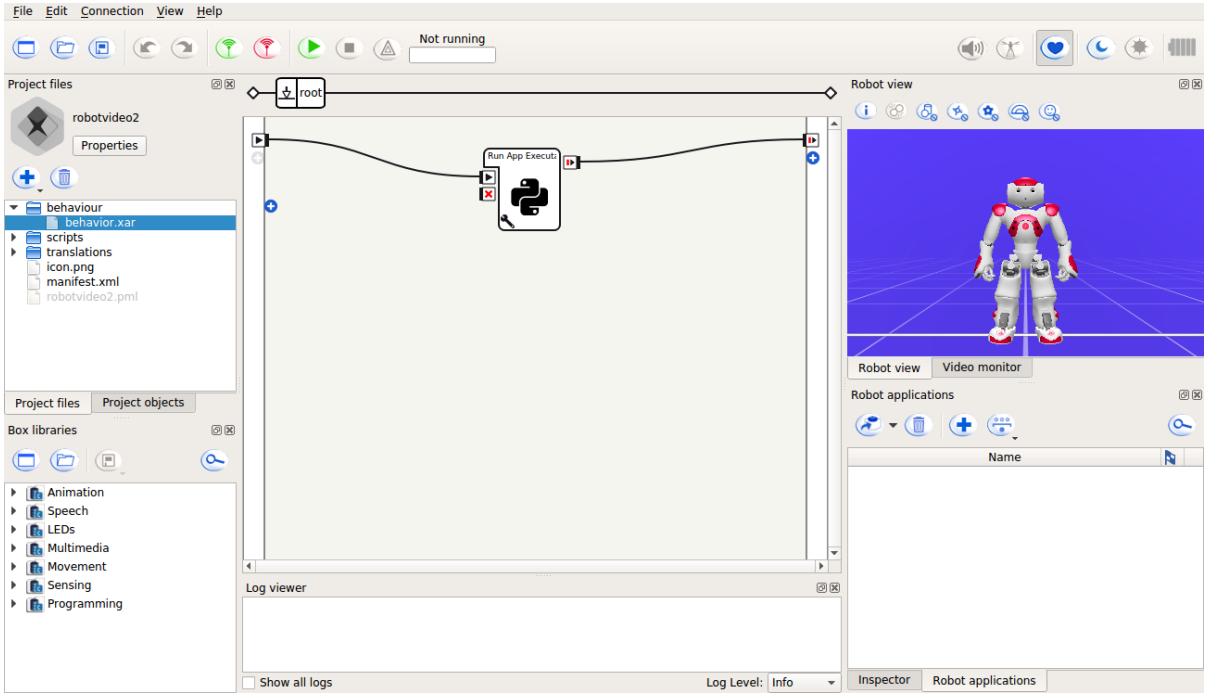

Figure 1.10: Choregraphe

1.2.1.9 CV2

cv2 is the latest opencv library built for Python. It is an extensive library for a range of computer vision applications. It is open source as the name suggests (Open Source Computer Vision Library). It is the most popular library for all computer vision applications. It has C++, Python and Java interfaces. It supports Windows, Linux, MacOS, iOS and Android operating systems. It is very popular because it can take advantage of multi-core processing available in almost all modern computers.

1.2.1.10 STK

STK stands for Student Tool Kit. It is a free library created by Aldebaran Robotics for students to access Choregraphe type programming structure outside Choregraphe. It is built for Python and Java. Their definition – "Studio tool kit is a set of libraries and tools used by Studio Team to develop awesome applications ".

1.2.1.10 PEPPER_BRINGUP

It is a ROS package built independent developers to integrate ROS's message passing service with the robots based on naoqi architecture (Built by Aldebaran).

1.2.2 FLOW STRUCTURE OF THE PROGRAM

Once all the required software and libraries are installed, one should execute the python file called 'gui_main.py'. This will open a Graphical User Interface (GUI). In this GUI there will be a text box, in which the IP address of the robot should be entered. Once the IP address is entered, the Roscore start button must be pressed. This will start a terminal in the right corner of the GUI and execute a roslaunch command in it.

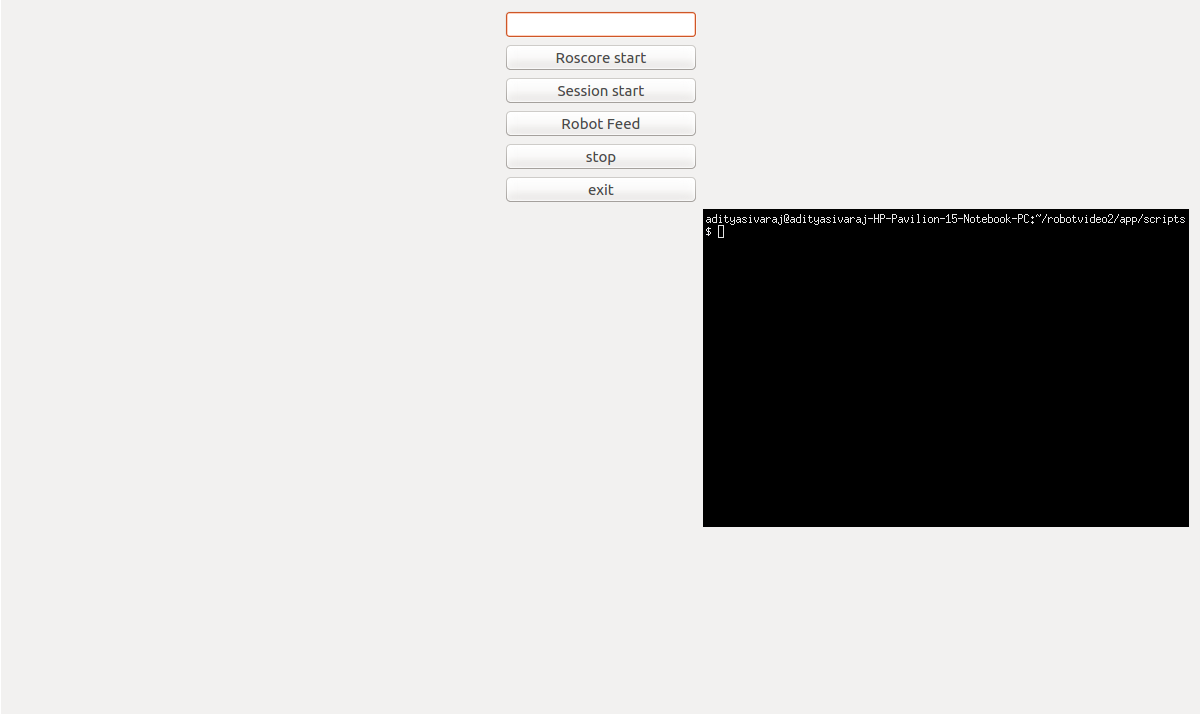

Figure 1.11: GUI Initial

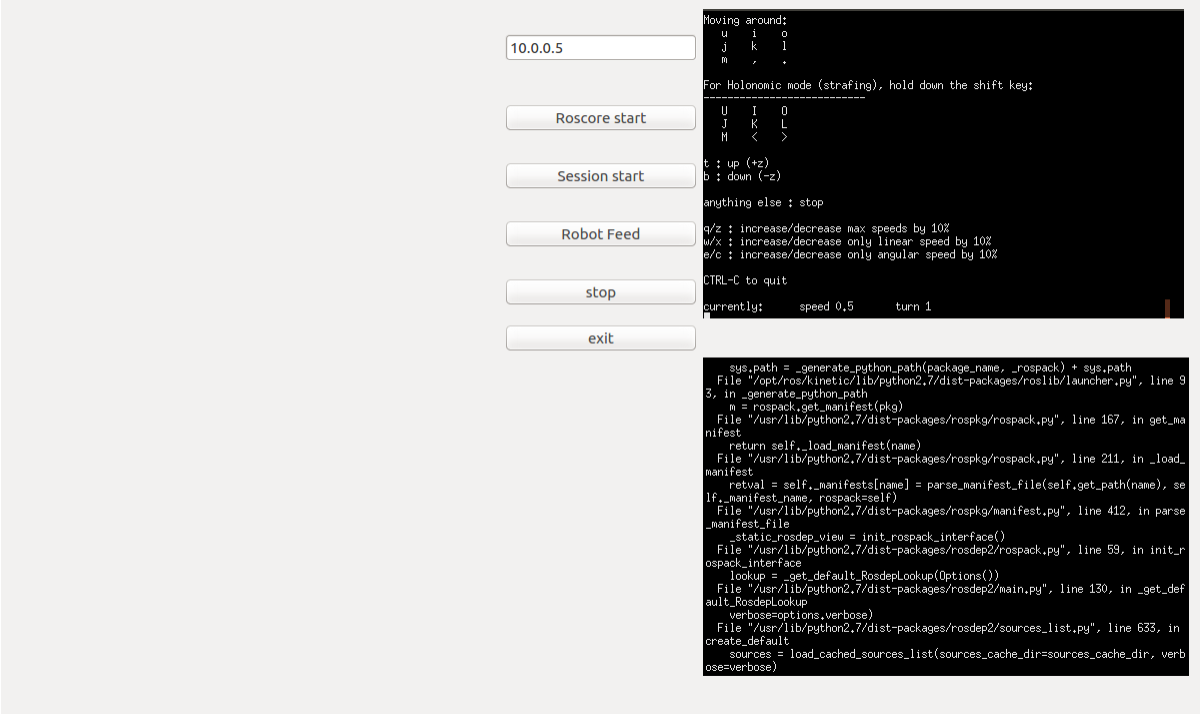

Figure 1.12: GUI after Roscore Start

This command launches eight files in total –

- **Pepper Publisher**
- **naoqi_driver**
- **SONAR sensor**
- **LASER Sensor**
- **pose_manager**
- **Perception**
- **Telepresence - 1. com_cam.py**

  2. teleop_twist_keyboard.py

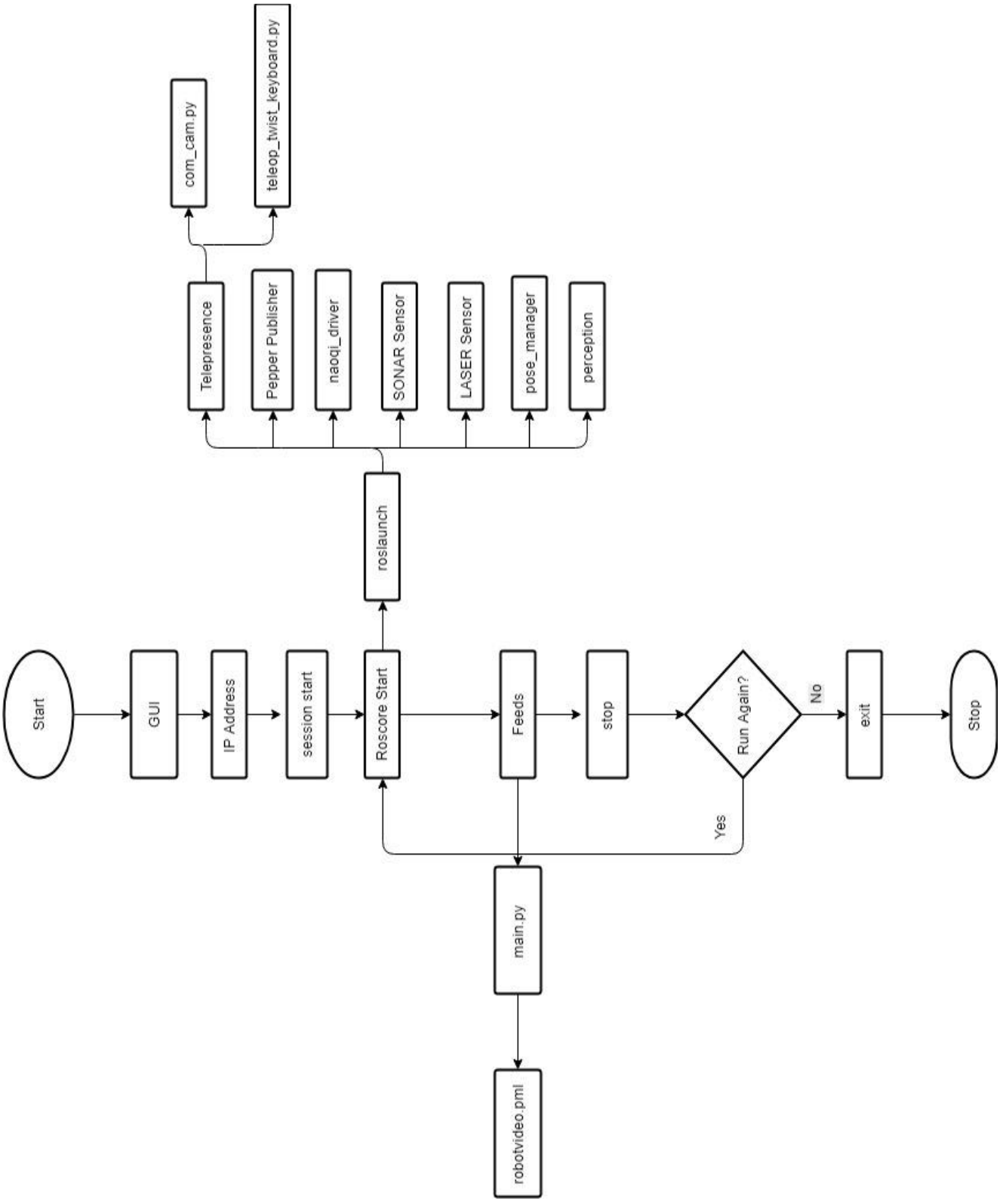

Figure 1.13: Flowchart of the Telepresence Setup

Pepper publisher takes care of the message passing between the roscore (the computer in which the program is executed). naoqi_driver is a ros based python program that connects the hardware of the robot to the computer (roscore). The SONAR Sensor program initiates and handles the SONAR sensors on the robot which aids in detecting obstacles during movement. The LASER sensor program initiates and handles the LASER sensors of the robot which also aids in obstacle detection during movement. The perception program takes care of the other cameras to detect human beings and other obstacles. The last one – Telepresence is another launch file which will launch two files – com_cam.py and teleop_twist_keyboard.py. The com_cam.py program is responsible for getting the computer's camera feed as frames using cv2 library, pastes it on a HTML page and sends it over the network. It also creates a pipeline for receiving audio feed from pepper. The teleop_twist_keyboard.py program is a ros based python program that takes keystroke as input. Based on the input, an appropriate command is published as a message for the Pepper robot.

When the Session start button is pressed a connection between the Pepper robot and the GUI program is established.

When the Robot feed button is pressed a video feed from the robot's top camera is displayed in the top left corner of the GUI. This button also executes another program called main.py which is connected to a Choregraphe executable program called robotvideo.pml. This program is basically executed on the robot to receive the video feed from the computer as a HTML page and display the feed on Pepper's tablet as webpage. This program also contains a pipeline that sends the audio feed data from the tablet's microphone.

The stop button will send a stop command to all the aforementioned programs. It can be started again by starting the process with Roscore start button.

The exit button will close all connections, terminate all programs and destroy the instance of the GUI.

CHAPTER 2

2.1 COMPUTER TO ROBOT FEED PROGRAM

com_cam.py

```
#!/usr/bin/env python
from flask import Flask, render_template, Response, stream_with_context
from camera import VideoCamera
import signal
import sys
#import argparse
import gi
gi.require_version('Gst', '1.0')
from gi.repository import GObject, Gst
from gevent.wsgi import WSGIServer
import os
import subprocess


video = 10000
audio = 10001
http = 10002


GObject.threads_init() #initialising gstreamer threads
Gst.init(None)

#initialising the audio pipeline
audio_pipeline = Gst.Pipeline('audio_pipeline')

#defining audio pipeline attributes
audio_udpsrc = Gst.ElementFactory.make('udpsrc', None)
audio_udpsrc.set_property('port', audio)

audio_caps = Gst.caps_from_string('application/x-rtp,media=(string)audio, clock-rate=(int)44100, width=16, height=16, encoding-name=(string)L16, encoding-params=(string)1, channels=(int)2, format=(string)S16LE, channel-positions=(int)1, payload=(int)96')
audio_filter = Gst.ElementFactory.make('capsfilter', None)
audio_filter.set_property('caps',audio_caps)

audio_depay = Gst.ElementFactory.make('rtpL16depay', None)
audio_convert = Gst.ElementFactory.make('audioconvert', None)

audio_sink = Gst.ElementFactory.make('alsasink', None)
audio_sink.set_property('sync',False)

#linking the various attributes
audio_pipeline.add(audio_udpsrc,audio_filter,audio_depay,audio_convert,audio_sink)

audio_udpsrc.link(audio_filter)
audio_filter.link(audio_depay)
audio_depay.link(audio_convert)
audio_convert.link(audio_sink)

```

```
# ip = "10.0.0.4"
# audio_port = 80

# s_audio_pipeline = Gst.Pipeline('s_audio_pipeline')

# s_audio_src = Gst.ElementFactory.make('autoaudiosrc')

# s_audio_convert = Gst.ElementFactory.make('audioconvert')
# s_audio_caps = Gst.ElementFactory.make('audio/x-raw-int,channels=2,depth=16,width=16,rate=44100')
# s_audio_filter = Gst.ElementFactory.make('capsfilter')
# s_audio_filter.set_property('caps',audio_caps)
# s_audio_pay = Gst.ElementFactory.make('rtpL16pay')
# s_audio_udp = Gst.ElementFactory.make('udpsink')
# s_audio_udp.set_property('host',ip)
# s_audio_udp.set_property('port',int(audio_port))

# s_audio_pipeline.add(s_audio_src,s_audio_convert,s_audio_filter,s_audio_pay,s_audio_udp)
# s_audio_src.link(s_audio_convert)
# s_audio_convert.link(s_audio_filter)
# s_audio_filter.link(s_audio_pay)
# s_audio_pay.link(s_audio_udp)

def signal_handler(signal, frame):
    # Free resources.
    audio_pipeline.set_state(Gst.State.NULL)
    #s_audio_pipeline.set_state(Gst.State.NULL)
    print('Exit')
    sys.exit(0)

#creating a webpage using flask framework
app = Flask(__name__)

@app.route('/')
def index():
    return render_template('index.html')

def gen(camera):
    # Set our pipelines state to Playing.
#    video_pipeline.set_state(Gst.State.PLAYING)
    audio_pipeline.set_state(Gst.State.PLAYING)
    #s_audio_pipeline.set_state(Gst.State.PLAYING)
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
            b'Content-Type: image/jpeg\r\n\r\n' + frame + b'\r\n\r\n') #sending frames to the webpage

@app.route('/video_feed')
def video_feed():
    try:
        return Response(stream_with_context(gen(VideoCamera())), mimetype='multipart/x-mixed-replace; boundary=frame')#accessing the camera.py program, which will reurn images.

    except Exception:
        return None

    if __name__ == '__main__':
        signal.signal(signal.SIGINT, signal_handler) #incase ^C is called.
```

```
102.       #   app.run(host='0.0.0.0', port = http, debug=True, threaded=True)
103.           http_server = WSGIServer(('0.0.0.0', http), app) #creating a server with
    open ip
104.           http_server.serve_forever()
```

camera.py

```
1.  import cv2
2.  #import cv
3.  import time
4.
5.  class VideoCamera(object):
6.      def __init__(self):
7.          # Using OpenCV to capture from device 0. If you have trouble capturing
8.          # from a webcam, comment the line below out and use a video file
9.          # instead.
10.         self.video = cv2.VideoCapture(0)
11.
12.
13.     def get_frame(self):
14.         success, image = self.video.read()
15.         # We are using Motion JPEG, but OpenCV defaults to capture raw images,
16.         # so we must encode it into JPEG in order to correctly display the
17.         # video stream.
18.         ret, jpeg = cv2.imencode('.jpg', image)
19.         #cv2.flip(jpeg, jpeg, 1)
20.         cv2.waitKey(10)
21.         return jpeg.tobytes()
22.
23.     def __del__(self):
24.
25.         self.video.release()
```

The second program camera.py is imported into the first program com_cam.py.

In com_cam.py all the libraries including the camera program is imported first. The version of GStreamer must also be mentioned (1.0 here). Then the port numbers are initialized separately for http, audio and video ports. Now the GObject and GStreamer threads are initialized. Now we create the receiving end of an audio pipeline. First, the name is initialized, then the source is defined as udpsrc which is basically a source from a network with UDP type packaging. Then the port number is linked with the pipeline. Then, the properties of the audio stream of declared, properties like encoding format, clock-rate, etc., are defined. Now that the properties are defined, the payload type is

declared, and the method of conversion is defined. Finally, the data sink and its type are defined. Now all these elements are added to the pipeline and linked to one another in a sequence. After the pipeline, a signal_handler method is defined, which is called if the program is terminated in the middle of execution. The pipeline is closed inside that method. In the next few lines the webpage for displaying the frames is built. To do this we must have already created a sample HTML page called index.html. In the method called video_feed() we return the result of another method gen() along with the content type defined. The gen() method is actually an abbreviation for generator. In gen() method the class VideoCamera() from the program camera.py is called. The get_frame() method inside the class VideoCamera() returns the recently captured frame. This frame is then embedded inside the HTML page and returned. Finally, in the main function, an open IP is created and the data is transmitted infinitely until another device calls for its stream, now the data is transmitted to this device infinitely.

In camera.py function, we make use of the opencv library to capture the video feed from the computer's camera. This image is then encoded into .jpeg format and returned to its caller in bytes. In case the instance of the class is destroyed then all the data is released.

.

2.2 GUI PROGRAM

gui_main.py

```
1.  import qi
2.  import sys
3.  import argparse
4.  from PyQt4.QtCore import *
5.  from PyQt4.QtGui import *
6.  from PyQt4.QtGui import QPixmap, QTextEdit, QPainter
7.  from PyQt4.QtCore import QTimer
8.  import vision_definitions
```

```
import main as act

session = qi.Session()

user_ip = "10.0.0.10"
user_http_port = 9559

class tele_pepper(QWidget):

       def __init__(self):
        QWidget.__init__(self)
        self._processes = []
        self.resize(1200, 1000)
        self.terminal = QWidget(self)
        layout = QGridLayout(self)

        layout.addWidget(self.terminal, 10, 8, 5, 5)
        self.nav1 = QPixmap('nav.png')
        self._start_process(
            'xterm',
            ['-into', str(self.terminal.winId()),
             '-e', 'tmux', 'new', '-s', 'my_session']
        )
        self.button = QPushButton('Roscore start')
        self.sessbutton = QPushButton('Session start')
        self.rbutton = QPushButton('Robot Feed')
        self.sbutton = QPushButton('stop')
        self.ebutton = QPushButton('exit')
        self.imgL = QLabel(self)
        self.ipbar = QLineEdit(self)
        self.nav = QLabel(self)

        layout.addWidget(self.imgL, 0, 0, 5, 5)
        layout.addWidget(self.nav, 0, 8, 5, 5)
        layout.addWidget(self.ipbar, 0, 5, 1, 2)
        layout.addWidget(self.button, 1, 5, 1, 2)
        layout.addWidget(self.sessbutton, 2, 5, 1, 2)
        layout.addWidget(self.rbutton, 3, 5, 1, 2)
        layout.addWidget(self.sbutton, 4, 5, 1, 2)
        layout.addWidget(self.ebutton, 5, 5, 1, 2)

        self.sess = session

        self.res = vision_definitions.kQVGA
        self.clrSpc = vision_definitions.kRGBColorSpace

        self.setLayout(layout)

        self.button.clicked.connect(self._list_command)
        self.sessbutton.clicked.connect(self.sessStart)
        self.rbutton.clicked.connect(self.session_init)
        self.sbutton.clicked.connect(self._exitprg)
        self.ebutton.clicked.connect(self._exitapp)

       def sessStart(self):
        self.pport = 9559
         self.sess.connect("tcp://" + str(self.ip) + ":" + str(self.pport))

       def session_init(self):

         self.video_service = self.sess.service('ALVideoDevice')
```

```
70.          self.memory = self.sess.service('ALMemory')
71.          self.video_service.setParam(vision_definitions.kCameraSelectID, 0)
72.          self.videoClient = self.video_service.subscribe("pyth", self.res, self.clr
    Spc, 15)
73.
74.          self.program = QProcess()
75.          prg = "python main.py --qi-url " + self.ip
76.          self.program.start(prg)
77.
78.          self.run()
79.
80.        def run(self):
81.
82.         self.timer = QTimer(self)
83.         self.timer.timeout.connect(self.upd)
84.         self.timer.start(5)
85.
86.        def _start_process(self, prog, args):
87.         child = QProcess()
88.         self._processes.append(child)
89.         child.start(prog, args)
90.
91.
92.        def _list_command(self):
93.         self.ip = self.ipbar.text()
94.         self.nav.setPixmap(self.nav1)
95.         self.nav.show()
96.
97.         self._start_process(
98.             'tmux', ['send-keys', '-
    t', 'my_session:0', 'cd', 'Enter', 'roslaunch pepper_bringup tele_pepper.launch nao
    _ip:=', self.ip, ' roscore_ip:=10.0.0.10', 'Enter'])
99.
100.
101.       def upd(self):
102.           self.naoImage = self.video_service.getImageRemote(self.videoClient)
103.           self.outImage = QImage(self.naoImage[6],           # Pixel array.
104.                                  self.naoImage[0],           # Width.
105.                                  self.naoImage[1],           # Height.
106.                                  QImage.Format_RGB888)
107.           self.disp(self.outImage)
108.
109.       def disp(self, img):
110.
111.           self.imgL.setPixmap(QPixmap.fromImage(img))
112.           self.imgL.setScaledContents(True)
113.
114.       def _exitprg(self):
115.               self._start_process('tmux', ['send-keys', '-
    t', 'my_session:0', '^C', 'Enter'])
116.           self.timer.stop()
117.           tabletService = self.sess.service("ALTabletService")
118.           tabletService.cleanWebview()
119.           #tabletService.resetTablet()
120.           self.video_service.unsubscribe(self.videoClient)
121.
122.       def _exitapp(self):
123.               self._start_process('tmux', ['send-keys', '-
    t', 'my_session:0', 'tmux kill-server', 'Enter'])
124.
125.               tabletService = self.sess.service("ALTabletService")
```

```
126.                 tabletService.hideWebview()
127.                 #tabletService.resetTablet()
128.                 self.video_service.unsubscribe(self.videoClient)
129.                 self.destroy()
130.
131.         def __del__(self):
132.                 self._start_process('tmux', ['send-keys', '-
    t', 'my_session:0', 'tmux kill-server', 'Enter'])
133.                 self.video_service.unsubscribe(self.videoClient)
134.                 self.destroy()
135.
136.
137.         if __name__ == "__main__":
138.
139.             app = QApplication(sys.argv)
140.             main = tele_pepper()
141.             main.show()
142.             sys.exit(app.exec_())
```

This program makes use of the Qt 4 library to build the GUI using PyQt4 bindings. This program is starting with definition of the GUI called tele_pepper which is of type QWidget. In the constructor of the class, first, the instance of the GUI is initialized. Next, an empty list for containing the various commands for the terminal is declared. After the dimensions of the GUI are set the terminal widget is declared. Then, the terminal is set at a place in the GUI and the current session is declared. Next the various buttons, text box and two labels, one for the directions image and the other for the video feed, are declared. These objects are now set inside the GUI at appropriate places. Now the signals to call functions when the appropriate buttons are clicked are defined. Also, the resolution and colorspace for the video feed from the robot is defined.

The sessStart() function will create the connection between the program and the robot.

In the session_init() function, services to various functions like videoservice and memoryservice is subscribed. Then, the video feed from the top camera is subscribed. Finally, the program that receives and displays the video feed from the computer, main.py, is called.

The run() function takes care of updating the frame. It creates an object for the QTimer class and calls the frame update function upd() every 5µs.

The start_process() command is called every time a command is supposed to be executed in the embedded tmux terminal. It basically creates a child process and runs the command.

In the function _list_command(), The IP address from the text box is read and initialized to a variable. Then it pastes a picture containing the instructions to maneuver the robot. Then it gives the roslaunch command, which will start the earlier mentioned eight programs together.

The upd() function receives the latest frame from the robot's top camera and the processes it to become a QImage (a Qt based object). Now, the QImage is passed on to disp() function.

The disp() function pastes the recently processed frame onto the imgL QLabel. It also makes sure that the image is scaled appropriately according to the dimensions of imgL.

In the _exitprg() function the services to various services previously subscribed to are unsubscribed. The webpage view on the tablet is closed. And a stop command is passed on to various running programs.

The _exitapp() function is similar to _exitprg() function, with this function having an additional command to destroy the instance of the GUI.

Finally, the destructor is declared which performs similar functions of _exitapp() program.

In the main function, first, the system arguments are read, then, an instance of the GUI class is created and displayed.

index.html

```
<html>

<head>
  <!--<title>Video Streaming Demonstration</title>-->
  <style>
    body,
    html {
      height: 100%;
      margin: 0;
    }

    #bg {
      /* Full height */
      height: 100%;

      position: fixed;
      top: 0;
      bottom: 0;
      left: 0;
      right: 0;

      margin: auto;
      overflow: auto;
    }
  </style>
</head>

<body>
  <!--<h1>Video Streaming Demonstration</h1>-->
  <img id="bg" src="{{ url_for('video_feed') }}">
</body>

</html>
```

2.3 TELEOPERATION PROGRAM

teleop_twist_keyboard.py

```
#!/usr/bin/env python
import roslib; roslib.load_manifest('teleop_twist_keyboard')
import rospy

from geometry_msgs.msg import Twist
```

```
6.
7.  import sys, select, termios, tty
8.
9.  msg = """
10. Reading from the keyboard  and Publishing to Twist!
11. ---------------------------
12. Moving around:
13.    u    i    o
14.    j    k    l
15.    m    ,    .
16.
17. For Holonomic mode (strafing), hold down the shift key:
18. ---------------------------
19.    U    I    O
20.    J    K    L
21.    M    <    >
22.
23. t : up (+z)
24. b : down (-z)
25.
26. anything else : stop
27.
28. q/z : increase/decrease max speeds by 10%
29. w/x : increase/decrease only linear speed by 10%
30. e/c : increase/decrease only angular speed by 10%
31.
32. CTRL-C to quit
33. """
34.
35. moveBindings = {
36.         'i':(1,0,0,0),
37.         'o':(1,0,0,-1),
38.         'j':(0,0,0,1),
39.         'l':(0,0,0,-1),
40.         'u':(1,0,0,1),
41.         ',':(-1,0,0,0),
42.         '.':(-1,0,0,1),
43.         'm':(-1,0,0,-1),
44.         'O':(1,-1,0,0),
45.         'I':(1,0,0,0),
46.         'J':(0,1,0,0),
47.         'L':(0,-1,0,0),
48.         'U':(1,1,0,0),
49.         '<':(-1,0,0,0),
50.         '>':(-1,-1,0,0),
51.         'M':(-1,1,0,0),
52.         't':(0,0,1,0),
53.         'b':(0,0,-1,0),
54.            }
55.
56. speedBindings={
57.         'q':(1.1,1.1),
58.         'z':(.9,.9),
59.         'w':(1.1,1),
60.         'x':(.9,1),
61.         'e':(1,1.1),
62.         'c':(1,.9),
63.           }
64.
65. def getKey():
66.     tty.setraw(sys.stdin.fileno())
```

```
    select.select([sys.stdin], [], [], 0)
    key = sys.stdin.read(1)
    termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
    return key

speed = .5
turn = 1

def vels(speed,turn):
    return "currently:\tspeed %s\tturn %s " % (speed,turn)

if __name__=="__main__":
        settings = termios.tcgetattr(sys.stdin)

    pub = rospy.Publisher('cmd_vel', Twist, queue_size = 1)
    rospy.init_node('teleop_twist_keyboard')

    x = 0
    y = 0
    z = 0
    th = 0
    status = 0

    try:
        print msg
        print vels(speed,turn)
        while(1):
            key = getKey()
            if key in moveBindings.keys():
                x = moveBindings[key][0]
                y = moveBindings[key][1]
                z = moveBindings[key][2]
                th = moveBindings[key][3]
                    elif key in speedBindings.keys():
                        speed = speed * speedBindings[key][0]
                        turn = turn * speedBindings[key][1]

                        print vels(speed,turn)
                        if (status == 14):
                            print msg
                        status = (status + 1) % 15
                    else:
                        x = 0
                        y = 0
                        z = 0
                        th = 0
                        if (key == '\x03'):
                            break

                    twist = Twist()
                    twist.linear.x = x*speed; twist.linear.y = y*speed; twist.linear
    .z = z*speed;
                    twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = th*t
    urn
                    pub.publish(twist)

            except:
                print e

            finally:
                twist = Twist()
```

```
126.                 twist.linear.x = 0; twist.linear.y = 0; twist.linear.z = 0
127.                 twist.angular.x = 0; twist.angular.y = 0; twist.angular.z = 0
128.                 pub.publish(twist)
129.
130.                     termios.tcsetattr(sys.stdin, termios.TCSADRAIN, settings)
```

This program basically reads the keystrokes from the keyboard and publishes the signal that corresponds to the keystroke. This program takes care of two aspects of movement i.e., the direction of movement and the speed of movement. The speed is increased gradually by 0.5 units. The speed is calculated manually in the program before publishing. The linear speed and the angular speed is done separately.

2.4 ROBOT PROGRAM

main.py

```
1.  __version__ = "0.0.8"
2.
3.  __copyright__ = "Copyright 2015, Aldebaran Robotics"
4.  __author__ = 'YOURNAME'
5.  __email__ = 'YOUREMAIL@aldebaran.com'
6.
7.  import stk.runner
8.  import stk.events
9.  import stk.services
10. import stk.logging
11.
12. import gi
13. gi.require_version('Gst', '1.0')
14. from gi.repository import GObject, Gst
15. from gevent.wsgi import WSGIServer
16.
17. class Activity(object):
18.     "A sample standalone app, that demonstrates simple Python usage"
19.     APP_ID = "com.aldebaran.activity"
20.     def __init__(self, qiapp):
21.         self.qiapp = qiapp
22.         self.events = stk.events.EventHelper(qiapp.session)
23.         self.s = stk.services.ServiceCache(qiapp.session)
24.         self.logger = stk.logging.get_logger(qiapp.session, self.APP_ID)
25.         self.user_ip = '10.0.0.10'
26.         self.user_video_port = 10000
27.         self.user_audio_port = 10001
28.         self.user_http_port = 10002
29.         GObject.threads_init() #gstreamer thread initialisation
30.         Gst.init(None)
```

```
31.
32.
33.     def on_start(self):
34.
35.         self.s.ALAutonomousLife.switchFocus("robotvideo/behaviour") #to shift the f
    ocus to this behaviour, otherwise the robot will preempt the program
36.
37.         self.s.ALTabletService.loadUrl('http://' + self.user_ip + ':' + str(self.us
    er_http_port) + '/') #loading the url of the webpage that hosts the live video feed
     from the computer.
38.         self.s.ALTabletService.showWebview() #displays the webpage on the tablet.
39.
40.         audio_pipeline = Gst.Pipeline('audio_pipeline')
41.
42.         audio_src = Gst.ElementFactory.make('autoaudiosrc')
43.
44.         audio_convert = Gst.ElementFactory.make('audioconvert')
45.         audio_caps = Gst.ElementFactory.make('audio/x-raw-
    int,channels=2,depth=16,width=16,rate=44100')
46.         audio_filter = Gst.ElementFactory.make('capsfilter')
47.         audio_filter.set_property('caps',audio_caps)
48.         audio_pay = Gst.ElementFactory.make('rtpL16pay')
49.         audio_udp = Gst.ElementFactory.make('udpsink')
50.         audio_udp.set_property('host',user_ip)
51.         audio_udp.set_property('port',int(user_audio_port))
52.
53.         audio_pipeline.add(audio_src,audio_convert,audio_filter,audio_pay,audio_udp
    )
54.         audio_src.link(audio_convert)
55.         audio_convert.link(audio_filter)
56.         audio_filter.link(audio_pay)
57.         audio_pay.link(audio_udp)
58.
59.         audio_pipeline.set_state(Gst.State.PLAYING)
60.
61.         # audio = 80
62.
63.         # r_audio_pipeline = Gst.Pipeline('r_audio_pipeline')
64.
65.         # #defining audio pipeline attributes
66.         # r_audio_udpsrc = Gst.ElementFactory.make('udpsrc', None)
67.         # r_audio_udpsrc.set_property('port', audio)
68.
69.         # r_audio_caps = Gst.caps_from_string('application/x-
    rtp,media=(string)audio, clock-rate=(int)44100, width=16, height=16, encoding-
    name=(string)L16, encoding-
    params=(string)1, channels=(int)2, format=(string)S16LE, channel-
    positions=(int)1, payload=(int)96')
70.         # r_audio_filter = Gst.ElementFactory.make('capsfilter', None)
71.         # r_audio_filter.set_property('caps',r_audio_caps)
72.
73.         # r_audio_depay = Gst.ElementFactory.make('rtpL16depay', None)
74.         # r_audio_convert = Gst.ElementFactory.make('audioconvert', None)
75.
76.         # r_audio_sink = Gst.ElementFactory.make('alsasink', None)
77.         # r_audio_sink.set_property('sync',False)
78.
79.         # #linking the various attributes
80.         # r_audio_pipeline.add(r_audio_udpsrc,r_audio_filter,r_audio_depay,r_audio_
    convert,r_audio_sink)
81.         # r_audio_udpsrc.link(r_audio_filter)
```

```
        # r_audio_filter.link(r_audio_depay)
        # r_audio_depay.link(r_audio_convert)
        # r_audio_convert.link(r_audio_sink)

        # r_audio_pipeline.set_state(Gst.State.PLAYING)


    def on_stop(self):
        "Cleanup"
        self.s.ALAutonomousLife.stopFocus() #give back the focus to the robot.
        self.logger.info("Application finished.")
        self.events.clear()

if __name__ == "__main__":
    stk.runner.run_activity(Activity)
```

robotvideo.pml

```
class MyClass(GeneratedClass):
    executable_id = "UNDEFINED"
    def onLoad(self):
        self.listener_id = None
        self.executable_manager = self.session().service("ALServiceManager")
        executable_name = self.getParameter("Executable Name")
        if ALProxy("ALSystem").systemVersion() < "2.3":
            self.executable_id = executable_name
            if "." not in executable_name:
                self.logger.info("Warning: You will have conflicts if several packages have executables called '%s'" % executable_name)
                self.logger.info("Use a newer version of NAOqi to have executables prefixed with the package ID, or prefix it yourself, in the form with <package>.<executable ID>")
        else:
            self.executable_id = self.packageUid() + "." + executable_name

    def disconnect(self):
        try:
            self.executable_manager.serviceStopped.disconnect(self.listener_id)
        except Exception as e:
            pass

    def onUnload(self):
        self.executable_manager.stopService(self.executable_id)
        self.disconnect()

    def onInput_onStart(self):
        self.listener_id = self.executable_manager.serviceStopped.connect(self.onExecutableStopped)
        if not self.executable_manager.startService(self.executable_id):
            self.logger.info("Failed to start App Executable '%s', stopping." % repr(self.executable_id))
            self.onStopped()
            self.disconnect()

    def onExecutableStopped(self, stopped_executable, reason):
        if stopped_executable == self.executable_id:
```

```
34.             self.logger.info("App Executable Stopped: " + self.executable_id)
35.             self.onStopped()
36.             self.disconnect()
37.
38.     def onInput_onStop(self):
39.         self.onUnload()
40.         self.onStopped()
```

The programs main.py and robotvideo.pml are actually connected. robotvideo.pml is a Choregraphe program which is packaged and installed inside the Pepper robot. robotvideo.pml is an executable file created for the main.py program. The reason this is done this way is because when the program is run normally, the robot preempts the video feed display program on the tablet because it assumes the program is stuck and needs to be preempted in case other users want to use it. To bypass this the focus of the system must be shifted towards this program. To shift the focus a function called switchFocus() must be called. This function is available only as a Choregraphe function and can be used only when the application is installed in the robot. Hence to use this function outside Choregraphe in a normal way in a python script, the two programs must be paired with the Choregraphe program being an executable file for the main.py program. A package called robotvideo.pkg is installed inside the robot as an executable.

In the main.py program, the stk library acts as a substitute for the naoqi library inside Choregraphe. The ALAutonomous is a class that looks after the focus of the robot. Using the function switchFocus of this class the focus of the system is shifted to the program. Then a subscription to the Tablet class is made through which the webpage that receives the video stream and is loaded on the tablet. Next, in a fashion similar to com_cam.py an audio pipeline is created, this time with the source as microphone data. In this pipeline the audio data from microphone is sent, which then would be received by the pipeline in com_cam.py.

2.5 ROBOT PROGRAM IF GUI IS NOT USED

rob_cam.py

```
#!/usr/bin/env python
import qi
import Image
import signal
import sys
#import argparse
import cv2
import numpy
from naoqi_driver.naoqi_node import NaoqiNode
import rospy

import gi
gi.require_version('Gst', '1.0')
from gi.repository import GObject, Gst
from gevent.wsgi import WSGIServer

i = 0
x = 10

#ip = ""
#port = 0

def main(session):

        video_service = session.service('ALVideoDevice') #to access peppers camera


    GObject.threads_init() #gstreamer thread initialisation
    Gst.init(None)


#   memory = session.service('ALMemory')

    user_ip = '10.0.0.10'
    user_video_port = 10000
    user_audio_port = 10001
    user_http_port = 10002

    audio_pipeline = Gst.Pipeline('audio_pipeline')

    audio_src = Gst.ElementFactory.make('autoaudiosrc')

    audio_convert = Gst.ElementFactory.make('audioconvert')
    audio_caps = Gst.ElementFactory.make('audio/x-raw-int,channels=2,depth=16,width=16,rate=44100')
    audio_filter = Gst.ElementFactory.make('capsfilter')
    audio_filter.set_property('caps',audio_caps)
    audio_pay = Gst.ElementFactory.make('rtpL16pay')
    audio_udp = Gst.ElementFactory.make('udpsink')
    audio_udp.set_property('host',user_ip)
    audio_udp.set_property('port',int(user_audio_port))

    audio_pipeline.add(audio_src,audio_convert,audio_filter,audio_pay,audio_udp)
    audio_src.link(audio_convert)
```

```
    audio_convert.link(audio_filter)
    audio_filter.link(audio_pay)
    audio_pay.link(audio_udp)

    audio_pipeline.set_state(Gst.State.PLAYING)

    # audio = 80

    # r_audio_pipeline = Gst.Pipeline('r_audio_pipeline')

    # #defining audio pipeline attributes
    # r_audio_udpsrc = Gst.ElementFactory.make('udpsrc', None)
    # r_audio_udpsrc.set_property('port', audio)

    # r_audio_caps = Gst.caps_from_string('application/x-
rtp,media=(string)audio, clock-rate=(int)44100, width=16, height=16, encoding-
name=(string)L16, encoding-
params=(string)1, channels=(int)2, format=(string)S16LE, channel-
positions=(int)1, payload=(int)96')
    # r_audio_filter = Gst.ElementFactory.make('capsfilter', None)
    # r_audio_filter.set_property('caps',r_audio_caps)

    # r_audio_depay = Gst.ElementFactory.make('rtpL16depay', None)
    # r_audio_convert = Gst.ElementFactory.make('audioconvert', None)

    # r_audio_sink = Gst.ElementFactory.make('alsasink', None)
    # r_audio_sink.set_property('sync',False)

    # #linking the various attributes
    # r_audio_pipeline.add(r_audio_udpsrc,r_audio_filter,r_audio_depay,r_audio_conv
ert,r_audio_sink)
    # r_audio_udpsrc.link(r_audio_filter)
    # r_audio_filter.link(r_audio_depay)
    # r_audio_depay.link(r_audio_convert)
    # r_audio_convert.link(r_audio_sink)

    # r_audio_pipeline.set_state(Gst.State.PLAYING)

    for i in range(x):
          tabletService = session.service('ALTabletService')
          tabletService.loadUrl('http://' + user_ip + ':' + str(user_http_port) + '
/')
          tabletService.showWebview()

    tabletService = session.service('ALTabletService')
    tabletService.loadUrl('http://' + user_ip + ':' + str(user_http_port) + '/')
    tabletService.showWebview()

    while True:

         videoClient = video_service.subscribe("pyth12", 2, 11, 15) #subscribe to p
epper's camera service
         naoImage = video_service.getImageRemote(videoClient) #reading the image
         video_service.unsubscribe(videoClient) #unsubscribing to the camera servic
e

              #reading different attributes
              imageWidth = naoImage[0] #width
              imageHeight = naoImage[1] #height
              array = naoImage[6] #image array
              image_string = str(bytearray(array))
```

```
106.                pil_img = Image.frombytes("RGB", (imageWidth, imageHeight), image_st
    ring) #making it an image
107.                img = cv2.cvtColor(numpy.array(pil_img), cv2.COLOR_RGB2BGR) #changin
    g channel order
108.
109.                img = cv2.resize(img, (640, 480))
110.
111.                cv2.imshow('image1',img) #printing the image on pc
112.
113.                if cv2.waitKey(1) & 0xFF == ord('q'):
114.                    break
115.
116.
117.
118.        if __name__ == "__main__":
119.
120.            args = rospy.myargv(argv=sys.argv) #reading the arguments from the termi
    nal (for ip and port number)
121.            pip = args[1]
122.            pport = args[2]
123.
124.            # pip = "10.0.0.4"
125.            # pport = 9559
126.
127.            session = qi.Session()
128.            session.connect("tcp://" + pip + ":" + str(pport)) #starting a connectio
    n with Pepper
129.
130.            main(session)
```

This program is used in case one does not want use the GUI. This program uses opencv library to get images from the robot's camera. This program also opens the webpage and loads the URL similar to the previous program.

To execute using this program the following command must be run in a Terminal/Command Prompt – "roslaunch pepper_bringup tele_pepper_ros.launch nao_ip:=<robot's ip> roscore_ip:=<pilot user's ip>"

2.6 LAUNCH FILES

tele_pepper.launch

```
<launch>

  <arg name="nao_ip"            default="$(optenv NAO_IP 127.0.0.1)" />
  <arg name="nao_port"          default="$(optenv NAO_PORT 9559)" />
  <arg name="namespace"           default="pepper_robot" />

  <!-- Use CPP node by default for nao_sensors -->
  <arg name="force_python" default="true" />
  <include file="$(find pepper_description)/launch/pepper_publisher.launch" />

  <!-- nao driver works for pepper -->
  <include file="$(find naoqi_driver_py)/launch/naoqi_driver.launch">
    <arg name="nao_ip"    value="$(arg nao_ip)" />
  </include>

  <include file="$(find pepper_sensors_py)/launch/sonar.launch" >
    <arg name="nao_ip"    value="$(arg nao_ip)" />
  </include>
  <include file="$(find pepper_sensors_py)/launch/laser.launch" >
    <arg name="nao_ip"    value="$(arg nao_ip)" />
  </include>

  <!-- telepresence -->
  <include file="$(find telepresence_1)/launch/telepresence.launch">
    <arg name="nao_ip"    value="$(arg nao_ip)" />
    <arg name="nao_port"    value="$(arg nao_port)" />
  </include>

  <!-- launch pose manager -->
  <include file="$(find naoqi_pose)/launch/pose_manager.launch" ns="$(arg namespace
)/pose" >
    <arg name="nao_ip"            value="$(arg nao_ip)" />
  </include>

  <!-- launch perception -->
  <include file="$(find pepper_bringup)/launch/perception.launch.xml">
    <arg name="namespace"         value="$(arg namespace)" />
  </include>

</launch>
```

tele_pepper_ros.launch

```
<launch>

  <arg name="nao_ip"            default="$(optenv NAO_IP 127.0.0.1)" />
  <arg name="nao_port"          default="$(optenv NAO_PORT 9559)" />
```

```
5.    <arg name="namespace"           default="pepper_robot" />
6.
7.    <!-- Use CPP node by default for nao_sensors -->
8.    <arg name="force_python" default="true" />
9.    <include file="$(find pepper_description)/launch/pepper_publisher.launch" />
10.
11.   <!-- nao driver works for pepper -->
12.   <include file="$(find naoqi_driver_py)/launch/naoqi_driver.launch">
13.     <arg name="nao_ip"    value="$(arg nao_ip)" />
14.   </include>
15.
16.   <include file="$(find pepper_sensors_py)/launch/sonar.launch" >
17.     <arg name="nao_ip"    value="$(arg nao_ip)" />
18.   </include>
19.   <include file="$(find pepper_sensors_py)/launch/laser.launch" >
20.     <arg name="nao_ip"    value="$(arg nao_ip)" />
21.   </include>
22.
23.   <!-- telepresence -->
24.   <include file="$(find telepresence_1)/launch/telepresence_ros.launch">
25.     <arg name="nao_ip"    value="$(arg nao_ip)" />
26.     <arg name="nao_port"    value="$(arg nao_port)" />
27.   </include>
28.
29.   <!-- launch pose manager -->
30.   <include file="$(find naoqi_pose)/launch/pose_manager.launch" ns="$(arg namespace
    )/pose" >
31.     <arg name="nao_ip"            value="$(arg nao_ip)" />
32.   </include>
33.
34.   <!-- launch perception -->
35.   <include file="$(find pepper_bringup)/launch/perception.launch.xml" >
36.     <arg name="namespace"         value="$(arg namespace)" />
37.   </include>
38.
39. </launch>
```

These launch files have the function to give directions to find and launch other programs. These files are used for roslaunch commands. They are also used to find values from the command line and pass it to the other programs listed in them as variables.

Here we are using these launch files to launch the previously mentioned eight programs together. The file tele_pepper.launch is used in the GUI program and the file tele_pepper_ros.launch file is used when the program is executed without the GUI. These two files should be added in the pepper_bringup launch folder.

telepresence.launch

```
<launch>
  <env name="PYTHONPATH" value="$(env PYTHONPATH)" />

  <arg name="nao_ip" default="$(optenv NAO_IP 127.0.0.1)" />
  <arg name="nao_port" default="$(optenv NAO_PORT 9559)" />

  <node pkg="telepresence_1" type="com_cam.py" name="com_cam" required="true" output="screen"/>

  <!-
- node pkg="telepresence_1" type="rob3.py" name="rob3" required="true" args="$(arg nao_ip) $(arg nao_port)" output="screen" / -->

  <node pkg="telepresence_1" type="teleop_twist_keyboard.py" name="teleop_twist_keyboard" required="true" output="screen" />
  <!--launch-prefix="xterm -e" output="screen"-->
</launch>
```

telepresence_ros.launch

```
<launch>
  <env name="PYTHONPATH" value="$(env PYTHONPATH)" />

  <arg name="nao_ip" default="$(optenv NAO_IP 127.0.0.1)" />
  <arg name="nao_port" default="$(optenv NAO_PORT 9559)" />

  <node pkg="telepresence_1" type="com_cam.py" name="com_cam" required="true" output="screen"/>

  <node pkg="telepresence_1" type="rob_cam.py" name="rob3" required="true" args="$(arg nao_ip) $(arg nao_port)" output="screen" />

  <node pkg="telepresence_1" type="teleop_twist_keyboard.py" name="teleop_twist_keyboard" required="true" output="screen" />
  <!--launch-prefix="xterm -e" output="screen"-->
</launch>
```

These are the launch files called in tele_pepper.launch and tele_pepper_ros.launch respectively. They further contain the directions to the two or three main programs used in this project.

2.7 AUDIO FROM COMPUTER TO PEPPER ROBOT

The following are the commented code from com_cam.py and main.py/rob_cam.py.

In com_cam.py

```
ip = "10.0.0.4"
audio_port = 80

s_audio_pipeline = Gst.Pipeline('s_audio_pipeline')

s_audio_src = Gst.ElementFactory.make('autoaudiosrc')

s_audio_convert = Gst.ElementFactory.make('audioconvert')
s_audio_caps = Gst.ElementFactory.make('audio/x-raw-int,channels=2,depth=16,width=16,rate=44100')
s_audio_filter = Gst.ElementFactory.make('capsfilter')
s_audio_filter.set_property('caps',audio_caps)
s_audio_pay = Gst.ElementFactory.make('rtpL16pay')
s_audio_udp = Gst.ElementFactory.make('udpsink')
s_audio_udp.set_property('host',ip)
s_audio_udp.set_property('port',int(audio_port))

s_audio_pipeline.add(s_audio_src,s_audio_convert,s_audio_filter,s_audio_pay,s_audio_udp)
s_audio_src.link(s_audio_convert)
s_audio_convert.link(s_audio_filter)
s_audio_filter.link(s_audio_pay)
s_audio_pay.link(s_audio_udp)
```

In main.py/rob_cam.py

```
audio = 80

r_audio_pipeline = Gst.Pipeline('r_audio_pipeline')

#defining audio pipeline attributes
r_audio_udpsrc = Gst.ElementFactory.make('udpsrc', None)
r_audio_udpsrc.set_property('port', audio)

r_audio_caps = Gst.caps_from_string('application/x-rtp,media=(string)audio, clock-rate=(int)44100, width=16, height=16, encoding-name=(string)L16, encoding-params=(string)1, channels=(int)2, format=(string)S16LE, channel-positions=(int)1, payload=(int)96')
r_audio_filter = Gst.ElementFactory.make('capsfilter', None)
r_audio_filter.set_property('caps',r_audio_caps)

r_audio_depay = Gst.ElementFactory.make('rtpL16depay', None)
r_audio_convert = Gst.ElementFactory.make('audioconvert', None)

r_audio_sink = Gst.ElementFactory.make('alsasink', None)
r_audio_sink.set_property('sync',False)

```

```
#linking the various attributes
r_audio_pipeline.add(r_audio_udpsrc,r_audio_filter,r_audio_depay,r_audio_convert,r_audio_sink)
r_audio_udpsrc.link(r_audio_filter)
r_audio_filter.link(r_audio_depay)
r_audio_depay.link(r_audio_convert)
r_audio_convert.link(r_audio_sink)

r_audio_pipeline.set_state(Gst.State.PLAYING)
```

These codes are not used because the audio stream from the pilot user's microphone to Pepper robot's tablet's speakers needs an appropriate port to transmit the data. After trying to use many different ports, out of which the usage of some ports resulted in the program crashing, it was determined that it would not work that way. Different libraries like PyAudio, ffmpeg, etc., have also been tried but they don't seem useful in two-way live audio streaming.

CHAPTER 3

3.1 EXECUTION

All the programs have been executed and is running successfully.

https://www.dropbox.com/s/b7vfcyo7z485z4r/pepper_exec.mp4?dl=0

Fig 3.14: GUI Final

Fig 3.2: Execution On Pepper

CHAPTER 4

4.1 CONCLUSION

All the programs have been integrated into a GUI and has been executed successfully except the audio stream from pilot user's computer to the Pepper robot.

REFERENCES

1. Kristoffersson, Annica. *Measuring the Quality of Interaction in Mobile Robotic Telepresence Systems Using Presence, Spatial Formations and Sociometry*. Örebro: Örebro Universitet, 2013.
2. "SoftBank Robotics Documentation What's New in NAOqi 2.5?" Your First Steps in Choregraphe - Aldebaran 2.4.3.28-r2 Documentation. Accessed February 1, 2018. http://doc.aldebaran.com/2-5/index_dev_guide.html.
3. "Wiki." Ros.org. Accessed February 1, 2018. http://wiki.ros.org/ROS.
4. "Wiki." Ros.org. Accessed February 1, 2018. http://wiki.ros.org/naoqi_driver.
5. "Wiki." Ros.org. Accessed February 1, 2018. http://wiki.ros.org/pepper_bringup.
6. "OpenCV Library." About - OpenCV Library. Accessed February 1, 2018. https://opencv.org/.
7. "How Does It Work?¶." Overview - PyGObject. Accessed February 1, 2018. https://pygobject.readthedocs.io/en/latest/.
8. "Welcome." Welcome | Flask (A Python Microframework). Accessed February 1, 2018. http://flask.pocoo.org/.
9. "Introduction — PyQt 4.12.1 Reference Guide." Accessed February 1, 2018. http://pyqt.sourceforge.net/Docs/PyQt4/introduction.html
10. "WSGIserver." PyPI. Accessed February 1, 2018. https://pypi.org/project/WSGIserver/.
11. Pepperhacking. "Pepperhacking/studiotoolkit." GitHub. October 24, 2017. Accessed February 1, 2018. https://github.com/pepperhacking/studiotoolkit/.
12. Pepperhacking. "Pepperhacking/robot-jumpstarter." GitHub. February 02, 2018. Accessed June 16, 2018. https://github.com/pepperhacking/robot-jumpstarter.