Documentation

Multisensory Navigational Aid for Visual Prosthesis Users

This document describes the hardware and software design developed by An Chi Chen for the project Multisensory Navigation Aid for Visual Prosthesis Users. The files and programmes developed for this project form supplementary to the system currently being developed by the Applied Physics Lab (APL). This document is divided into hardware and software requirements which are further categorised by feedback system – haptic, auditory and visual. This document should be read in conjunction with the final report which can be found here.

Hardware

The following hardware were provided by APL:

- Jetson Processing unit with custom 8-channel haptic driver
- Monitor and mouse
- RGB and depth cameras
- Haptic actuators (LRAs)
- Aftershokz Aeropex open-ear bone conduction headphones
- Oculus GO VR headset
- TP-Link Wireless N Nano router
- Extension cord

Haptic Feedback System

The haptic headband consists of 7 LRAs (linear resonant actuators) in total. These actuators connect into a custom 8-channel haptic driver. The haptic patterns implemented in the code developed for this project are designed assuming the following actuator indexing / connection to the haptic driver – with the numbers representing which pin they are connected to.

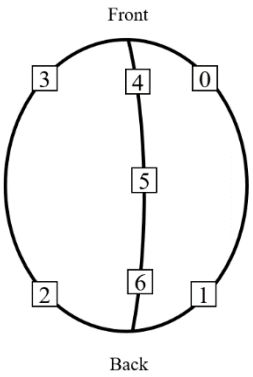

Figure 1: Indexing of Haptic Actuators on Headband

Auditory Feedback System

The open-ear bone conduction headphones used in this project were the Aftershokz Aeropex. These are connected to the processing unit via Bluetooth. This is the only hardware required for the auditory system.

Visual Feedback System

The virtual reality (VR) headset used for the visual feedback system is the Oculus GO. A router is required to connect the VR headset to the processing unit via Wi-Fi. The router used in this project is the TP-Link Wireless N Nano Router.

Software

System Requirements

The following is required to run the developed programmes:

- Linux
- Python3.6

The programmes developed for this project can be found in the Bitbucket repository for the associated APL project. It is under the branch named ‘anchi.’

The following section lists the developed programme files as well as describes the functionality of each function.

Files

File: audio_demo_node1.py
Description: Node that demonstrates how audio feedback works

File: audio_feedback1.py
Description: Python file that contains all functions in audio_demo_node1.py to be used

File: haptic_demo_node1.py
Description: Node that demonstrates how haptic feedback works

File: haptic_feedback1.py
Description: Python file that contains all functions in haptic_feedback_node1.py to be used for integration

File: vr_feedback1.py
Description: Python file that contains all functions needed to display visual feedback based off of camera input

File: landmark_driver_node1.py
Description: Node that demonstrates the integration of the haptic and audio feedback in one system

File: audio_demo1.launch
Description: Used to run audio_demo_node1.py

File: haptic_demo1.launch
Description: Used to run haptic_demo_node1.py

File: landmark_demo1.launch
Description: Used to run landmark_driver_node1.py

Haptic Feedback System

Required Libraries

No other additional libraries were required

Setting Haptic Patterns

The haptic patterns can be created after a HapticPattern() has been initialised:

```
pattern = HapticPattern()
```

The haptic pattern can be set using the addActuation function:

```
Pattern.addActuation(0, 0, 0.015, 127)
    First input = haptic actuator index (refer to Figure 1)
    Second input = start time
    Third input = stop time
    Fourth input = intensity (0 – minimum, 127 – maximum)
```

Functions

Function: **get_body_pos**
Description: Function that determines orientation and position of body (camera) with respect to world frame.
Inputs:
    msg: message from SLAM which contains body pose with respect to world coordinates
Outputs:
    dir_bodylook = vector describing the direction in which the user is looking
    t_map2body = vector from world coordinate frame to body coordinate frame

Function: **get_landmark_pos**
Description: Function that determines orientation and position of landmark found from SLAM mapping with respect to world frame.
Inputs:
    landmark = pose of landmark with respect to world coordinates
Outputs:
    t_map2landmark = vector from world coordinate frame to landmark

Function: **get_haptic_dir**
Description: Function that uses body position and position of landmark to determine which direction target is with respect to body.
Inputs:
    dir_bodylook = vector describing the direction in which the user is looking
    t_map2body = vector from world coordinate frame to body coordinate frame
    t_map2landmark = vector from world coordinate frame to landmark
Outputs:
    x_angle = horizontal angle between user look direction and landmark
    z_angle = vertical angle between user look direction and landmark

Function: **play_haptic**
Description: Plays the required haptic pattern.
Inputs:
    x_angle = horizontal angle between user look direction and landmark
    z_angle = vertical angle between user look direction and landmark
Outputs: Will play haptic pattern on haptic headband

Auditory Feedback System

Required Libraries

- scipy
  - pip install scipy
- sounddevice
  - pip install sounddevice
- soundfile
  - pip install soundfile
- sofa
  - pip install python-sofa

Loading Required Files

Currently, the sound and hrtf files are saved two directories up from the python files in files titled /Sounds and /HRTFs respectively. If this location is changed the source_dir and sofa_dir need to be changed accordingly.

Sound file(s):

- Loading in files
  - source_dir = '…../…/*.wav' # could be other formats if not .wav files
  - Source = glob.glob(source_dir)
  - source.sort() # to sort files in alphabetical order
- Read file
  - src, freq = sf.read(source[x]) # where x is index of desired file, freq not used here
  - # if sound file is stereo, need to make it mono by taking mean of both channels
    src = np.mean(src, axis=1)

HRTF sofa file

- Loading in file:
  - sofa_dir = '…../…/*.sofa'
  - sofa = glob.glob(sofa_dir)
  - sofa.sort() # to sort files in alphabetical order
- Open file
  - hrtf = sofa.Database.open(sofa[x]) # where x is index of desired file
  - positions = hrtf.Source.Position.get_values() # gets matrix of all azimuth and elevation, azimuths in first column and elevation in second column (may vary depending on file source)

Functions

Function: **get_body_pos**
Description: Function that determines orientation and position of body (camera) with respect to world frame.
Inputs:
msg: message from SLAM which contains body pose with respect to world coordinates
Outputs:
dir_bodylook = vector describing the direction in which the user is looking
t_map2body = vector from world coordinate frame to body coordinate frame

Function: **get_landmark_pos**

Description: Function that determines orientation and position of landmark found from SLAM mapping with respect to world frame.
Inputs:
landmark = pose of landmark with respect to world coordinates
Outputs:
landmarkinfo = array with first element being the ID of the landmark and second element being the same as t_map2landmark

Function: **get_sound_dir**
Description: Function that uses body position and position of landmark to determine from which direction sound should be played.
Inputs:
dir_bodylook = vector describing the direction in which the user is looking
t_map2body = vector from world coordinate frame to body coordinate frame
t_map2landmark = vector from world coordinate frame to landmark
Outputs:
x_angle = horizontal angle between user look direction and landmark
z_angle = vertical angle between user look direction and landmark
dist = distance between user and landmark in meters
src = location of sound source file to play (dependent on landmark ID)

Function: **play_sound**
Description: Plays the sound file with the required direction.
Inputs:
x_angle = horizontal angle between user look direction and landmark
z_angle = vertical angle between user look direction and landmark
Outputs: Will play spatialised sound

More Details:

- Find hrtf with position with closest correspondence to landmark position
  - find_nearest(array, val) # array = array to find in, val = target value
    # will return closest value to val in array and index
  - [azimuth, az_index] = find_nearest(positions[:,0], az_val)
  - azPos = positions[np.where(positions[:,0]==azimuth)] # because find_nearest only returns one value and we need all hrtfs with azimuth value of azimuth
  - [elevation, el_index] = find_nearest(azPos[:,1], el_val) # gets hrtf that closest corresponds to desired elevation
- Create left and right sounds to play
  - # Get left and right hrtf -> HRIR
    HRIR[:,0] = hrtf.Data.IR.get_values(indicies={"M":az_index+el_index, "R":0, "E":0})
    HRIR[:,1] = hrtf.Data.IR.get_values(indicies={"M":az_index+el_index, "R":1, "E":0})
  - sound_left = signal.fftconvolve(src, HRIR[:,0]) # left sound
  - sound_right = signal.fftconvolve(src, HRIR[:,1]) # right sound
  - mix = np.vstack([s_r, s_L]).transpose() # combine left and right sound
  - sd.play(mix, 44100) # play sound, original sound and HRTF frequency are 44.1kHz
  - status = sd.wait() # ensures sound is finished playing

Visual Feedback System

Required Libraries

No other additional libraries were required

Functions

Function: **get_body_pos**
Description: Function that determines orientation and position of body (camera) with respect to world frame.
Inputs:
msg: message from SLAM which contains body pose with respect to world coordinates
Outputs:
dir_bodylook = vector describing the direction in which the user is looking
t_map2body = vector from world coordinate frame to body coordinate frame

Function: **get_landmark_pos**
Description: Function that determines orientation and position of landmark found from SLAM mapping with respect to world frame.
Inputs:
landmark = pose of landmark with respect to world coordinates
Outputs:
landmarkinfo = array with first element being the ID of the landmark and second element being the same as t_map2landmark

Function: **get_landmark_dir**
Description: Function that uses body position and position of landmark to determine where pixel should be placed on VR display to represent position of landmark in FOV.
Inputs:
dir_bodylook = vector describing the direction in which the user is looking
t_map2body = vector from world coordinate frame to body coordinate frame
t_map2landmark = vector from world coordinate frame to landmark
Outputs:
x_angle = horizontal angle between user look direction and landmark
z_angle = vertical angle between user look direction and landmark

Function: **get_vr_img**
Description: Adds pixel location relating to landmarks visible in FOV to array
Inputs:
x_angle = horizontal angle between user look direction and landmark
z_angle = vertical angle between user look direction and landmark
y = array containing y coordinates of pixels to display
x = array containing x coordinates of pixels to display
Outputs: No outputs

Function: **play_vr**
Description: Displays VR image onto VR headset display
Inputs:
y_img = array containing y coordinates of pixels to display
x_img = array containing x coordinates of pixels to display
Outputs: No outputs