Development of Robot-enhanced Therapy for Children with Autism Spectrum Disorders

Project No. 611391

DREAM

Development of Robot-enhanced Therapy for Children with Autism Spectrum Disorders

Agreement Type: Collaborative Project
Agreement Number: 611391

D4.2 Evaluation of Multi-Sensory Data Perception

Due Date: **01/04/2016**
Submission date: **01/04/2016**

Start date of project: **01/04/2014** Duration: **54 months**

Organisation name of lead contractor for this deliverable: **University of Portsmouth**

Responsible Person: **Honghai Liu** Revision: **2.0**

Project co-funded by the European Commission within the Seventh Framework Programme		
Dissemination Level		
PU	Public	**PU**
PP	Restricted to other programme participants (including the Commission Service)	
RE	Restricted to a group specified by the consortium (including the Commission Service)	
CO	Confidential, only for members of the consortium (including the Commission Service)	

Contents

Executive Summary

Objectives

Deliverable D4.2 aims to evaluate multi-sensory data perception. Its main objectives are:

- To document algorithm specification, design, implementation, and validation of a suite of multi-modal sensory data acquisition modules derived from the sensory requirements set out in "Deliverable D1.1 Interaction Definition" (with particular regard to the sensory cues that characterize the action triggers, action components, and action goal states.)
- To deliver the results from task T4.2 and provide inputs for tasks T3.3, T4.3, T4.4, T6.1, and T6.2.

Implementation

The above objectives in D4.2 have been fulfilled and the involved tasks are summarized as follows:

We have proposed methods/algorithms for multi-modal sensory data acquisition. For visual data acquisition, a gaze estimation method is proposed to obtain ASD children's head poses as well as their gaze directions; a joint detection method is introduced to get ASD children's upper body skeletons and a joint-based method is presented to track their hands in real time; a simple blob detection method is employed for detecting the objects on the table and an efficient Gaussian mixture probability hypothesis density tracker is employed for object tracking; a face and expression recognition method is proposed to recognize children's faces and expressions. For audio data acquisition, the Microsoft SDK is utilized to recognize the speech and determine the sound direction. We have experimentally evaluated the feasibility and effectiveness of the proposed methods/algorithms.

- We have completed the YARP framework to implement the 25 perception primitives defined in "Deliverable D1.3 Child Behaviour Specification" and "Deliverable D3.1 System Architecture". We have developed three additional components to make the system more flexible and efficient.
- We also proposed methods for both the camera selection module and the coordinate transformation module, which are the foundation for multi-sensory data fusion.

Principal Contributors

The main authors of this deliverable are as follows (in alphabetical order)

Haibin Cai, University of Portsmouth
Yinfeng Fang, University of Portsmouth
Dongxu Gao, University of Portsmouth
Zhaojie Ju, University of Portsmouth
Honghai Liu, University of Portsmouth
Ting Wang, University of Portsmouth
Yiming Wang, University of Portsmouth
Hui Yu, University of Portsmouth
Shu Zhang, University of Portsmouth
Xiaolong Zhou, University of Portsmouth

Revision History

Version 2.0 (Zhou, X., Fang, Y., Cai, H., Gao, D., Wang, Y., Ju, Z., Yu, H., Liu, H., 31-03-2016)

1. Introduction

This deliverable, D4.2, describes the process and analyse of the sensory data, including images and sound, obtained from the multi-camera system designed. The processed data is further taken as the input for YARP implementation and multi-data fusion.

First, an overview of the multi-sensory data perception framework is introduced. Five individual sensors that include three RGB cameras and two Kinects are used for data sensing. Twenty-five interface functions and twenty-nine related variables are defined for sensory data organization and perception.

Then, data perception methods corresponding to the 25 functions described in D1.3 and D3.1 are presented. Specifically,

- A gaze estimation method is proposed to determine where a child with Autism Spectrum Disorders (ASD) is actually looking at.
- A method is employed to estimate ten skeleton joints of the child's upper body. The estimated joints can then be used to track the child's hands, which will be further needed for action recognition.
- A blob detection method is utilized for object detection and the Gaussian Mixture Probability Hypothesis Density (GM-PHD) tracker is incorporated to track the objects on the table.
- The Local Binary Patterns (LBP) is used to represent facial appearance cues and the SVM is applied for identity facial expression classification.
- Based on the Microsoft Kinect SDK, the speech in English as well as the sound direction can be recognized.
- In addition, experimental evaluation of the proposed data perception methods is also presented.

Finally, a YARP implementation of the sensory data perception is presented. A preliminary of multi-sensory data fusion that includes camera selection and coordinate transformation is also presented as a foundation of the next deliverable D4.3.

2. Multi-Sensory Data Perception Framework

In this chapter, an overview of multi-sensory data perception framework is presented. Three RGB cameras and two Kinects are used for data sensing according to D4.1. Twenty-five interface functions and twenty-nine related variables are defined for sensory data organization and perception. The captured multi-sensory data will be employed as an input for tasks T3.3, T4.3, T4.4, T6.1, and T6.2.

Figure 2.1 shows a framework of multi-sensory data capturing and processing. As documented in D4.1, this project employs five individual sensors: Camera0, Camera1, Camera2, Kinect0 and Kinect1. Camera0 and Kinect0 are located in the middle of the designed platform, which are frontally facing to an ASD child. Camera1 and Camera2 are on the left and right of the platform respectively, and Kinect1 is equipped on the top of the platform.

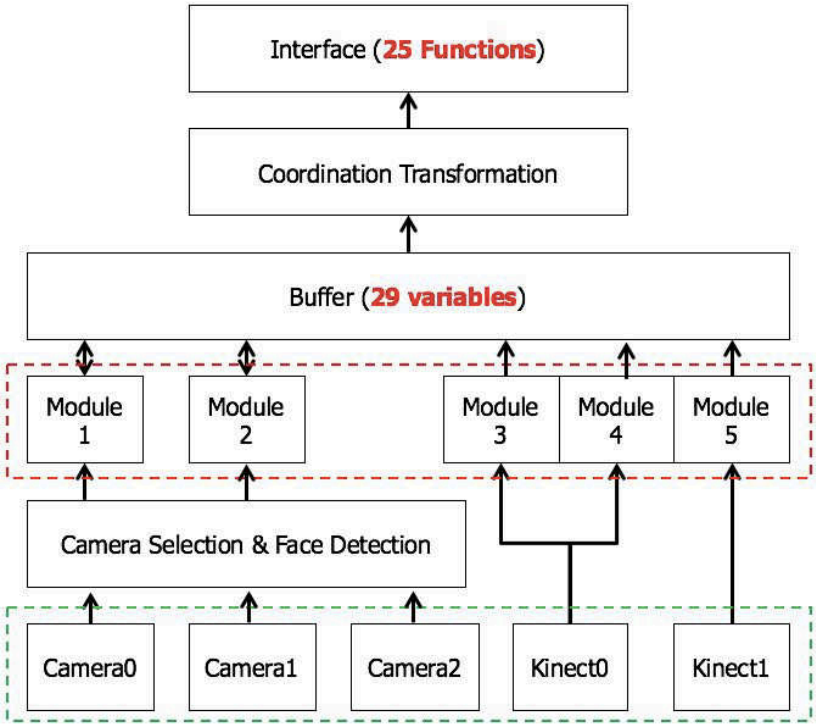


Figure 2.1: A framework of multi-sensory data capturing and processing.

Camera0, Camera1 and Camera2 form a functional unit to get the face location, eye locations, gaze direction, head direction, etc.

- The Camera Selection Module (CSM) captures image frames from three cameras and selects the best camera with the highest face detection probability, and meanwhile the CSM module also functions to obtain facial feature points from the selected frame.
- The selected camera ID, the original frame and the calculated feature points will be simultaneously saved in a global buffer and be updated according to the speed (fps).

- Module 1 and module 2 serve to implement the primary functions, like calculating face/eye locations, head/gaze directions, face IDs, facial expression IDs and etc. These separate modules will run through several main algorithms, which are proposed in this deliverable. It should be noted that Module 1 and Module 2 could communicate with each other by sharing data in the global buffer.
- The function of Kinect0 is two-fold: voice analysis (Module 3) and subject's skeleton joints extraction (Module 4). Module 3 can be further separated into two parts: speech recognition and sound direction tracking. Kinect1 focuses on object tracking, and the objective is to get the object (toys) locations, object IDs and robot's head location.
- An example definition of the global buffer with 29 variables is shown as below.

global_buffer

{

V1: *cv::VideoCapture cap;*
V2: *CameradeviceID*
V3: *Mat ::image*
V4: *Eyes(rightx, righty, rightz, left x,left y, left z)*
V5: *Head pose(roll, yaw, pitch)*
V6: *Face(vector(x,y,z))*
V7: *Gaze(roll, yaw, pitch)*
V8: *Coordinate transform Mat(R,T)---camera1,camera2,camera3,Kinect2*
V9: *Frame 3D points(x,y,z)*
V10: *Object position(x,y,z)*
V11: *Head position(x,y,z)*
V12: *Hand position(x,y,z)*
V13: *cv::Mat X; //face 49 feature points*
V14: *int numberkernel; //number of kernel used for eye centre detection*
V15: *vector<cv::Mat> kernel_filter; //kernel used for eye centre detection*
V16: *Robot head position*
V17: *Sound Direction*
V18: *Face source*
V19: *Desk_point vector // 3 points*
V20: *Skeleton joint vector*
V21: *Object_location*
V22: *Object_id*
V23: *Face_id*
V24: *Face_expression_id*
V25: *Object_id*
V26: *Object_history_location vector*
V27: *Voice_descriptor_id*
V28: *Voice_text_id*
V29: *Skeleton_history_joint vector*

}

- The coordinate transformation component is employed to determine the position and orientation of each camera and transform all the local sensory data to a global coordinate system.

While running the program, all the variables would be updated by the modules and CSM in real-time. With these 29 variables, the predefined 25 interface functions can be easily implemented through accessing the variables. Table 2.1 lists relationships between these 25 interface functions (described in D3.1.3) and the defined variables.

Table 2.1: Functions and global variables.

25 Interface functions	Related variables
F1: checkMutualGaze	V7,V8,V17
F2: getArmAngle	V20
F3: getBody	V20
F4: getBodyPose	V20
F5: getEyeGaze	V4,V7,V8,V13
F6: getEyes	V3,V4,V6,V8,V13,V14,V15
F7: getFaces	V3,V6,V8
F8: getGripLocation	V22,V21,V22
F9: getHands	V20
F10: getHead	V6,V8,V13
F11: getHeadGaze1	V6,V8,V11,V13,V19
F12: getHeadGaze2	V6,V8,V13
F13: getObjects1	V21,V22
F14: getObjects2	V19,V21,V22
F15: getObjectTableDistance	V21,V22
F16: getSoundDirection	V8,V17
F17: indntifyFace	V11,V23
F18: IdentifyFaceExpression	V11,V23,V24
F19: identifyObject	V21,V25
F20: identifyTrajectory	V25,V26
F21: identifyVoice	V27
F22: recognizeSpeech	V28
F23: trackFace	V6,V8,V18
F24: trackHand	V29
F25: trackObject	V21,V22

3. Multi-Sensory Data Perception Methods

In this chapter, specific methods for multi-sensory data perception are proposed. These methods, including gaze estimation, upper body skeleton joints detection and joint-based hand tracking, object detection and tracking, face and facial expression recognition, and speech and sound direction recognition, are investigated and employed to realise the 25 functions introduced in Chapter 2.

3.1. Gaze Estimation

3.1.1. Method

Face Location Method

The boosted cascade face detector [1] is employed with default parameters in order to obtain the approximate location of the face. This method corresponds to the getFaces(x,y,z) function.

Eye Location Method

This method corresponds to the getEyes(eyeLx, eyeLy, eyeLz, eyeRx, eyeRy, eyeRz) function. We present an improved integro-differential solution to localize the eye centres. The proposed method is computationally much cheaper than the original integro-differential method [2] and it also achieves a higher accuracy in lower-resolution images.

The original integro-differential method is one of the most popular eye localization methods in the literature. The integro-differential operator (IDO) is defined as follows.

$$\max_{(r,x_0,y_0)} \left| G_\sigma(r) * \frac{\partial}{\partial r} \oint_{r,x_0,y_0} \frac{I(x,y)}{2\sigma r} ds \right| \tag{3-1}$$

where $G_\sigma(\mathrm{r})$ is a smoothing Gaussian with a scale of σ and $*$ represents convolution, $\mathrm{I}(\mathrm{x},\mathrm{y})$ is the image of an eye and ds is the contour of the r radius circle with the centre point of (x_0, y_0). The operator searches for the maximum along the circle path in the blurred image via the Gaussian kernel, partial derivative with respect to increasing radius r. In order to deal with the obscure of upper and lower limbus by the eyelids, the angular arc of contour integration s is restricted in range to two opposing 90 cones centred on the horizon.

In discrete implementation of the IDO, the order of convolution and differentiation is interchanged and concatenated to improve the speed. After replacing the convolution and contour integrals with sums, the equation is derived as follows.

$$\max_{(\mathrm{n}\Delta\mathrm{r},x_0,y_0)} = \left| \frac{1}{\Delta \mathrm{r}} \sum_k \left\{ \left(G_\sigma\big((n-k)\Delta r\big) - G_\sigma\big((n-\mathrm{k}-1)\Delta r\big) \right) \sum_m I[(k\Delta r \cos(m\Delta\theta) + x_0), (k\Delta r \sin(m\Delta\theta) + y_0)] \right\} \right| \tag{3-2}$$

where Δr means a small increment in radius, $\Delta\theta$ is the angular sampling interval along the circular arcs.

As can be seen from the Equation (3-2), an angular sampling interval Δθ is used to find points along the circular arc ds. However, this makes it hard to choose an appropriate value of $\Delta\theta$. If $\Delta\theta$ is too small the computational cost would be very high, on the other side the accuracy would decrease. Besides, the original IDO only uses the optimization of the circle curve integral of the gradient magnitudes, the centre point of pupil, which is also an important information, is not taken into account. The proposed method utilizes all the pixels along the circular by convoluting different sizes of circle kernels in the eye region image. The grayscale of the eye centre is also considered by designing the kernel with a weight in the centre point. The size of the kernel is $2r + 1$ where r stands for the radius of circle. The pixels along the circular are assigned a normalized value. In order to cope with the obscure of eye lids, the upper and lower part is not assigned.

Instead of using a differential method at the integral of circle intensity, this project calculates a ratio derivative between neighbour curve magnitudes, which is formulated as follows.

$$\begin{cases} I_r = K_r * I(x,y) \\ I'_{r+1} = K'_{r+1} * I(x,y) \\ D_r = \frac{I'_{r+1}}{I_r} \\ argmax_{(r,x,y)}(D_r) \\ r\epsilon[r_{min}, r_{max}] \end{cases} \tag{3-3}$$

where K_r is the kernel with a centre weight and r stands for the radius of the circle inside the kernel.

The kernel without a centre weight is represented as K'_{r+1} whose radius is $r + 1$. I_r and I'_{r+1} are the results of convolution of the different kernels with eye image $I(x, y)$. D_r means the ratio derivative calculated by the division of the convolution result image. r_{min} and r_{max}, which are set according to the size of eye image, represent the minimum and maximum of the radius r. The weights of the points around the circular arcs are of equal value and normalized to 1, and the weight of the centre point is settled to a valid value. In order to locate the eye centre and radius, the proposed method searches the maximum of different radius of D_r and the smoothing function in the original IDO is not employed. By using FFT in the realization of convolution, the computation complexity can be reduced.

Gaze Estimation Method

We propose a real-time gaze estimation method by constructing multi-sensor fusion system to handle the large head movement. Three cameras and two Kinects are used in this system. In

the gaze estimation task, the cameras are used to capture the face of the child. The Kinect0 is used to capture the head position in the world coordinate and the Kinect1 is used to capture the positions of the robot head and objects. All the image data are captured simultaneously by creating 8 handles in programming. Each handle deals with different kinds of data. The data captured in these handles contains two Kinect RGB image data, two Kinect depth data, three camera data and one Kinect audio data. The resolution of the camera, Kinect RGB image, Kinect depth image are 1280*960, 640*480 and 640*480, respectively.

To estimate the gaze direction, the facial features should be located at first. We employ the method proposed by Xiong et al. [3] to locate the feature points in the human face. In order to deal with head movements, the head poses need to be determined. We employ the object pose estimation method (POSIT) proposed by Dementhon et al. [4] to calculate the direction of head pose (corresponding to the getHead(headx, heady, headz) function). Then the eye centre is located by applying the proposed convolution based intergo-differential eye centre localization method. It should be noted that the gaze direction differs from the head pose by two angles, the horizontal direction θ and the vertical direction φ. The gaze direction is finally determined by adding the angles to the head pose. The following is the equation to calculate the gaze direction (corresponding to the getEyeGaze(eye, x, y, z) function).

$$\begin{cases} \theta = \tan^{-1}(\gamma * \sqrt{(x_p - x_c)^2 + (y_{p-}y_c)^2} * \frac{\cos\alpha}{L}) \\ \varphi = \tan^{-1}(\varepsilon * \sqrt{(x_p - x_c)^2 + (y_{p-}y_c)^2} * \frac{\cos\beta}{H}) \end{cases} \tag{3-4}$$

where (x_c, y_c) denotes the centre of eye corner, (x_p, y_p) denotes the centre of eye pupil, α is the angle between the line of two eye corners and the line of two centres. β is the complementary angle of α. L is the distance of two eye corners, γ and ε are determined through experiments.

3.1.2. Experimental Results

Eye Location Results

Rough eye regions are extracted through anthropometric relationships with the face as stated in [5] and [6]. The proposed method is validated on the naturally captured ASD children images. The main challenge is caused by a large variety of illumination conditions, backgrounds, scales and poses. Some children in the database are wearing glasses, while in some images the eyes are partly closed. Some experimental snapshots are illustrated in Figure 3.1. The located facial landmarks are marked as green points and the located eye centres are marked with a red cross. The results demonstrate that our eye location method can effectively and accurately detect and locate ASD children's eyes even in challenging cases.

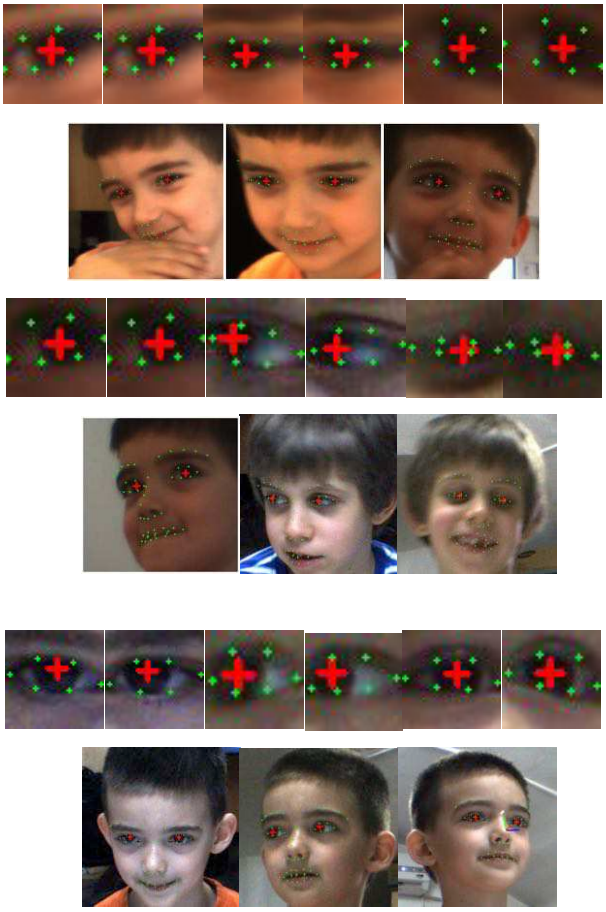


Figure 3.1: Snapshots with accurate eye centre estimation.

Gaze Estimation Results

Figures 3.2-3.5 show the gaze estimation results of our project. The Kinects are used to acquire the RGB colour images as well as the depth images; three high-resolution cameras are used to acquire high quality RGB colour images. The images we used to calculate the gaze direction are selected according to the angle of head poses. The most frontal face image is the image that has the smallest angle of yaw. This image is then chosen to estimate the gaze direction. In the Figures 3.2-3.5, green dots, red crosses and white lines represent the feature points, eye locations and gaze directions, respectively. The results demonstrate that the proposed method can successfully and correctly locate children's eyes and estimate the gaze directions with different head movements.

Figure 3.2: Gaze estimation result from the middle camera.

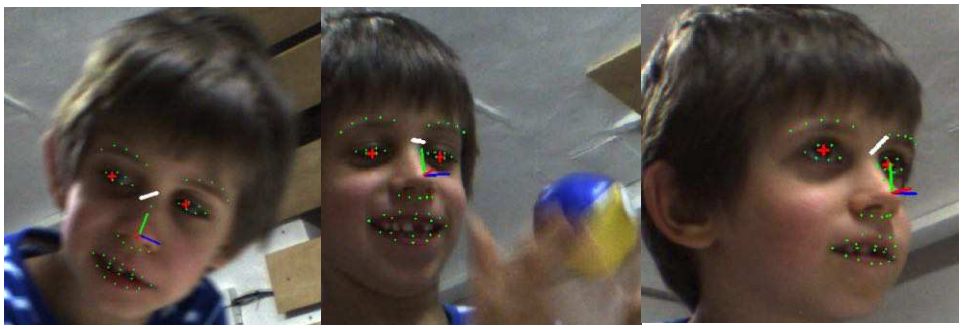


Figure 3.3: Gaze estimation result from the right camera.

Figure 3.4: Gaze estimation result from the left camera.

Figure 3.5: Gaze estimation on other ASD children.

Mutual Gaze Estimation Results

The estimated gaze direction can be further used for mutual gaze determination by incorporating the position of the robot head. In the project, the mutual gaze function is designed to judge whether the child is looking at the robot or not. As shown in Figure 3.6, if the child is facing towards the head position of the robot, then the system can automatically output a status of the child as "gaze robot head".

Figure 3.6: Mutual gaze results.

3.1.3. Related Functions

Related functions are F1, F5, F6, F7, F10, F11, F12 and F23.

Relationship of above functions and global variables are shown in Figure 3.7.

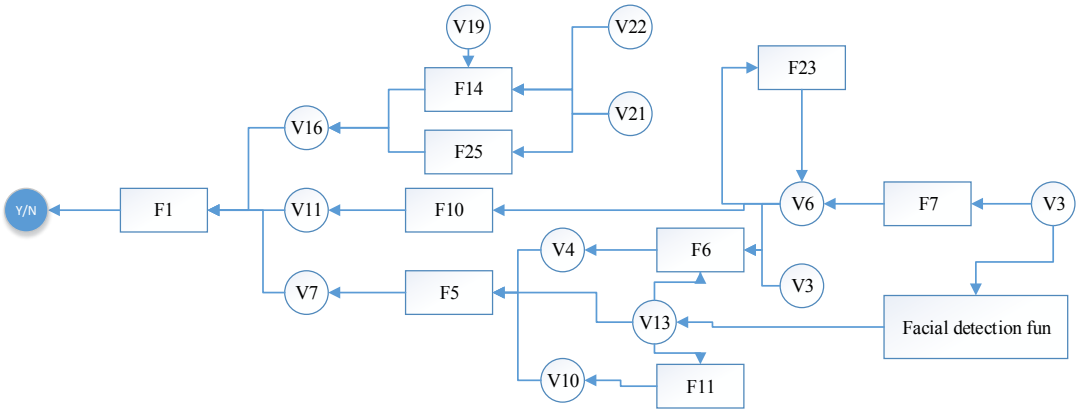


Figure 3.7: Relationship between gaze related functions and variables.

3.2. Upper Body Skeleton Detection and Joint-Based Hand Tracking

3.2.1. Method

Human upper body skeleton detection

To recognise the action of human upper body, this project will use the skeleton information acquired by Kinect0. The main idea is to represent the movement of human upper body using the pairwise relative positions of the joints feature. A depth image is employed to accurately and quickly infer 3D positions of body joints. Different body parts are labelled for classifying joints. Per-pixel information is processed and pooled to generate reliable states of skeletal joints.

The acquirement of the skeletal joints is a fundamental part for competing some functions like getting upper body poses, computing arm angles and locating the positions of the hand. This project utilises the Kinect SDK in the process of programming for acquiring the skeleton data. The skeleton data can be captured for use in real-time and/or can also be persisted for offline processing. Both real-time online and offline data processing can be useful in the detection of the skeleton and provide excellent results for activity recognition.

For a human subject, 10 joint positions are tracked by the skeleton tracker when seating in front of the sensor, and each joint i has 3 coordinates $p_i(t) = (x_i(t), y_i(t), z_i(t))$ at a frame t. The coordinates are normalized so that the motion is invariant to the initial body orientation and the body size. Using relative joint positions is actually a quite intuitive way to represent human motions. For example, it can be interpreted as "arms above the shoulder and moving left and right" for recognising the action "waving". This can be effectively characterized through the pairwise relative positions.

Human hand tracking

Based on the detected skeletal joints, we can easily track the 3D position of a hand, frame by frame. Hand tracking can assist in estimating the location of object to grasp and is a key step for tracking the trajectory of the hand. This will be used to analyse which object is grasped by the ASD child and further to help with the activity classification.

3.2.2. Experimental Results

The skeleton acquired includes 10 joints (head, neck, left and right shoulder, left and right elbow, left and right wrist, and left and right hand) as shown in Figure 3.8.

Figure 3.8: Captured skeletal joints with different actions.

From the results of captured skeletal joints, it is clear that all these ten joints can be estimated accurately with different actions. These joints can be further used to track the hand and analyse the activities. As shown in Figure 3.9, the selected skeletal joint data extracted from the recorded data can be used to indicate body movements.

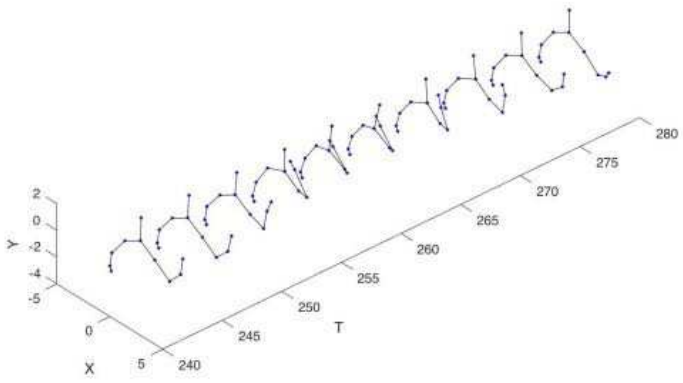


Figure 3.9: The skeletal joints detection from the recorded data.

As the upper body joints include hand joint information, frame by frame movement of the hand can be tracked. Figure 3.10 shows the trajectory of the tracked hand by only considering the position of the hand from the detected skeletal joints for different actions. The results demonstrate the good performance of tracking hand by skeleton joints.

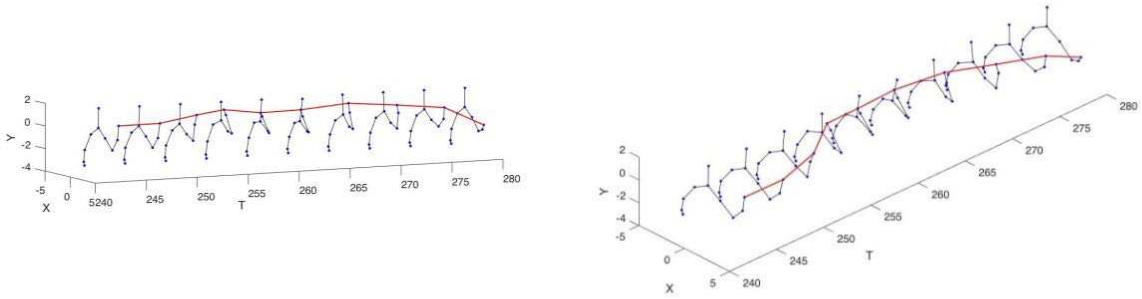


Figure 3.10: The trajectory of tracked hand based on the detected skeleton joints.

3.2.3. Related Functions

Related functions are F2, F3, F4, F8, F9 and F24.

Relationship of above functions and global variables are shown in Figure 3.11.

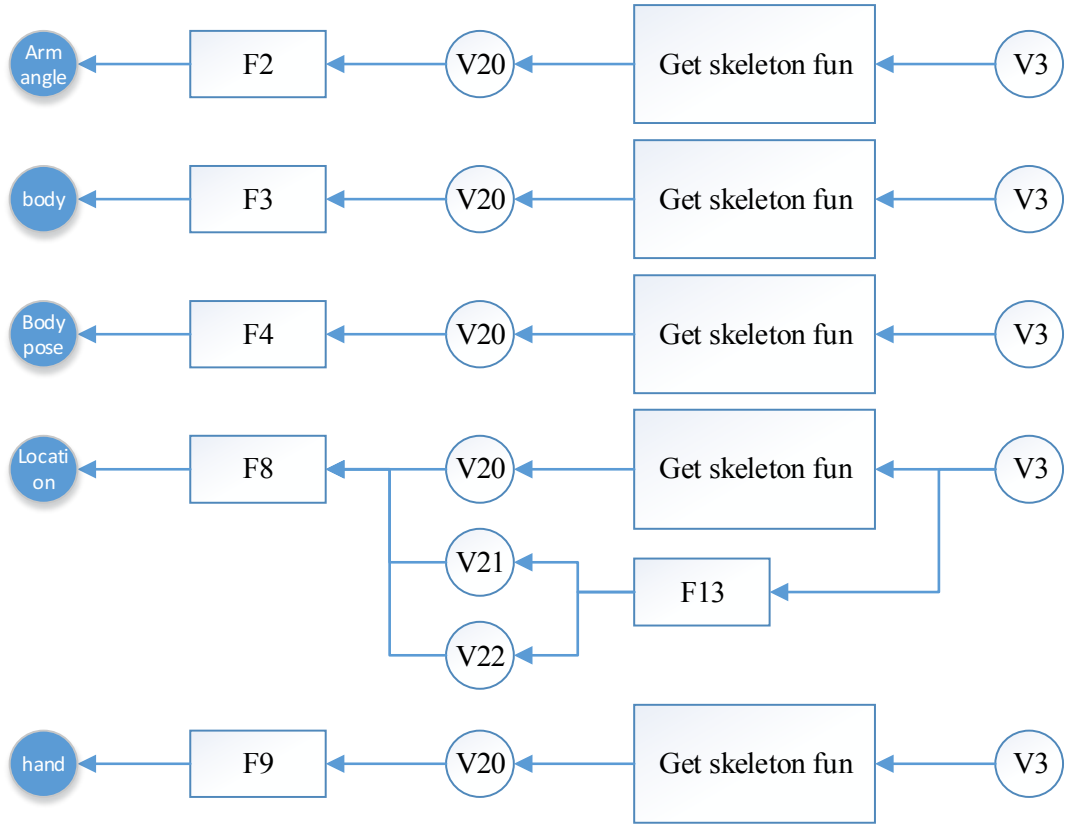


Figure 3.11: Relationship between skeleton related functions and variables.

3.3. Object Detection and Tracking

3.3.1. Method

Numerous object detection and tracking algorithms have been proposed in the literature. In this project, the objective is to detect and track the objects (toys) on the table and finally to judge whether the objects are picked up by the ASD child or not. The main challenges raised in this project are object variety, illumination and occlusion. Fortunately, only two objects are involved at one time and they are on two separate positions of the table. To effectively detect and track the objects in real time, a simple blob based Otsu object detection method [8] is employed at each frame and an efficient GM-PHD tracker [9] is used for tracking objects over time.

The main steps of object detection are summarized as follows.

- Input the video RGB image and transform it to HSV image.
- Use the global threshold method for image binarization with respect to the V-channel of the HSV image.
- Employ the blob algorithm to recognize the maximum boundary of the table, based on the fact that the table is white. Once the table area is detected, it is saved and all object detections are operated within this area.
- At each time step, transform the table area to grey scale, and then use the Otsu algorithm for adaptively image binarization.
- Within the table area, employ the blob algorithm to detect the candidate regions of the objects. The centre of each blob is regarded as the position of each object.

Object detection can find all the locations of objects on the table at each frame. To correctly associate the objects in consecutive frames, an efficient GM-PHD tracker is utilized for object tracking. The main steps of object tracking are given here.

- Input the video image and use object detection method to detect all the objects.
- Use entropy distribution-based method [10] to estimate the birth intensity of the new objects.
- Predict object states according to the state transition model. In this project a constant velocity model is used.
- Update object states according to the new detected measurements. By doing so, the same object between two consecutive frames will be associated with the highest weight.
- Output the states of the objects and the corresponding identities.

The abovementioned method is based on one 2D RGB image and it outputs 2D locations of the objects. To obtain the 3D locations of the objects, a 2D-3D correspondence according to the depth information captured by the Kinect could be incorporated.

3.3.2. Experimental Results

The proposed object detection and tracking method is tested on the recorded video data provided by the UBB. Considering the following scenario, a therapist puts two objects on the table and guides an ASD child to pick up one of them. The objects are varied and sometimes occluded by child's hands. As shown in Figure 3.12, the proposed method can successfully detect and track the objects when they appear on the table as well as when the child grasps them.

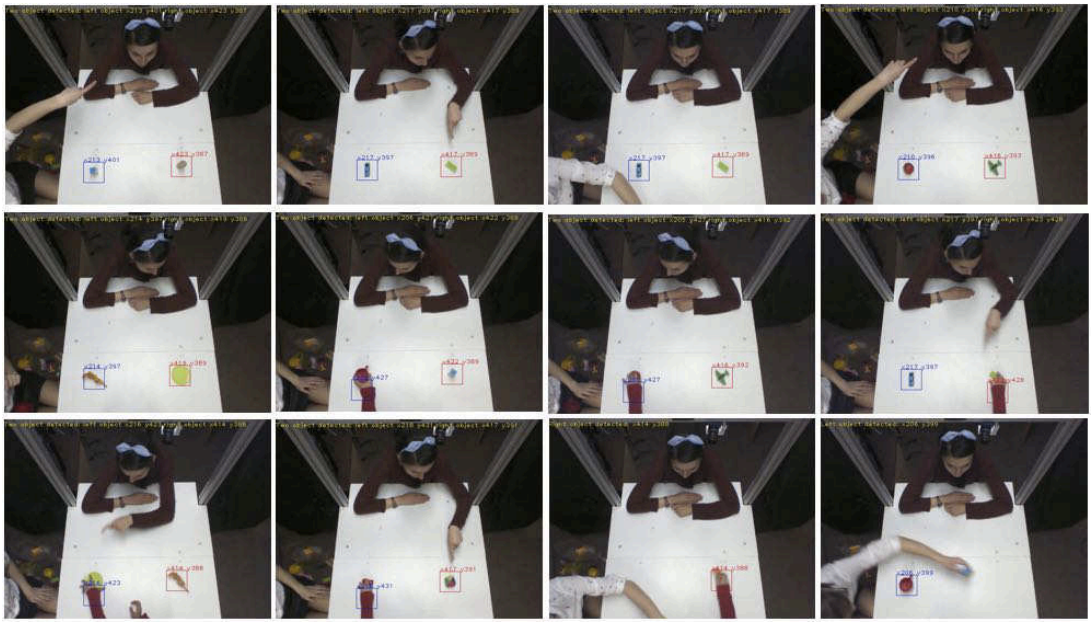


Figure 3.12: Detecting and tracking objects on the table.

3.3.3. Related Functions

Related functions are F13, F14, F15, F19, F20 and F25.

Relationship of above functions and global variables are shown Figure 3.13.

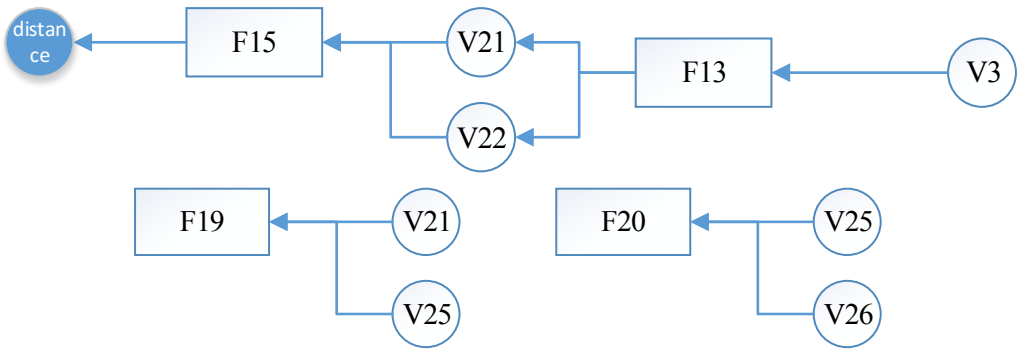


Figure 3.13: Relationship between object tracking related functions and variables.

3.4. Face and Facial Expression Recognition

3.4.1. Method

Task 4.4 provides some advices that the facial appearance cues should be captured and Support Vector Machine (SVM) is considered as a classifier. So we use Local Binary Patterns (LBP) to represent facial appearance cues and apply SVM for identity and facial expression classification.

LBP is a nonparametric method and has been proven as a powerful descriptor in representing the local textural structure [11]. The main advantages of LBP are the strong tolerance against illumination variations and the computational simplicity. This method has been successfully used in both spatial and spatio-temporal domains in face recognition and facial expression recognition.

The original LBP operator labels the pixels of an image with decimal numbers. Each pixel is compared with its eight neighbours in a 3∗3 neighbourhood, considering the centre pixel value as a threshold; bigger values are encoded with 1 and the others with 0. A binary number is obtained by concatenating all these values. Its corresponding decimal number is used to compute LBP histogram. Figure 3.14 shows an example of LBP operator.

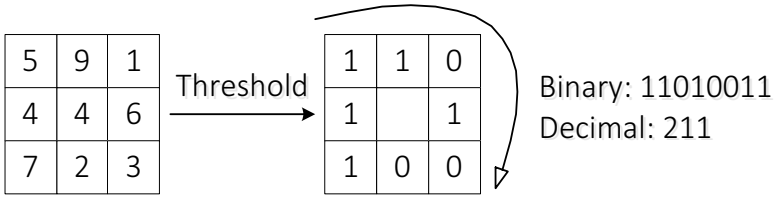


Figure 3.14: An example of LBP operator.

In order to emphasize spatial relationships of a face image, the holistic LBP histogram is extended to a spatially enhanced histogram by using block-based LBP strategy. The detected face image is divided into 8-by-8 blocks and the LBP feature is extracted in each block. All the LBP histograms are concatenated into a single histogram. The resulting spatially enhanced LBP descriptor will be the input of SVM.

SVM is considered as one of the most powerful machine learning techniques for data classification. It achieves a good balance between structural complexity and generalization error. It offers a great performance under the circumstance of very few training samples, high dimensionality and nonlinear classification.

In a two-class learning task, SVM finds a maximal margin hyperplane as its decision boundary. For a linear separable dataset, SVM assumes that the best classification results are obtained by maximizing the margin of hyperplane between two classes. It allows not only the best partition on the training data, but also leaves much room for the correct classification of the future data. In order to guarantee the maximum margin hyperplanes to be actually found, an SVM classifier attempts to maximize the following function with respect to $\vec{w}$ and b:

$$L_P = \frac{1}{2}\|\vec{w}\| - \sum_{i=1}^{t} \alpha_i y_i (\vec{w} \cdot \vec{x_i} + b) + \sum_{i=1}^{t} \alpha_i \quad (3\text{-}5)$$

where t is the number of training examples, α_i are the Lagrange multipliers. The vector $\vec{w}$ and constant b define the hyperplane.

3.4.2. Experimental Results

The face database is created by manually extracting a number of frames from videos, which yields 204 images of six children. Basically, the face images are laid in frontal or near-frontal view. There is no large-scale occlusion or head pose in all the images, whilst illuminations vary.

For the facial expression database, the raw data is also manually extracted from videos. Meanwhile, the NIMH Child Emotional Faces Picture Set (NIMH-ChEFS) is used to complement the facial expression database. The resulting database includes 437 images of five emotional categories (83 Angry, 78 Fear, 111 Happy, 73 Neutral and 92 Sad).

We evaluate our method using 10-fold cross-validation. For face recognition, the preliminary research is based on the identity classification of six children and experimental results show that this method can successfully identify them and the recognition rate is around 97%. Considering that the face database is in a small scale, the accuracy may reduce when applying this method to the real-world face recognition.

For facial expression recognition, the confusion matrix is shown in Table 3.1. The overall recognition rate is 0.6371. It is very difficult to achieve a clear partition of emotions. The child tends to perform a combination of emotions (most frequently a combination of fear and angry). It therefore is difficult to distinguish the negative emotions of children.

Table 3.1: The confusion matrix for facial expression recognition on ASD children.

	Neutral	Angry	Fear	Happy	Sad
Neutral	0.5778	0.1765	0.0415	0.1107	0.0934
Angry	0.2035	0.5196	0.0536	0.1161	0.1071
Fear	0.1509	0.0943	0.4906	0.1509	0.1132
Happy	0.0491	0.0552	0.0773	0.7796	0.0387
Sad	0.0636	0.0909	0.1515	0.1060	0.5879

3.4.3. Related Functions

Related functions are F17 and F18.

Relationship of above functions and global variables are shown in Figure 3.15.

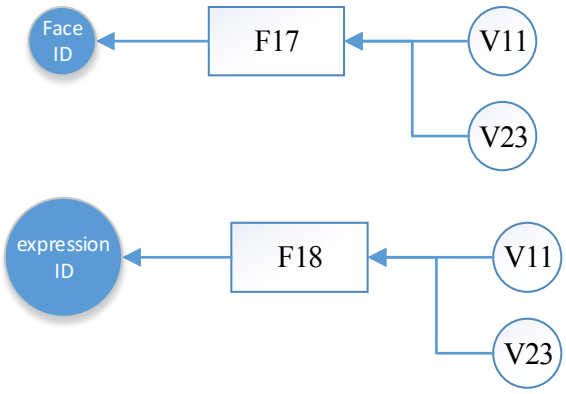


Figure 3.15: Relationship between face related functions and variables.

3.5. Speech and Sound Direction Recognition

3.5.1. Method

The speech recognition method is based on Microsoft Kinect SDK. The codes are written to utilize the trained model provided by the SDK to recognize the speech. A dictionary is designed to store the predefined key words and related short sentences, to make the speech recognition individually independent. The dictionary is fully customizable. This will bring convenience to users to recognize what sentences the subject say by key words. It will start to recognize the speech and returns a textual representation on the screen when the subject speaks. However, the proposed method cannot recognize speech in Romanian since there are no training samples provided by the SDK.

The direction of the incoming sound is identified based on the different places of microphones in the Kinect. The positions of microphones are shown in Figure 3.16. The sound will arrive at each of the microphones in a chronological order as the distances are different between microphones and the sound source.

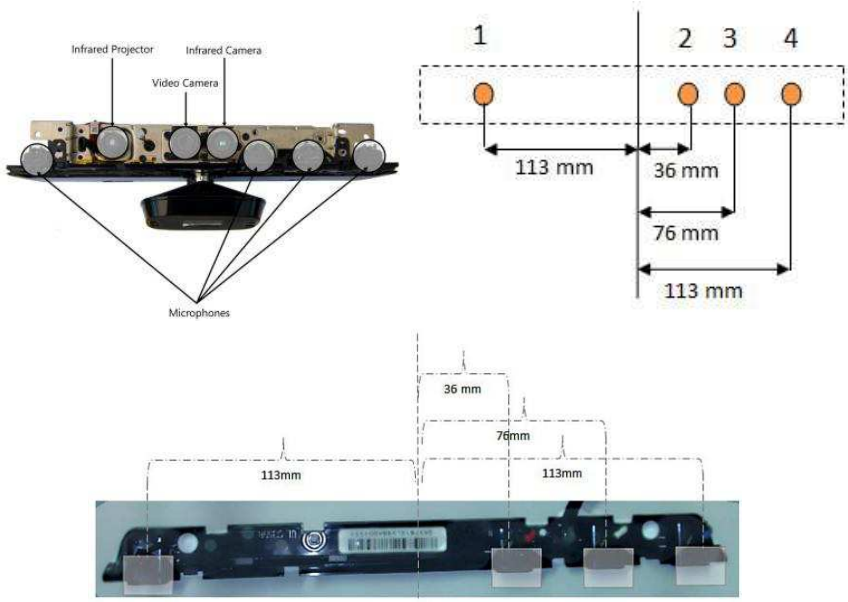


Figure 3.16: An illustration of distances between Kinect microphones.

A signal with higher-quality sound will be produced by processing the audio signals of all microphones after calculating the source and position of the sound. Two significant properties, which are the sound angle and the confidence of the sound angle, will be identified and then the system outputs the direction of the most crucial sound. The angles, including the sound source angle and the beam angle, are defined in the x-z plane of the sensor perpendicular to the z-axis of the sensor from the sensor location to analyse the sound source effectively; this will provide the direction of the sound but not the location of the sound. The range of the confidence is from one to zero, which represents the full confidence and no confidence respectively.

3.5.2. Experimental Results

The isolated words or continuous sentences can be recognized with this implementation of speech recognition that built on top of word recognition technology. The proposed speech recognition system displays the content of recognition results in plain text as output when the subject is speaking. Figure 3.17 shows the experiment results on speech recognition in English. The results validate that the system can successfully recognize the words and sentences in English.

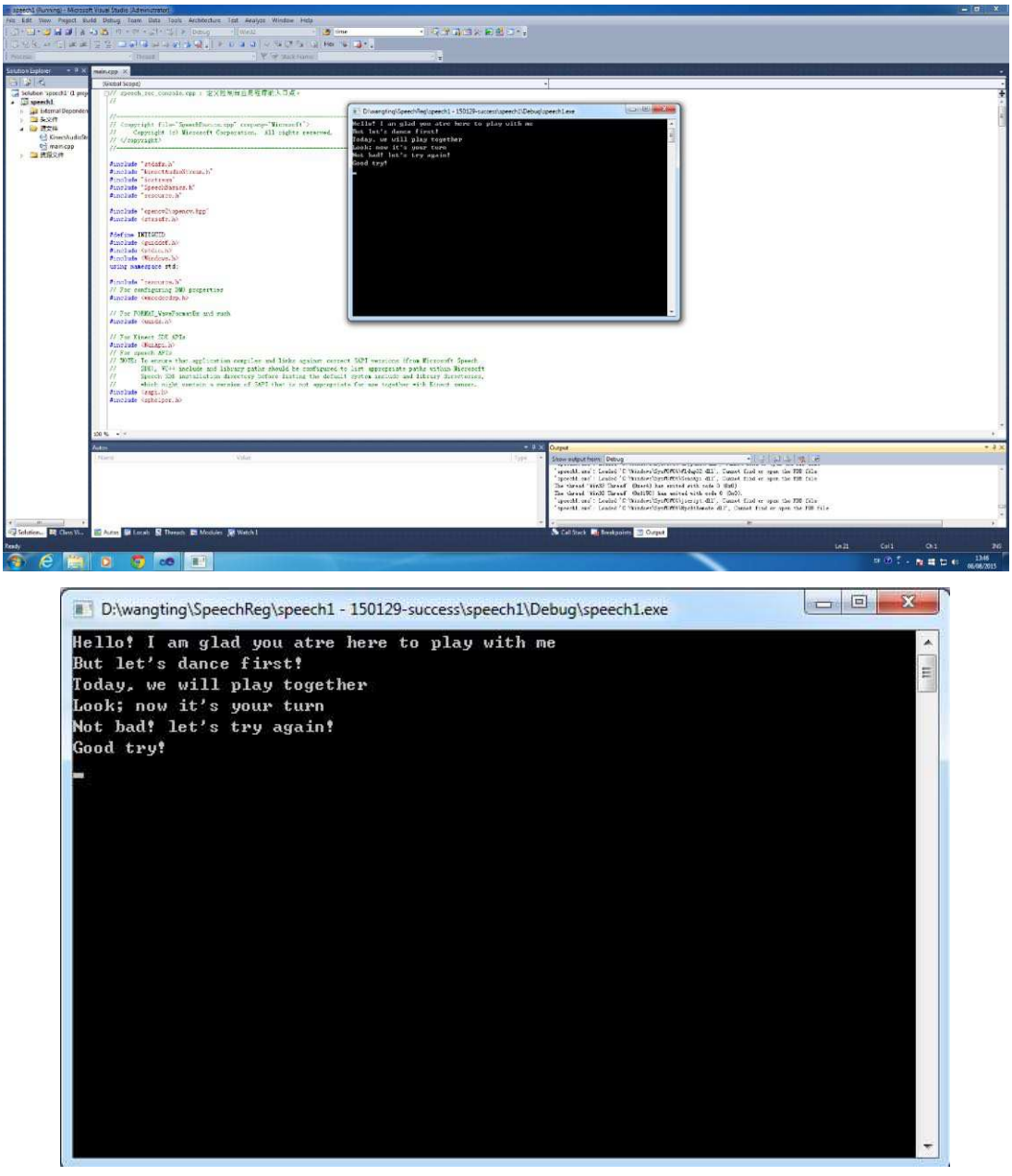
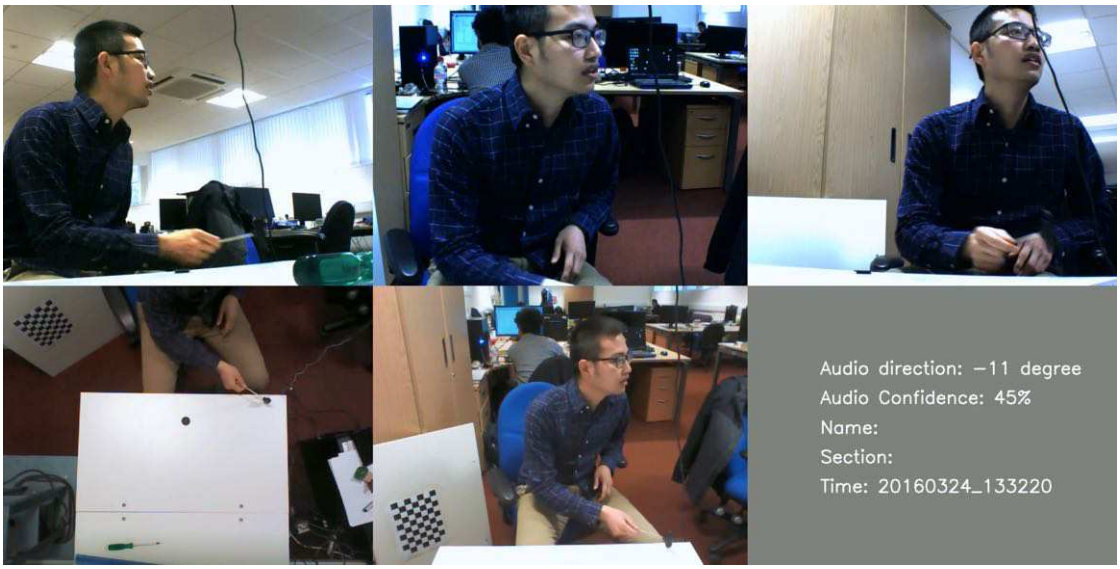
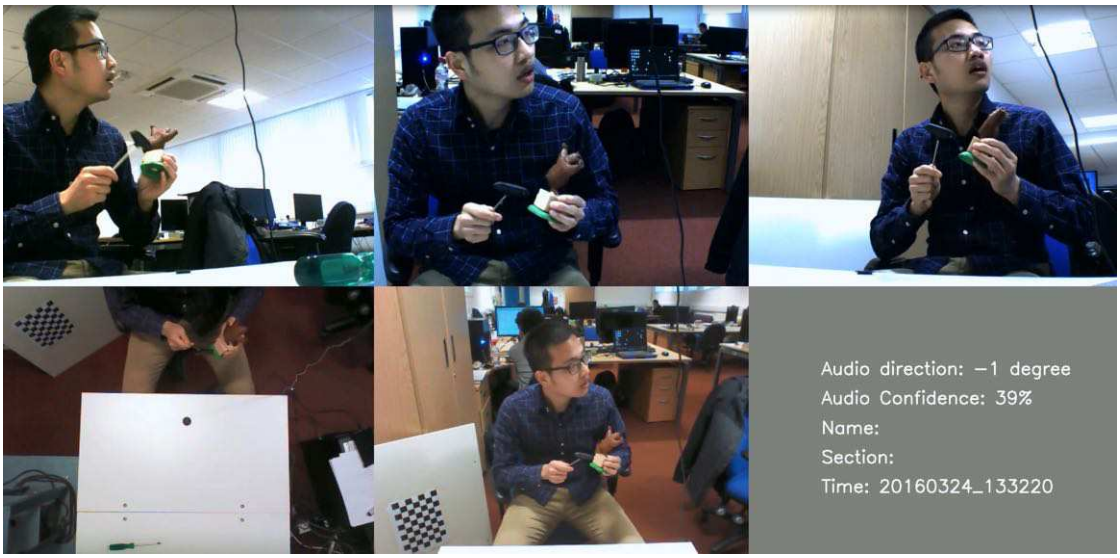


Figure 3.17: Speech recognition results in English.

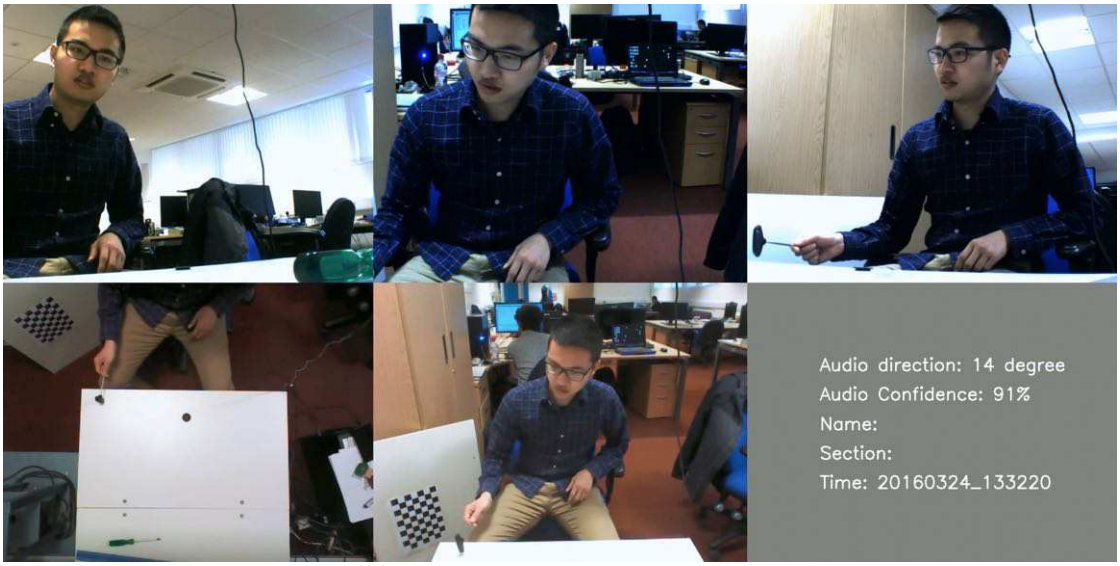
In Figure 3.18(a), the user makes a sound to the left of the Kinect and the system outputs the direction of the sound as originating from -11 degrees. The "-" indicates that the sound source is located to the left of the Kinect as measured from the front and the absolute value indicates the angle of the source of sound from the normal, which is perpendicular to the mid-point of the Kinect's front face. Similarly, the system can successfully recognize different sounds from the middle and right sides of the Kinect (as shown in Figure 3.18(b) and 3.18(c)). The sound from the middle side can be determined by setting a threshold of the angle of the source of sound. For example, if the angle of the source of sound is between -5 degrees and 5 degrees, then this sound could be regarded as a middle sound.



(a) Sound from the left side of the Kinect



(b) Sound from the middle of the Kinect



(c) Sound from the right side of the Kinect

Figure 3.18: Sound direction recognition results.

3.5.3. Related Functions

Function: Related functions are F16, F21 and F22.

Relationship of above functions and global variables are shown in Figure 3.19.

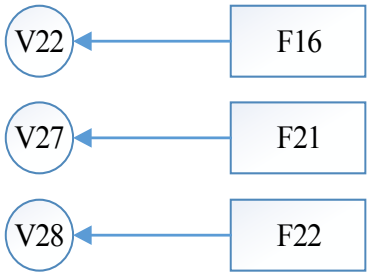


Figure 3.19: Relationship between audio related functions and variables.

3.5.4. Challenge

Currently, the system is able to identify the sound direction in Romanian, but it cannot recognize the speech in Romanian due to the lack of related training database. It is also a challenge to collect and train the Romanian language data without a native speaker's assistance.

4. YARP Implementation

The component of *sensoryInterpretation* aims to implement 25 perception primitives defined in Section 2 of Deliverable D1.3 (Child Behaviour Specification), and Section 3 of Deliverable D3.1 (System Architecture). We developed three additional components to support *sensoryInterpretation*. They are *usbCameraSource*, *usbCameraSelection* and *KinnectSource*. The structuralized design makes the whole system more flexible and allows different components to share the workload that originally belongs to one component, which improves the real-time performance in sensing children's behaviours. The complete system architecture of *sensoryInterpretation* is shown in Figure 4.1.

4.1. usbCameraSource

1) Component Description

It reads images directly from a camera and streams them to a YARP port. An instance of *usbCamerSource* can only read one USB camera.

2) Input Port

None.

3) Output Port

- */usbCameraSource/Cam:o*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The output port to which the images are updated.

4.2. usbCameraSelection

1) Component Description

It inputs images from 3 YARP ports synchronously, and selects an image with the best frontal face for further processing to get local information, like eye location, eye gaze, etc.

2) Input Port

- */cameraSelection/camMid:i*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The input port to which the images from **camera0** are streamed.
- */cameraSelection/camLeft:i*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The input port to which the images from **camera1** are streamed.
- */cameraSelection/camRight:i*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The input port to which the images from **camera2** are streamed.

3) Output Port

- */cameraSelection/cam:o*
- *BufferedPort<ImageOf<PixelRgb>>*
  Note: The output port to which the selected image with the best frontal face is streamed. In this image, 49 facial landmarks, eye location *etc.* can be visualized for demonstrations.

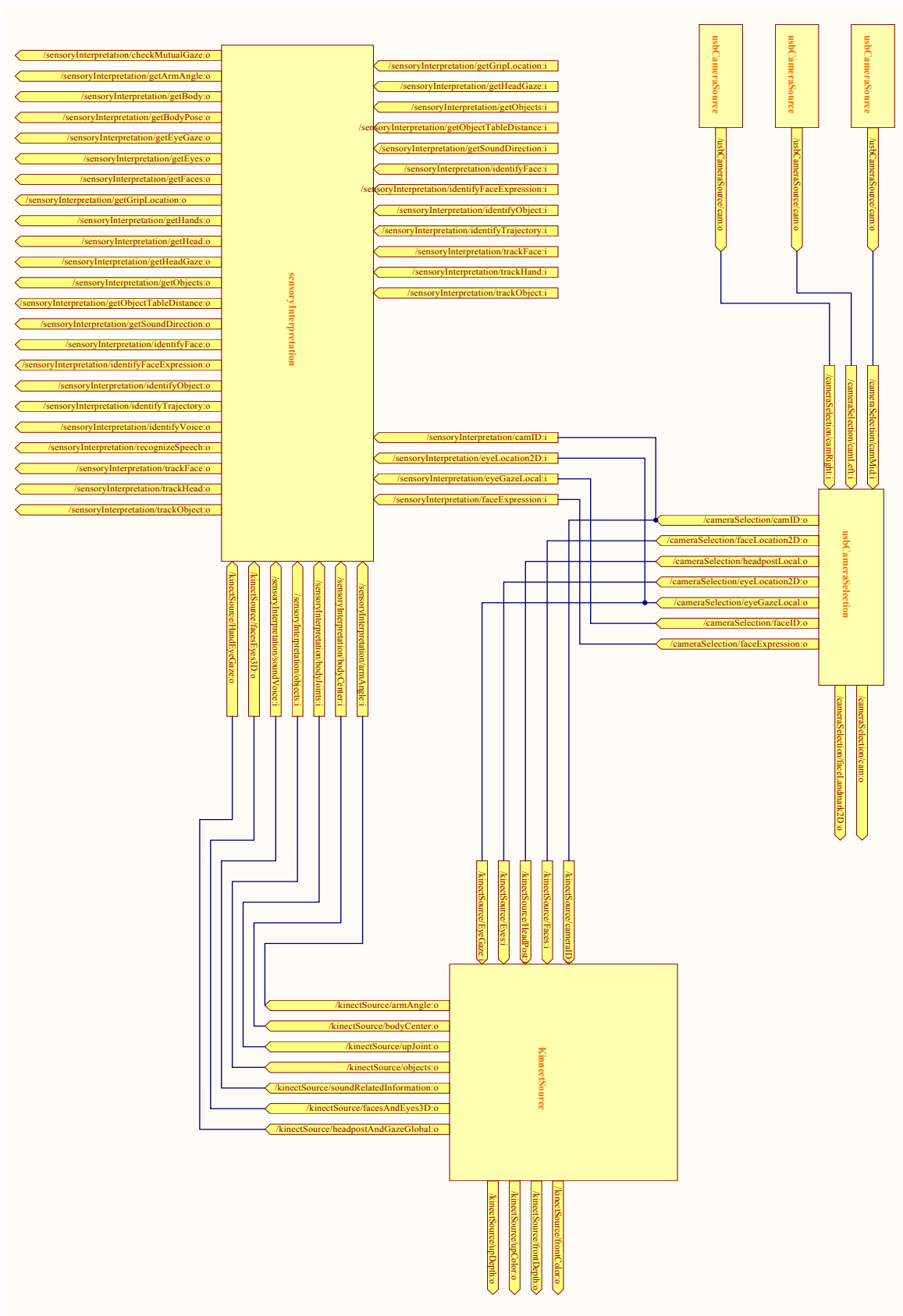


Figure 4.1: The *sensoryInterpretation* component architecture.

- */cameraSelection/camID:o*
  BufferedPort<VectorOf<int>>
  Note: The complete output port to which the **ID of the selected camera (1, 0 and 2 refer to the left, middle, and right cameras respectively)** is streamed. The length of output vector is 2 with an additional valid state indicator, and it is formatted as [*id, state*], where *state = 0* indicates an invalid camera ID information and *state = 1* indicates a valid camera ID information. When face is not detected from any camera, *state* is set to 0. But the invalid information is still output to YARP server, to implement unblock reading in YARP. (This design can be found in the following contents, and we will not explain it again.)
- */cameraSelection/faceLocation2D:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **face locations** in the selected image are streamed. The length of the vector changes with the number of faces being detected, and it is formatted as [*N, face1.x, face1.y, face2.x, face2.y, …*]. *N* refers to the number of faces.
- */cameraSelection/faceLandmark2D:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **98 landmarks of a face** are streamed. The length of the vector is 99, formatted as [*mark1.x, mark1.y, mark2.x, mark2.y, … , mark48.x, mark49.y, state*].
- */cameraSelection/headpostLocal:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which the **head gaze** with respect to the post of the selected camera is streamed. The length of the vector is 4, and it is formatted as [*pitch, yaw, roll, state*].
- */cameraSelection/eyeLocation2D:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **eyes' location** of a face is streamed. The length of the vector is 5, and it is formatted as [*righteye.x, righteye.y, lefteye.x, lefteye.y, state*].
- */cameraSelection/eyeGazeLocal:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which the **eye gaze** with respect to the post of the selected camera is streamed. The length of the vector is 3, and it is formatted as [*pitch, yaw, roll, state*].
- */cameraSelection/faceID:o;*
  BufferedPort<VectorOf<int>>
  Note: The output port to which the **face ID (0,1,2 … stand for different individuals)** is streamed. The length of the vector is 2, and it is formatted as [*ID, state*].
- *cameraSelection/faceExpressID:o*
  BufferedPort<VectorOf<int>>
  Note: The complete output port name to which the **facial expression (0, 1, 2, 3 and 4 stand for neutral, happy, sad, angry and fearful)** is streamed. The length of the vector is 2, and it is formatted as [*ID, state*].

4.3. KinectSource

1) Component Description

This component functions in two parts: a) reads RGB and depth information from 2 Kinects, and obtains the locations of upper body joints, body centre, arm angles and sound direction of a child; b) reads the selected camera ID, 2D face, eye locations, local head and eye gazes from *usbCameraSelectioin*, and transfers these coordinates to the world coordinate system.

2) Input Port

- */ kinectSource /cameraID:i*, connecting to */cameraSelection/camID:o*
- */ kinectSource /faces:i*, connecting to */cameraSelection/faceLocation2D:o*
- */ kinectSource /headPost:i*, connecting to */cameraSelection/ headPostLocal:o*
- */ kinectSource /eyes:i*, connecting to */cameraSelection /eyeLocation2D:o*
- */ kinectSource /eyeGazeLocal:I*, connecting to */cameraSelection /eyeGaze:o*

3) Output Port

- */kinectSource/frontColor:o*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The output port to which the RGD image of the front Kinect (Kinect0) is streamed.
- */kinectSource/frontDepth:o*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The output port to which the depth image of the front Kinect (Kinect0) is streamed.
- */kinectSource/upColor:o*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The output port to which the RGB image of the upper Kinect (Kinect1) is streamed.
- */kinectSource/upDepth:o*
  BufferedPort<ImageOf<PixelRgb>>
  Note: The output port to which the depth image of the upper Kinect (Kinect1) is streamed.
- */kinectSource/ArmAngle:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **the azimuth and elevation angles of upper left and right arms** of a child are streamed. The length of the vector is 5, and it is formatted as [*left_elevation, left_azimuth, right_elevation, right_azimuth, state*].
- */kinectSource/bodyCenter:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **the body centre** of a child is streamed. The length of the vector is 4, and it is formatted as [*bodyCenterOut.x, bodyCenterOut.y, bodyCenterOut.z, state*].
- */kinectSource/upJoints:o*
  BufferedPort<VectorOf<double>>

Note: The output port to which **10 upper body joints** of shoulder centre, head, left shoulder, left elbow, left wrist, left hand, right shoulder, right elbow, right wrist and right hand are streamed. The length of the vector is 31, and it is formatted as [*joint1.x, joint1.y, joint1.z, ..., joint10.x, joint10.y, joint10.z, state*], where joint1, joint2 …, and joint10 refer to shudder centre, head, left shoulder, left elbow, left wrist, left hand, right shoulder, right elbow, right wrist and right hand respectively.

- */kinectSource/soundRelatedInformation:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **the horizontal and vertical angles defining the direction to the loudest sound of the environment, voice ID and speech ID** are streamed. The length of the vector is 5, and it is formatted as [*horizontal_sound_direction, vertical_sound_direction, voice_ID and speech_ID, state*]. *Voice_ID* refers to the person who makes voice, and *speech_ID* refers to one of several predefined sentences being identified.
- */kinectSource/Objects:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **the locations of objects in the view of upper Kinect** are streamed. The length of the vector is 8, and it is formatted as [*o1.x, o1.y, o1.z, o1.state, o2.x, o2.y, o2.z, o2.state*], where o1 refers to the left side object on the table, and o2 refers to the right side object. *o1.state* and *o2.state* indicate whether the object is identified.
- /kinectSource/facesAndEyes3D*:o*
  BufferedPort<VectorOf<double>>
  Note: The output port to which **the locations of two eyes, multiple faces and camera ID** are streamed. The length of the vector is N*3+9, and it is formatted as [*state, CamID, leye.x, leye.y, leye.z, reye.x, reye.y, reye.z, N, face1.x, face1.y, face1.z, ..., faceN.x, faceN.y, faceN.z*], where *N* refers to the number of faces, *state* shows whether this data is valid.
- */kinectSource/headpostAndGazeGlobal:o*
  BufferedPort<VectorOf<double>>
  Note: The output port name to which **the directions of head and eye gaze in the world** are streamed. The length of the vector is 7, and it is formatted as [*state, headgaze.x, headgaze.y, headgaze.z, eyegaze.x, eyegaze.y, eyegaze.z, o2.s*].

4.4. sensoryInterpretation

1) Component Description

This component reads data *usbCameraSelection* and *KinectSource*, and then reorganizes the data format according to the definition of 25 perception primitives. It also encapsulates the internal outputs of *usbCameraSelection* and *KinectSource* into standard outputs according to the primitives defined in Section 3 of Deliverable D3.1 (System Architecture).

2) Input Port

Internal input ports from *cameraSelection* and *kinectSource*

- */sensoryInterpretation/camID:i*, connecting to */cameraSelection/camID:o*

- */sensoryInterpretation/eyeGazeLocal:i*, connecting to */cameraSelection/eyeGazeLocal:o*
- */sensoryInterpretation/faceID:i*, connecting to */cameraSelection/faceID:o*
- */sensoryInterpretation/faceExpression:i,* connecting to */cameraSelection/faceExpression:o*
- */sensoryInterpretation/armAngle:i*, connecting to */kinectSource/armAngle:o*
- */sensoryInterpretation/bodyCenter:I*, connecting to */kinectSource/bodyCenter:o*
- */sensoryInterpretation/bodyJoints:i* ,connecting to */kinectSource/upJoint:o*
- */ sensoryInterpretation/objects:i* ,connecting to */kinectSource/objectsLocation:o*
- */ sensoryInterpretation /soundVoice:i* ,connecting to */kinectSource /soundRelatedInformation:o*
- */ sensoryInterpretation/facesEyes:i* ,connecting to */kinectSource /facesAndEyes3D:o*
- */ sensoryInterpretation/headEyeGaze:i* ,connecting to */kinectSource /headPostAndGazeGlobal:o*

External input ports from *users' component*

- */sensoryInterpretation/getGripLocation:i* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/getHeadGaze:i* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/getObjects:i* */sensoryInterp* BufferedPort<VectorOf<double>>
- */sensoryInterpretation/getObjectTableDistance:i* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/getSoundDirection:i* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/identifyFace:o* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/identifyFaceExpression:o* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/identifyObject:o* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/identifyTrajectory:o* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/trackFace:o* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/getHead:o* *BufferedPort<VectorOf<double>>*
- */sensoryInterpretation/trackObject:o* *BufferedPort<VectorOf<double>>*

3) Output Port

1. */sensoryInterpretation/checkMutualGaze:o*
   BufferedPort<VectorOf<int>>
   Note: The length of the vector is 1, and it is formatted as [*state*]. *state = 0* and *state = 1* refer to no mutualGaze and mutualGaze, respectively.
2. */sensoryInterpretation/getArmAngle:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 4, and it is formatted as [*left_elevation, left_azimuth, right_elevation, right_azimuth*], referring to the azimuth and elevation angles of the upper left and right arms of a child.
3. */sensoryInterpretation/getBody:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 3, and it is formatted as [*x, y, z*].
4. */sensoryInterpretation/getBodyPose:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 30, and it is formatted as [*joint1.x, joint1.y, joint1.z, ..., joint10.x, joint10.y, joint10.z*]. 10 joint positions are listed in the order of shudder centre, head, left shoulder, left elbow, left wrist, left hand, right shoulder, right elbow, right wrist and right hand.
5. */sensoryInterpretation/getEyeGaze:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 3, and it is formatted as [*x, y, z*]. Together with the input from */sensoryInterpretation/getEyeGaze:i* containing eye's position, the gaze direction originated from child's eye can be given.
6. */sensoryInterpretation/getEyes:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 6, and it is formatted as [*leftEye.x, leftEye.y, leftEye.z, rightEye.x, rightEye.y, rightEye.z*].
7. */sensoryInterpretation/getFaces:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 3**N*+1, and it is formatted as [*N, face1.x, face2.y, face3.z, faceN.x, faceN.y, faceN.z*]. *N* refers to the number of faces being detected.
8. */sensoryInterpretation/getGripLocation:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 3, and it is formatted as [*x, y, z*], referring to the location of the incidence of gripping an object.
9. */sensoryInterpretation/getHands:o*
   BufferedPort<VectorOf<double>>
   Note: The length of the vector is 6, containing the locations of two hands. The vector is formatted as [*left.x, left.y, left.z, right.x, right.y, right.z*].
10. */sensoryInterpretation/getHead:o*
    BufferedPort<VectorOf<double>>
    Note: The length of the vector is 3, containing child's head location. The vector is formatted as [*head.x, head.y, head.z*].
11. */sensoryInterpretation/getHeadGaze:o*

BufferedPort<VectorOf<double>>
Note: When there is no input from */sensoryInterpretion/getHeadGaze:i*, this port outputs a vector formatted as [*x, y, z*]. It reflects a child's gaze direction originated from the head position.

12. */sensoryInterpretation/getHeadGaze:o*
BufferedPort<VectorOf<double>>
Note: When receiving the input from */sensoryInterpretion/getHeadGaze:i*, this port outputs the location on a flat surface. The output vector is formatted as [*x, y, z*].
13. */sensoryInterpretation/getObjects:o*
/sensoryInterp BufferedPort<VectorOf<double>>
Note: The length of the vector is 4, and it is formatted as [*object1.x, object1.y, object1.z, objects1.state, object2.x, object2.y, object2.z, objects2.state*]. If *objects1.state* = 1, an object on the left side of the table is detected and the position of this object is given by the coordinate (*object1.x, object1.y, object1.z*). If *objects1.state* = 0, no object is detected on the left side of the table. If *objects2.state* = 1, an object on the right side of the table is detected and the position of this object is given by the coordinate (*object2.x, object2.y, objec2.z*). If *objects1.state* = 0, no object is detected on the right side of the table.
14. */sensoryInterpretation/getObjects:o*
BufferedPort<VectorOf<double>>
Note: When receiving data from */sensoryInterpretation/getObjects:i,* this port outputs locations of objects in a certain area. The length of the vector is 3**N+1*, and it is formatted as [N, *x1*, *y1*, *z1*, …, *xN*, *yN*, *zN*]. *N* is the number of objects being identified.
15. */sensoryInterpretation/getObjectTableDistance:o*
BufferedPort<VectorOf<double>>
Note: With the input of an object' location from */sensoryInterpretation/getObjectTableDistance:i,* this port outputs the vertical distance between the table and the object. The length of the output vector is 1, and it is formatted as [*vectical_distance*]. If no data is received form the input port, no data will be output.
16. */sensoryInterpretation/getSoundDirection:o*
BufferedPort<VectorOf<double>>
Note: The length of the output vector is 2, and it is formatted as [*horizontal_sound_direction, vertical_sound_direction*]. This angle directs to the loudest sound (higher than a threshold) of the environment from the front Kinect0. If no input data is received from */sensoryInterpretation/threshold:i*, a default threshold will be applied.
17. */sensoryInterpretation/identifyFace:o*
BufferedPort<VectorOf<double>>
Note: The length of the output vector is 1, and it is formatted as [*faceID*]. If no face is identified, no data will be output form this port.
18. */sensoryInterpretation/identifyFaceExpression:o*
BufferedPort<VectorOf<double>>
Note: The length of the output vector is 1, and it is formatted as [*ExpressionID*]. If no face is identified, no data will be output form this port.

19. */sensoryInterpretation/identifyObject:o*
    BufferedPort<VectorOf<double>>
    Note: The length of the vector is 1, and it is formatted as [*objectID*].
20. */sensoryInterpretation/identifyTrajectory:o*
    BufferedPort<VectorOf<double>>
    Note: The length of the vector is 1, and it is formatted as [*state*], where 0 and 1 stand for the incidence and non-incidence of hand waving.
21. */sensoryInterpretation/identifyVoice:o*
    BufferedPort<VectorOf<int>>
    Note: The length of the vector [*ID*] is 1, and the person's ID who arises the voice is streamed.
22. */sensoryInterpretation/recognizeSpeech:o*
    BufferedPort<Bottle>
23. */sensoryInterpretation/trackFace:o*
    BufferedPort<VectorOf<double>>
    Note: The length of the vector is 3, and it is formatted as [*face.x, face.y, face.z*]. It outputs exactly the same values as */sensoryInterpretation/trackFace:o*
24. */sensoryInterpretation/getHead:o*
    BufferedPort<VectorOf<double>>
    Note: The length of the vector is 3, and it is formatted as [*x, y, z*] to represent the head position in the world.
25. */sensoryInterpretation/trackObject:o*
    BufferedPort<VectorOf<double>>
    Note: The length of the vector is 3, and it is formatted as [*x, y, z*] to represent the location of a specified object (left or right object).

5. Multi-Sensory Data Fusion - A Preliminary

In this chapter, a preliminary that includes camera selection and coordinate transformation for multi-sensory data fusion is presented. As shown in Figure 2.1, there are two main modules involved in the framework for sensory data fusion: Camera Selection Module (CSM) and Coordination Transformation Module (CTM). The CSM is employed to determine which sensor is optimal for capturing the best view of a subject's face, while the CTM is utilized to transform all individual sensory data to a global coordinate system. By doing so, the user can directly collect and use the sensory data output by the system.

5.1. Camera Selection

5.1.1. Method

All the data captured by three cameras and two Kinects needs to be synchronized for further analysis. A multi-sensor selection strategy [12] is used to keep the synchronization of each sensor while at the same time keep the system running in real time. To deal with the synchronization problem, a multi-threaded programing strategy is employed, where each sensor owns a separate thread, and a controlling thread is used to coordinate the start and end of all other threads. To acquire real time performance, the multi-sensor selection strategy is divided into two stages, namely detection stage and tracking stage. The detailed procedures of the two stages are shown in Figure 5.1 and Figure 5.2, respectively.

In the detection stage, the first step is to calibrate available sensors, and then capture the sensory data in parallel. With the captured sensory data, methods for the face detection, face features extraction, head pose estimation, and object detection can be invoked. The camera that captures the most frontal face is selected for gaze estimation, face recognition and facial expression analysis. The face features extraction and head pose estimation methods are applied on the selected images. The data captured by Kinect0 is used for child's head detection and the data captured by the Kinect1 is for objects and robot head detection.

In the tracking stage, based on the selected camera and the two Kinects, methods for facial feature points and head pose tracking, child's head tracking, and objects and robot head tracking can be performed in real time. Since the processing speed of detection stage is relatively slow, the system can recall the tracking stage to improve the efficiency and thus lead to a real time performance.

By combining the detection and tracking stages together, the whole system can run in real time. In order to perform optimal camera selection, a face confidential score of each camera is defined. This score is acquired by measuring the variation of facial landmarks of detected face with respect to facial landmarks of a predefined frontal face. The detection and tracking stages will output a face confidential score for each camera. The camera with the highest face confidential score will be selected as the optimal camera.

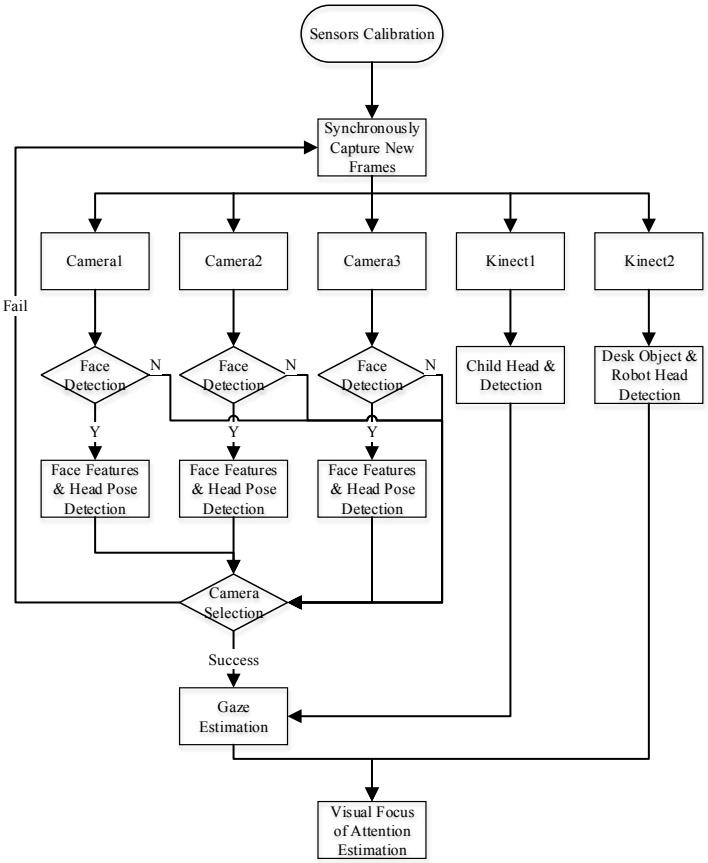


Figure 5.1: The detection stage.

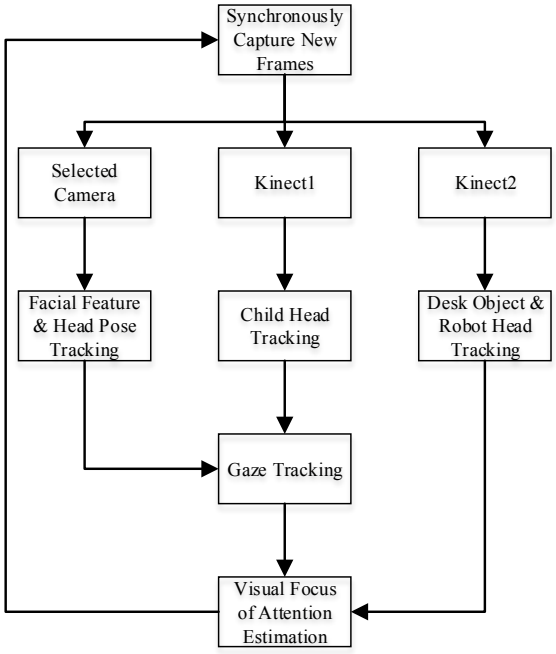


Figure 5.2: The tracking stage.

5.1.2. Experimental Results

The results of the camera selection module are shown in Figure 5.3. These results show that the camera can be correctly selected based on the face confidential score. Figures 5.3(a), 5.3(b) and 5.3(c) demonstrate the selected results when a subject is facing forward, left and right, respectively.

(a) Selected results when a subject is facing forward.

(b) Selected results when a subject is facing left.

(c) Selected results when a subject is facing right.

Figure 5.3: Results of optimal camera selection strategy.

5.2. Coordinate Transformation

The goal of CTM is to transform all the local data captured from the individual cameras to a global coordinate system. To this end, we first need to determine the position and orientation of each camera, given its intrinsic parameters and a set of n correspondences between 3D points and their 2D projections. Then, any 3D point coordinate in the camera coordinate system can be transformed to the global 3D coordinate system with the rotation and translation matrices of the camera. The workflow of the proposed camera pose estimation is shown in Figure 5.4.

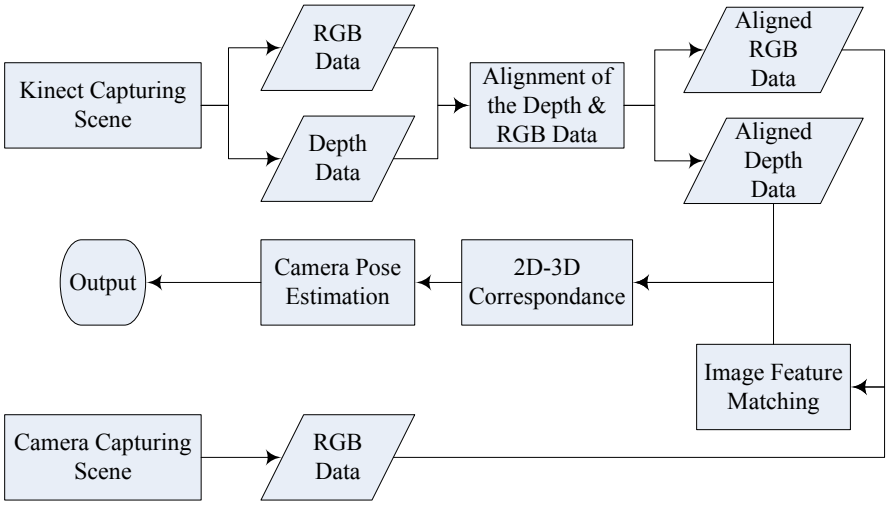


Figure 5.4: Workflow of the proposed camera pose estimation.

5.2.1. Method

Our implementation is based on an Efficient Perspective-n-Points (EPnP) algorithm proposed by Vincent et al. [13]. Our camera pose estimation method has a robust result when different camera poses are encountered. It only requires users to mark corresponding points between Kinect images and Camera images manually for about 20 pairs. This method is prefered because it is more reliable than all the other feature-matching algorithms tested. With the intrinsic parameters of the cameras, the poses of those cameras related to the Kinect can be determined robustly. As shown in the following equation

$$\mathbf{m}_i \approx \mathbf{K}(\mathbf{R},\mathbf{t})\widehat{\mathbf{M}}_i \tag{5-1}$$

where $\mathbf{m}_i$ is the projection of the 3D point $\mathbf{M}_i$ onto the camera image with $\mathbf{K}$ being intrinsic parameters of the camera. $\mathbf{R}$ is the rotation matrix and $\mathbf{t}$ is the translation matrix. $\mathbf{m}_i$, $\mathbf{K}$ and $\mathbf{M}_i$ are known in the equation. With more than 3 pairs of $\mathbf{m}_i$-$\mathbf{M}_i$ correspondences, the $\mathbf{R}$ and $\mathbf{t}$ can be estimated using optimization algorithms. In our implementation, the $\mathbf{m}_i$-$\mathbf{M}_i$ correspondences are more than 20 pairs to improve the robustness of the process.

Therefore, the first step for camera pose estimation is to find the 2D-3D correspondence between the 2D points in the camera image and the 3D points in the space. Because the Kinect

can generate both RGB and depth images, the 2D-3D correspondence can be done through an intermediate step of 2D-2D correspondence between the camera RGB image and the Kinect RGB image. Then the relationship between points in the Kinect RGB image and the Kinect depth image will provide the 2D-3D correspondence mentioned above.

To ensure the accuracy of the estimation, the 2D-2D correspondence is achieved by manually marking corresponding 2D points on the RGB image from the normal camera and RGB image captured by the Kinect. A calibration object is used to assist this marking process. This object is shown in the field of view (FOV) of both camera and Kinect. The same point on the object is marked in RGB images from both the camera and Kinect. With this process, the accurate 2D-2D correspondence can be obtained.

In the meanwhile, the process of alignment between the RGB image and the depth image both generated from Kinect is carried out. However, the shift of the location of the different sensors causes a shift between the RGB image and Depth Image. This presents an obstacle for searching from 3D points in space to 2D points in the camera image, which has 2D-2D correspondence to Kinect RGB image. This could be solved by taking into account of the constant distance between the RGB sensor and the infrared sensor in the Kinect device. With the knowledge of FOV of the Kinect, we can modify every pixel in the depth image accordingly to make them aligned with the pixels in RGB image. After alignment, for every coordinate of 2D point in RGB image, we can retrieve the corresponding 2D coordinate in Depth image. Then coordinate of 3D point in the space can be obtained with the Equation (5-2).

$$\frac{x_p}{u-u_0} = \frac{y_p}{v-v_0} = \frac{z_p}{f} \tag{5-2}$$

where (u_0, v_0) is the depth image centre of the Kinect, and f is the focal length of the infrared camera. (x_p, y_p, z_p) is the 3D coordinate of a point in the space corresponding to the 2D point of (u, v) in the depth image. The alignment result of RGB image and Depth image is illustrated in Figure 5.5.

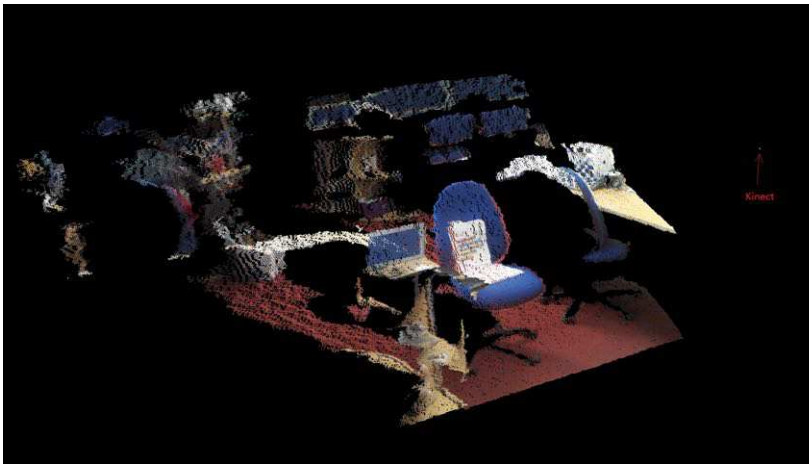


Figure 5.5: The point cloud collection by a Kinect after the RGB image and the Depth image has been aligned. The Kinect is in front of the chair.

When 2D-3D correspondence is obtained, the next process is to estimate the camera pose. In our method, this process is mainly based on an iterative process. In every loop of iterations,

a Perspective-n-Points (PnP) algorithm is applied along with the 2D-3D correspondence calculated by the previous process. There is a wide range of PnP algorithm implementations in the community. We choose an EPnP algorithm according to its high efficiency in calculation. The EPnP algorithm is an O(n) non-iterative process in the first place. We put it into a sequence of loops because the main process of the PnP algorithm is about parameterization and quadratic equations solving, which will also bring in errors when outliers are input. To minimize this, in each iteration, we firstly apply the EPnP algorithm with the 2D-3D correspondences. Then a projection process from every 3D point in space to 2D points is conducted with the estimated camera rotation and translation in the current loop. By comparing the projected 2D points and the true 2D points in the camera image, the outliers of the 2D-3D pairs can be counted. If the number of outliers is larger than a predefined threshold, such as the 40% of the total number of the point-pairs in our implementation, then randomly down sample the 2D-3D point pairs to a predefined number of count, such as the 60% of the total number of the point-pairs in our implementation. After randomly down sampling, next loop starts. If the number of outliers is less than the threshold, or the total count of the loop is larger than a predefined number, the iteration should end, and the final results of the camera pose can be output.

Furthermore, we have also provided an implementation for transforming local 3D coordinates to global 3D coordinates. The transformation process is based on the Rotation and Translation of the Camera relative to the global coordinate system.

With the previously obtained results of the camera poses, the coordinates of the 3D points can be easily transformed from camera coordinate system (local 3D coordinate) to Kinect coordinate system (global 3D coordinate). To achieve unified 3D coordinates in same coordinate system when the points from different cameras, the following equation can be used.

$$\mathrm{P} = R * \mathrm{P}' + t \tag{5-3}$$

where P’ is a 3D point in camera coordinate system and P is the corresponding 3D point in unified coordinate system. R and t are the rotation and translation matrices of the camera, which are also known as the pose of the camera. Similarly, the same process can be applied for other cameras.

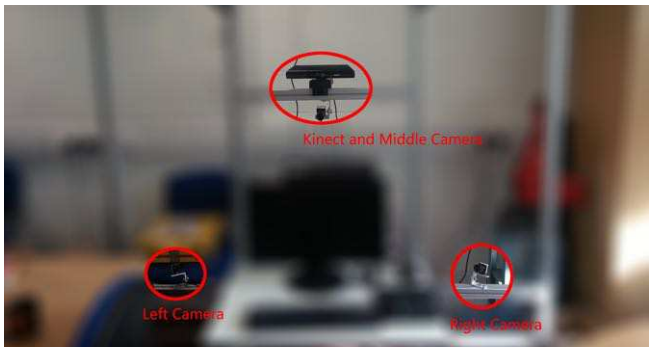
For those facial points where one of the cameras and the middle Kinect can both capture, it is easy to find their global 3D coordinates. However, it is sometimes hard for both devices to capture the same facial points in many situations because of the large head movement. Thus a 2D to 3D coordinate transformation for these located 2D facial points is necessary. The transformation can be performed using the following equation.

$$\begin{cases} P_C = R^{-1} * P_W - R^{-1} * T \\ \dfrac{X_C^{'}}{u - u_0} = \dfrac{Y_C^{'}}{v - v_0} = \dfrac{Z_{PC}^{'}}{f} \\ Z_C^{'} = Z_{PC}^{'} \end{cases} \tag{5-4}$$

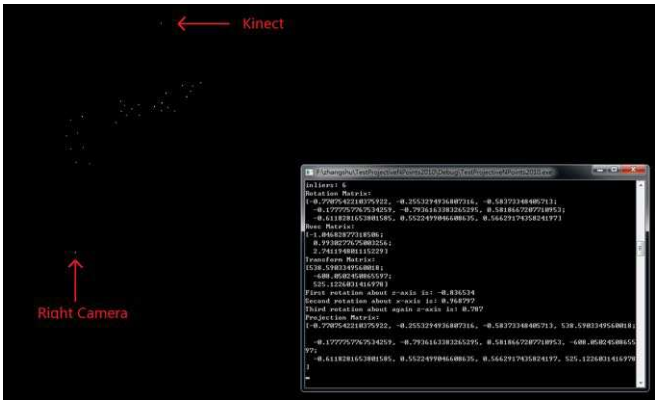
where P_W refers to the head centre position in world coordinate, P_C is the head centre position in the local coordinate; (u_0, v_0) is the image centre of the camera, and f is the focal length of the camera. $\left(X_C^{'}, Y_C^{'}, Z_C^{'}\right)$ is the 3D coordinate of a point in the local coordinate of the camera, which is corresponding to the 2D point of (u, v) in the image. $Z_{PC}^{'}$ is the depth value of head centre point in the local coordinate system. The depth value of the facial points is replaced by the depth value of head centre in local coordinate for the calculation of its 3D points in local coordinate. Its global coordinate can be acquired by using Equation (5-2).

5.2.2. Experimental Results

The experimental results of camera pose estimation are shown in Figure 5.6. The origin of the 3D coordinate system is seated in the Kinect. Compared to the ground truth shown in Figure 5.6(a), the poses of the cameras in the middle, left and right are estimated accurately and some matched points between each camera and Kinect can be seen in Figures 5.6(b), 5.6(c) and 5.6(d).

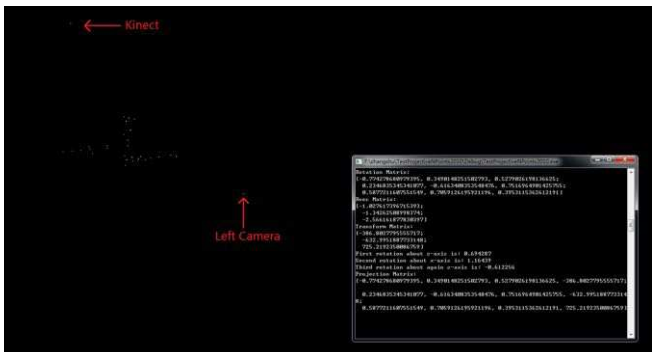


(a) Relative Positions between Kinect and Three Cameras



(b) Calculated Relative Positions between the Kinect and the Right Camera

(c) Calculated Relative Positions between the Kinect and the Middle Camera



(d) Calculated Relative Positions between the Kinect and the Left Camera

Figure 5.6: The experimental results by the proposed method.

With the proposed method, the parameters of the rotation and translation matrices for the three cameras with respect to the Kinect0 can be obtained in Table 5.1.

Table 5.1: The rotation and translation matrices of three cameras with respect to the Kinect0.

	Rotation matrix	Translation matrix
Camera0	[-0.98808, 0.08217, 0.13015; -0.01150, -0.88261, 0.46996; 0.15349, 0.46287, 0.87304]	[-23.4745; -139.0706; 36.2407]
Camera1	[-0.94568, 0.24830, 0.20983; 0.01914, -0.60182, 0.79841; 0.32452, 0.75906, 0.56438]	[-265.7251; -640.1735; 251.9052]
Camera2	[-0.83212, -0.43233, -0.34736; -0.11177, -0.48276, 0.86859; -0.54321, 0.76160, 0.35339]	[457.1840; -634.0433; 509.5746]

To validate the correctness of obtained rotation and translation matrices, a coordinate transformation experiment between the Camera0 and Kinect0 is devised. Figure 5.7 shows the colour images captured by the Kinect0 and Camera0. First, the colour and depth images captured by the Kinect0 are aligned using the Kinect SDK, the aligned result is shown in Figure 5.8. Then, the 3D point cloud is recovered according to the Equation (5-2). The recovered 3D points are then transformed to the Camera0 local coordinate system to get their relative colour information using Equation (5-3). Finally, the 3D points cloud is transformed

back into the world coordinate system after the colour information has been updated. The coordinate transformation result is shown in Figure 5.9, which visually validates the acceptable performance of the proposed method.

Figure 5.7: Colour images captured by the Kinect0 (left) and the Camera0 (right).

Figure 5.8: The aligned result of colour and depth images captured by the Kinect0.

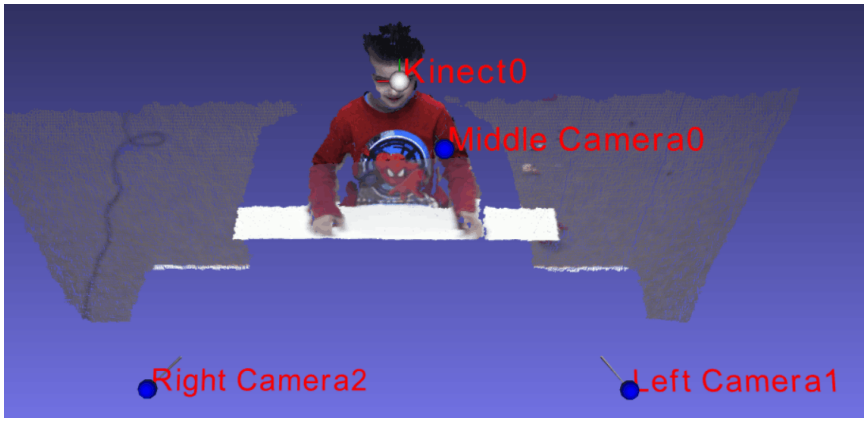


Figure 5.9: Coordinate transformation result by transforming the data captured by the Camera0 to the data captured by the Kinect0

References

[1] P. Viola and M.J. Jones, Robust real-time face detection. *International Journal of Computer Vision*, 2004, 57(2): 137-154.

[2] J.G. Daugman, High Confidence Visual Recognition of Persons by a Test of Statistical Independence. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 1993, 15(11): 1148-1161.

[3] X.H. Xiong and F. De la Torre, Supervised Descent Method and its Applications to Face Alignment. *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2013, pp. 532-539.

[4] D.F. Dementhon and L.S. Davis, Model-Based Object Pose in 25 Lines of Code. *International Journal of Computer Vision*, 1995, 15(1-2): 123-141.

[5] R. Valenti and T. Gevers, Accurate Eye Center Location through Invariant Isocentric Patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2012, 34(9): 1785-1798.

[6] N. Markus, et al., Eye pupil localization with an ensemble of randomized trees. *Pattern Recognition*, 2014, 47(2): 578-587.

[7] F. Jan, I. Usman, and S. Agha, A non-circular iris localization algorithm using image projection function and gray level statistics. *Optik*, 2013, 124(18): 3187-3193.

[8] L. Agarwal and K. Lakhwani, Optimization of frame rate in real time object detection and tracking, *International Journal of Scientific & Technology Research*, 2013, 2(7): 132-134.

[9] X. Zhou, H. Yu, H. Liu, and Y.F. Li, Tracking multiple video targets with an improved GM-PHD tracker, *Sensors*, 2015, 15(12): 30240-30260.

[10] X. Zhou, Y.F. Li, and B. He, Entropy distribution and coverage rate-based birth intensity estimation in GM-PHD filter for multi-target visual tracking, *Signal Processing*, 2014, 94: 650-660.

[11] D. Huang, C. Shan, M. Ardabilian, Y. Wang, and L. Chen. Local Binary Patterns and Its Application to Facial Image Analysis: A Survey. *IEEE Transactions on Systems, Man, and Cybernetics, part C: Applications and Reviews*, 2011, 41(6):765-781.

[12] H. Cai, X. Zhou, H. Yu, and H. Liu, Gaze estimation driven solution for interacting children with ASD. *26th 2015 International Symposium on Micro-Nano Mechatronics and Human Science (MHS2015),* Nagoya, Japan, 2015.

[13] L. Vincent, F. Moreno-Noguer, and P. Fua, Epnp: An accurate o(n) solution to the pnp problem. *International Journal of Computer Vision*, 2008, 81(2):155-166.