Face Detection and Tracking Control with Omni Car

Jheng-Hao Chen, Tung-Yu Wu

CS 231A Final Report
June 31, 2016

Abstract

We present a combination of frontal and side face detection approach, using deep learning with Nvidia TX1 platform and an embedded GPU, providing the Omni robot with an efficient deep model of face detection with low computation and a provable detection accuracy for the robot's motion planning. The similar pipeline and framework is general and can also be applied to other object detection system for the robot usage. In addition, we also provide a new method to control the Omni robot equipped with four Mecanum wheels. By using orientation feedback and face's position estimation, the robot is able to follow and point to human's frontal face in any direction.

1. Introduction

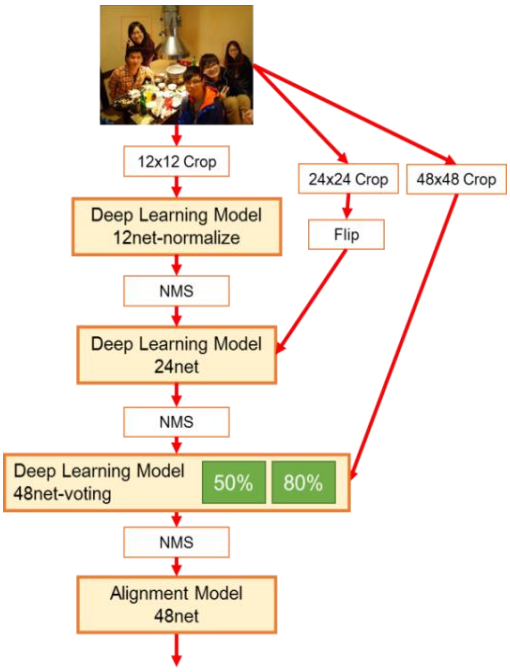
Nowadays, most robots are able to follow the moving object indoor with camera already set in the environment. Putting camera on the robot and track the moving object outside dynamically makes this problem hard to solve since the vibration of the robot and noisy estimation of the object's position in the real-time. We would like to propose a proper control algorithm to solve this problem. Tracing human face becomes much more difficult since robot needs to detect the human face on board and estimate the position of the face dynamically in the real time. When people tend to turn around with the side face, the existing algorithm for the face detection fails to recognize the face completely. As a result, we'd like to solve this problem with side face detection, estimation of the face position in the world coordinate to make the robot track face outdoor successfully.

2.1 Review of previous work

For the autonomous robot, it is important that the robot can recognize things by its own device without any other device apart from it. Running a recognition algorithm on an embedded system need a lot of trade-off. We need to consider both computation limitation and recognition accuracy. There are lots work which using opencv face recognition package for face detection and tracking on an embedded system. Even if the computational efficient is good, the accuracy is too poor to provide accurate information to the robot's control. It is also hard to detect the side face with opencv package. Recently, some researches indicates that deep neural network provides the high accuracy for face detection. Nevertheless, the computation resource cannot fit the requirement for the robot's tracking usage [1-3].

2.2 Contributions

We use deep learning approach for the robot detection. To overcome the computation efficiency, we use Nvidia TX1 platform and port the torch package and run our model on an embedded GPU. Furthermore, we use a cascade model [3] in our deep neural network to speed up the face detection. However, the performance for the pure cascade model is still not good for our control scenario. To overcome this problem, we use two kinds of cascade models and apply the boosting method to increase the accuracy. Furthermore, we try to combine the weights of two models ((1) Face labeled: true if the overlap between bounding box and the face >50% and (2) Face labeled: true if the overlap between bounding box and the face >80%) by transfer learning. It saves a lot of memory usage and computation loading on an embedded system. The following figure summarizes our deep architecture. By the above approach, we provide an efficient deep model of face detection model running on a computation limited platform and also provide a provable accuracy for the robot control. The similar pipeline and framework is general and can also be applied to other object recognition system for the robot usage.



3.1 Summary of the technical solution

3.1.1 Face Detection:

We provide a deep learning approach to train the model. With this ensemble method, we try to deliver higher accuracy to detect multi-face in the video. Collecting large amount of different angles of face, we can detect the faces in different angles by the deep learning model and port this model on a small footprint (embedded system) to make it run on real-time.

3.1.2 Position Estimation and Tracking Control:

Once we retrieve the face position in the camera coordinate, we are able to set a desired position of the human face relative to the camera coordinate. When the face is moving, resulting in a position and velocity deviation between the face and the desired face. With this information, we can use a linearization controller to design the proper control input to eliminate the tracking error. As a result, we can make the robot follow the human face successfully in the real time.

3.2 Technical Details

3.2.1 Omni Robot Hardware Design

1. **Mechanical Design**

The CAD design of the Omni Car with Solidworks 2015 is illustrated in Figure 3. The main material of our mechanical system is made of acrylic, and cut by laser-cutter in the PRL. A rotary encoder is attached on each wheel. The purchased 60mm Mecanum Wheel Set is mounted on the base plate to provide the agile movement in any direction.

2. Mechatronic Design

For the electronic part, we use 2 packs of L298N as the motor driver to control 4 DC motors with 12 volts input, and a 20 to 5 volts LM2596 digital voltage regulator to provide 5 volts to the Arduino mega micro controller. Each Jameco DC motor can provide 300g-cm torque under 12 volts. Adafruit BNO055 Absolute IMU Sensor is used to estimate the orientation of the robot for the navigation. We also used 11.1V LiPo Battery (3S 5000mAh 50C) as the power supply for the whole system. The Jetson TX1, Logitech HD Webcam C615 are mounted on the upper plate.

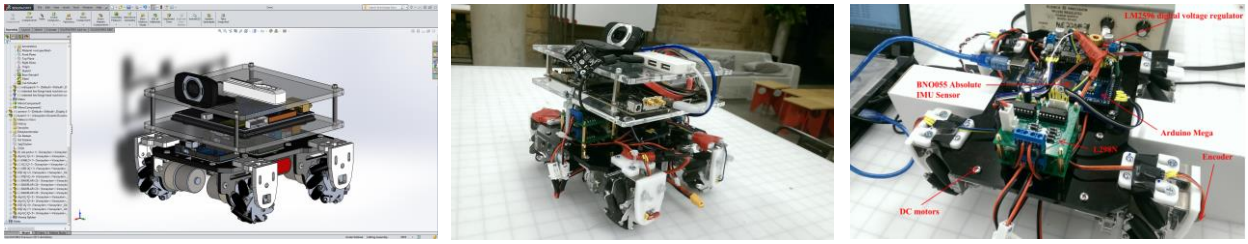


Figure 3. CAD model and Omni Robot Car

3.2.2 Control Strategy

1. Lower Level Individual Wheel Control

The lower-level control block diagram is shown in Figure 6. Given a desired speed of the wheels (RPM) and the wheel speed feedback from the encoder, the PID controller is able to compute the control input, which is represented in PWM signal to control both the direction and magnitude of the motor's rotational speed. We can formulate the control law as $F = k_p(\omega_d - \omega) + k_d(\alpha_d - \alpha) + k_i(\theta_d - \theta)$, where F is the PWM signal sent from the micro controller, ω_d is the desired rotational speed, and ω is the current rotational speed measured from the encoder. k_p, k_d, k_i are the proportional, derivate, integral gain respectively. Figure 6 shows how the speed of the four wheels influences in the movement of the robot.

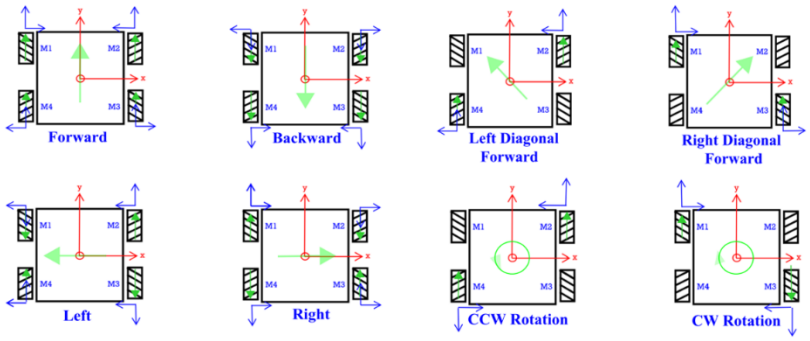


Figure 6. Individual Wheel Control Diagram

2. Higher Level Motion Control

The Mecanum wheel enhances the mobility of the conventional four wheels' car. The force from the wheel to the robot is at a 45 degree angle which is shown in Figure7, meaning that we could manipulate and vary the magnitude and direction of the force vectors to achieve the translation, rotational movement; therefore, the robot could move in any direction while keeping the head of the robot in a specified direction. The most different part between the Mecanum drive and conventional cart or tank drive is that it does not require the robot to turn the head of the robot to travel in another direction.

The robot dynamic analysis is important for our controller design. Based on the free body diagram analysis, we can formulate the Translational and rotational dynamics of the robot.

$$ {}^{L}\ddot{x} = \begin{bmatrix} \frac{(F_{1x} - F_{2x} + F_{3x} - F_{4x})}{M} \\ \frac{(F_{1y} - F_{2y} + F_{3y} - F_{4y})}{M} \\ 0 \end{bmatrix}, \qquad {}^{G}\ddot{x} = {}^{G}_{L}R\,{}^{L}\ddot{x} $$

$$ \ddot{\theta} = \frac{(-F_{1x} + F_{2x} + F_{3x} - F_{4x})I_{xx} + (-F_{1y} + F_{2y} + F_{3y} - F_{4y})I_{yy}}{I_{zz}} $$

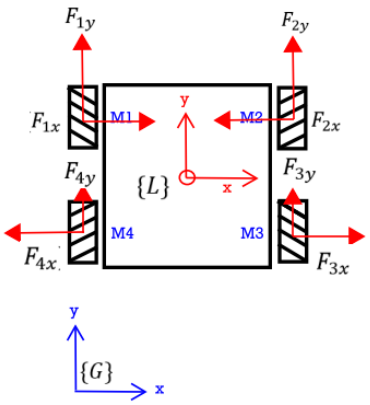


Figure 7: Free Body Diagram of the robot

,where F is the force vector in each wheel represented in local frame{L}, M is the mass of the robot, ${}^{L}\ddot{x}$ is the acceleration of the robot represented in local frame{L}, ${}^{G}\ddot{x}$ is the acceleration of the robot represented in global frame{G}, ${}^{G}_{L}\mathbf{R}$ is the rotation matrix that maps the local frame into global frame. I is the inertia of the robot and $\ddot{\boldsymbol{\theta}}$ is the angular acceleration of the robot.

The main idea of High Level Motion Control strategy is that we keep our face in the center of the image and remain a fixed distance between the robots by controlling the wheels simultaneously, the idea is shown in Figure 8. In addition, the robot keeps track of the human's face orientation so that it is able to point right in front of us.

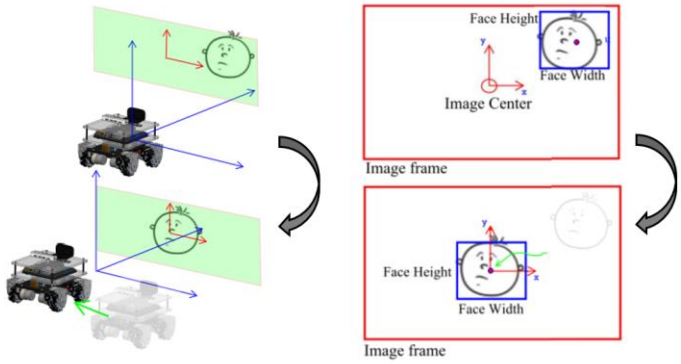


Figure 8: Control Strategy

Based on the idea above and the observation, we could find out that the robot's dynamics could be decoupled and we can formulate the control algorithm as

$$\begin{bmatrix} F_1 \\ F_2 \\ F_3 \\ F_4 \end{bmatrix} = \begin{bmatrix} V_d sin\left(\theta + \frac{\pi}{4}\right) - V_\theta \\ V_d cos\left(\theta + \frac{\pi}{4}\right) + V_\theta \\ V_d sin\left(\theta + \frac{\pi}{4}\right) + V_\theta \\ V_d cos\left(\theta + \frac{\pi}{4}\right) - V_\theta \end{bmatrix}$$

,where $\boldsymbol{F}$ is the force vector in each wheel, $\boldsymbol{V_d}$ is the robot's desired translational speed, $\boldsymbol{\theta}$ is the robot's desired translational angle and V_θ is a calibration factor that help the robot change its orientation to the desired angle. The $\boldsymbol{\theta}$ and $\boldsymbol{V_d}$ can be calculated by the following equations:

$$\begin{aligned} {}^G f &= {}^G_L R\, {}^L f = {}^G_L R \begin{pmatrix} f_x \\ f_y \\ 1 \end{pmatrix} \\ \theta &= atan2({}^G f_x, {}^G f_x) \\ V_d &= \sqrt{{}^G f_x{}^2 + {}^G f_x{}^2} \end{aligned}$$

,where $\boldsymbol{{}^L f}$ is the local total force vector exerted to the robot. We can use a simple transformation to map this force vector to the global coordinate by using the rotation matrix $\boldsymbol{{}^G_L R}$. The parameter of the rotation matrix can be computed based on the information of the orientation from the IMU sensor.

But how to we decide the control input, global vector $\boldsymbol{f_x}$ and $\boldsymbol{f_y}$ and the calibration factor $\boldsymbol{V_\theta}$. The idea is obvious and we could take advantage of the linearized control theorem. The problem can be simplified as a linear control problem since there is no unknown dynamic term involved, which means the only thing we need to control is the force vector. The larger deviation between the face's image and the center of the camera, the larger global force we need to provide to our system. The orientation control is based on the same idea mentioned above.

$$\begin{bmatrix} f_x \\ f_y \\ V_\theta \end{bmatrix} = K_p e + K_d \dot{e}$$

, where error vector and its derivatives are 3 by 1 vector and $\boldsymbol{K_p}$ and $\boldsymbol{K_d}$ is 3 by 3 positive definite matrix. The diagonal term is related to the x, y and orientation proportional/ derivative gain respectively. $\boldsymbol{p_d}$ represents the desired position in the image and the desired orientation we specify, $\boldsymbol{p}$ denotes the current face's center in the image and the current orientation of our face.

$$e = p_d - p = \begin{bmatrix} 0 \\ 0 \\ \theta_d \end{bmatrix} - \begin{bmatrix} {}^c p_x \\ {}^c p_y \\ \theta \end{bmatrix}, \qquad \dot{e} = \dot{p}_d - \dot{p} = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix} - \begin{bmatrix} {}^c \dot{p}_x \\ {}^c \dot{p}_y \\ \theta \end{bmatrix}$$

$$K_p = \begin{bmatrix} K_{px} & 0 & 0 \\ 0 & K_{py} & 0 \\ 0 & 0 & K_{p\theta} \end{bmatrix}, K_d = \begin{bmatrix} K_{dx} & 0 & 0 \\ 0 & K_{dy} & 0 \\ 0 & 0 & K_{d\theta} \end{bmatrix}$$

3.2.3 Control Block Diagram

The high level control block diagram is shown in Figure 9. The Arduino Mega Controller is able to calculate the desired speed of each wheel as our control input to the low-level controller. The IMU sensor is used as an orientation estimator to provide the controller with the orientation and angular velocity feedback. Jetson TX1 runs the face detection algorithm and estimates the position of the human's face, used as the position feedback for the robot.

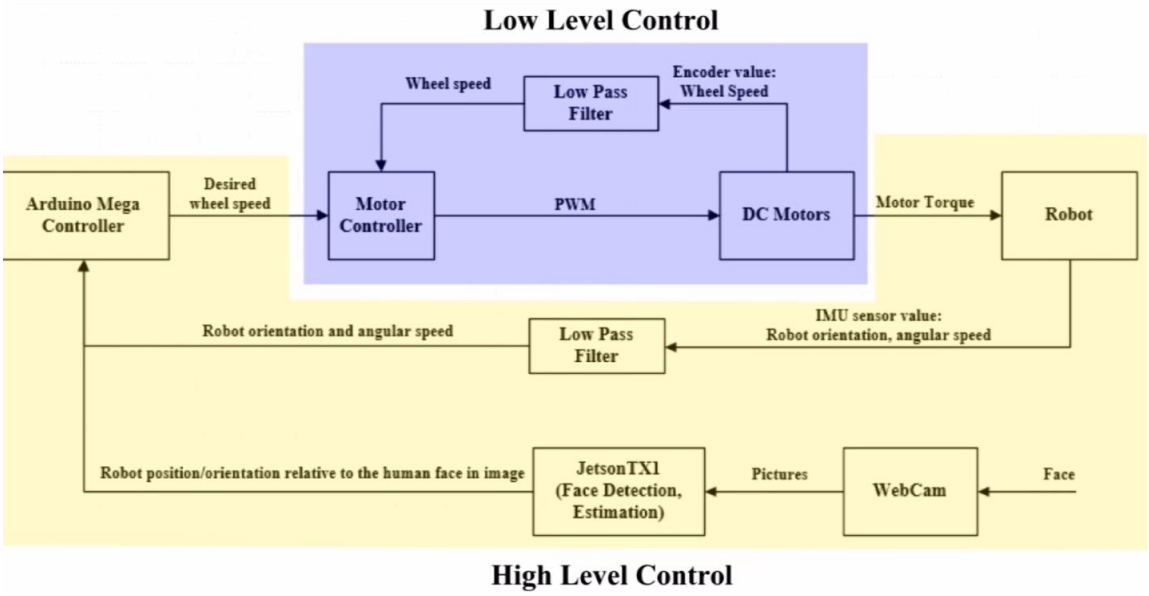


Figure 9: Control Block Diagram

3.2.4 Face Detection

1. **Dataset Preparation**
   - Dataset

   Annotated Facial Landmarks in the Wild [5]
   - Data Description

     25k annotated faces, a wide range of natural face poses is captured the database is not limited to frontal or near frontal faces.

	Face	NonFace
Training Set	90%	
	25%	75%
Testing Set	10%	
	25%	75%

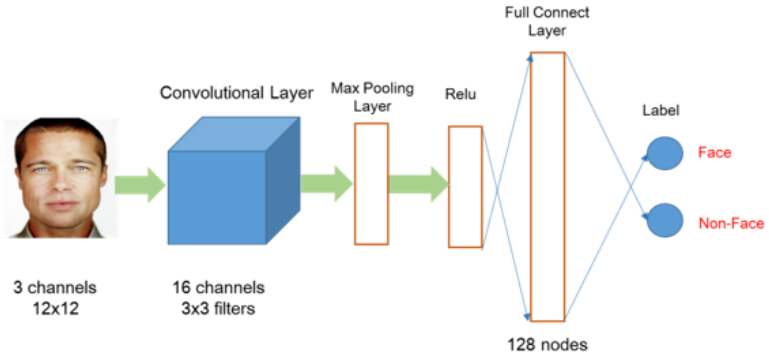
Table 1: Positive & Negative Set

   - Face Cutting

     Crop out the face pictures or non-face pictures from AFWL dataset.
     We revise the code from [2].
   - Face Cutting Steps
     1. Read one image and coordinates of bounding boxes.
     2. Random shift around bounding box. (Shift Range: 0~ face width)

3. Compute the IOU with original bounding box.
4. If IOU is larger than threshold (Some Percentage), save the picture to the folder “face”.
5. If IOU is smaller than threshold, save the picture to the folder “non-face”.

2. CNN Model



3. Learning Phase

● Architecture

We will generate two training labeled data set from AFLW [5]. One is that we labeled the face if the overlap between bounding box and the ground truth (face) > 50%. The other one is we labeled the face if the overlap between bounding box and the ground truth (face) > 80%. We will not train two separated models for these two data sets for our boosting approach due to the weights in these two model sharing similar weights (filters). We combine two models in our architecture. That is, the two model share the layers (convolutional layer, pooling, ReLu etc) before the fully connected layers in 12-net, 24-net and 48-net respectively. In the final 48-net layer, we use a boosting algorithm to decide if a patch in the image is a face or not. This sharing architecture reduces the memory usage (weights) and increase the utilization of GPU in the platform.

● Dropout

Due to the limited amount of the data, we need to prevent the over-fitting issue in our training phase. We apply the dropout method [7] to achieve this. This approach is similar to cross-validation in machine learning. The following table shows the results.

Accuracy	Training	Testing
Without Dropout	97.2%	90.7%
With Dropout	96%	92.9%

● Batch Normalization Layer

Training a CNN model always takes time even using a powerful GPU. We apply the batch normalization layer [8] to speed up the convergence rate in the training phase. However, we need to remove this layer when we apply the trained model in testing phase. This is because this layer will slow down the testing time. The following is the equations which are used to adjust the weights after removing the batch normalization layers where x, y and w are input patch, output patch and the filer respectively. We have added this feature in our torch platform for automatically removing this layer.

$$y = \frac{x - mean(x)}{Std(x)} \times gamma + beta$$

- Training: $mean(x)$ is the mean of the batch.
  $Std(x)$ is the std of the batch.
- Testing: $mean(x)$ is the mean of the whole training dataset.
  $Std(x)$ is the std of the whole training dataset.

$$x = w \times x' + bias \rightarrow \quad y = \frac{(w \times x' + bias) - mean}{Std} \times gamma + beta$$

$$\rightarrow y = wx' \times \frac{gamma}{Std} + (bias - mean) \times \frac{gamma}{Std} + beta$$

$$\rightarrow y = w \times \frac{gamma}{Std} \times x' + (bias - mean) \times \frac{gamma}{Std} + beta$$

- Sliding Window in GPU

GPU is a SIMD architecture. If we apply the sliding window method in GPU, it always takes time and is hard to meet the real-time requirement. It also will cause the utilization of GPU lower. To solve this problem, we combine all the windows as an input image and create a bigger model by repeating our CNN model. This idea is inspired by YOLO [9] which a grid-base CNN model. We also share the weights to save the cache and memory usage since we copy them from the same model. This part can be done because of the flexibility of Torch.

4.1 Control Simulation Results

The simulations result shows the main idea of the control strategy, which is shown in Figure 10. The Blue bars represent human's face with a certain orientation. The green bar indicates the orthogonal line between the face's position and its projected to the camera. The blue square represents the Omni robot car with four wheels. And the dash blue line and red line represents the human's face trajectory and robot's trajectory respectively.

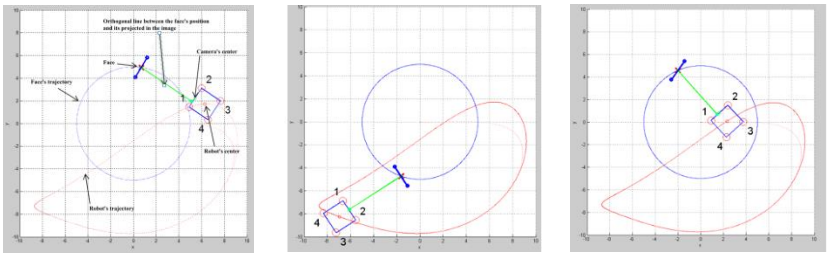


Figure 10: 2D Simulation results

Figure 11 shows that with a proper proportional and derivative gain about 40 and 20 for the orientation tracking, the robot is able to follow both human's face position and orientation successfully. The distance between the robot is set as 150 cm. Due to the nonlinearity of dynamics and coupling effect, there is a little fluctuation around the desired point. Figure 12 shows the robot without orientation controller; therefore, the robot can only do the position tracking. Figure 13 illustrates that if we begin to increase the proportional and derivate gain, the robot starts to point to the human's frontal face with specified orientation.

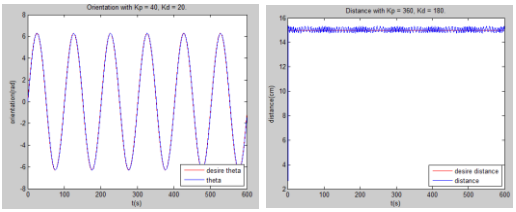


Figure 11: Orientation and Position Controller

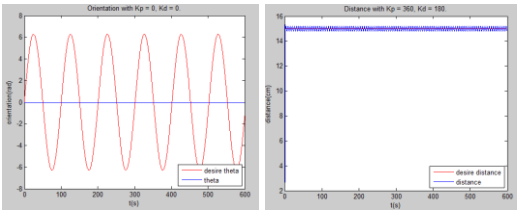


Figure 12: Position Controller

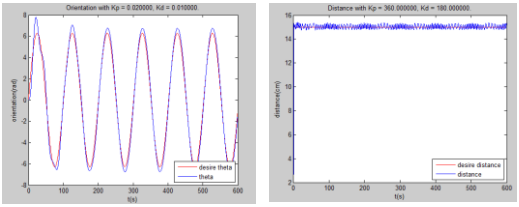


Figure 13: Orientation and Position Controller with lower gain

4.2 Face Detection Experiments and Results

Figure 14 shows the Face detection results with different side and frontal faces detection. Table2 illustrates the computation and memory usage on the Embedded System (Jetson TX1). Table3 indicates the accuracy of the training and testing set for CNN model.

Speed	20 fps (800*600 resolution from the camera)
Memory Usage	2GB

Table 2: Computation and Memory Usage on the Embedded System

Accuracy	**Training**	**Testing**
Boosting	96%	95%
Model (50% overlap data)	94%	92.9%
Model (80% overlap data)	93.2%	92.5%

Table 3: CNN

Figure 14: Face Detection Results

5. Conclusions

In this work, we provided a compact deep neural model for face detection. It can be executed on an embedded platform for the autonomous robot's control. The high accuracy and performance of this model can fit the real-time computational requirement. This framework can be also applied to general object detection by changing the training data for autonomous robots. The tracking control is able to make the robot do the motion planning and point to the human's frontal face in the real time even if people are trying to rotate their faces while moving.

6. References

[1] S. S. Farfade, Md. Saberian and Li-Jia Li, "Multi-view Face Detection Using Deep Convolutional Neural Networks," International Conference on Multimedia Retrieval (ICMR), 2015

[2] S. Yang, P. Luo, C. C. Loy, and X. Tang. From facial parts responses to face detection: A deep learning approach. ICCV, 2015

[3] Omkar M Parkhi, Andrea Vedaldi, and Andrew Zisserman, "Deep face recognition," Proceedings of the British Machine Vision, 2015.

[4] H. Li, Z. Lin, X. Shen, J. Brandt, and G. Hua. A convolutional neural network cascade for face detection. In CVPR, 2015.

[5] https://lrs.icg.tugraz.at/research/aflw/

[6] https://github.com/guoyilin/FaceDetection_CNN/blob/master/image_preprocess.py

7. Video Links:

[1] Tracking Control with Face detection Demo: https://www.youtube.com/watch?v=x1YkVRYlOAw

[2] Tracking Control Simulation with MATLAB: https://www.youtube.com/watch?v=RfqmdIC9LWk

8. Code Links:

We train the model by Torch and use opencv to get image frames from camera.
A robot control unit is also programmed by us.
https://drive.google.com/file/d/0B2PnxbFvd8Tda3UtYWt3d3IyQk0/view?usp=sharing