EECS 442 Final Project Report

Chunkai Yao   Chang Xu   Runyu Gao
University of Michigan

{nickyao, changx, runyugao}@umich.edu

Abstract

This report is an overall summary of our final project. We use the concepts in computer vision (specifically, image recognition) and robotics to implement a self-moving robot that is able to take the best pictures of a designated model. In order to find the best picture, we used open-source Python library ImageAI to find the target in the scene, depth estimation to locate the position of the target, A star search algorithm to generate the path for motion simulation around the target and face pose estimation to calculate facial orientation. We used python, Dlib library, ResNet50 convolutional network and pretrained models to implement the computer vision and image processing part of the program and we use C++ and Unity for motion control and simulation. Our distance estimation calculated by the program is very close to the actual simulation distance. We are able to take good frontal pictures of the model and we have also observed a linear relationship between offset values and robot orientation.

1 Introduction

Currently there are some selfie robots that enable people to take selfies, but it requires the model to do some manual operation in order to take the pictures, and most of them cannot actively move and search for the model. Our project aims to develop a program that makes a robot to actively move and find a designated model and take an optimal picture of the model. To achieve this, we use and implement the knowledge in computer vision. We have also built a robot model and run many simulations. Our project is divided into three chunks: person detection, distance estimation and face pose estimation. Person detection differentiates the person we want to take the picture of from the environment setting and marks the person with a rectangle. Distance estimation figures out the distance required between the robot and the model in order to take an optimal (defined in the later part) picture of the model. Face pose estimation calculates the angle between the target person's face and the robot to control the orientation of the model in the picture taken. More details and our progress on each part will be discussed in more detail in the later sections.

2 Person Detection

In our project, the first step is to find the person we want to take pictures of in the environment setting. Due to the limitation of frame size, the camera mounted on our robot may not be able to capture the person at the beginning. In addition, there may be other objects and obstacles that are distracting. Therefore, we are working on developing a strategy to locate our target and aim the camera at that person. In order to do that, we rely on computer version technology to help us detect people in our first-person-view camera.

We find an open-source Python library named ImageAI [1], which can be used to classify objects in images and videos. During testing, we find out that ImageAI is able to correctly detect the person, even if that person is not facing the camera. Using ImageAI, we are able to locate the person we want to take pictures of and mark her with a rectangle.

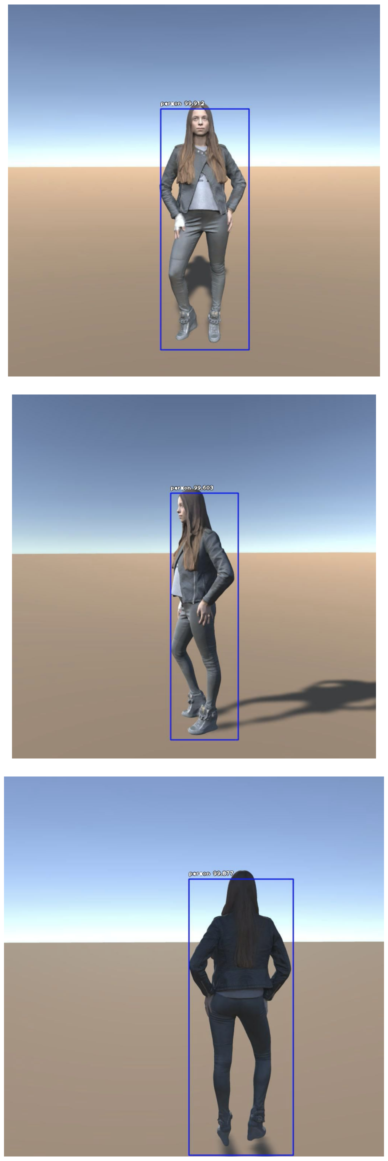


Figure 1: Persona Detection from Different Angles

For the model, we are using ResNet50 [2]. It is a convolutional neural network that is 50 layers deep. Using this model, we can load a pre-trained version of the network trained on more than a million images from the ImageNet database. The pretrained network can classify images into 1000 object categories, such as keyboard, mouse, pencil, and many animals.

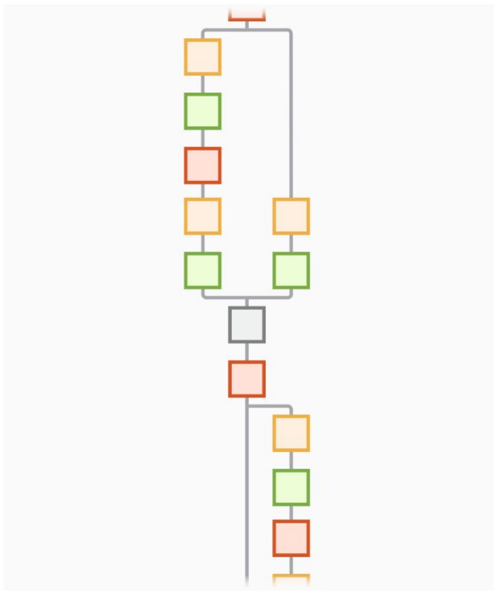
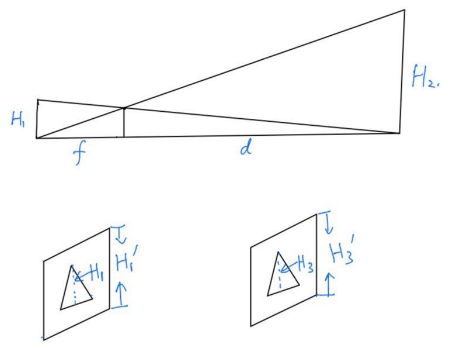


Figure 2: ResNet50 Convolutional Neural Network

3 Distance Estimation

In the previous part, we are able to locate the person in images and we get a rectangle indicating the position information of that person. With information of camera settings, we are able to estimate the distance between that person and camera, which can be referred to as a parameter to adjust the position of the robot.

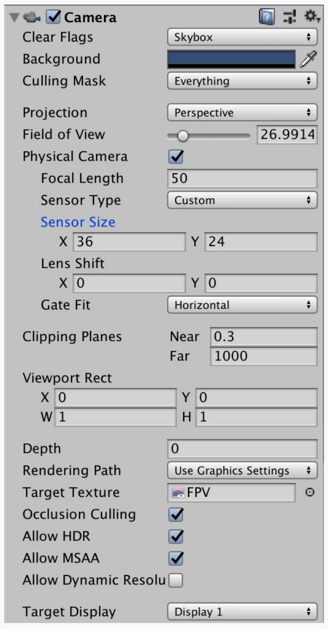
The estimation is based on the following idea. From the FPV camera, we are able to determine the proportion of the person within the frame. Using this proportion, we can figure out the height of the person image on the CMOS sensor by referring to the sensor size. Further, knowing the rough height of the person and the focal length, we are able to use similarity transformation to calculate the distance between the person and the camera.



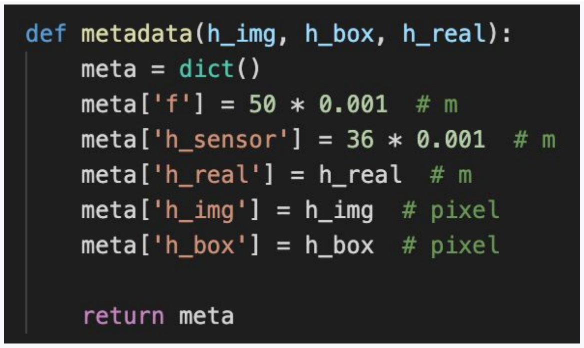
H1: height of the object on sensor (m)
H2: actual height of the object (m)
H1': height of the sensor (m)
H3: height of object in the image (pixels)
H3': height of the image (pixels)
f: focal length of the sensor
d: distance (depth) from the object to the sensor

Figure 3: Depth Estimation

In Unity, we are able to simulate physical cameras as well as adjusting camera parameters, which provides us more flexibility in testing.



Camera	✓
Clear Flags	Skybox
Background	
Culling Mask	Everything
Projection	Perspective
Field of View	26.9914
Physical Camera	✓
Focal Length	50
Sensor Type	Custom
Sensor Size	X 36 Y 24
Lens Shift	X 0 Y 0
Gate Fit	Horizontal
Clipping Planes	Near 0.3 Far 1000
Viewport Rect	X 0 Y 0 W 1 H 1
Depth	0
Rendering Path	Use Graphics Settings
Target Texture	FPV
Occlusion Culling	✓
Allow HDR	✓
Allow MSAA	✓
Allow Dynamic Resolu	
Target Display	Display 1



```python
def metadata(h_img, h_box, h_real):
    meta = dict()
    meta['f'] = 50 * 0.001  # m
    meta['h_sensor'] = 36 * 0.001  # m
    meta['h_real'] = h_real  # m
    meta['h_img'] = h_img  # pixel
    meta['h_box'] = h_box  # pixel

    return meta
```

Figure 4: Camera Settings in Unity and Python Script

During testing, we set the distance between the person and the camera to 8 m, and the result of our estimation is 8.011251758087202 m. The estimated distance and the actual distance are pretty close.

4 Face Pose Estimation

The pose of an object refers to the object's orientation and position with respect to a camera. Pose can be changed by moving the object with respect to the camera or moving the camera with respect to the object. In order to get the face pose estimation, we first need to extract facial feature landmarks.

To detect facial features, we use the model shape_predictor_68_face_landmarks.dat from the Dlib library [3], a library capable of giving us 68 points (landmarks) of the face with pre-trained models. In our program, we detect these 68 feature points (landmarks) from the picture we took of the model's face, and display these landmarks.

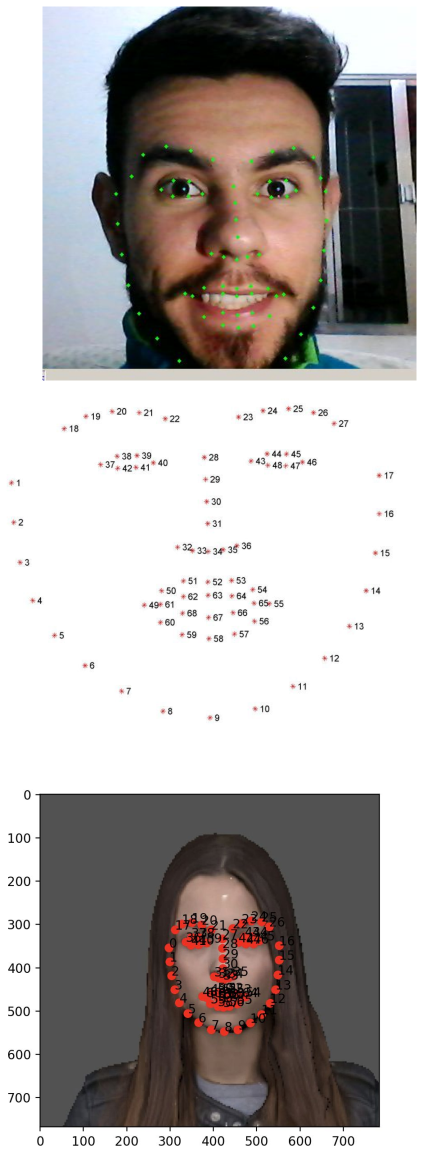


Figure 5: Example of Facial Feature Detector Output

Our goal is now to find the pose of the model, and we know the locations of the 3D points on the object and the corresponding 2D projections in the image as they are detected as landmarks earlier. Thus, we need to calculate the translation and rotation in order to represent the camera motion.

Our implementation is based on a paper "Optimum Fiducials Under Weak Perspective Projection" [4] and the idea we learned from an example implemented by others[5]. The paper investigates how a given fixed number of points should be located in space so that the pose of a camera viewing them from unknown locations can be estimated with the greatest accuracy. The conclusion shows that optimum solutions are obtained when the points form concentric complete regular polyhedra. In our implementation, we designate where we place several feature points Pi = [Xi Yi Zi ….]^ T in the environment and minus the value of all points to calculate an arbitrary rotation R and translation T. We are hence able to calculate the angle of the person's face orientation.

5 Simulation and Evaluation

We decided to use Unity platform to simulate our project. We develop the testing scenario. We use the 3D model of a girl as our target person. In the bottom-left side of the screen, we include the real-time image from FPV camera. During the simulation, we process the image from the FPV camera to generate parameters to control the robot. The simulation is divided into two parts: locating the target and moving around the target to get optimum position.

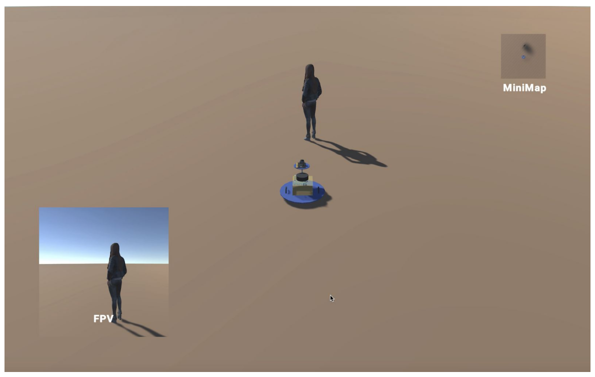


Figure 6: Screenshot of Simulation

For the first part, the main goal is to rotate the robot to face the target person and generate the estimated position of that person. The original plan was to constantly rotate the robot and process the real-time FPV camera video to check if the target person appears in the center of the camera frame. However, due to the limitation of CPU capability, we process one image every time the robot rotates 13 degrees. Once we detect the person in the image, we calculate the offset between human center and image center.

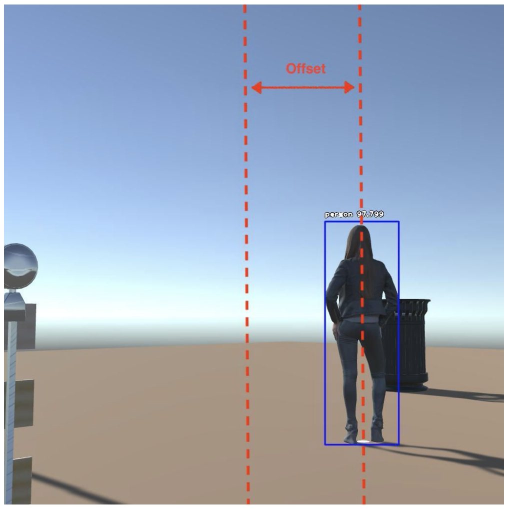


Figure 7: Offset between Centers

During the experimentation, we set the person at position (0, 0, 0) and robot at (-5.51, 0, -0.3) and rotate the robot. For the images in which the target can be found, we record the offset values and orientation values. We figure out that there is a linear relationship between that offset value and robot orientation. For the tables below, our prediction of the angle facing the target is 87.144 and theoretical value is arctan(5.51/0.3) = 86.884. The predicted value is close to the actual value.

Offset Val	Angle
451	69
431	72
303.5	75
223.5	78
145.5	81
68	84
-4	87
-78	90
-151.5	93
-230	96
-308.5	99
-387.5	102
-415.5	105

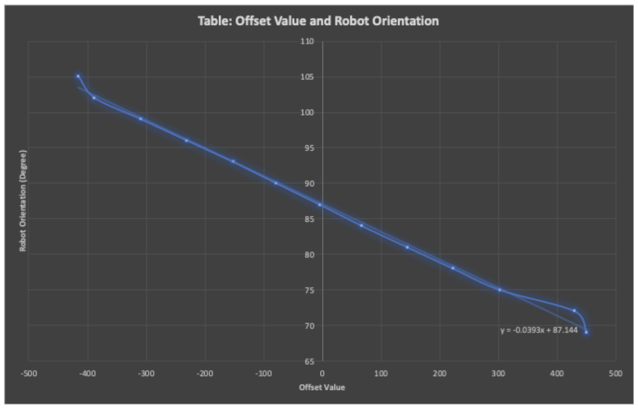
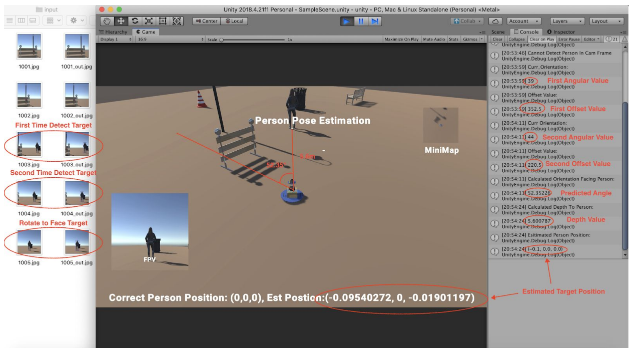


Figure 8: The Relationship Between Offset Values and Robot Orientation

In the practice, we need only two sets of offset and orientation values to predict the optimum angle for the robot to face the target. Once we detect the target in the image, we record the values and rotate another 5 degrees to get another set of values. Then, we use these two sets of values to predict the optimum angle to face the target. After the camera faces the target, we call depthEstimation script to calculate the distance between the target person and the robot. Using the distance, angle and robot current position information, we are able to predict the position of the target person. In the testing, we move the robot to another position (-4.53, 0, -3.44) and the estimated position of the person is (-0.095, 0, -0.019), which is very close to the actual position (0, 0, 0).



```
[20:53:46] Cannot Detect Person In Cam Frame
UnityEngine.Debug:Log(Object)
[20:53:59] Curr_Orientation:
UnityEngine.Debug:Log(Object)
[20:53:59] 39
UnityEngine.Debug:Log(Object)
[20:53:59] Offset Value:
UnityEngine.Debug:Log(Object)
[20:53:59] 352.5
UnityEngine.Debug:Log(Object)
[20:54:11] Curr Orientation:
UnityEngine.Debug:Log(Object)
[20:54:11] 44
UnityEngine.Debug:Log(Object)
[20:54:11] Offset Value:
UnityEngine.Debug:Log(Object)
[20:54:11] 220.5
UnityEngine.Debug:Log(Object)
[20:54:11] Calculated Orientation Facing Person:
UnityEngine.Debug:Log(Object)
[20:54:11] 52.35226
UnityEngine.Debug:Log(Object)
[20:54:24] Calculated Depth To Person:
UnityEngine.Debug:Log(Object)
[20:54:24] 5.600787
UnityEngine.Debug:Log(Object)
[20:54:24] Esitimated Person Position:
UnityEngine.Debug:Log(Object)
[20:54:24] (-0.1, 0.0, 0.0)
UnityEngine.Debug:Log(Object)
```

Figure 9: Position Estimation

The second part of the simulation is to move around the target to find the best position to take pictures. This part can be divided into two parts: path generation and navigation. We use A Star algorithm to check grids in the map to find the path between starting point and ending point.

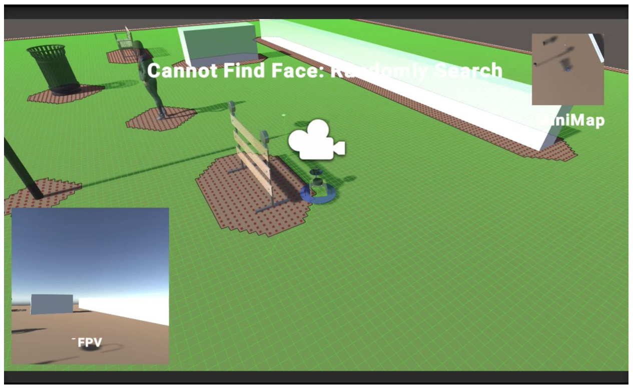


Figure 10: A Star Algorithm to Generate Path

In the previous part, we estimate the position of the target person. The basic idea of navigation is to first navigate the robot to a point near the target and then rotate the robot to face the target and take pictures. If the human face is not detected in the image, the next point will be generated by rotating the current position of the robot around the target by 30°. The robot repeats this process until it detects the human face and generates human face pose estimation value. Then, we let the robot rotate around the target by that angle and take pictures again. Once the absolute value of angular value sent from the robot is less than 9 degrees, we consider that the robot is at the position where it can take the best picture.

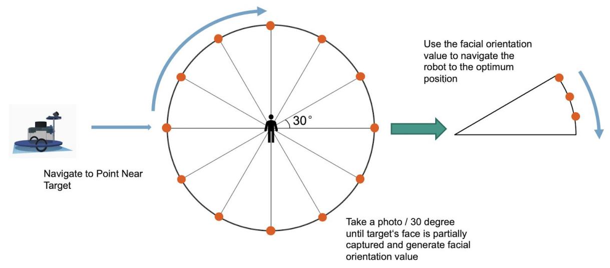

Navigate to Point Near Target
30°
Take a photo / 30 degree until target's face is partially captured and generate facial orientation value
Use the facial orientation value to navigate the robot to the optimum position

Figure 11: Idea of Navigation

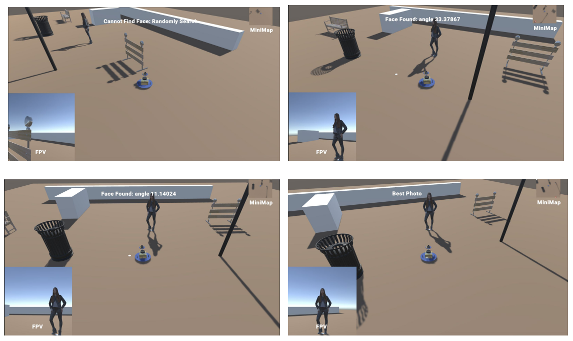

Cannot Find Face: Randomly Search
Face Found: angle 33.37867
Face Found: angle 11.14024
Best Photo
MiniMap
FPV

Figure 12: Different Phases of Navigation

6 Video

Link

7 Conclusion

We developed a self-moving robot that is able to differentiate a target model from the environment setting, actively move to the front of the model and figure out the best position by angles and distance to take the best picture of the model. We use Python Dlib library and ImageAI open-source library, ResNet50 convolutional neural network to facilitate our implementation. With our implementation, we record the offset values and the orientation values in images in which the target model can be found, and we have observed a linear relationship between that offset value and robot orientation. Based on the stimulation result, the algorithm we developed is quite effective in getting an optimal picture of the model. However, there is something we need to improve in the future. When we are calling ImageAI to find the target person and when we are doing depth estimation, our program written in C++ calls another program in Python, which makes the model load again every time the call happens. For every step the robot moves, we need to recalculate and reconstruct the path to the target. This is a huge amount of calculation and takes a great portion of CPU usage, making the simulation process slow and not smooth. For this reason, the image processing of the pictures taken by our robot cannot be achieved in real-time. In order to improve this problem, we will try different models and buy a better computer.

References

[1] M. Olafenwa, “ImageAI,” 14-Jan-2020. [Online]. Available: https://github.com/OlafenwaMoses/ImageAI.

[2] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep Residual Learning for Image Recognition. In *ICPR,* 2015.

[3] Davis E. King. Dlib-ml: A Machine Learning Toolkit. Northrop Grumman ES, ATR and Image Exploitation Group in 2009.

[4] Alfred M. Bruckstein, Robert J. Holt Bell, Thomas S. Huang, and Arun N. Netravali. Optimum Fiducials Under Weak Perspective Projection. In *CVPR*, 1999.

[5] J. How, “Face Yaw Roll Pitch from Pose Estimation using OpenCV,” 12-Dec-2018. [Online]. Available: https://github.com/jerryhouuu/Face-Yaw-Roll-Pitch-from-Pose-Estimation-using-OpenCV