A Novel Cyclist Safety Aide Using Deep Neural Net of Computer Vision to Identify Bike Obstacles

< Master Graduation Thesis >

Student ID: S20015 Student Name: Lingzhi_Xie
Tutorial Professor: Ryosuke_Okuda
Submission Date: August 2, 2022

神戸情報大学院大学/情報技術研究科/情報システム専攻
Kobe Institute of Computing / Graduate School of Information Technology

Table of contents

Manifest of illustration figures

List of explication tables

Abstract

Student ID	20015	Name	Lingzhi Xie
Title of Thesis	A Novel Cyclist Safety Aide Using Deep Neural Net of Computer Vision to Identify Bike Obstacles		

Abstract of Thesis

Bike riding safety is an interesting topic attracting some researchers' attention. In some theses, papers, and product developments, researchers or developers introduced a number of cycling safety improvement methods which remind cyclists of 1~2 kinds of road dangers incorporated in the scope of abrupt roadbed bumps and dents, overtaking vehicles, relatively approaching objects coming from multiple directions including vehicles, people and even animals, and other types' road obstacles, relying on the use of technological equipment such as radar, lidar laser, ultrasonic wave, or camera plus computer vision. And in all the mentioned methods, camera plus computer vision has been quite a new one and has been not deeply explored from the perspective of the author of this thesis. Hence, he chose the methodology of deep neural net of computer vision for his research on facilitating bike riding safety.

In his research the author established a prototype which integrated small electronics like RaspberryPi_4B, usb_camera, and neural net accelerator, and deployed the computer vision technology of Yolo3_tiny onto his prototype bicycle. During prototype experiments, the usb_camera was mounted on bike basket for capturing real-time images from front view angle, while the RaspberryPi_4B processed captured images using deep neural net for vision analysis and in turn generated alarm to indicate highly weighted dangers above risk weights threshold. To simplify the prototype experiment and to circumvent the issue of insufficient recognition accuracies on types of curbs and ditches deriving from the earlier stage of Yolo3-tiny DNN model training, the author only concentrated on the danger type of electricity pole for final proposal evaluation using function-minimized prototype machine.

The later evaluation work proved that, the author's proposed solution is effective to urge cyclists to early perceive inconspicuous obstacles and suddenly emerging obstacles in front, and it also can alleviate cyclists' burden of having to watch front side and inner lateral-to-rear side simultaneously, and tellingly notify cyclists to avoid unexpected front looming obstacles, on the occasion when cyclists are making big rotation steering.

1. Research introduction

1.1 Overview

This study was to explore the methodology which was using deep neural net of computer vision to identify and classify road dangers for the ultimate purpose to continuously remind cyclists of avoiding bike obstacles in real time. The author probed into the research background, problem analysis, research objectives, and a proposed solution compared with existing solutions. Subsequently, a prototype machine was developed for in-field testing to evaluate the proposed solution's effectiveness. And the post-evaluation conclusions were given at the end.

1.2 Background

One day in a sunny noon when the author was on his way to his Japanese language academy in Fukuyama city, Japan, in 2019, he saw an elder lady who looked around 50 to 60 years old riding a bicycle crashed her face to an electricity pole and got badly injured. The accident cause was that the elder lady didn't notice the inconspicuous electricity pole camouflaging itself in the backdrop of environment. And this accident turned to be the author's cause to start his research explicated in this thesis.

According to the statistic data from the Traffic Bureau of National Police Agency of Japan in figure 1, about 200 traffic accidents involved cyclists every day from 2016 to 2020 on average, and some of the cyclists got victimized. And in many bicycle accidents, cyclists were found not having taken correct contingency measure ahead of time.

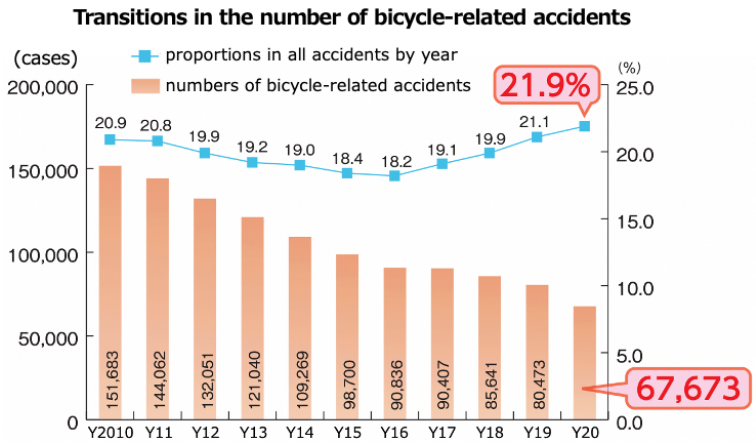


Figure 1: Bicycle-related accident numbers in Japan by year

On the other side, autonomous automobile driving systems have been successfully developed by technology giants like Waymo and Tesla, and with further R&D investments in the future they will adapt to commercialization standards better. And the author's idea was to imitate part of the conception of autonomous

automobile driving system to facilitate bicycle riding safety using the computer vision technology of Yolo3_tiny.

Although modern people live in a high-technology era, ordinary bike riders still rely fully on humans' physiological functions to handle risky contingencies in most situations. However, compared to computer computations humans' physiological reactions move at lower speed and with lower reliability. So the author's consideration was that, to facilitate bike riding safety for plain riders especially for elder people with physiological degeneration, introducing deep neural net of computer vision on obstacles detection for cyclists should be worth of experimentation. As figure 2 illustrates, cyclists surely can perceive road danger with their own eyes and ears. But in the cases of distraction, insufficient safety confirmation, fuzzy sight in poor illumination conditions, etc., the mobile app of cycling safety aide can help detect road dangers and urge cyclists to take necessary reactions for self-protection.

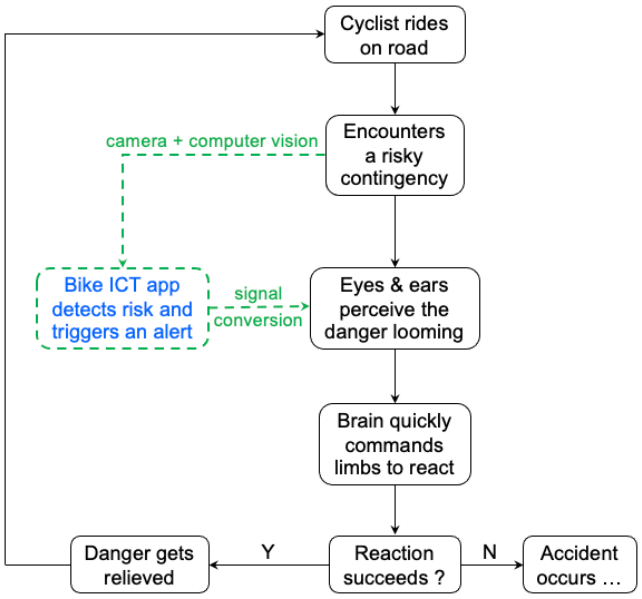


Figure 2: Application frame of cycling safety aide using computer vision

2. Problem analysis

2.1 Problem profile

When people ride bicycles on roads, their brains must rapidly process the information of ephemeral road conditions. Of course, bicycle accidents could be caused by many reasons in various traffic conditions. But the research in this thesis only took the perspective to inspect what kinds of mistakes cyclists commit in the way of initiating bicycle accidents, and with what ICT methodologies' help they could do better to avoid accidents. For example, on some occasions, cyclists give rise to accidents because of their own negligence or misjudgment. The factors such as mental issues, inadvertently violating traffic rules, inconspicuous obstacles neglected, rear overtaking vehicles unnoticed, etc., can contribute to potential risks to cyclists. And the purpose of this problem analysis was to figure out which road risks with what methodologies the author's research could facilitate cyclists to effectively circumvent to a considerable extent.

2.2 Problem tree

Even though this research took the angle from only the bicycle accidents initiated by cyclists themselves for problem analysis, the structure and venation of the whole problem hierarchy don't appear to be simple. As the

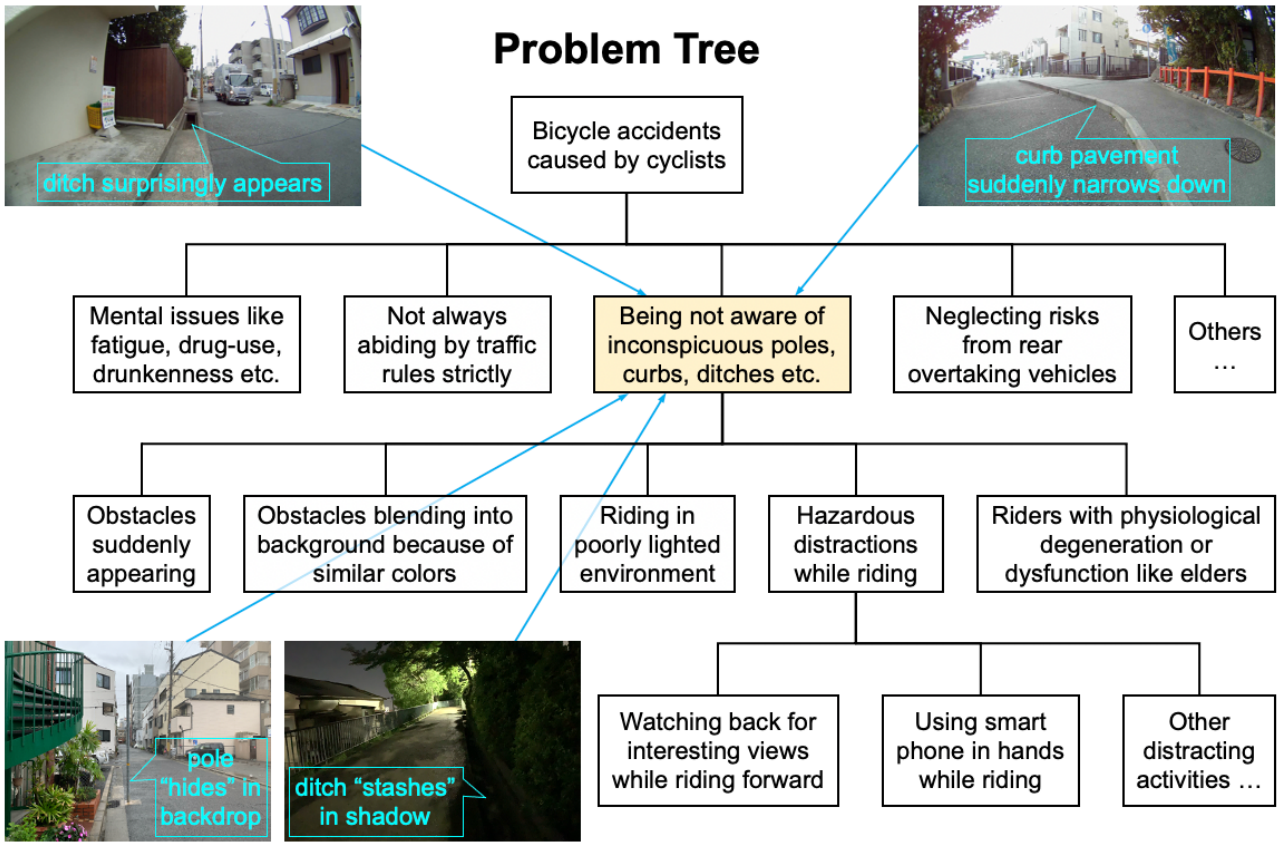


Figure 3: Problem tree of accidents for which cyclists are primarily responsible

problem tree in figure 3 shows us, in the first downward stratum, the major constituents were generalized as reason 1 -- mental issues like fatigue, drug use, drunkenness, etc., reason 2 -- not always abiding by traffic rules strictly, reason 3 -- being not aware of inconspicuous poles, curbs, ditches, etc., and reason 4 -- neglecting risks from rear overtaking vehicles. Since reasons 1 & 2 basically result from cyclists' individual behaviors which obviously can't be corrected by the author's research work, unnecessary downward elaborations for these 2 reasons were ruled out at first. And because the problem of inspecting rear overtaking vehicles had been effectively resolved by other existing solutions which will be illustrated in chapter 3, plus two-dimensional computer vision technologies have bottlenecks for detailed tasks like measuring relative speed and determining obstacles' instantaneous positions on purpose of detecting risky overtaking vehicles, reason 4 was excluded as well for further breaking down. Therefore, the remaining and critical constituent in the first downward stratum was reason 3 saying "being not aware of inconspicuous poles, curbs, ditches etc." that required further anatomy.

The 4 scene photos in figure 3 illustrate sectional situations that electricity poles, curbs, and ditches could quietly set up cyclists in jeopardy when they don't look conspicuous out of background. Though the scenes in the 4 photos are typical somehow, they don't exhaustively explain how the inconspicuous road dangers might hazard cyclists. As the second downward stratum describes, these dangers were fully enumerated as factor 1 -- obstacles suddenly appearing, factor 2 -- obstacles blending into background because of similar colors, factor 3 -- riding in poorly lighted environment, factor 4 -- hazardous distractions while riding, and factor 5 -- riders with physiological degeneration or dysfunction like elders. Furthermore, the factor 4 was broken down in the bottom stratum into 2 major kinds of behaviors often seen in our daily life, which are watching back for interesting views while riding forward and using smart phone in hands while riding.

2.3 Target problem

As mentioned once in explaining the problem tree in figure 3, only the issue class of "being not aware of inconspicuous poles, curbs and ditches" was highlighted by the author as the target problem in this research based on technology limitations and existing solutions. From the watching angle of cyclists, there are probably many types of obstacles may bring risks to them. To simply problem generalization for purpose of exploring possible solutions for the target problem class, the author picked out 3 typical problem scenarios illustrated in figures 4, 5 and 6 as specified target problems.

In figure 4, neither the abrupt ditch nor the suddenly shape-changing curb is regarded as perils for cyclists in general situations. But in the case of distracted cyclists, they might turn out to be harmful to bike riders. For instance, if a cyclist is watching back for being attracted by a fair view or a pretty lady while he didn't notice his bicycle is approaching an abrupt ditch or a longitudinal curb with a dramatic shape change intersects his cycling route, he may get badly bumped by the ditch or the curb. Real accidents that cyclists got injured for having fallen into ditches or bumped with curbs can be easily searched out on some websites of Japanese news.

Figure 4: Suddenly showing-up obstacles could bump back-watching cyclists

In figure 5, the slim pole blending into background for color resemblance which stands protrusive in bicycle lane can vehemently crash reckless cyclists. The author did some in-field investigations in Japan about this. Many electricity poles have fluorescent marker plates with yellow-black stripes attached to the lower-middle of pole bodies in Japan. The fluorescent markers act a good job to remind vehicles coming and going to notice the existence of those poles, which avoids many vehicle-pole collisions. But somehow some electricity poles and traffic sign poles just appear with naked lower-middle bodies without fluorescent markers. Even in daytime, this type of poles especially the poles with slim shapes, can lead the cyclists who are not 100% focusing on front road conditions or have visual degeneration to neglect their existence when they have colors close to background. As mentioned in chapter 1, the author ever saw an elder female cyclist crashed her face into an electricity pole, and similar accidents could be searched out as instances of evidence.

Figure 6 illustrates the common sense that in poorly lighted environments cyclists can easily get bumped by ditches or bulging objects hiding in shadows in case of careless riding. Although Japan's public places are provided with streetlight illumination in nighttime until dawn, many corners in cities are not well light-casted, not even to mention the roads in suburbs and rural areas.

Figure 5: A pole blending into background color may strike reckless cyclists

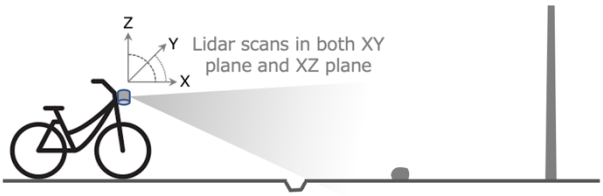
Figure 6: A ditch hiding in shadow can badly trap careless cyclists

3. Existing solutions

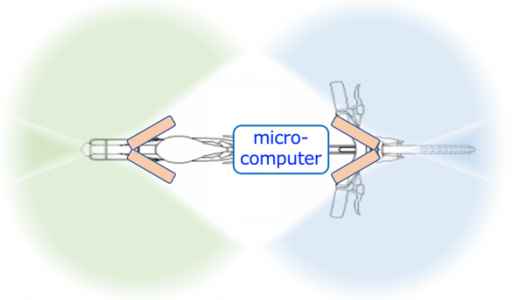
3.1 Existing solutions introduction

Since every cyclist is concerned about cycling safety, facilitating cycling safety with extra equipment plus engineering technologies is a meaningful research subject. And in this aspect, researchers and developers have done a lot of productive work which is worthy of reference for peer-to-peer study. Based on the target problem defined in chapter 2, the author consulted extensive information from theses, academic papers, and product development conceptions through the internet, and in turn he comprehensively searched out 4 classical existing solutions which have connections to his target problem, namely being not aware of inconspicuous poles, curbs, ditches, etc.

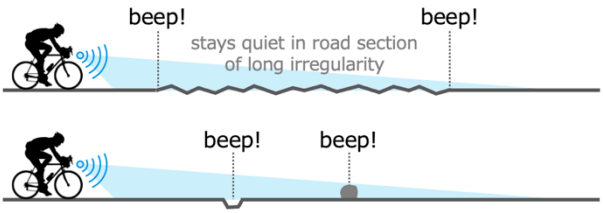
(a). Ultrasonic wave scan to detect road surface's dents and bumps



(b). Lidar scan to detect surroundings and road surface's dents & bumps



(c). Radar capture to detect moving objects getting closer in multiple orientations



(d). Camera plus computer vision to inspect road surface's irregularities

Figure 7: Diagrams of existing solutions to detect bike dangers

As diagrams in figure 7 demonstrate, solution_a takes advantage of ultrasonic waves to scan road surfaces in order to find out risky dents like potholes and bumps like stones, solution_b uses Lidar laser scanning in both horizontal plane and longitudinal vertical plane to detect obstacles and road surface's dents and bumps, solution_c utilizes conventional radar devices but deploys them in 4 positions to detect moving objects getting closer in multiple orientations, and solution_d adopts computer vision technology to inspect road surface's irregularities such as suddenly appearing road dents, road bumps, and surprisingly emerging football or animal, etc. The foregoing 3 solutions are presented in theses and academic papers while the last one of solution_d has been applied in real product development.

3.2 Characteristics of existing solutions

As the visual impressions presented in figure 7, the 4 classical existing solutions have some function features different than others and other function features in common. Because ultrasonic wave generally is not capable to probe objects further than 2.5 meters, so basically solution_a is in the limitation of inspecting very near road surface ahead in the distance of 2.5 meters. Based on the currently developed technology competence of Lidar laser, solution_b tends to work effectively on inspecting road surface quality which could be compromised by dents and bumps, rather than detecting surrounding objects. Solution_c is an impressive idea which assembles multiple radars turning out to be competent to detect objects in multiple orientations. But it mostly concentrates on detecting moving vehicles getting closer to cyclists. Solution_d pioneeringly integrated computer vision technology for inspecting road surface irregularities which are broadly and ambiguously defined. That is to say, risky objects with long irregularities which are actually regarded as regularities by the designed recognition algorithm of this system will not trigger any beep sound. The common places of the 4 solutions are first all of them are of the function to detect objects ahead of bicycles, and second all the systems adopt the same pattern of working mode that detection device relays inspection signals to a micro-computer which generates final alerts that can be easily and instantly understood by users.

4. Solution proposal

4.1 Research motive

Most people take it for granted that humans can easily detect road dangers in front of their bicycles with their own eyes. But the granted things just occasionally fail in case of cyclists' own mistakes such as sighting miss, attention on distractions, and some other unconventional reasons. All the existing solutions presented in chapter 3 theoretically take care of road dangers in front of bicycles, but actually they just can bear limited jobs with confined capabilities according to their working mechanisms. If there is a third artificial eye stares at bicycle's front side ceaselessly to help cover cyclists' occasional sighting misses and deconcentrating as a nonstop safety aide, it would be a new gospel for the cyclists who prefer to equip the third eye. And this was exactly the author's motive to proceed with this research.

4.2 Proposed solution

Figure 8 succinctly paints a rough image of what the author's proposal looks like and what kinds of devices need to be used to realize the proposed solution. The proposed system's total hardware consists of 3 components which are a front view camera mounted on the bike forepart, a man-machine interface fixed on the steering handle, and a micro-computer plus a portable battery powering the whole system attached underneath the seat pad. The man-machine interface is supposed to output final speech alerts and provide optimal man-machine interaction access, however, it can be left out of the proposal in the case that, as an alternative means, a well-configured micro-computer assumed employing GPIO functions can handle the 2 jobs. And the author picked out 3 classical obstacle types of ditches, curbs, and electricity poles as detection targets as shown in the proposal diagram. In real practice for model achievement, many other obstacle types could be set as detection targets just depending on how a developer's computer vision model is trained. However, considering limited time for this research practice, the author selected only electricity poles as detection targets in order to minimize prototype development, which will be referred to again in section 5.1.

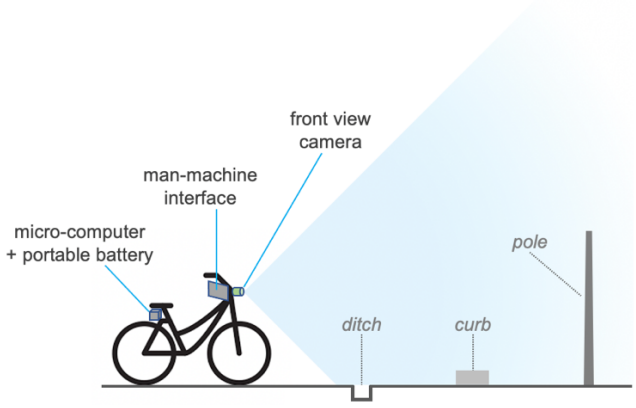


Figure 8: Proposed system of bike obstacles detection using deep neural net of computer vision

The front view camera consecutively captures images in real time, and in turn the captured images are analyzed using deep neural net of computer vision inside the brain of the whole system which is the portable micro-computer. The image analysis work includes judging what types the front looming obstacles are and weighting how dangerous they are. If a detected obstacle's risk weight is greater than the predefined risk weights threshold depending on the type of this obstacle, a task integration app in the micro-computer will immediately trigger a speech alert to remind user of what type the looming dangerous obstacle is and what orientation it is located in. Furthermore, the author must make things clear to say that multiple obstacles inside any single image can be instantaneously recognized using deep neural net of computer vision according to this computing technology's current development level, therefore computing multiple simultaneously detected obstacles' risk weights in a very short moment is not a challenging task, and of course, the final speech may indicate multiple dangerous obstacles. All these successive processes compose an action loop in figure 9.

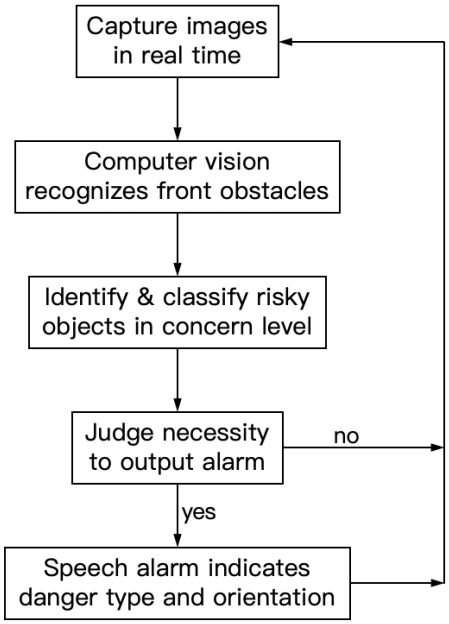


Figure 9: The workflow of the proposed bike obstacles detection system

4.3 Algorithm for weighting pole risk

In the case of deploying deep neural net of computer vision for bike obstacles detection, a general algorithm used for computing obstacles' risk weights is a sequent job must be done. And figure 10 illustrates how the proposed solution calculates risk weight under the situation of having detected an electricity pole. At the beginning, the author hypothesizes that the raw images captured by camera are of resolution with $2a*b$ pixels in image width and height respectively. And the blue box UVAB represents a raw image. The bottom middle point O of the blue box stands for coordinate origin while lines OX and OH stand for X positive and Y positive

respectively. The rectangle $B_1B_2B_3B_4$ stands for the bounding box output by deep neural net to predict the existence of the pole. To quantify the detected pole's risk weight, 5 vital assignments need to be done in steps. The 1st one is to mark out a risk-concerned area, the 2nd one is to locate a target point whose x & y coordinates are used for weight computation, the 3rd one is to determine a virtual projection angle including the central line OH plus the fictive camera position outside the blue box and the target point located in last assignment, the 4th one is to measure the previously located target point's relative positions inside the risk-concerned area and the blue box, and the 5th one is to formulate a simplified algebraic equation for computing pole risk weight.

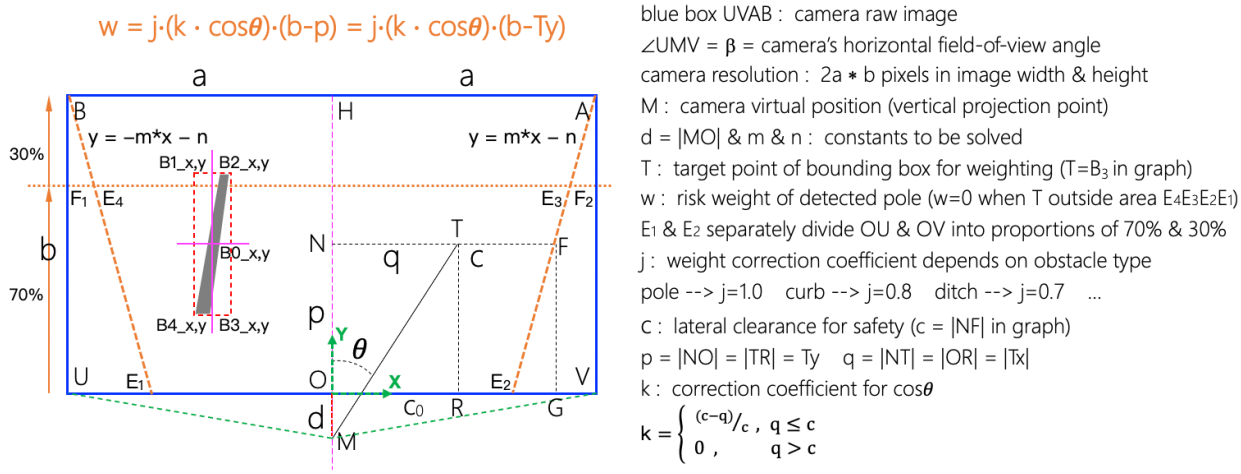


Figure 10: Pole risk weight calculational algorithm

As shown in figure 10, the risk-concerned area of the isosceles trapezoid $E_4E_3E_2E_1$ with a wider roof is enclosed by 4 lines, which are the blue box's bottom line, the horizontal line F_1F_2 that divides the blue box into 70% and 30% in the height direction, and the 2 inclined lines of $y = \pm m*x - n$ (m and n are positive constants to be determined by the camera imaging resolution of 2a*b pixels) that respectively go through the 2 top corner points A & B of the blue box and respectively incorporate the 2 points E_1 & E_2 which separately divide line segments OU & OV into 70% & 30%.

Because electricity poles are roughly vertical objects with respect to the ground, only the bottom-line segment of B_4B_3 is necessary to be taken into account. Assuming B_0 is the geometrical centroid of the bounding box $B_1B_2B_3B_4$ which is used by computer vision to outline a detected object, if B_0 locates in raw image's right half (B_0_x>0), point B_4 will be used as target point T for pole weight calculation, otherwise, if B_0 locates in raw image's left half (B_0_x<=0), point B_3 will be used as target point T for pole weight calculation. Based on this rule, point B_3 is regarded as target point T in the instantiation of figure 10, where points B_3 and T are deliberately located at different positions on purpose of avoiding layout congestion.

Stepping further is to define a point M at Y negative axis to virtually represent camera position which is out of camera image. If camera's field view angle in horizontal plane is β based on product specifications when

camera lens axis is well leveled in a horizontal plane, then d = |MO| = a ∗ cotangent (β/2) locates the position of M. And in turn as a critical factor in the final equation the virtual projection angle θ = ∠TMN could be calculated using the expression θ = arctan[q/(p+d)] = arctan[|Tx|/(Ty+d)] where Tx and Ty are the target point T's coordinates inside the camera coordinate system defined in figure 10.

The next moves approaching the final equation are measuring how far T is to the blue box's top line BA and how proportionally close T is to the blue box's central line OH. Given p = Ty, then b-p = b-Ty determines how vertically far T is to the top blue line BA. Supposing c = |NF| in image width direction defines lateral clearance, k = (c-q)/c = (c-|Tx|)/c indicating how laterally close T is to line OH is used as correction coefficient for cosθ. In addition to this, k is assigned 0 when |Tx| > c.

Finally, the algebraic equation could be written as w = j·(k · cosθ)·(b-p) = j·(k · cosθ)·(b-Ty) to compute risk weight in case T locates inside the risk-concerned area $E_4E_3E_2E_1$. The constant of j depends on obstacle type, and in case of pole obstacle j is set to 1. And the weight w is regarded as 0 under the situation that T locates outside the risk-concerned area.

4.4 Methodology to determine pole risk weights threshold

Once an algorithm for computing pole weights is determined, a closely subsequent question is how to determine pole risk weights threshold by which any detected pole can be classified as a risky object in concern level or a less risky object which can be temporarily ignored by the proposed system. Figure 11 illustrates a simplified method to define pole risk weights threshold. The key part of this job is to determine a critical distance between an electricity pole and a riding bicycle which is straightly heading the pole in line. This critical distance will be used for positioning bike on purpose of capturing image through bike front-view camera to decide on final pole risk weights threshold.

Ordinary bike riding speed except for competitive cycling sports is roughly in the range of 10~20 km/h. The author grabs the top speed 20 km/h (≅ 5.56m/s) to estimate the critical distance for purpose of giving cyclists longer buffering distance to respond to looming dangers in real rides. Based on common technological senses, supposing the prototype system totally takes 0.2~0.3 second to recognize a pole including camera photographing time, deep neural net analysis time, and the time for subsequent risk level evaluation and triggering final alarm etc., and given that a general cyclist needs 1 second on average to make reactions, then the critical distance could be estimated using the expressions (20km/h) ∗ (1+0.2)second ≅ 6.67meters, and (20km/h) ∗ (1+0.3)second ≅ 7.22meters. After rounding off, the final critical distance is defined to be 7 meters.

The next move is to find a front-to-back level ground with an upright electricity pole which can be easily detected by the bike's equipped computer vision module at various distances in numerous orientations. At this point, an easily detectable pole might be a little important, since the deep neural net of computer vision might not be 100% capable of detecting a random target obstacle attributed to a well-trained object type in deep neural

net at an arbitrary position in camera image, even though the deep neural net has been trained hard. Besides, as shown in figure 11, the camera needs to be tightly fixed with camera lens axis positioned along front-to-rear horizontal line. And following up it is to place the prototype bike straightly heading the selected pole at the critical distance of 7 meters to the pole, and in turn to capture and save an image through the front bike camera which will be used for determining pole risk weights threshold later. The final move of this step is to place the prototype bike straightly heading the selected pole as well, but at 2 distances which are respectively a little nearer and farther than the critical distance (just set the tantamount distance bias to 1 meter based on the top bike speed ≅ 5.56m/s) by 2 times, and to subsequently capture and save images through the front bike camera as well.

With a professional software tool of raw images analysis like ImageJ, framing boxes containing poles could be precisely drawn in the 3 images captured at the 3 specified distances as illustrated in figure 11, and the bounding boxes' all corner coordinates in pixel units could be acquired as well. And then by following the algorithm explicated in section 4.3 step by step, 3 pole weight values can be obtained after mathematical calculations. Then, the weight value determined at the critical distance can be decided as the pole risk weights threshold, while the other 2 weight values determined at the shorter and farther distances can be used for comparison to see the proximity extents of the 3 determined weight values for purpose of verifying how easily or hard false risk level judgements can be made by the proposed system.

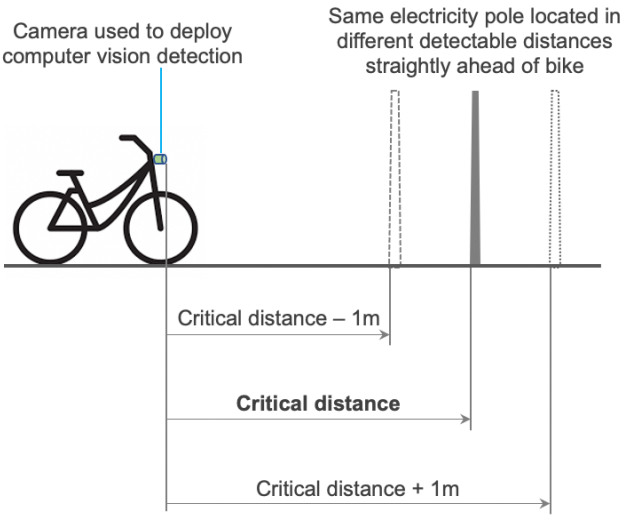


Figure 11: Pole risk weights threshold determination at critical distance

4.5 Comparison with existing solutions

To recap the discussions in chapter 3, ultrasonic wave inspection is just capable of detecting road bumps & dents in very short range less than 2.5 meters, Lidar laser scan does well for examining road surface quality but this technology's development hasn't determined what specific surroundings it can distinguish up to now,

multiple radars detection is good at moving vehicles approaching from multiple orientations but not capable of still obstacles relative to the ground, solution_d based on a vague concept of road surface irregularities could miss risky obstacles with long irregularities and with simple beeping sounds it does not inform cyclists detail types and orientations of detected dangers. The author's proposed solution is similar to solution_d since it adopts computer vision technology as well. But it is designed to do more complex and pragmatic jobs than solution_d, which means by warning cyclists of obstacle types and orientations in advance it can give users premeditated implications of how to react to looming dangers according to obstacle types and orientations. For example, in case of getting reminded of a suddenly appearing long ditch close to and parallel to his or her riding route from a speech alert in proposed solution_e, a cyclist just needs to keep appropriate lateral clearance with optional speeding down. Obviously, solution_d probably can't detect a long regular ditch along a cyclist's lateral side, not even to mention giving the cyclist a premeditated indication for how to handle the situation according to identification information of detected danger. Table 1 underneath gives readers a concise impression of the comparison between existing solutions and the author's proposed solution. And the novelty of the author's proposal lies in that, with the effort of unconventionally deploying computer vision technology on ordinary bicycles, it can notify cyclists more detailed information about front looming dangers such as obstacle types and obstacle orientations, leading to the consequence that cyclists can do better in reacting to the dangers to protect themselves.

Table 1: Comparison between existing solutions and proposed solution

Solution	Item	Inspection tool	Inspection target	Alerting mode	Primary user
existing	a	Ultrasonic wave	bumps & dents of road surface		
	b	Lidar laser	road surface qualities + bike surroundings		
	c	Multiple radars	approaching objects in multiple orientations		
	d	Camera + computer vision	lane surface irregularities	beep sound	road racers
proposed	e	Camera + deep neuralnet of computer vision	electricity pole + curb + ditch	speech indicating obstacle type & orientation	common riders

4.6 Research objective tree

As the objective tree presents in figure 12, the overall objective of the author's research practice was to develop a bike aide system of obstacle identification using deep neural net of computer vision based on the purpose to predict inconspicuous poles, curbs, and ditches to cyclists for early responses. And the author expected the bike aide within this context could handle 4 classical jobs which are predicting abruptly appearing obstacles to cyclists, urging cyclists to notice obstacles blending into backdrops for close colors ahead of time,

pulling back cyclists' concentration from distractions to front looming dangers, and reminding the cyclists with visual degeneration to discern front obstacles in time.

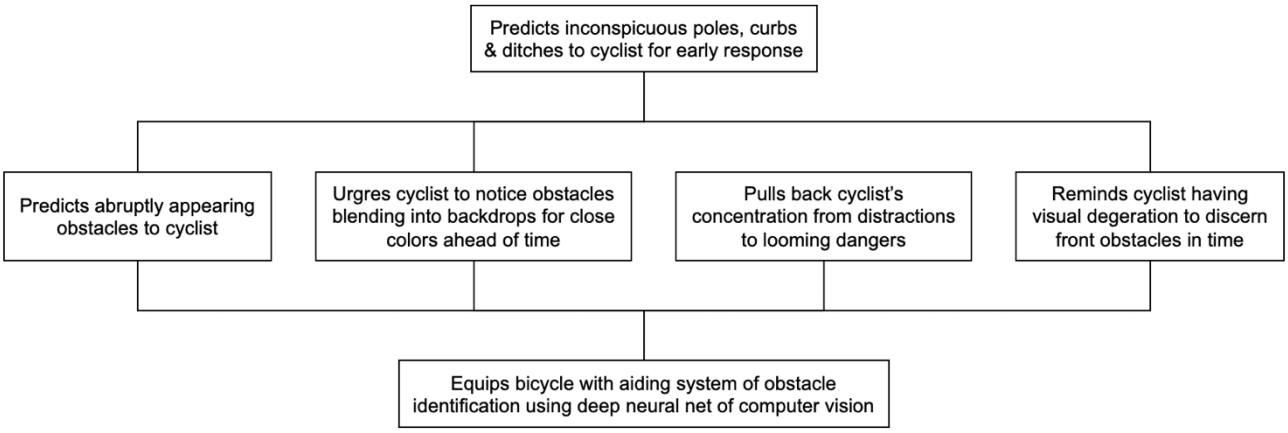


Figure 12: Objective tree for research practice

4.7 Tankyu chart of research orientation

To set up a strategic orientation to direct his research work on macro scales, the author portrayed a Tankyu chart presented in figure 13. Simply speaking, the whole research practice roughly consisted of the 3 vital procedures which were identifying issue as research subject, conceiving possible solution, and using exploitable resources and methods as solution enablers to testify proposed solution's effectiveness.

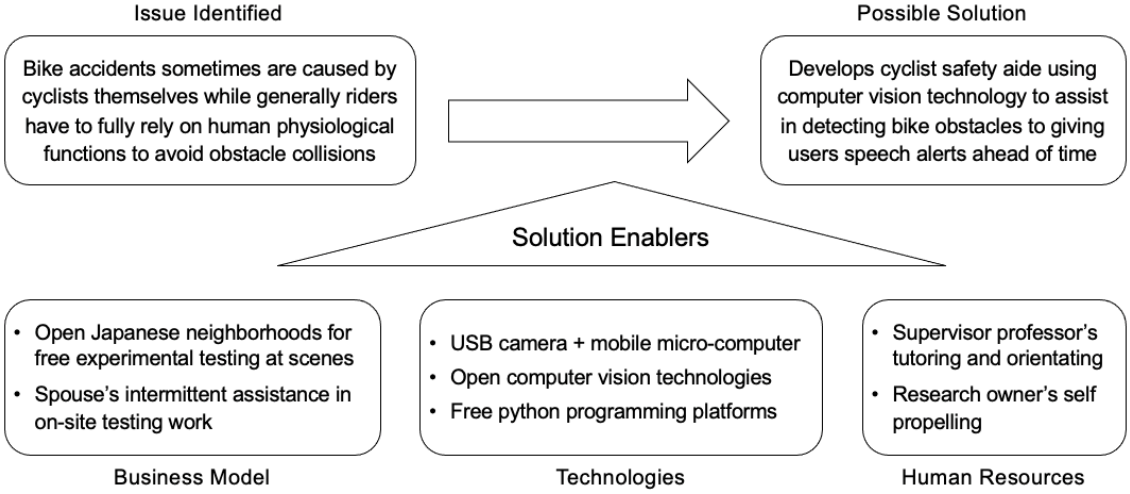


Figure 13: Tankyu chart to orientate research practice

In the author's research scheme, the identified issue was that bike accidents sometimes are caused by cyclists themselves while riders generally have to fully rely on human physiological functions to avoid obstacle collisions. Since the author thought that computer vision technology had some extraordinary features over other technological measures and it had not been deeply developed for helping on cycling safety, he decided on

developing cyclist safety aide using computer vision technology to assist in detecting bike obstacles ahead of time with speech alerts to facilitate cycling safety as his possible solution. Moreover, several optimistic estimations on the research practice could be firstly the author's supervisor professor had been experienced with deploying computer vision in depth, secondly, mobile micro-computers had become pretty functionally powerful to process relatively complex jobs so far, thirdly, open computer vision technologies and free python programming platforms developed by IT vanguards provided amicable underlying architectures for coding, and fourthly open Japanese neighborhoods were also favorable for prototype testing in fields.

5. Prototype development

5.1 Objective of prototype

Considering the complexity of the research subject and the limited time allowed for prototype experimenting, the author had to simplify his prototype development. As shown in figure 14, the prototype function was designed to remind cyclists of only detected electricity poles above concerned risk level using computer vision detection. With this limited function, the prototype objective was set to verifying that the proposed solution was workable for the 2 typical problem scenarios described in chapter 2, which are suddenly showing-up obstacles that could bump back-watching cyclists, and poles blending into background color may strike reckless cyclists. And the author decided to adapt his prototype to working in only daytime environments, after discovering ordinary infrared usb_cameras are not capable to capture images of any object further than 2 meters in full darkness through in-field testing.

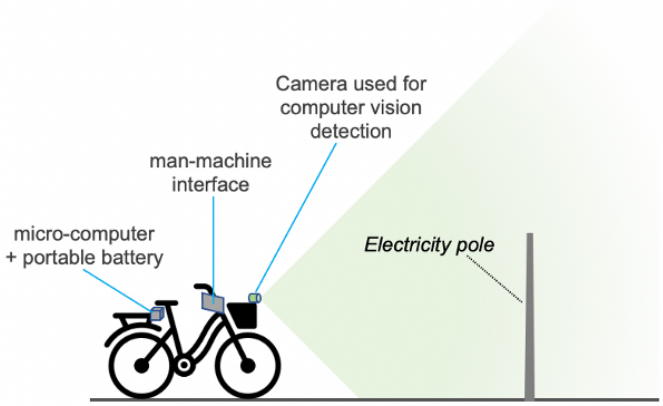


Figure 14: Minimum prototype function to detect electricity poles

5.2 Hardware configuration

As figure 15 demonstrates, to develop a prototype machine, the author assembled 5 hardware components together which were a RaspberryPi-4B micro-computer, a wide-angle USB camera, a neural net accelerator, a mobile battery box, and a mini-LCD touch screen. And table 2 lists the critical features of the 5 electronic devices used in assembling. As the newest generation of RaspberryPi microcomputer series so far, RaspberryPi_4B can process complex jobs with speeds not much lower than general family laptops, and it provides multiple connection port types to support other 4 devices, moreover, as a very important factor, it has good compatibility to co-work with the Intel Compute Stick 2, namely the neural net accelerator. This was why the author chose RaspberryPi_4B as the computation brain of the whole system. In order to seize as much road & traffic information as possible for objects detection, a wide-angle camera with a popular aspect ratio of 16/9 capturing images in landscape mode with a maximum resolution of 1920x1080 was adopted. To build a user-friendly man-machine interaction interface, the author selected a 7-inch capacitive LCD touch screen. In case

Figure 15: Electronic devices used to develop prototype machine

of using no mini LCD screen, other type's man-machine interaction method would have to be achieved through configuring GPIO pins on RaspberryPi_4B. And the mobile battery had been proved to have the capacity to support the whole system in working of consecutive real-time object detection for more than 12 hours. And the next section will explain the necessity of using the neural net accelerator.

Table 2: Hardware components used to assemble prototype machine

Item	Functional device	Model	Critical features for prototype expriments
①	Mobile micro-computer	RaspberryPi-4B	1.Micro SD card slot for loading operating system and data storage 2.Quad-core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5GHz processor 3. Multiple USB ports including 2 USB 3.0 ports can support Neural Compute Stick 2 4. 2 mini HDMI video ouput ports & 1 USB-C 5V power input port
②	Front view camera	Wide-angle USB camera	1. Default 30fps dynamic imaging resolution of 1920x1080 pixels 2. Diagonal field-of-view angle of 160 degrees
③	Neuralnet computation accelerator	Intel Neural Compute Stick 2	1. Hardware-compatible for RaspberryPi-4B & MacBook Pro 2. Support OS platforms of macOS_Catalina & Raspbian_Linux11
④	Power supply	Mobile battery of power bank	1. Max power capacity of 20000mAh(74Wh) 2. Output ports support both USB-A and USB-C with automatic voltage switching amid 5V, 9V and 12V 3. Maximum power supply of 18W
⑤	Man-machine interface	Mini LCD touch screen	1. HDMI video input port 2. 5V micro-USB power input ports 3. 7 inch portable mini screen 4. Supports RaspberryPi

5.3 Necessity of neural net accelerator

Before assembling the prototype machine, the author experimented to evaluate the object detection speed of computer vision on RaspberryPi-4B without deep neural net acceleration. The deep neural net model used in the experiment was Yolo3-tiny with pre-trained neural net weights provided by its developers. After the work

of file format conversion on the pre-trained neural net weights, the infrastructural Yolo3-tiny model was capable to detect up to 80 classes of ordinary objects often seen in our daily life like person, bicycle, car, motorbike, bus, truck, etc. And the author used this Yolo3-tiny model to detect ordinary object like persons in a prepared still image. As illustrated in figure 16, 2 tests were performed separately in 2 means which were once_run execution and 10 loops execution. The 2 tests did a same object detection task with a same still image, whereas the difference was that the 1st test implemented the task just for once in 1 execution of program while the 2nd test repeatedly implemented the task for total 10 loops in 1 execution of program.

once_run execution code

```
def detect_img(yolo):

    #while True:
    #    img = input('Input image filename:')
    img = 'pic1.jpg'
    #    try:
    image = Image.open(img)
    #    except:
    #        print('Open Error! Try again!')
    #        continue
    #    else:
    r_image = yolo.detect_image(image)
    #        r_image.show()
    yolo.close_session()
```

10 loops execution code

```
def detect_img(yolo):

    #while True:
    #    img = input('Input image filename:')
    img = 'pic1.jpg'
    #    try:
    image = Image.open(img)
    #    except:
    #        print('Open Error! Try again!')
    #        continue
    #    else:
    for i in range(10):
        r_image = yolo.detect_image(image)
    #        r_image.show()
    yolo.close_session()
```

Figure 16: Python codes for once_run and 10 loops repeating on same object detection task

The author measured the 2 tests' time cost under 3 program overhead types of "real", "user" and "sys" using the Linux command "time". The measurement result in table 3 shows that, under the situation of continuous repeating object detection tasks in 1 time's program execution, the average time cost of deep neural net running for 1 single detection task was 1.846 seconds. This time cost was high, because with it plus other factors like camera imaging delay, other program blocks' time cost outside the running of deep neural net, and cyclist' physiological reaction time, in case of detecting obstacles in real-time live image in field, a cyclist may spend more than 3 seconds to handle a risky obstacle with such a prototype machine. This would make the prototype meaningless for being helpless. Therefore, a deep neural net accelerator was necessary in order to sharply shorten deep neural net's object detection speed.

Table 3: Object detection speed estimation on RaspberryPi-4B without deep neural net acceleration

overhead type	Time cost by unit of second				
	task execution		offset for 9 loops	loop average	readiness for execution
	once_run	10 loops	10 loops - once_run	offset for 9 loops / 9	once_run - loop average
real	34.080	50.694	16.614	1.846	32.234
user	33.798	65.819	32.021	3.558	30.240
sys	2.179	6.072	3.893	0.433	1.746

5.4 Prototype working environment and setting-up flow

Figure 17 gives readers a terse impression of what the prototype working environment appears to be. The RaspberryPi_4B microcomputer was loaded with the operating system of the Raspbian_GNU version of Linux11_(bullseye)_32bit. The author adopted the open computer vision model of Keras Yolo3-tiny as his groundwork code to build on his own computer vision model of deep neural net. Considering the necessity of extensive compatibilities between Python language and other necessary program packages like Keras, Pillow-PIL, openvino-dev, opencv-python, etc., on the foundation of RaspberryPi_4B, the author decided to use the Python version of 3.7. And the free IDE platform of PyCharmpy-community-2021.3.2 was selected to provide a convenient Python programming environment in which the author could easily load and remove program packages. Besides, the program packages used for building the prototype will be listed in the thesis appendices.

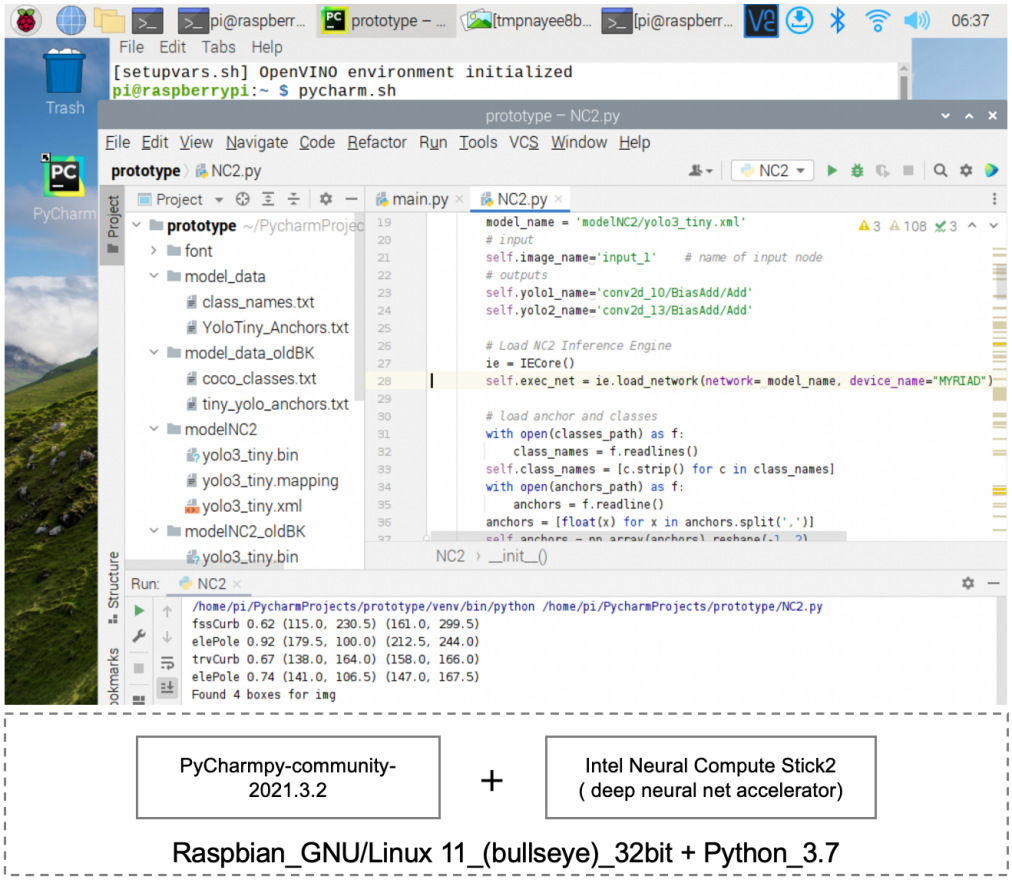


Figure 17: Critical hardware specifications of integrated electronic devices

Figure 18 concisely illustrates the setting-up flow of how the author worked up the prototype step by step. First, a well-adapted deep neural net model of computer vision was primarily needed as the infrastructure of the whole prototype. The DNN model of Keras Yolo3-tiny which was used as groundwork for this prototype can detect up to 80 classes of objects based on the COCO dataset freely provided by other developers. But unfortunately, the expected detection classes like electricity pole, curb, and ditch are not included in the COCO

dataset. Hence, the author photographed more than 1200 raw images with the themes of electricity poles, curbs, and ditches using the bike front camera equipped in his prototype machine. And then 800 raw images were picked out as data set to train Yolo3_tiny as described at the beginning of the flow chart in figure 18.

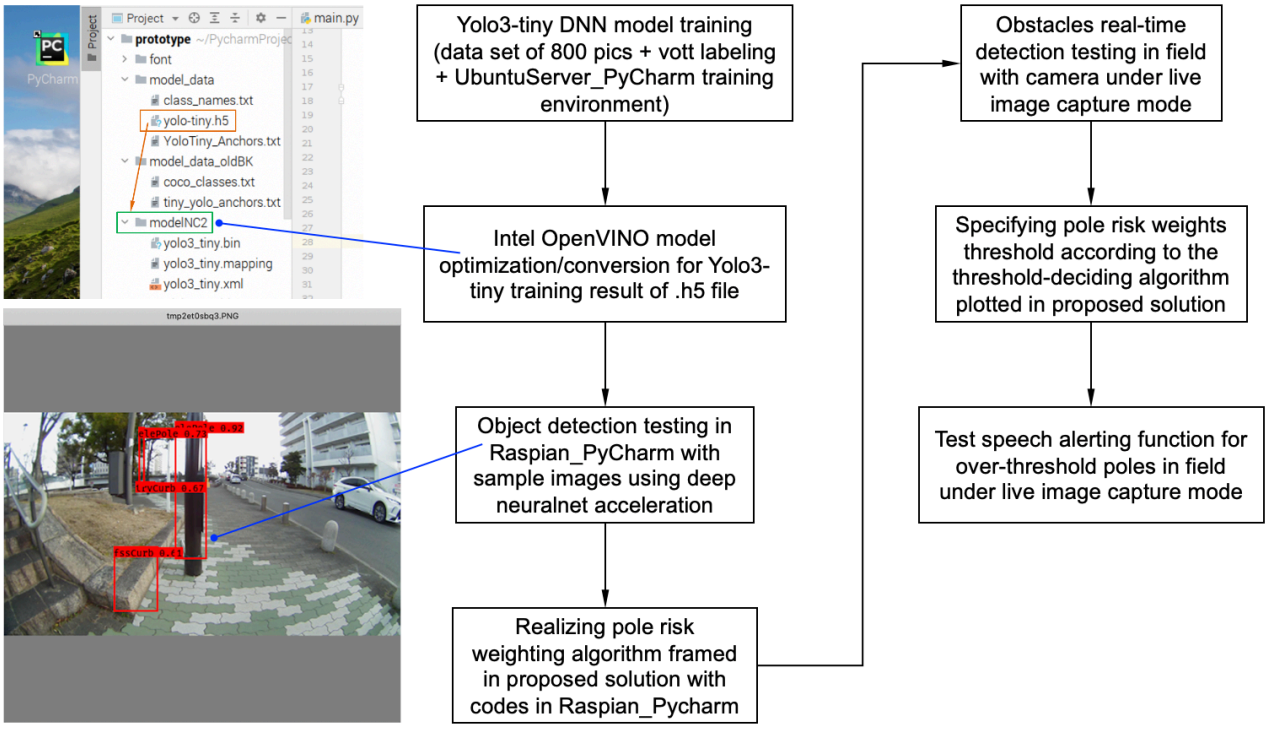
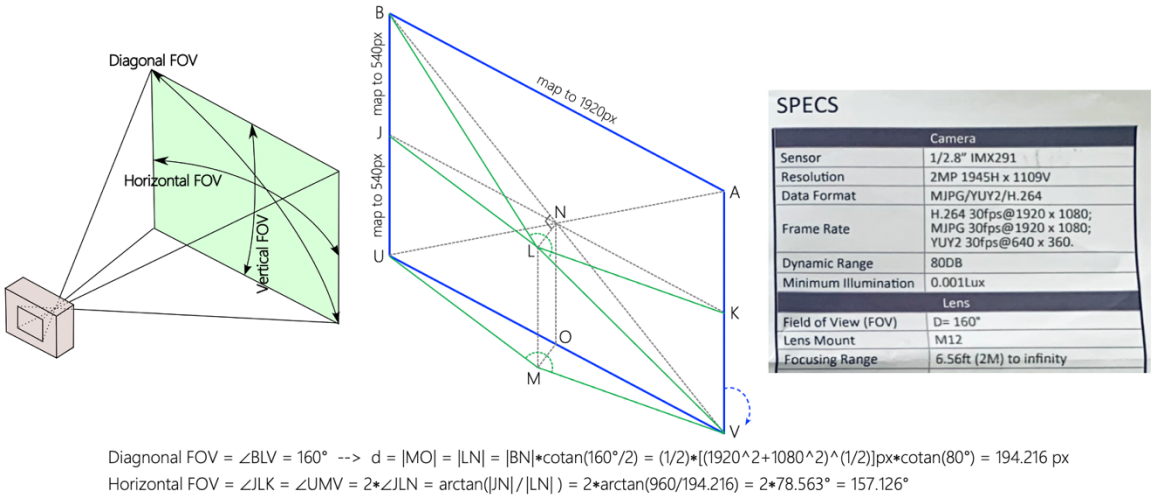


Figure 18: Setting-up flow to function prototype

After the DNN model training, the training result of .h5 file had to be converted to the format which could be read by the integrated deep neural net accelerator named Intel Compute Stick 2. This step was shorted called Intel OpenVINO model optimization/conversion. The left bottom picture in figure 18 gives readers a rough impression of what the result of object detection testing with still sample pictures looked like. The red solid-line boxes are the bounding boxes containing the obstacles detected by the trained deep neural net. And each detected obstacle was shortly tagged with 2 information pieces which were firstly an obstacle class defined by the DNN model trainer, namely the author, and later a confidence number indicating how much confident the trained DNN model was on the obstacle class judgement. As the supplementary instructions to this flow chart, section 5.5 will introduce the process of realizing the previously framed pole risk weighting algorithm and present a result demonstration of this task, section 6.2 will present the process and result of specifying pole risk weights threshold according to the plotted methodology explicated in section 4.4, and 6.3 will illustrate the in-field object recognizing experiment about measuring detection delays in bike-handle steering rotations.

5.5 Pole risk weighting algorithm implementation

The pole risk weighting algorithm implementation depended on the front view camera's 2 key performance indices, which were the imaging resolution and the horizontal field-of-view angle under dynamic fps capture mode. According to the camera's user manual, the dynamic fps imaging resolution was 1920∗1080 pixels, and the diagonal FOV angel was 160°. And the author's in-field photography tests proved that the imaging resolutions under dynamic fps capture mode and still one-by-one capture mode were both 1920∗1080 pixels. Nevertheless, the other performance index of horizontal field-of-view angle had to be determined by doing some solid-geometrical calculations.



Diagnonal FOV = ∠BLV = 160° --> d = |MO| = |LN| = |BN|∗cotan(160°/2) = (1/2)∗[(1920^2+1080^2)^(1/2)]px∗cotan(80°) = 194.216 px
Horizontal FOV = ∠JLK = ∠UMV = 2∗∠JLN = arctan(|JN|/|LN|) = 2∗arctan(960/194.216) = 2∗78.563° = 157.126°

Figure 19: Camera horizontal FOV angle and camera virtual position determination

The author re-plotted the front view camera's field-of-view angles in a 3-dimensional diagram as shown in figure 19. Supposing the camera lens' longitudinal axis was well leveled along the horizontal line LN, point L represented the camera lens' forepart, the blue box UVAB represented a complete vertical field-of-view plane in front of the camera, and the horizontal FOV angle to be solved was represented by ∠JLK. Based on the imaging resolution = 1920∗1080 pixels and the diagonal FOV angle = ∠BLV = 160°, furthering geometrical calculations decided that the horizontal FOV angle ∠JLK was equal to 157.126°, and the constant d to determine the camera virtual position which meant the same as the previous constant d in figure 10 was equal to 960∗cotan(157.126/2) = 194.215 pixels. The line segment LM actually stood for the main height of the prototype bicycle. And it was reasonable to assume that every point on the bicycle's main height was virtually watching front with a same horizontal FOV angle equal to ∠JLK just as the camera lens did. This indirectly proved that using detected poles' bounding boxes' bottom edges for risk weight calculation was reasonable. And in case of pushing the vertical plane UVAB down to the ground plane with a 90° forward rotation around the axis line UV, the new UVAB plane containing scenes and objects at different frontal distances could be used for pole risk weight calculation as the same as the previous UVAB plane in figure 10.

With the decided camera imaging resolution which was 1920∗1080pixels, the risk-concerned area defined in figure 10 could be fully determined. As shown in the right graph in figure 20, after some geometrical calculations the constants m and n were decided to be m=3.75 and n=2520, which meant the 2 inclined lines including the risk-concerned area were written as linear equations y = ±3.75∗x – 2520 .

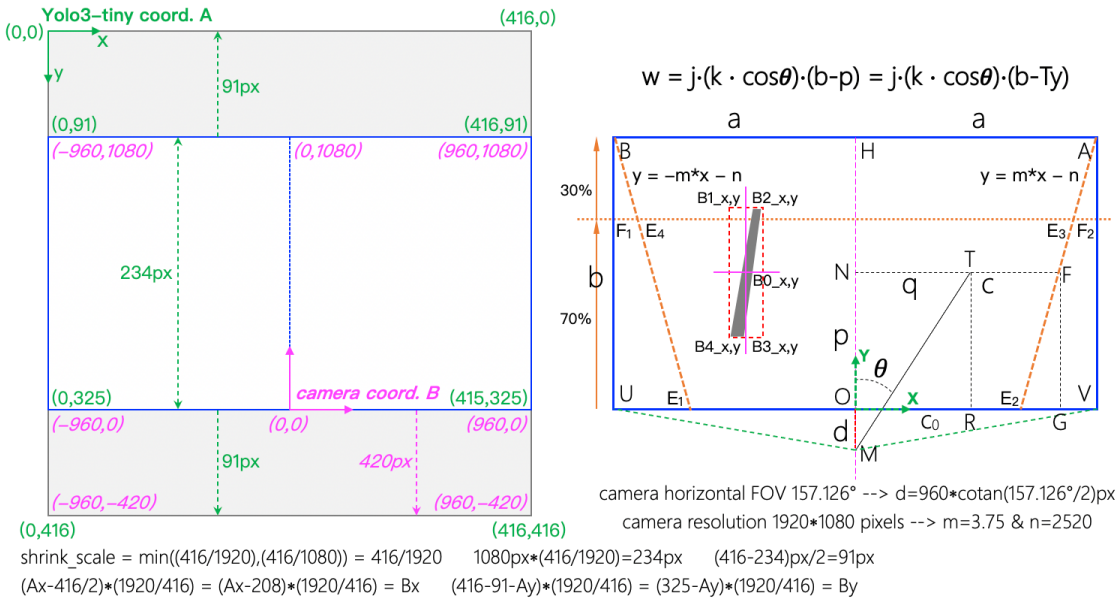


Figure 20: Coordinate systems conversion and risk-concerned area specifying

The other significant matter was coordinate systems conversion. The Yolo3-tiny DNN model only accepts square images with the resolution of 416∗416 pixels as input for deep neural net computations based upon its system design. Before being transmitted to the Yolo3-tiny model as process input, any raw image must be resized with an equal scaling factor decided by the function min(416/img_w, img_h) for both width and height. In the case of this prototype, the camera image's width was close to 2 times of its height, and even the image's height was at least 1 time bigger than the edge length of Yolo3-tiny's square input image. Hence, as commented at the left bottom of figure 20, camera raw images were resized using shrink_scale = 416/1920 in both width and height directions. After resizing, the camera image was placed into the Yolo3-tiny's square input box and centered in the height direction. Because of the inequality of resized image's width and height, 2 blank areas were left at the top and bottom of the square box. And these 2 blank areas were automatically filled with mono-color gray by the Yolo3-tiny system. On the other side, as the left graph of figure 20 illustrates, Yolo3-tiny selects the left top corner of its square input box as its coordinate origin with X positive facing right and Y positive going down, while the author picked out camera raw image's bottom-middle point as the coordinate origin of the camera coordinate system with X positive facing right and Y positive going up. And the author used the camera coordinate system as the ultimate computational coordinate system to implement pole risk

weighting algorithm. Therefore, the coordinate systems conversion formulas were crucial, and figured out as (Ax-416/2)∗(1920/416)=(Ax-208)∗(1920/416)=Bx & (416-91-Ay)∗(1920/416)=(325-Ay)∗(1920/416)=By.

Figure 21 demonstrates a sample of pole risk weights calculation results. The detected pole's bounding box's left & top & right & bottom corner coordinates were initially given by the author-trained deep neural net in the Yolo3-tiny coordinate system. And the corner coordinates were converted into 4 new coordinate values in the camera coordinate system defined in figure 20 where remaining calculations were executed. As shown in figure 21, in this pole detection case, the detected pole's risk weight was finally determined to be 412.465. The author wouldn't like to say but has to admit that this risk weight value did have a small problem of accuracy based on the issue that the red bounding box enclosing the detected pole had a little size contracted and position shifted to a non-serious level. But readers get the impression of pole risk weighting results with this calculation sample. And the problem of bounding boxes' size contracting & position shifting could be corrected by a very well-trained deep neural net model.

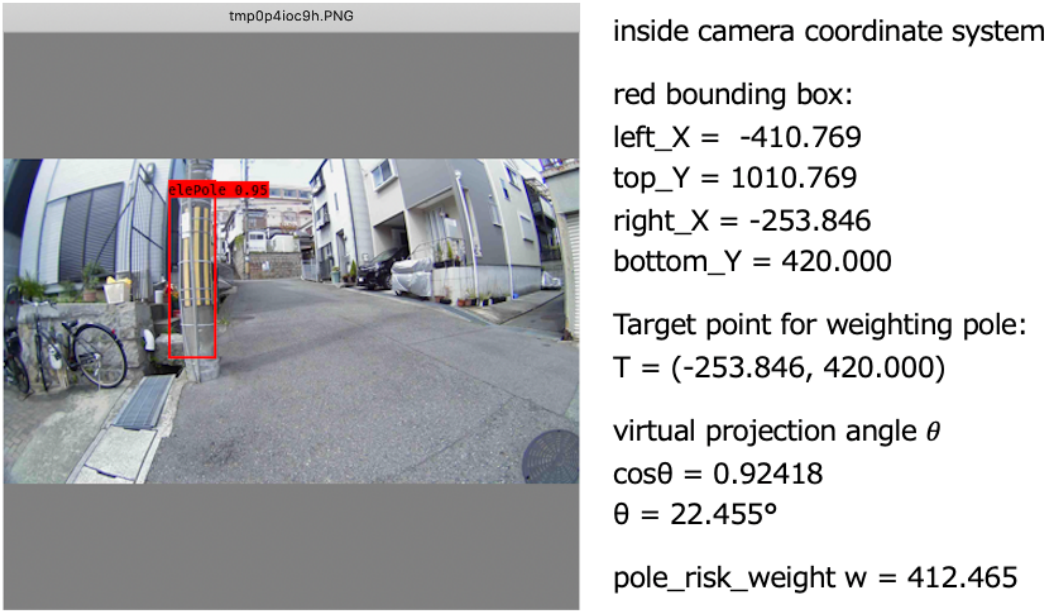


Figure 21: Pole risk weight calculation sample

6. Proposed solution evaluation

6.1 Neural net acceleration verification

To verify how much object recognition speed could be improved by using deep neural net accelerator, the author re-did a confirmatory experiment using his own author-trained Yolo3-tiny model accelerated by the Intel Neural Compute Stick 2 mentioned in section 5.2. The detection targets were curbs and poles in a still image captured by the bike front camera in the prototype machine. As shown in figure 22, the experimental method was same as the experimental method in section 5.3, which means, again, same object detection tasks with a same still image were respectively implemented in 2 tests, while the difference was that the 1st test implemented the task just for once in 1 execution of program while the 2nd test repeatedly implemented the task for total 10 loops in 1 execution of program.

once_run execution code + time cost record

```
if __name__ == '__main__':
    # initialization
    nc2=NC2()

    # Prepare input image
    img_name = 'testPics/2035.jpg'
    img = Image.open(img_name)

    # Execute
    out_boxes, out_scores, out_classes, boxed_image = nc2.execute( img, drawBB=True )
    print('Found {} boxes for {}'.format(len(out_boxes), 'img'))
    boxed_image.show()
```

```
(venv) pi@raspberrypi:~/PycharmProjects/prototype $ time /
home/pi/PycharmProjects/prototype/venv/bin/python /home/pi
/PycharmProjects/prototype/NC2.py
fssCurb 0.62 (115.0, 230.5) (161.0, 299.5)
elePole 0.92 (179.5, 100.0) (212.5, 244.0)
trvCurb 0.67 (138.0, 164.0) (158.0, 166.0)
elePole 0.74 (141.0, 106.5) (147.0, 167.5)
Found 4 boxes for img

real    0m4.317s
user    0m2.169s
sys     0m0.773s
(venv) pi@raspberrypi:~/PycharmProjects/prototype $ time /
```

10 loops execution code + time cost record

```
if __name__ == '__main__':
    # initialization
    nc2=NC2()

    # Prepare input image
    img_name = 'testPics/2035.jpg'
    img = Image.open(img_name)

    # Execute
    for i in range(10):
        out_boxes, out_scores, out_classes, boxed_image = nc2.execute( img, drawBB=True )
        print('Found {} boxes for {}'.format(len(out_boxes), 'img'))
        boxed_image.show()
```

```
trvCurb 0.67 (138.0, 164.0) (158.0, 166.0)
elePole 0.74 (141.0, 106.5) (147.0, 167.5)
Found 4 boxes for img
fssCurb 0.62 (115.0, 230.5) (161.0, 299.5)
elePole 0.92 (179.5, 100.0) (212.5, 244.0)
trvCurb 0.67 (138.0, 164.0) (158.0, 166.0)
elePole 0.74 (141.0, 106.5) (147.0, 167.5)
Found 4 boxes for img

real    0m7.092s
user    0m4.292s
sys     0m1.887s
(venv) pi@raspberrypi:~/PycharmProjects/prototype $
```

Figure 22: Once_run & looping run codes plus time cost records

The time cost measurement result in table 4 shows that, with the facilitation from the deep neural net accelerator, the average time cost of 1 single detection task was decreased to 0.308 second. This was a great improvement compared to the previous average time cost for a single detection task which was 1.846 seconds. Moreover, with the performance of 0.308 second, 4 obstacles in the testing image were simultaneously detected in 1 detection task, which was also a good performance. Considering that the bike front camera's default frame rate at continuous live imaging mode was 30 fps, plus supposing the camera was desired to normally work at

an image sampling rate of 1/4 for continuous live imaging mode, the camera imaging delay for a single detection task was determined to be 4/30 second ≅ 0.133 second. So, the total time cost for a single object detection task on 1 adopted camera image was estimated to be 0.308 + 0.133 = 0.441 second on average, which was a great performance index that could make the prototype system leave cyclists sufficient buffering time to respond to detected front looming obstacles. That was to say that the deep neural net accelerator's performance achieved the author's expectation.

Table 4: Object detection speed estimation on RaspberryPi-4B with deep neural net acceleration

overhead type	Time cost by unit of second				
	task execution		offset for 9 loops	loop average	readiness for execution
	once_run	10 loops	10 loops - once_run	offset for 9 loops / 9	once_run - loop average
real	4.317	7.092	2.775	0.308	4.009
user	2.169	4.292	2.123	0.236	1.933
sys	0.773	1.887	1.114	0.124	0.649

6.2 Pole risk weights threshold determination

Although the prototype machine had been proved to be competent to detect electricity poles in field experiments, to achieve the function of generating final warning messages to cyclists depending on pole risk extent, there was a critical procedure named pole risk weights threshold determination that must be done. As the methodology described in section 4.4, the author had made it clear how to determine pole risk weights threshold, the following related vital job to implement the algorithm with in-field experiments and relevant mathematical calculations.

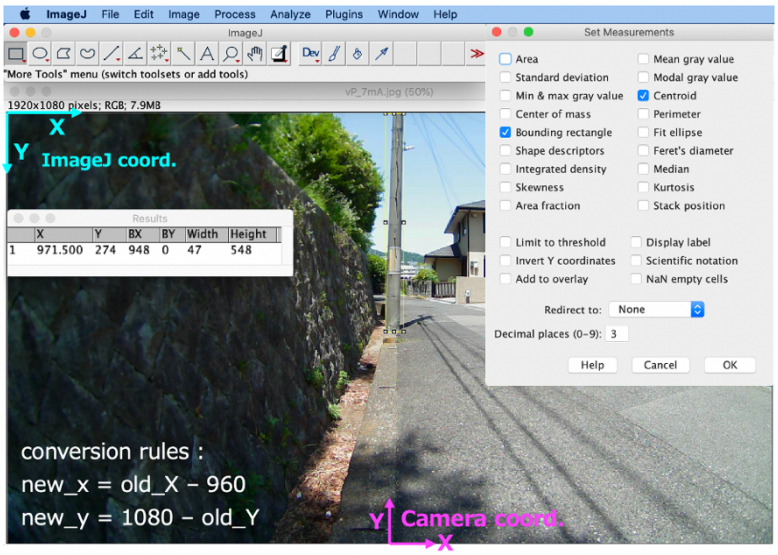


Figure 23: Border box coordinates extraction for pole at risk weights threshold deciding position

As shown in figure 23, a picture was captured by the bike front camera with the prototype bike positioned at pole risk weights threshold deciding position of 7 meters distance. Then the author extracted the target pole's border box's corner coordinates with the raw image analysis software tool of ImageJ. As can be seen in figure 23, the yellow filament box is the manually drawn border box which exactly contains the target pole, and the border box's size and position information was provided by the measurement function embedded in the ImageJ software. Because ImageJ coordinate system and the author-defined camera image coordinate system used different coordinate origins and Y axis positive directions, coordinate conversion equations had to be formulated to obtain the border box's 4 corners' final coordinates in the camera image coordinate system. The whole process was repeated both for the shorter distance of 6 meters and further distance of 8 meters at which the prototype bike was positioned as figure 11 instructs. And with the prototype bike at every distance, 2 images were captured from the bike front camera for pole border box coordinates extraction as redundant data sources for mutual confirmation.

With the border boxes' coordinate information acquired in the last step, the pole risk weights at different distances were determined as the green numbers listed in table 5, according to the computational methods explicated in sections 4.3, 4.4, and 5.5. Based on the fact that the distance of 7 meters was the critical distance decided in section 4.4, the final pole risk weights threshold was determined to be 537.44 after a simple averaging calculation. As the table shows, the weights trend down as distance increases, which is a good sign making sense. Whereas, the bad sign is that the weight values at 8 meters distance are very close to weight values at 7 meters distance. This also makes sense because of the perspective principle in 2-dimensionally imaging. However, the authentic imperfection here needs to be concerned in this table is that the risk weights' proximities at the critical distance and slightly farther distances could lead to a consequence that the prototype machine's risk level classification mechanism can't largely distinguish poles with risk weights slightly greater than risk weights threshold and poles with risk weights slightly smaller than the risk weights threshold. As a result, the final alarm generation module may easily trigger false alarms under the circumstance of detected poles having risk weights close to the risk weights threshold while a cyclist is riding on a bumpy road.

Table 5: Pole risk weights threshold determination table

distance	pic_ID	Coordinates in camera coordinate system				Virtual projection angle	Pole risk weight
		left_X	top_Y	right_X	bottom_Y	cosθ	w
6 meters	vP_6mA	-42	1080	25	503	0.99936	558.75
	vP_6mC	-39	1080	30	502	0.99907	555.97
7 meters	vP_7mA	-12	1080	35	532	0.99986	539.85
	vP_7mB	-19	1080	30	532	0.99966	535.02
8 meters	vP_8mA	-26	1080	14	536	0.99982	534.56
	vP_8mC	-24	1080	15	536	0.99979	533.87

Trend down as distance increases

537.44 on average

6.3 Detection delays in bike-handle steering rotations

The author implemented an experiment to measure object detection delays in bike-handle steering rotations as illustrated in figure 24. Before the beginning of the experiment, the author found an upright electricity pole standing on a front-to-rear horizontal ground. Like the positioning method in figure 11, the prototype bike was firstly placed at the distance of 8 meters straightly heading to the target pole. And then the bike handle was posed at a skew angle of 45° on left or right side. After that, the stopwatch in smartphone 1 was clicked on to start time counting, and subsequently the slow-motion filming was started in smartphone 2. Next, the author rapidly rotated the bike handle and restored it to its central straight position. 2~3 seconds later the author clicked off the slow-motion filming on smartphone 2, and then he played the just filmed video and found out the object detection delay time from the moment the bike handle started to rotate to the moment a blue bounding box started to appear in the mini-LCD screen indicating the target pole was detected by the camera restored to straight front position. In this way, the first object detection delay in bike-handle 45° steering rotation was measured. Later on, the author repeated same measurements with the bike placed at different distances of 7 meters, 6 meters, 5 meters, 4 meters, 3 meters, and 2 meters, and with no other condition changed.

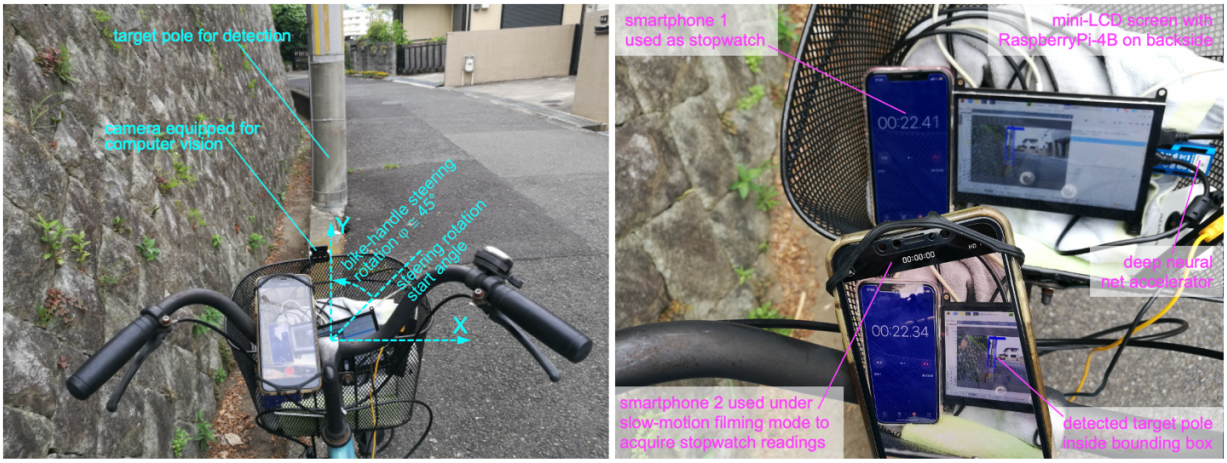


Figure 24: Object detection delays measurement settings for steering rotation tests

After the whole process was finished for the first target pole, same measurement processes were repeated for other target poles at different locations. After the whole experiment's completion, all the delay data were summarized as shown in table 6. Except failed measurements and a few discrete data points with values over 0.9 second or even greater than 1 second probably caused by the author's imperfectly trained Yolo3-tiny DNN model, most of the pole detection delays located in or stayed very close to the range of 0.6 ~ 0.8 second. As the comment in table 6 explains, every object detection delay consisted of 3 partitions which were bike handle steering rotation time, camera transmission delay, and computer vision detection time. The sum of the 2 latter items was equal to or close to 0.441 second according to the related explanation in section 6.1. Therefore, considering the inevitable measurement tolerances, the steering rotation time's first incorporated item which

depended on the author's hand driving speed approximately lay in the range of 0.2~0.35 second, which appears to make sense. And the key point is that, based on the previous analysis, the author could make an inference saying that, the author-trained deep neural net model of the prototype system can effectively detect dangerous front obstacles in the time of 0.6 ~ 0.8 second since a cyclist starts to largely turn his bike handle with a steering rotation of 45°. But in case of same bike handle steering rotation of 45° without deploying this prototype system, cyclists' human eyes may take several seconds to positively sight front looming obstacles in some situations because they must watch their inner lateral-to-rear side from time to time as they are making big rotation turns.

Table 6: Pole detection delays of bike handle steering rotations

pole detection delays of bike handle steering rotations (by unit of second)							
pole ID	bike camera to pole straight distances						
	2m	3m	4m	5m	6m	7m	8m
pole 1	0.68	0.76	0.63	0.65	0.91	0.75	0.79
pole 3	X	0.81	0.69	0.60	0.72	1.33	X
pole 4	0.72	0.80	0.76	0.77	0.68	0.74	0.69
pole 2	0.71	0.62	0.64	0.73	0.82	0.72	1.02
pole 6	0.70	0.71	0.65	0.79	0.74	0.69	0.58
pole 5	X	0.90	0.74	0.77	0.95	0.65	0.62
pole 7	0.68	0.67	0.60	0.75	0.80	0.69	0.61
pole 8	0.75	0.66	0.66	0.62	0.76	0.85	0.68
pole 9	0.73	0.61	0.75	0.72	0.65	0.78	0.94
pole 10	0.64	0.59	0.68	0.66	0.72	0.70	0.82
1. X: measurement failed because computer vision didn't recognize target pole 2. object detection delay = bike handle steering rotation time + camera transimision delay + computer vision detection time 3. camera worked at default frame rate of 30 fps and images sampling rate of 1/4, which resulted in camera transimission delay = (1/30)x4 second = 0.133 second							

6.4 Final speech alert experiment

With the pole risk weights threshold determined in section 6.2, the alarm module can reasonably generate speech alerts upon detected dangerous poles with risk weights above the threshold and, dismiss detected riskless poles with weights under the threshold. Plus considering the virtual projection angle θ and the positive or negative of the X coordinate of the target point T in figure 10, the final speech alarms could be set to 3 short voice messages of "pole middle", "pole left" and "pole right" based on bicycle's current forwarding direction which is represented by line MH in figure 10. As a supplement, table 7 tersely instructs the standards for triggering the 3 kinds of voice alerts.

To verify the feasibility of this speech alert mechanism, the author implemented confirmatory experiments in fields with his prototype machine after programming completion inside Raspbian_PyCharm. And the experiment results proved that, once a dangerous front pole was detected, the prototype machine could instantaneously output a speech warning which shortly states obstacle type of pole and obstacle orientation of

left, middle, or right in accordance with table 7. It ought to be better to optimize the speech alerting rules in table 7 to better cope with the complex situations that multiple poles are detected simultaneously in different orientations. Nevertheless, the final confirmatory experiments already demonstrated that, the conception of jointing the speech alert function to the bike obstacle detection module of computer vision is viable, and the methodology of clarifying obstacle types and orientations with momentary short voice messages, can indeed help cyclists obtain premeditated implications of how to react to front looming dangers as the author stated in section 4.5.

Table 7: Alerting speech message depending on T point's orientation

speech messages of alerts		X coordinate of T point	
		Tx <= 0	Tx > 0
virtual projection angle	θ < 7.5°	pole middle	pole middle
	θ > 7.5°	pole left	pole right

6.5 Evaluation result

Based on the neural net accelerator's facilitation on the minimized prototype machine discussed in section 6.1, the proposed solution was proved to be effective for the 2 application requirements, which are first helping cyclists early perceive inconspicuous obstacles like the scenario in figure 5, and second urging distracted cyclists to timely watch front suddenly emerging obstacles like the scenario in figure 4. And the explication on the measurement result of object detection delays in bike-handle steering rotations in section 6.3 suggests that, on the occasions when cyclists are making big rotation steering, the proposed solution can alleviate cyclists' burden of having to watch front side and inner lateral-to-rear side simultaneously, and can tellingly help cyclists to circumvent front looming dangers as well.

The minor problem of the pole risk weighting inaccuracy caused by the imperfectly trained deep neural net model discussed in section 5.5, and the sensitive alarm triggering issue caused by pole risk weights close to the pole risk weights threshold discussed in section 6.2, may compromise the proposal's utility a little bit. But the whole effectiveness of the proposal wasn't impacted. After all, as a cyclist safety aide, the proposed system doesn't need to be extremely accurate in non-critical computing processes. Starting to remind cyclists of risky obstacles since marginal safe distances and consequently leaving cyclists enough time to react to the looming obstacles is good enough. And the proposed solution meets this criterion.

7. Research practice conclusion

7.1 Closing statement

The evaluation work in chapter 6 qualitatively confirmed the effectiveness of the whole solution proposed in chapter 4. Metaphorically speaking, the proposed solution gives ICT developers a feasible conception that, deploying a third artificial eye stares at a bicycle's front side ceaselessly with the computer vision technology of deep neural net could efficaciously help cover cyclists' occasional sighting misses and deconcentrating.

Evidently, the free technology resource of computer vision like Yolo3-tiny, and low-cost high-performance portable micro-computers like RaspberryPi-4B, make this conception possible to be realized without worrying about budget management for a passionate research practitioner like the author. In spite of some imperfections in the prototype system, continuous furthering work can make the newer version of the prototype system tend to be better or even become visible with market potential.

7.2 Future work scheme

Besides the factor of limited time, the other reason why the author evaluated his proposal with a function-minimized prototype focusing on poles detection only, was that his own author-trained Yolo3-tiny DNN model couldn’t recognize the other 2 trained object classes of curb and ditch with high enough success rates. This was probably caused by the insufficient amount of training data set for Yolo3-tiny. Hence, the author plans to expand his training data set from the previous 800 pictures up to 3000 pictures and to incorporate more object classes in training process for purpose of making a final prototype system capable of recognizing more obstacle types with considerably high success rates. On the other side, adopting the newest generation of the Yolo-tiny technology series for now like Yolo5-tiny to replace Yolo3-tiny as the infrastructural DNN frame in the newer version of the prototype system is also a considerable option for prototype performance enhancement.

Furthermore, the much bigger ambition is to achieve a vehemently reinforced prototype version which can cover obstacle detection both in daytime and nighttime. With no doubt, this goal requires the author to equip his prototype machine with a high-performance night vision usb_camera costing a high price which can clearly capture the images of poorly lighted objects at distance up to 20 meters in dark nights. Regardless of rising-up project budget, soaring-up research value tempts the author to try it out with more enthusiasm and tenacity.

Acknowledgement

The author would like to firstly express his sincere appreciation to his research supervisor, professor Ryosuke Okuda, for his patient and enthusiastic tutoring on the author's research practice. As an excellent scholar, professor Ryosuke Okuda has been selflessly demonstrating and sharing his expertise with those KIC students who chose to learn ICT knowledge from him. And besides, his rigorous attitude toward academic research and his amiable communication skills are also good samples worthy of KIC students' learning.

Furthermore, the author is supposed to say a loud thanks to the KIC school which provides a good academic platform and hub for those passionate students gathering from many corners of this globe with their dreams of learning ICT knowledge. It was at KIC that the author got his priceless chance to learn precious professional knowledge in his favorite intellectual field and to exchange minds with brilliant professors and students.

And the author wouldn’t be grudging to say thanks to his beautiful beloved lovely wife. The study life in KIC was fortunate but toilsome, the author’s wife gave his husband irreplaceable support and encouragement.

Days before the author started to write these acknowledging words, the sudden leaving of the prominent Japanese national leader Abe Shinzo shocked and grieved the whole world except totalitarian nations. Mr. Abe Shinzo set up a superb example for billions of people around the world with his non-stop service and sacrifice to his great civilized mother country till the last moment of his life based on his deep belief in the universal values of democracy, liberty, and love. Hence, a late-coming appreciation to Mr. Abe Shinzo is deserved as well. And Mr. Abe Shinzo's aristocratic personality reminded the author of the sentence that he ever discussed with professor Ryosuke Okuda, which was "The knowledge students learned in KIC is just their tool, the key point is how KIC students use the tool to benefit others with the ultimate purpose of making a better world after they make good enough versions of themselves." The author hopes one day he could glorify KIC school with his dedication to Japanese society.

In the end, the author would like to send his final gratitude to the Almighty GOD for giving him the splendid fortune to land in Japan with the stunning consequences of meeting a lot of kind people and joining KIC for the study of the master's degree. GOD guides the author in the paths of righteousness for the sake of His name.

References

[1] “Neural Networks and Deep Learning” [online] by Michael Nielsen.Available at: http://neuralnetworksanddeeplearning.com/index.html

[2] Keras-yolo3 DNN model self-learning coach [online]. Available at: https://github.com/qqwweee/keras-yolo3/

[3] “Evaluation of a Bicycle-Mounted Ultrasonic Distance Sensor ...” [online]. Available at: https://ijssst.info/Vol-16/No-6/paper1.pdf

[4] “Laser-Based Obstacle Avoidance and Road Quality Detection for Autonomous Bicycles” [online]. Available at: https://www.researchgate.net/publication/285611991_Laser-Based_Obstacle_Avoidance_and_Road_Quality_Detection_for_Autonomous_Bicycles

[5] “Radar system for bicycle -a new measure for safety” [online]. Available at: https://www.researchgate.net/publication/337771867_Radar_system_for_bicycle_-a_new_measure_for_safety

[6] Production introduction of a branded device of bike collision avoidance [online]. Available at: https://www.indiegogo.com/projects/byxee-the-smart-active-safety-device-for-bicycle#/

[7] “Two-Dimensional Active Sensing System for Bicyclist-Motorist Crash Prediction” [online]. Available at: https://par.nsf.gov/servlets/purl/10038725

[8] “Introducing Python” [paper book] by Bill Lubanovic, first edition of November

[9] Camera field-of-view angle introduction [online]. Available at: https://en.wikipedia.org/wiki/Field_of_view

[10] User manual of the cross-platform IDE tool of PyCharm [online]. Available at: https://www.jetbrains.com/help/pycharm/2021.3/installation-guide.html

[11] User manual of Intel OpenVINO toolkit for Raspbian* OS [online]. Available at: https://docs.openvino.ai/2021.4/openvino_docs_install_guides_installing_openvino_raspbian.html

Appendix 1: prototype main program [main.py]

```
import cv2
import numpy as np
from PIL import Image
from Camera import Camera
from NC2 import NC2
from playsound import playsound
from judgeRisk import JR

draw_image_for_debug=True   # set False in real use

if __name__ == '__main__':

    try:
        #initialize
        cam=Camera()    # intialize USB Camera
        nc2=NC2()       # initialize NC2 accelerator

        while True:
            # delay.
            cam.skip_frames( 4 )  # image sampling rate of 1/4, which causes camera delay
            # Capture images with camera
            numpy_image = cam.capture( drawAtt=True)

            # DNN Prediction
            pil_img = Image.fromarray(numpy_image)
            out_boxes, out_scores, out_classes, boxed_image = nc2.execute(pil_img, drawBB=True)
            print('Found {} boxes for {}'.format(len(out_boxes), 'img'))
            print(out_boxes)
            print(out_scores)
            print(out_classes)

            # Identify dangerous objects in concern level
            pL, pM, pR = 0, 0, 0
            if len(out_boxes) != 0:
                pL, pM, pR = JR.judgeRisk(out_boxes, out_scores, out_classes)
            # Judge necessity to output alarm
            if pM > 0:
                playsound('/home/pi/AndyAudios/PoleMiddle.mp3')
            if pL > 0:
                playsound('/home/pi/AndyAudios/PoleLeft.mp3')
            if pR > 0:
                playsound('/home/pi/AndyAudios/PoleRight.mp3')

            # draw images just for debug
            if draw_image_for_debug is True:
                numpy_img = np.array(boxed_image)
                cv2.imshow( 'Frame', numpy_img )
                cv2.waitKey(1)  # 1ミリ秒キーボードの入力を待ち受ける
                #time.sleep(0.5)
    except KeyboardInterrupt:
```

```
            pass
        finally:
            cam.release () # VideoCapture の Close
            cv2.destroyAllWindows()   # cv2 window を削除する
```

Appendix 2: prototype package program [Camera.py]

```
    import cv2
    import time

    interval=0
    class Camera():
        def __init__(self):
            self.camera = cv2.VideoCapture(0)

        def skip_frames(self, num_frames ):
            while( self.camera.isOpened()):
                num_frames -= 1
                if num_frames < 0:
                    return
                retval, image = self.camera.read()
                if retval is False:
                    raise IOError

        def capture(self, drawAtt=False):
            while( self.camera.isOpened()):
                retval, image = self.camera.read()
                if retval is False:
                    raise IOError
                if drawAtt is True:
                    text = 'WIDTH={:.0f} HEIGHT={:.0f}
FPS={:.0f}'.format(self.camera.get(cv2.CAP_PROP_FRAME_WIDTH),
                                                    self.camera.get(cv2.CAP_PROP_FRAME_HEIGHT),
                                                    self.camera.get(cv2.CAP_PROP_FPS))
                    # 元 Image,文字列,位置,フォント,サイズ（スケール係数）,色,太さ,ラインの種類
                    cv2.putText(image, text, (10, 30), cv2.FONT_HERSHEY_SIMPLEX, 1, (255, 255, 255), 1, 4)
                return image

        def release(self):
            self.camera.release();

    if __name__ == '__main__':
        try:
            camera=Camera()
            while True:
                camera.skip_frames( 4 )
                image = camera.capture( drawAtt=True)
                cv2.imshow( 'Frame', image )
                cv2.waitKey(1)  # 1 ミリ秒キーボードの入力を待ち受ける
```

```
        #time.sleep(0.5)
    except KeyboardInterrupt:
        pass
    finally:
        camera.release ()   # VideoCapture の Close
        cv2.destroyAllWindows()   # cv2 window を削除する
```

Appendix 3: prototype package program [NC2.py]

```
from openvino.inference_engine import IECore
import numpy as np
from PIL import Image, ImageFont, ImageDraw
import math

class NC2():
    def __init__(self):
        classes_path = 'model_data/class_names.txt'
        anchors_path = 'model_data/YoloTiny_Anchors.txt'
        model_name = 'modelNC2/yolo3_tiny.xml'
        # input
        self.image_name='input_1'    # name of input node
        # outputs
        self.yolo1_name='conv2d_10/BiasAdd/Add'
        self.yolo2_name='conv2d_13/BiasAdd/Add'

        # Load NC2 Inference Engine
        ie = IECore()
        self.exec_net = ie.load_network(network= model_name, device_name="MYRIAD")

        # load anchor and classes
        with open(classes_path) as f:
            class_names = f.readlines()
        self.class_names = [c.strip() for c in class_names]
        with open(anchors_path) as f:
            anchors = f.readline()
        anchors = [float(x) for x in anchors.split(',')]
        self.anchors = np.array(anchors).reshape(-1, 2)

    def letterbox_image(self, image, size):
        '''resize image with unchanged aspect ratio using padding'''
        iw, ih = image.size
        w, h = size
        scale = min(w/iw, h/ih)
        nw = int(iw*scale)
        nh = int(ih*scale)

        image = image.resize((nw,nh), Image.BICUBIC)
        new_image = Image.new('RGB', size, (128,128,128))
        new_image.paste(image, ((w-nw)//2, (h-nh)//2))
        # new_image.show()
```

```
    return new_image

# decode output
def sigmoid( self, z ):
    return 1.0 / (1.0 + math.exp(-z))

def yolo_eval( self, yolo1, # [255,13,13]
          yolo2, # [255,26,26]
          anchors,
          classes_names,
          score_threshold=.6 ):
    """Evaluate YOLO model on given input and return filtered boxes."""
    num_layers = 2
    anchor_mask = [[3,4,5], [1,2,3]] # default setting
    boxes = []
    scores = []
    classes = []
    grid_num=[13,26]
    num_classes = len(self.class_names)

    for l in range(num_layers):
        output = yolo1 if l==0 else yolo2;
        num_grid=grid_num[l]
        grid_size = 416.0/float(num_grid)
        for row in range(num_grid):
            for col in range(num_grid):
                grid_output = output[:,row,col]
                for anchor in range(3):
                    stride = 5+num_classes
                    anc_output = grid_output[ anchor*stride : (anchor+1)*stride  ]
                    bc = anc_output[ 4 ]
                    if bc > 0.0:  # corresponds to 50% confidence
                        # Now true box found
                        box_confidence = self.sigmoid( bc )
                        # search most probable label
                        label = -1
                        max_conf = -10000.0
                        for lb in range(num_classes):
                            conf = anc_output[ 5+lb ]
                            if conf > max_conf:
                                label = lb
                                max_conf = conf
                        if label < 0 or max_conf < 0.0:
                            continue

                        class_confidence = self.sigmoid( max_conf )
                        total_confidence = box_confidence * class_confidence
                        if total_confidence < score_threshold:
                            continue

                        # decode position and size of boundering box
                        anchor_box = anchors[ anchor_mask[l][anchor] ]
```

```
                    tx = anc_output[ 0 ]
                    ty = anc_output[ 1 ]
                    tw = anc_output[ 2 ]
                    th = anc_output[ 3 ]
                    box_center_x = int( grid_size*( self.sigmoid(tx)+float(col) ) )
                    box_center_y = int( grid_size*( self.sigmoid(ty)+float(row) ) )
                    box_w = int( anchor_box[0]*math.exp(tw) )
                    box_h = int( anchor_box[1]*math.exp(th) )
                    box_half_w = box_w / 2
                    box_half_h = box_h / 2
                    box_x0 = box_center_x - box_half_w
                    box_y0 = box_center_y - box_half_h
                    box_x1 = box_center_x + box_half_w
                    box_y1 = box_center_y + box_half_h

                    # add to result
                    boxes.append( [ box_x0, box_y0, box_x1, box_y1 ])
                    scores.append( total_confidence )
                    classes.append( self.class_names[label] )
        return boxes, scores, classes

    def execute( self, img, drawBB=False ):
        #input_image_shape = [img.size[1], img.size[0]]  # size of original image
        model_image_size = [416, 416]
        boxed_image = self.letterbox_image(img, tuple(reversed(model_image_size)))
        image_data = np.array(boxed_image, dtype='float32')
        image_data /= 255.
        image_data = np.expand_dims(image_data, 0)  # Add batch dimension.
        image_data = np.transpose(image_data, (0, 3, 1, 2))  # (1,416,416,3) -> (1,3,416,416)

        # Execute network
        result = self.exec_net.infer({self.image_name: image_data})
        #print( result )

        # get result
        yolo1 = result[self.yolo1_name][0]
        yolo2 = result[self.yolo2_name][0]
        # print( yolo1.shape )    # (1, 255, 13, 13)
        # print( yolo2.shape )    # (1, 255, 26, 26)

        out_boxes, out_scores, out_classes = self.yolo_eval( yolo1, yolo2, self.anchors, self.class_names )

        if drawBB :
            font = ImageFont.truetype(font='font/FiraMono-Medium.otf', size=10 )
            thickness = 2

            for i, c in reversed(list(enumerate(out_classes))):
                predicted_class = c
                box = out_boxes[i]
                score = out_scores[i]

                label = '{} {:.2f}'.format(predicted_class, score)
```

```
            left, top, right, bottom = box
            print(label, (left, top), (right, bottom))

            draw = ImageDraw.Draw(boxed_image)
            label_size = draw.textsize(label, font)

            if top - 416 >= 0:
               text_origin = np.array([left, top - 416])
            else:
               text_origin = np.array([left, top + 1])

            # My kingdom for a good redistributable image drawing library.
            for i in range(thickness):
               draw.rectangle(
                  [left + i, top + i, right - i, bottom - i],
                  outline=(255,0,0))
            draw.rectangle(
               [tuple(text_origin), tuple(text_origin + label_size)],
               fill=(255,0,0))
            draw.text(text_origin, label, fill=(0, 0, 0), font=font)
            del draw

      return out_boxes, out_scores, out_classes, boxed_image

if __name__ == '__main__':
   # initialization
   nc2=NC2()

   # Prepare input image
   img_name = 'testPics/2035.jpg'
   img = Image.open(img_name)

   for i in range(10):
   out_boxes, out_scores, out_classes, boxed_image = nc2.execute( img, drawBB=True )
   print('Found {} boxes for {}'.format(len(out_boxes), 'img'))
   boxed_image.show()
```

Appendix 4: prototype package program [judgeRisk.py]

```
import math

class JR():
   def __init__(self):
      pass

   def judgeRisk(outBoxes, outScores, outClasses):
      pl, pm, pr = 0, 0, 0
      for idx, box in enumerate(outBoxes):
         score = outScores[idx]
         className = outClasses[idx]
```

```
            leftX = float(box[0])
            rightX = float(box[2])
            topY = float(box[1])
            bottomY = float(box[3])

            lX = (1920.0 / 416.0) * (leftX - 208)
            rX = (1920.0 / 416.0) * (rightX - 208)
            tY = -(1920.0 / 416.0) * (topY - 325)
            bY = -(1920.0 / 416.0) * (bottomY - 325)
            print('left_X, right_X, top_Y, bottom_Y of the bbox in new coordinate system are successively:')
            print(lX, rX, tY, bY)

            d = 960 / math.tan(math.radians(157.126/2))
            centX = (lX + rX) / 2
            # if 'Curb' in className:
            #     pass
            # if 'Step' in className:
            #     pass
            if 'Pole' in className:
                j = 1.0
                Ty = bY
                c = (Ty + 2520) / 3.75
                if centX > 0:
                    Tx = lX
                else:
                    Tx = rX
                if abs(Tx) > c:
                    k = 0
                else:
                    if Ty > 0.7 * 1080:
                        k = 0
                    else:
                        k = (c - abs(Tx)) / c
                cosθ = (Ty + d) / (((Ty + d) ** 2 + Tx ** 2) ** 0.5)
                w = j * (k * cosθ) * (1080 - Ty)
                print('cosθ = %f, risk_weight w = %f' % (cosθ, w))
                if w > 537.44:
                    if cosθ > math.cos(math.radians(7.5)):
                        pm += 1
                    else:
                        if Tx > 0:
                            pr += 1
                        else:
                            pl += 1
        return(pl,pm,pr)

if __name__ == '__main__':
    out_boxes0 = [[0, 91, 415, 324], [207, 207, 207, 207]]
    out_classes0 = ['dummy', 'dummy']
    out_scores0 = [0.98, 0.88]
```

```
        # img_name='/home/andy/labeledPics/srcPic2000-2099/2032.jpg'
        out_boxes2032 = [[137.0, 68.5, 153.0, 241.5], [132.5, 89.0, 147.5, 239.0]]
        out_classes2032 = ['elePole', 'elePole']
        out_scores2032 = [0.8951907304070944, 0.8407036674118816]

        # img_name='/home/andy/labeledPics/srcPic2000-2099/2035.jpg'
        out_boxes2035 = [[141.0, 97.0, 147.0, 181.0], [181.0, 63.0, 211.0, 285.0], [111.5, 229.0, 164.5,
301.0]]
        out_classes2035 = ['elePole', 'elePole', 'fssCurb']
        out_scores2035 = [0.6701313079194877, 0.8886470658492421, 0.6795031061903412]

        # img_name='/home/andy/labeledPics/srcPic2000-2099/2065.jpg'
        out_boxes2065 = [[147.0, 88.5, 155.0, 179.5], [168.0, 100.0, 176.0, 180.0]]
        out_classes2065 = ['elePole', 'elePole']
        out_scores2065 = [0.8059001722113326, 0.7047918864967497]

        pic0 = [out_boxes0, out_classes0, out_scores0]
        pic2032 = [out_boxes2032, out_classes2032, out_scores2032]
        pic2035 = [out_boxes2035, out_classes2035, out_scores2035]
        pic2065 = [out_boxes2065, out_classes2065, out_scores2065]

        testPic = pic2065
        out_boxes = testPic[0]
        out_classes = testPic[1]
        out_scores = testPic[2]

        jr = JR()
        jr.judgeRisk(out_boxes, out_scores, out_classes)
        print("The method 'judgeRisk' has been executed so far.")
```

Appendix 5: prototype package program [pwThreshold.py]

```
import math

class JR():
   def __init__(self):
      pass

   def judgeRisk(self, outBoxes, outScores, outClasses):
      wR = []  # weighted risks
      dR = []  # dangerous risks with weights greater than threshold
      for idx, box in enumerate(outBoxes):
         score = outScores[idx]
         className = outClasses[idx]

         print('\nleftX, topY, rightX, bottomY of the bbox in yolo coordinate system are respectively:\n', box)
         print('the confidence score of this bbox is:\n', score)
         print('the class name of this bbox is:\n', className)

         lX = float(box[0])
```

```
            rX = float(box[2])
            tY = float(box[1])
            bY = float(box[3])

            d = 960 / math.tan(math.radians(157.126/2))
            print('The constant locates camera virtual position is determined as constant d = %f' % d)
            centX = (lX + rX) / 2
            if 'Pole' in className:
                j = 1.0
                Ty = bY
                c = (Ty + 2520) / 3.75
                if centX > 0:
                    Tx = lX
                else:
                    Tx = rX
                if abs(Tx) > c:
                    k = 0
                else:
                    if Ty > 0.7 * 1080:
                        k = 0
                    else:
                        k = (c - abs(Tx)) / c
                cosθ = (Ty + d) / (((Ty + d) ** 2 + Tx ** 2) ** 0.5)
                w = j * (k * cosθ) * (1080 - Ty)
                print('cosθ = %f, risk_weight w = %f' % (cosθ, w))
                wR.append([className, w])
        print('\nThe weighted risks are listed as:\n', wR)
        return(wR)

if __name__ == '__main__':
    # img_name = '/Users/andx/PycharmProjects/keras-yolo3/testPics/vP_6mA.jpg'
    out_boxes6mA = [[-42, 1080, 25, 503]]
    out_classes6mA = ['elePole']
    out_scores6mA = [1.00]

    # img_name = '/Users/andx/PycharmProjects/keras-yolo3/testPics/vP_6mC.jpg'
    out_boxes6mC = [[-39, 1080, 30, 502]]
    out_classes6mC = ['elePole']
    out_scores6mC = [1.00]

    # img_name = '/Users/andx/PycharmProjects/keras-yolo3/testPics/vP_7mB.jpg'
    out_boxes7mA = [[-12, 1080, 35, 532]]
    out_classes7mA = ['elePole']
    out_scores7mA = [1.00]

    # img_name = '/Users/andx/PycharmProjects/keras-yolo3/testPics/vP_8mA.jpg'
    out_boxes7mB = [[-19, 1080, 30, 532]]
    out_classes7mB = ['elePole']
    out_scores7mB = [1.00]

    # img_name = '/Users/andx/PycharmProjects/keras-yolo3/testPics/vP_8mB.jpg'
    out_boxes8mA = [[-26, 1080, 14, 536]]
```

```
out_classes8mA = ['elePole']
out_scores8mA = [1.00]

# img_name = '/Users/andx/PycharmProjects/keras-yolo3/testPics/vP_8mC.jpg'
out_boxes8mC = [[-24, 1080, 15, 536]]
out_classes8mC = ['elePole']
out_scores8mC = [1.00]

pic6mA = [out_boxes6mA, out_classes6mA, out_scores6mA]
pic6mC = [out_boxes6mC, out_classes6mC, out_scores6mC]
pic7mA = [out_boxes7mA, out_classes7mA, out_scores7mA]
pic7mB = [out_boxes7mB, out_classes7mB, out_scores7mB]
pic8mA = [out_boxes8mA, out_classes8mA, out_scores8mA]
pic8mC = [out_boxes8mC, out_classes8mC, out_scores8mC]

testPic = pic6mA
out_boxes = testPic[0]
out_classes = testPic[1]
out_scores = testPic[2]

jr = JR()
jr.judgeRisk(out_boxes, out_scores, out_classes)
```

Appendix 6: prototype package file [class_names.txt]

```
fssCurb
bssCurb
trvCurb
elePole
saPole
spPole
fssStep
bssStep
trvStep
```

Appendix 7: prototype package file [YoloTiny_Anchors.txt]

```
2,2, 2,5, 3,2, 6,9, 13,5, 40,30
```

Appendix 8: prototype package file [yolo3_tiny.xml]