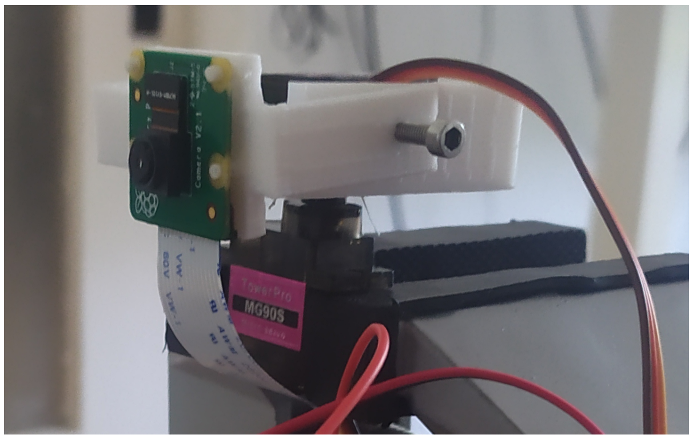
An Analysis on Object Detection and Tracking with a Tilt-Pan Camera on an Embedded Device

Fynn Luca Maaß

Hamburg University of Applied Sciences
fynnluca.maass@haw-hamburg.de



Abstract. This paper presents a complete solution for real time, 30 frames per second, object detection and tracking using a SSD architecture for object detection in combination with a proportional controller for tilt-pan camera adjustments for tracking. It addresses several problems including the lack of real time capability of the SSD architecture on low powered devices and the choice of the right feedback controller for this application. The real time aspect is analyzed by comparing different input resolutions at different frame rates across the entire system. It is found, that the resolution size correlates with the inference time of the neural network in a proportional manner, achieving up to 3.5ms at a resolution of 160x120x3 pixels with special hardware acceleration. It is also found that a proportional controller is better suited than a proportional-integral-(derivative) controller in this context by looking at the unit step responses. The interaction between object detection and tracking is investigated in tests of oscillating motions. The system can track a red ball in a pendulum motion with more than 160°per second. It proves the real time capability of the developed system and builds a foundation that can be adopted in other applications.

Keywords: Real Time Object Detection · Object Tracking · Single Shot Detection · Embedded System · Deep Learning · Neural Network · Proportional Integral Derivative Controller · 3D-Printing · Prototyping

1 Introduction

Object detection is part of machine vision and a popular task in computer science with a wide variety of applications such as autonomous driving, face detection, video surveillance or human-

robot interaction. Many of those tasks are time critical, require expensive hardware and are not suited for real time computation. In 2015 this problem was addressed by an architecture named "You Only Look Once" also known as "YOLO" [7] and later in the same year with a different architecture called "Single Shot multibox Detectior" or "SSD" [5], outperforming the accuracy and speed of "YOLO" while using lower resolution images. As the name of both indicates they are so called one-stage detectors, meaning they are capable of recognizing object(s) and predict their position(s) in one run through the neural network whereas previous two-stage architectures first predicted a region of interest and then ran a classification on that region. One-stage approaches have a simpler architecture and a much faster inference time. However, this methods still take a high-end GPU to achieve a reasonable performance of 30+ Frames Per Second (FPS). In order to run a real time object detection on a low-performance device the architectures and underlying networks still need to be tweaked.
This is done in this paper and then used to perform actual object following or rather tracking with a camera sitting on a fixed pan-tilt construction keeping the desired object in the center of the camera while keeping good speed and accuracy in terms of the neural network output as well as the camera alignment.
In 2016 NVIDIA proposed an also potentially for this problem applicable End to End solution for self driving cars [1] by just using the images taken from the front camera and the current steering wheel angle as a label to train a neural network. As a result the car was able to keep track. The used neural network is relatively small and fast, because there where no unnecessary human features to learn such as lane markers and certain traffic signs. This would essentially eliminate the need of a controller that performs the object following in this application. Unfortunately due to the lack of labeled training data this method is not feasible for this use case.
Therefore two major tasks need to be accomplished. The first one being the optimisation of overall computation time in respect to the neural network's inference time and image taking- and loading time. The second one being the optimisation of the controller used to steer the pan-tilt motors. Both will then be evaluated given response times, accuracy and general speed when following a red ball. The target is to archive full 30FPS object detection and following as this is considered "real time object detection" in this context.

2 Materials and Methods

This section covers all steps and methods necessary to understand and reproduce the results in section 3. There is an existing repository [6] for this paper containing, source code, the trained neural network, further examples from the training dataset, raw measurement data and 3D model files.

2.1 Baseline for Object Detection

Neural Network Architecture The baseline model is a ssd-mobilenet-v2-quantized-coco from the TensorFlow detection model zoo [8]. The underlying Neural Network (NN) is a mobilenet, whitch is a special architecture for low power mobile devices. The model is pre-trained with the COCO dataset and capable of recognizing 80 different objects. In addition the model is quantized, which improves the speed. By default the model expects input in the shape of [300,300,3]. The model is converted and compiled for the target hardware (see section 2.1 Target System) and then used in the object detection example of TensorFlow Lite for the Raspberry Pi [9], written in Python3, using the TensorFlow Lite runtime.

Target System The foundation of the target system is a Raspberry Pi 4 with 2GB Ram and Raspberry Pi OS. A Raspberry Pi Camera v2.1 is plugged into it for capturing images. This camera has a field of view of 62.2°horizontally and 48.8°vertically and is capable of filming with up to 30FPS while maintaining the Field Of View (FOV) and up to 90FPS while downsizing it. A Coral Edge TPU is plugged into the USB 3 port of the Raspberry Pi. This special piece of hardware is responsible for the computation of the NN and decreases the inference time significantly. In order to run a NN on that TPU it needs to be compiled using the Edge TPU Compiler.

Transfer Learning the Neural Network The pre-trained ssd-mobilenet-v2-quantized-coco is transfer leaned in order to recognize a red ball that is used in further testing. The key aspect is to lower the expected input dimension from images with 300x300x3pixels to images with 160x120x3pixels that are matching the cameras aspect ratio. The self made dataset contains 450 images in total with a variety in distance, lightning, backgrounds and positions all containing a red ball. They are manually labeled with a class and coordinates. A few examples from the images are shown in Figure 1. The dataset is split into 350 training and 100 test images. To encounter the problem of distinguishing fore- and background that arises when training a (regional based)classifier only using one class, a technique called "hard example mining" is applied. Even though reducing the input dimension does not reduce the trainable parameters of a Convolutional Nerual Network (CNN) significantly, it does help with the computation time. The inference time as well as the overall time the system needs for preprocessing the image from start to finish within one frame is compared between both 300x300x3 and 160x120x3 images. With this information we will also decide, whether or not a trade off between the FPS and FOV is useful for the final application. For time measurement we use timestamps generated by our Python code. This might not be very accurate, but gives us a good enough impression. The Python program used for this is a heavily modified version of the object detection example [9] from TensorFlow with additional arguments such as resolution and FPS for better testing. Additionally the confidence of the classifier is analyzed over distances ranging from 0cm to 200cm with the red ball in the center of the camera.

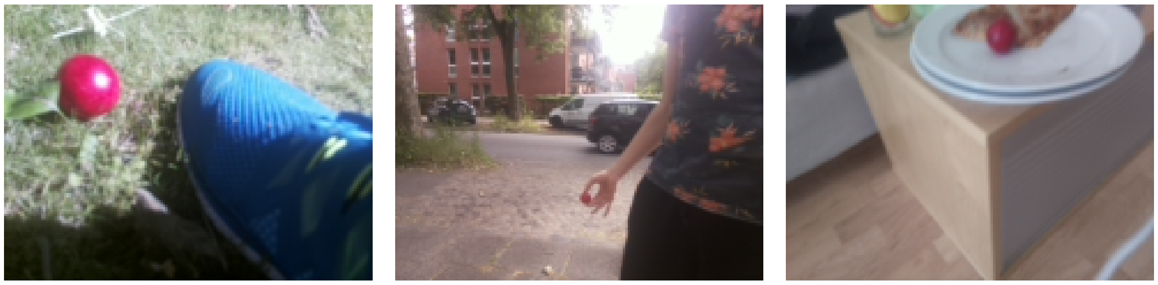


Fig. 1: Examples from the Test Dataset

2.2 Adding the Object Tracking Feature

Mechanical Hardware The construction responsible for moving the camera in order to follow the object consists of two MG90S servo motors attached to custom 3D-Printed brackets and are controlled via hardware Pulse Width Modulation (PWM) from the Raspberry Pi. The first one is moving the camera on the horizontal axis, the second one in the vertical axis. The resting position of the system is defined by the angles of the servos when the system is started. These are 94.5°horizontal and 108°vertical, aligning the cameras facing direction perpendicular to the background wall.

Feedback Controllers The location of the red ball in the image is given by the neural network in pixels and is described as the process variable (PV). The desired setpoint (SP) of the object is the middle of the camera. An error value (e(t)) is calculated by the difference from PV and SP. Therefore a control loop with feedback is needed to adjust the camera based on the error so that the object appears in the center of it.
A range of different Feedback Controller (Proportional (P), Proportional Integral (PI), Proportional Derivative (PD), Proportional Integral Derivative (PID) respectively) are tested for this use case because of the relatively easy implementation on a discrete system. The proportional gain (K_p), integral gain (K_i) and derivative gain (K_d) where adjusted by the Ziegler-Nichols heuristic method [3]. Therefore we need to start with a P controller at a low proportional gain and increase it until the point is reached, where steady oscillation occurs as a response to a minor change in PV. We take this proportional gain and set it as the ultimate gain (K_u) and measure the oscillation period T_u. This is done with both axes independently while the other one is turned off. It is found that the $K_u = 0.95$ and $T_u \approx 0.3s$ (297ms/302ms) is approximately equal for the horizontal and the vertical axis. According to Ziegler-Nichols we calculate our K_p, K_i and K_d for both axes as seen in table 1. The existing Python source code is expanded by the implementation of the PID controller. The

Controller Type	K_p	K_i	K_d
P	$0.5K_u = 0.5 \cdot 0.95 = 0.475$		
PI	$0.45K_u = 0.45 \cdot 0.95 = 0.4275$	$T_u/1.2 = 0.3/1.2 = 0.25$	
PD	$0.8K_u = 0.8 \cdot 0.95 = 0.76$		$T_u/8 = 0.3/8 = 0.0375$
PID	$0.6K_u = 0.6 \cdot 0.96 = 0.57$	$T_u/2 = 0.3/2 = 0.15$	$T_u/8 = 0.3/8 = 0.0375$

Table 1: Formulas given by the Zigler-Nichols method for calculating the controller components

evaluation of the P, PI, PD and PID controller is done with a look at the response to a unit step for the horizontal axis. This unit step is generated by hanging a red ball 35cm away and placing it as close to the edge of the FOV as possible on the investigated axis and then turning on the system.

2.3 Testing Object Following

In order to track the red ball in motion two different motion patterns are used. These two are pendulum motion and free fall motion with bouncing. Both are fairly similar but compared to a "best effort moving by hand" they are deterministic and repeatable whitch is very important for comparison. A disadvantage is that they are non linear motions and are therefore not good for analysing the sheer speed an object can pass in front of the camera. Nevertheless, they are easy to reproduce and are the most accurate movements that can be performed at home.
The pendulum motion is performed with an amplitude of 45°and different lengths of the rope (15cm, 30cm 45cm) across distances 10cm, 20cm, 30cm 40cm and 50cm away from the camera with the equilibrium position right in the center of the camera in its resting position. There are multiple start setups.

- The camera is in its resting position when the pendulum starts, meaning the ball will enter the FOV.
- The camera is adjusted to the starting point of the pendulum, meaning the ball is within the FOV.
- The camera is in its resting position when the pendulum starts, meaning the ball will enter the FOV. In addition there is an obstacle in the middle of the resting position covering about 12°FOV horizontal and 100% vertical FOV.
- The camera is adjusted to the starting point of the pendulum, meaning the ball is within the FOV. In addition there is an obstacle in the middle of the resting position covering about 12°FOV horizontal and 100% vertical FOV.

The free falling bouncing motion is accomplished by the ball falling 50cm above the ground at various distances to the camera whitch is 18cm above the ground. Likewise to the pendulum experiment, the ball will enter the FOV as well as be within it when the experiment starts. In both motions the angles of the motors, a timestamp and a flag whether or not a ball is recognized in a sample are logged. For simplicity reasons the background is a white wall. Due to the perspectives there are not always experiments where the system starts in the resting position (e.g. pendulum length 30cm - distance to camera 50cm), because the ball is already visible.

3 Results

3.1 Time decrease

Table 2 shows that the inference time improves with a lower input dimension. Independently from the FPS the inference time of the custom transfer learned mobilenet_ssd_v2 (expects 160x120x3 as input dimension) is around 3.5ms and roughly 5 times lower than the equivalent architecture requiring 300x300x3 inputs and therefore roughy 5 times more pixels. It is to be noted that the remaining processing time is dependent on the frame rate of the camera and cant be lower than the sampling time. Noteworthy is that we in no case achieve faster speeds than 30ms for the processing time, this is confirmed with even lower resolutions than 160x120pixels. In addition to that, it is also interesting to see that a significant increase in time takes place at 300x300pixels 30FPS. There is no improvement in time observed on different frame rates at a 160x120pixels and therefore no need to use higher frame rates, because the FOV would be downsized (e.g. 30FPS to 90FPS at about the factor 2.5 [4]) essentially reducing our maximum theoretical speed of tracking an object.
Based on the results we use the custom trained mobilnet_ssd_v2 with inputs of 160x120x3 at 30FPS for the further project.

Transfer Learned ssd_v2			
resolution and frame rate	inference / ms	remaining processing / ms	total / ms
160x120x3@90FPS	3.6	32.2	35.8
160x120x3@60FPS	3.6	32.7	35.8
160x120x3@30FPS	3.5	29,8	33.3

Pre Trained ssd2̌			
resolution and frame rate	inference / ms	remaining processing / ms	total / ms
300x300x3@90FPS	18.8	30.3	49.1
300x300x3@60FPS	17.6	32.5	50.1
300x300x3@30FPS	17.4	50.1	67.5

Table 2: Time comparison between different image resolutions and frame rates averaged on 100 samples

3.2 Controller Type

Figure 3.2 shows the response to a unit step on the implemented P and PI controller. The diagrams are showing the angle of the horizontal axes of the system with respect to the time, each sample represented as a dot takes around 33ms. It can be seen that the P controller performs very well in terms of speed and overshoot. The system nearly settles completely within 0.8 seconds with coming close to the desired set point within 0.13 seconds. The complete process is done at around 5.5 seconds with minor changes to the angle, barley noticeable for an observant next to the system.
The PI controller shows a different behaviour with long term zig-zag oscillation. Within the first 0.16 seconds the PI controller also comes close to the set point, however the settling time is too long for the time frame. You can see that the impact of the Integral component decreases as time goes on seen on the lowering gradient with that the integral tries to act against. This is counter acted by the proportional controller that lowers the angle. Notable is, that this counter action is not continuous rather taking place at a fixed angles.

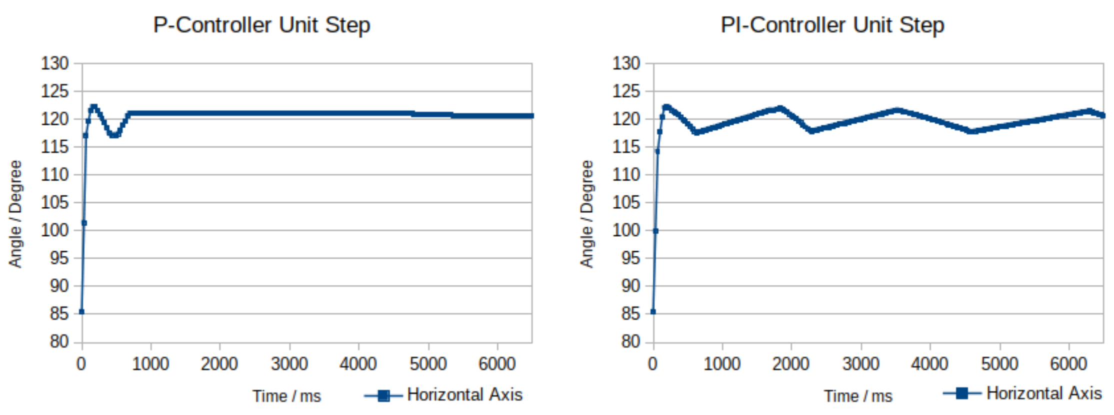


Fig. 2: Unit Steps on Horizontal Axis Compared on P and PI Controller

The response to a unit step for a PD and PID controller is very random and neither came close to the desired setpoint. The output of the controllers are exceeding the maximum values the system can handle and is due to the rapid changes dangerous to the hardware. Lowering the D component shows no effect in its behaviour. In further testing the P controller is used.

3.3 Classifier Certainty

Figure 3 shows the overall classifier certainty over distance as described in section 2.1. The best precision was achieved within 25cm to 35cm and 65cm to 80cm with well above 0.7. Very noticeable is the dip around 50cm where the certainty drops significantly down to 0.2. Everything below 20cm and above 110cm shows very low and noisy confidence. The red colored part in the graph illustrates the "area of interest", meaning the range in where the most tests for the following section are performed. In the further experiments, the classifier certainty threshold is 0.2 whitch is still enough to have no false positives in the test setup. Figure 4 illustrates the classefier certaitinty on three images of the test dataset.

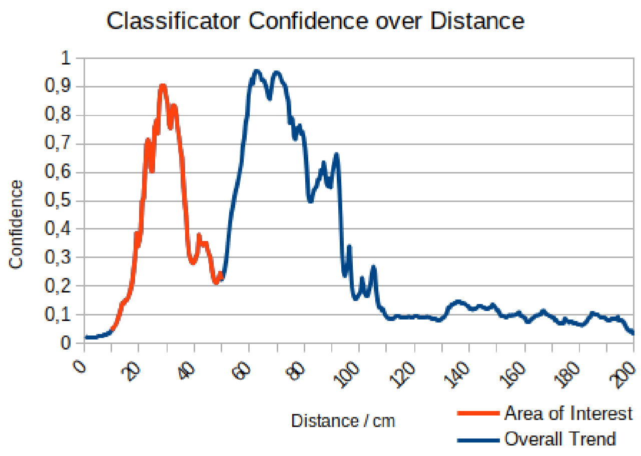


Fig. 3: Classifier Precision over Distance with Moving-Average (n = 3) post processing

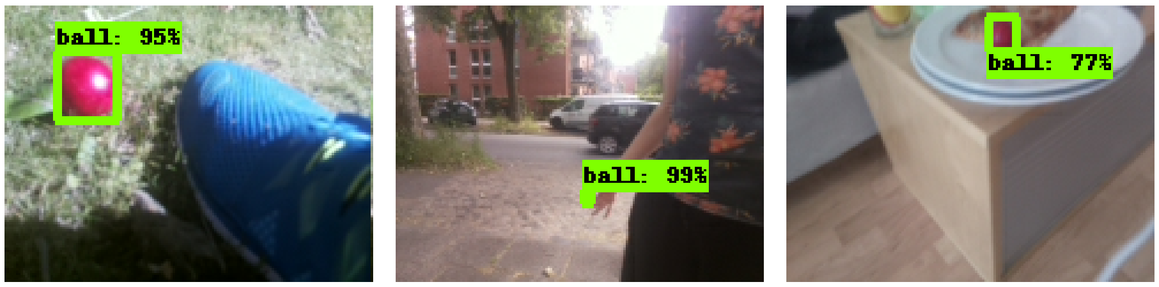
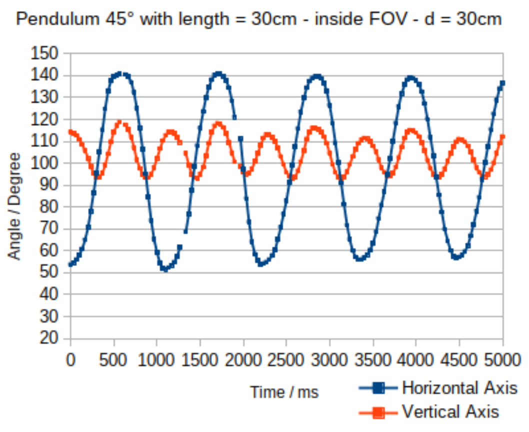


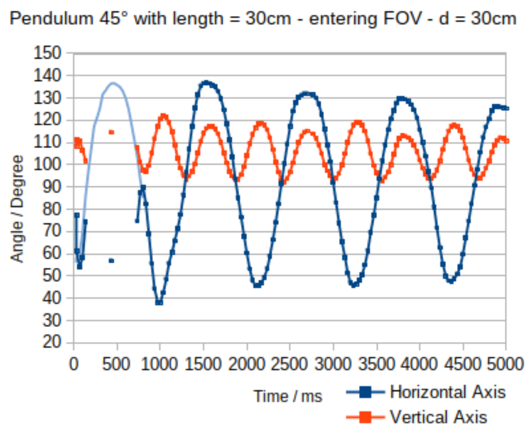
Fig. 4: Evaluated Examples from the Test Dataset (distances ≈ 25cm, 80cm, 35cm)

3.4 Tracking Behaviour on Oscillating Motions

Pendulum Motion Figure 3.4 shows the tracking behaviour in a scenario where the ball oscillates on a 30cm long rope 30cm away from the camera with an starting angle of 45°. Both graphs differ in the starting scenarios in regard to the resting vs. focused starting position. There is no obstacle in the frame. In addition, the right figure also shows a predicted path in light blue for better reference. Both graphs show similar results on the axis in terms of the horizontal deflection (±45°) and vertical deflection (±25°). The object detection failed three times at random places in the left graph and 17 times, 16 of whitch within the first period of the pendulum resulting in skipping it, on the right graph. Figure 5b is interesting, because it appears that the system at first adjusts against the moving direction of the pendulum as soon as the ball enters the FOV and shortly after that changing the direction and moving with it, until it looses the ball. This can be seen as the little parabola on the horizontal axis right at the start. At about 800ms the same effect can be observed, but this time the system is able to keep up.



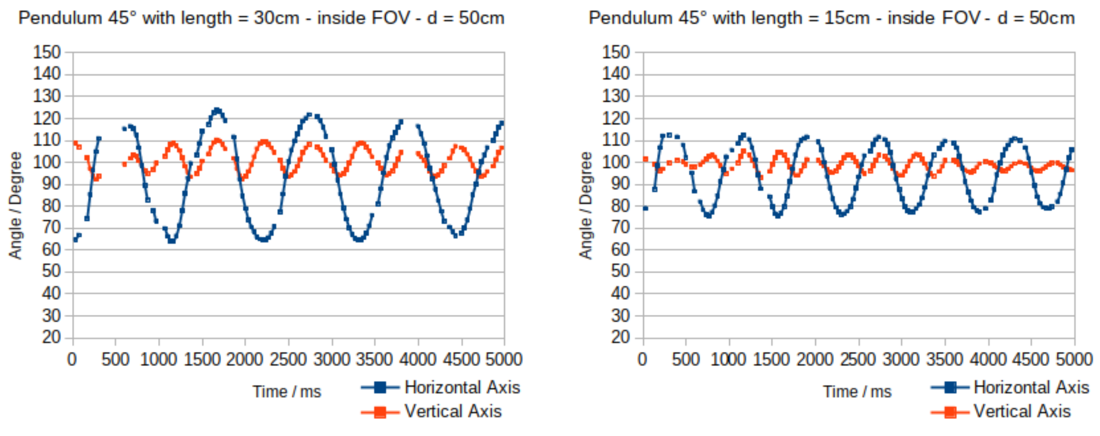
(a)



(b)

Fig. 5: Pendulum Tracking at 30cm Distance with Different Starting Scenarios

The two graphs in figure 3.4 are also showing the tracking behaviour of a pendulum movement. In both graphs the red Ball is 50cm away and the system starts in its resting position without an obstacle. The pendulum length in Figure 6a is 30cm and in Figure 6b 15cm. In 32 of the 150 samples (0.21%) in Figure 6a the ball could not be recognized by the system. None of whitch where missed at the fastest points in the pendulum period. A similar result is in 6b where the ball is not recognized in 27 out of 152 samples (0.18%). The maximum deflection for the systems angles are ±30°horizontal and ±15°vertical in Figure 6a and ±20°horizontal and ±10°vertical in Figure 6b. Even though a significant amount of samples are not recognized, by looking at the the system in person while performing this particular experiment, the lower recognition rate is not noticeable to the human eye.



(a) (b)

Fig. 6: Pendulum Tracking at 50cm Distance with Different and Pendulum Length

Pendulum experiments with a distance ranging from 20cm to 40cm are very similar to the presented results for 30cm. Distances lower than 20cm are similar to worse than distances around 50cm. Everything closer than 5cm is definitely not usable.

Pendulum Movement with Obstacle When the line of sight is partially covered due to an obstacle as described in 2.3 no remarkable effect is found. As soon as the red ball is behind the object, the system does not recognize it and also is not adjusting until it appears on the other side. The System has no problem on catching up when it appears on the other side with a pendulum length of 30cm with an starting angle of 45°while its standing 30cm away from the camera in both cases, entering the FOV and already starting inside the FOV. It is harder for the system to catch up, when the starting position of the system is the resting position though. The obstacle does amplify the trouble of tracking the red ball at a distance of 50cm slightly, but it is still fine for the human eye.

Free Falling Movement Investigating the tracking of a free falling ball does not give any additional insights. The same effects are seen in the pendulum experiments across the distances.

4 Discussion and Conclusion

4.1 Time Analysis

Decreasing the resolution of the images is directly visible in the decrease of the inference time in a proportional manner and is expected. The result that the remaining processing is stuck at a constant level independently from the FPS is not expected and points to a bottleneck issue on the Raspberry Pi itself. Further investigations on this issue in regard to the video format and encoder reinitialization for each capture [2] could be very beneficial towards this problem.

4.2 Classifier Confidence

The classifier confidence being relatively low, in many cases under 0.5, may seem not very good. But a confidence threshold of 0.2 is enough to perform good object recognition with a white background. If the background is more diverse like in the training data, a higher threshold, e.g. 0.6, can be applied in a trade off to the recall. With this findings it can be stated, that the confidence score in this application does not say a lot about the real world performance of the system. This is probably due to the "single-class-problem" that was encountered while training this network. It is assumed, that with more classes, the confidences would be better.
The lag of confidence below 20cm is correlated to the training logs of this network addressing the mean average precision of small, medium and large bounding boxes. The average precision of large bounding boxes is -1. This is because the training set does not contain enough close up images of the ball. This might also explain the confidence drop around distances of 50cm, but can not be backed with the training logs. The solution to this problems is more or evener distributed training data. Everything further away than 110cm is to small to be even detected by a small bounding box.

4.3 Controller

The results show that more feature components in a controller could end up worsen the application a lot. The designed tracking system is a rapidly changing system, also due to the fact that the servo motors can not perform continuous rotation. An integral component is more feasible in such continuous motions without a lot of directional change. A rapid change of the system is something like the unit step performed in section 2.2 where the PI controller is measurably worse than the P controller. An effect called "integral windup" can be observed in this case whitch is responsible for the long term zig-zag oscillation. It is unclear why the counter action of the proportional component against the windup is at fixed angles. A controller with a derivative component however could be useful in this use case, but because it depends on the derivative of our motion it is sensitive to noise and measurement errors whitch is not handled in any pre processing. In combination with a relatively low sample frequency compared to the rapid changes in between the samples at the beginning of the unit step, it builds the reason for its unsuitability.

4.4 Oscillating Motions

The investigations on oscillating motions prove the real time object tracking capability of the overall system. Although some flaws are within the test setup including the pendulum motion not be perfectly symmetrical to the camera and the falling ball not hitting the same spot over and over again it is fair to say that the result is very good. In the best cases the system is easily capable of tracking the ball across a total of 160°within one second without any indication that the sheer speed of the ball is a problem. Problematic is however when the ball enters the FOV from outside and the system at first adjusts in the opposite pendulum direction effectively increasing the speed of the ball and maybe loosing it due to the controller overshoot. It is important to keep in mind, that missing a certain amount of detections can inevitably result in missing even more until the ball is visible again. The immediate stop of adjustment when the ball is not visible is the expected behaviour when using a P controller. With a PI controller this would not necessarily happen. Thoughtful is the use of additional data preprocessing that comes from the NN. For instance a low pass filter against noise or the use of a Kalman filter for predicting the objects motion.

Glossary

CNN Convolutional Nerual Network.

Feedback Controller The in this paper described P, PI, PD, PID controllers are commonly used in the industry. The P component means, that the controller proportionally adjusts the system to the measured error (deviation of the supposed value). The Integral component takes the previous errors into account by summing them up to an integral, essentially eliminating a steady offset. This could be useful when the P component adjusts by the same rate as the red ball object in this case moves away. The derivative component responses to how fast the changes taking place by approximating the current gradient base on the current minus the last error.
If the current error gives you a velocity, then the integral component is the position and the derivative is the acceleration.

FOV Field Of View.

FPS Frames Per Second.

NN Neural Network.

P Proportional.

PD Proportional Derivative.

PI Proportional Integral.

PID Proportional Integral Derivative.

PWM Pulse Width Modulation.

Bibliography

[1] Bojarski, M., Del Testa, D., Dworakowski, D., Firner, B., Flepp, B., Goyal, P., Jackel, L. D., Monfort, M., Muller, U., Zhang, J., Zhang, X., Zhao, J., and Zieba, K. (2016). End to End Learning for Self-Driving Cars. *arXiv e-prints*, page arXiv:1604.07316.

[2] Hughes, D. (2017). Rapid capture and processing. `https://picamera.readthedocs.io/en/release-1.8/recipes2.html#rapid-capture-and-processing`.

[3] Jenkins (Unknown). Tuning for pid controllers. `http://faculty.mercer.edu/jenkins_he/documents/TuningforPIDControllers.pdf`.

[4] Jones, D. (2017). Camera hardware. `https://picamera.readthedocs.io/en/release-1.12/fov.html`.

[5] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C.-Y., and Berg, A. C. (2015). SSD: Single Shot MultiBox Detector. *arXiv e-prints*, page arXiv:1512.02325.

[6] Maaß, F. L. (2020). Project repository. `https://git.haw-hamburg.de/acn814/eml-tilt-pan-object-tracking`.

[7] Redmon, J., Divvala, S., Girshick, R., and Farhadi, A. (2015). You Only Look Once: Unified, Real-Time Object Detection. *arXiv e-prints*, page arXiv:1506.02640.

[8] Tensorflow (2020a). Tensorflow detection model zoo. `https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/detection_model_zoo.md`.

[9] Tensorflow (2020b). Tensorflow lite python object detection example with pi camera. `https://github.com/tensorflow/examples/tree/master/lite/examples/object_detection/raspberry_pi`.