Robin Yan - 7520

Discovery Log for Using EdgeTPU on Pi to Perform Note Detection

Introduction

In order to do complex autonomous motion, the robot must be able to know its position at all times, and how to get to a specific location. This requires vision detection of april tags using photonvision and game pieces. Previously, using a HD 1080p microsoft logi webcam connected to a raspberry pi 4 could detect April tags, but not well enough for us to use. When we want to approach an april tag, often we must stop for a second before photonvision can detect it. Additionally, the robot must move really slowly towards the april tag otherwise it becomes too blurry and loses track. With the resources we had then, we simply assumed that it was a waste of time to integrate camera detection because it wasn’t fast enough to benefit us in game - our human senses worked much better and faster.

When looking at the problem of blurry images coming from the camera, someone might assume that it’s because the frame rate is too low, or in other words the camera shutter is too slow and thus causing a blur. In photography, a faster shutter allows someone to take more precise pictures for an object in motion. Technically speaking, if we could increase the shutter speed, the blur is removed and the frame rate is increased too. However, blur is also created as a result of how a camera loads each pixel. Commonly, cameras do not load all pixels in an entire frame at the same time, but instead it updates sections of pixels one at a time within the frame; meaning the first row of pixels updates, then the second row, then the third row. Thus, when an object moves while the sections of pixels are being updated, each previously updated row or column technically delayed compared to the incoming updated pixel sections; this can distort a frame and contribute to blur. Most cameras have a Rolling Shutter because they are cheaper, less power consuming, and simpler than what we call Global Shutters - as the name defines it captures all pixels at once. A global shutter is expensive, but allows for faster shutter speeds and distortionless frames. Historically, global shutters had slower readout speeds, but nowadays it’s faster and often performs better than rolling shutters.

Shutter Speed - the speed of how fast the camera shutter is open for. Responsible for motion blur and exposure

Frame Rate - the number of frames a camera can capture a second. This is dependent on readout speed

Readout Speed - the rate of how fast a frame can be captured and processed. Generally, faster readout speed means capability to get higher frame rates

Rolling Shutter Distortion - the jello effect, slant of an object in motion, caused from reading pixels row by row instead of all together

Motion Blur - produced from a slow shutter which lets in light for longer, and thus the light of moving objects trail across the image sensor, producing a blur

Previously, the rasp pi 4 + webcam did not produce fast enough results. This is both dependent on the camera quality and the processing power of the pi. The global shutter is mainly to target april tags since it requires precision, and to make the most out of its ability requires a stronger computer; thus using the orange pi. Additionally, the global shutter camera is black and white, reducing the data transfer size of colour information - which is why game piece detection cannot use this camera as the AI model requires colour. A rolling shutter is fine for game piece detection as the model doesn't require perfectly clear frames to discern a note. Now with the global shutter camera tested for frame rate and accuracy as well as edgeTPU for game pieces running, there are still a few things to resolve. Just to clarify, the reason we used a TPU instead of photonvision for game piece detection is 1. More accurate AI model compared to simple colour detection (though this one is debatable because the AI Model currently being used isn't the best) and 2. Higher performance speed by letting accelerator perform heavy calculations rather than forcing all of it on CPU of pi, and 3. Better flexibility using default ubuntu.

First, both april tag and game piece detection must be working at the same time. We can have one pi access two camera feeds and process both at the same time, but two cameras introduce an ID problem, Ubuntu version incompatibility with TPU, and problems with running photonvision-ubuntu first before TPU. If we use two pi's, we need two ethernet ports but the radio only supports a max 2 etherports, one already occupied by the Rio. Either we need another legal radio with 3 or more ports, or we use an ethernet switch. According to WestCoast Products, there is $180 4 port radio, but these radio's are in testing phase, planned to go official for 2025.

For CNE 2024, this radio was listed as not supported until 2 weeks prior, so we'll set up a switch and the new radio.

If we decide to use two separate pi's for the two detection tasks, we can use the Raspberry Pi 4 + EdgeTPU to perform game piece detection. We'll start by setting up ubuntu 20.04 on the pi, installing/downloading required repositories and environment properties for the TPU, then modifying certain elements of the system and code to make it suitable for usage.

NOTICE: This is a log for setting up a raspberry pi 4 with the edge tpu. All notes here are for the command line in linux. Additionally, I tested out installing requirements for orange pi, but I did not finish them. As such, this log contains information which may not necessarily be useful to the reader who does not care about orange pi. The log is basically an entire history of trial and error - **it is not a linear step-by-step guide but rather a documented learning process.**

Process

Installing Basics

```
# To install python 3.7 onto Ubuntu 22.04...

sudo apt install software-properties-common -y

sudo add-apt-repository ppa:deadsnakes/ppa -y

sudo apt update

sudo apt install python3.7


# To install OpenCv. ..

pip3 install opencv-python


# To install Mjpeg-streamer

pip install mjpeg-streamer
```

```
# To install EdgeTpu Runtime. *

echo "deb https://packages.cloud.google.com/apt coral-edgetpu-stable
main" | sudo tee /etc/apt/sources.list.d/coral-edgetpu.list


# Make sure curl is installed so you can install the repositroy from
google. Also check if you have spelled everything correctly. Note*
"curl https://packages.cloud.google.com/apt/doc/apt-key.gpg | sudo
apt-key add -" has been changed because apt-key is depecrated...

sudo apt-get install curl

curl -fsSL https://packages.cloud.google.com/apt/doc/apt-key.gpg |
sudo gpg --dearmor -o
/etc/apt/trusted.gpg.d/google-cloud-packages-archive-keyring.gpg

sudo apt-get update

sudo apt-get install libedgetpu1-std

sudo apt-get install python3-pycoral
```

Python Version Problem For Ubuntu 22.04

Attempts to Change System Python

```
# To downgrade the base python on ubuntu so you can install pycoral
library

sudo update-alternatives --install /usr/bin/python python
/usr/bin/python3.7 1


# To downgrade python3 to python 3.7 instead of 3.10

sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.10 1
```

```
sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.7 2

sudo update-alternatives --config python3

sudo apt-get remove python3.10

sudo update-alternatives --remove python3 /usr/bin/python3.10


# Even downgrading to 3.7, still fails to download pycoral library.

sudo apt-get purge python3.10

sudo apt-get autoremove

sudo apt-get purge python3.10 python3.10-minimal

sudo apt-get autoremove

sudo apt-get clean

sudo apt-get clean


# Purging all of 3.10, a new error displays, and python3 was removed
in the process. Must reinstall, and repurge

sudo apt-get install python3.7 python3.7-venv python3.7-dev

sudo update-alternatives --install /usr/bin/python3 python3
/usr/bin/python3.7 1

sudo apt-get purge python3.10 python3.10-minimal

sudo apt-get autoremove

sudo apt-get clean
```

Using Isolated Environments

```
# It seems like the system simply doesn't allow pycoral to work.
We'll need to set up an environment using Miniconda. First install
```

```
mkdir -p ~/miniconda3

wget
https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
-O ~/miniconda3/miniconda.sh

bash ~/miniconda3/miniconda.sh -b -u -p ~/miniconda3

rm -rf ~/miniconda3/miniconda.sh

~/miniconda3/bin/conda init bash

~/miniconda3/bin/conda init zsh


# After reopening the terminal, create the environment named py37 and
dependencies for pycoral

conda create -n py37 python=3.7

conda activate py37

pip install numpy pillow tflite-runtime pycoral
```

Running Projects and Debugging

```
# Continue with running a test example from Coral.ai. Make sure you
install the git command

sudo apt-get install git

mkdir coral && cd coral

git clone https://github.com/google-coral/pycoral.git

cd pycoral

bash examples/install_requirements.sh classify_image.py

python3 examples/classify_image.py \
--model test_data/mobilenet_v2_1.0_224_inat_bird_quant_edgetpu.tflite
\
--labels test_data/inat_bird_labels.txt \
--input test_data/parrot.jpg
```

```
# Run into an error saying ModuleNotFoundError for "from
pycoral.adapters import classify" in line 37 of classify.py in
examples folder. When the python script is under the "examples"
folder, this issue occurs. When run outside in the pycoral folder, no
error occurs.

# I discovered that pycoral supports python 3.9, so I tried updating
to see if that helped. Made a new environment. Installed pycoral, and
moved the classify.py out from examples into pycoral main. A new
error, No module named 'PIL.' So my guess is that it's trying to
import things from different folders and theres some error happening.
When I switch back to python 3.7 conda environment, the same script
runs PIL import but not pycoral.adapters...

# ChatGPT has tried told me to add ".." in front of the line to
return to the parent package, but pycoral main folder is not a
pacakge, so it doesn't work. I've never run into this issue on
windows, so I'm not sure whats going on. Again, all these error is
coming from the example project to classify an image.


# To install the 7520 edge tpu stuff and network tables

pip install pynetworktables

git clone https://github.com/Team7520/EdgeTPU-Vision


# Make sure the ip address you use to stream your cam feed is valid.
Each internet or ethernet connection has a different ip address. If
you just want to test the camera feed and not stream to another
device, use the local ip address 127.0.0.1:8080. Otherwise, expect a
very long error message. Use the command, ip a, to check what ip
addresses are available.

# cap.release() from line 23 is not defined for main.py from EDGETUP
folder? Worked on windows, don't know why it has a problem here. I
removed the exit_handler function and the line that calls it, and
added the exit lines (cap.realse() and cv2.destroy... to the very
bottom). Be sure to connect your edgetpu while you run.

# I ran main.py again, and ran into some error, INFO: Created
TensorFlow Lite XNNPACK delegate for CPU.
```

Segmentation fault (core dumped), so I installed tensor flow to see if it helped

```
pip install tensorflow==2.5.0
```

# Didn't seem to resolve the issue. Apparently, this error comes from the fact that the computer is redirecting the processing stuff to the CPU from the GPU, but I literally have the TPU connected, why is it not using that?

# Changed the main pycoral folder name to EdgeCoral to reduce confusion for pc. Doing import sys and redirecting the path ran the pycoral.adapter line, but within the import other file also could not find another module...

# Seems like the issue was that the pycoral version installed, which is 0.1.0, was too low. By updating pycoral to 2.0.0, the NoModuleFoundError no longer appeared. The example project for classifying an image now works!

```
python3 -m pip install --extra-index-url
https://google-coral.github.io/py-repo/ pycoral~=2.0
```

# Yipeeeeee! Everything's working now, even the note detection script! Seems like I just needed to update pycoral to 2.0. How did I install 0.1.0 before?

# Apparently, it is possible to run the note detection script in the main system without an environment because the script doesn't make use of pycoral library, which is the biggest hassle here... Perhaps there is another solution?

Restrictions Unique to Pi

Editing Files on Pi

# June 25, trying to run mjpeg on rasp4 on 20.04, MAKE SURE python3.7, also be sure to install the library it needs.

Additionally, make sure ip address inside script matches applicable address for streaming

```
sudo apt-get install libgl1-mesa-glx
```

```
# To write into a python script using the terminal

vim [name].py

:w # To write, or just click "a" on keyboard to insert

:x # To save
```

```
# Alternatively, you can use vi instead of vim. You must save before you quit

vi [name].[type]

i # insert

:w # save

:q # exit
```

```
# Also alternatively, you can use nano instead of vi or vim. In fact, nano is probably the best editor. Please use nano

nano [name].[type]
```

Accessibility With No GUI

```
# Download missing libraries. It says xcb plugin cannot be run, but that is because raspberry pi does not have a gui for ubuntu. On pc, a window would pop up for camera feed. We just want to send stream to an address, so remove imshow line in script.

sudo apt-get install libgl1

sudo apt-get install libxcb-xinerama0 libxcb-xinerama0-dev
sudo apt-get install libxcb1 libxcb1-dev libx11-xcb1 libx11-xcb-dev

sudo apt-get install libglu1-mesa libegl1-mesa


# BE sure to use Ctrl+C and not Ctrl+Z to exit and KILL/end a
program. Ctrl+Z leaves the program running. Otherwise:

sudo lsof -i :8080 # or any port you're using

sudo kill -9 [pid] # program ID usually is 4 digits


# Ran into an issue where the python script to test camera streaming
worked, but when trying to actually view the stream from another
computer, script displays error saying it has trouble resizing... but
really it's just that I had the wrong camera index, because second
time I changed index it worked. LOL

# To make life easier, you can remote to the raspberry pi through
ssh. By default the pi uses a public key to sign in which must be set
up, but if you want to log in using password you can do the
following:

cd /etc/ssh

sudo nano sshd_config # Change to "PasswordAuthentication yes"
instead of no, save

sudo systemctl restart ssh


# After that, go your computer (I assume windows) command prompt and
type in

ssh pi@192.168..... # If pi is not the username of the raspberry pi,
change it hence. The username is the word before the @ on the linux
terminal. Also be sure to know what ip address the raspberry pi is
using

# Login as required, and now you can control remotely
```

Frame Rate Optimization

```
# You can check what formats your webcam supports, and the modes for
frame rates using the following commands:

sudo apt-get update

sudo apt-get install v4l-utils

v4l2-ctl --device=/dev/video0 --list-formats-ext # List all possible
formats and fps modes

v4l2-ctl --device=/dev/video0 --list-formats-ext > [name].txt # Write
the output to a text file, use "vi [name].txt" to read/edit

v4l2-ctl --device=/dev/video0 --get-fmt-video # List current format
and fps mode

v4l2-ctl --device=/dev/video1
--set-fmt-video=width=1280,height=720,pixelformat=MJPG # To literally
change camera type to mjpeg, hard change rather than by code


# Something I discovered is that the camera will always take in the
maximum fps it can, almost always 30 fps, except the stream may not
be maxed out because of many reasons. The smaller the res, the more
fps for the stream.

# I tried streaming the stream to see how many fps the stream had,
but that didn't work because you can't get mjpeg from an https, or
something like that according to chatGPT. So trying to stream a
stream doesn't work.

# The stream resolution does not affect the processing resolution,
but the processing will affect the stream.

# Try changing the format, or confirming the format captured as mjpeg
and not yuv. However, in order to do this, make sure your video
camera sets the driver for it:

cap = cv2.VideoCapture([index], cv2.CAP_V4L2)

cap.set(cv2.CAP_PROP_FOURCC, cv2.VideoWriter_fourcc('M', 'J', 'P',
'G'))
```

```
# After doing some digging, it seems like setting the "cap"
resolution will affect the fps, and the TPU seems like is not
responsible for the low fps

# You can also use code to change the camera type to mjpeg, but it
doesn't really work!

import os

os.system("v4l2-ctl --device=/dev/video1
--set-fmt-video=width=1280,height=720,pixelformat=MJPG") # Make sure
video1 is the uses the index of the camera e.g. video0


# So by setting different cap resolutions, and alternating between
setting mjpeg format, we've come to the conclusion: the raspberry pi
4 is not strong enough to handle Mjpeg

# By theory, if processing strength was not a boundary, mjpeg would
allow higher frame rates for higher resolutions that YUV cannot
support

# Normally, Mjpeg's compression is what makes it faster than YUV, but
compression also requires decompression to extract and use data -
that process takes time too

# For a raspberry pi 4, it is actually faster to just use YUV
directly than to compress and decompress Mjep, which is why YUV gives
faster FPS - this was tested using a performanceTest script

# The camera does it's job by recording 30fps as that is what it
supports, and certainly the stream does not effect what the camera
originally feeds

# But the process to transfer the data as mjpeg or YUV is what is
limits the fps the most. On an orange pi, converting to mjpeg truly
does gives a higher fps, but it's actually a downgrade for a rasp pi

# At a certain resolution, the downgrade effect starts becoming
pretty clear --> best to stick with YUV at 640x480 fps.

# Also, though not debunked otherwise, we can assume the TPU doesn't
play a role in downgrading fps.
```

# noted June 27th 2024

Automatic Command Execution

# Alright, so we basically have everything we need now. With a low enough resolution, we can run 30 fps - but how are we going to run it when it's on a bot?

# We'll need to have so that when the raspberry pi boots up, it automatically runs the main.py script to do note detection.

```
nano myscript.sh

chmod +x myscript.sh # turn it executable

./myscript.sh # to run the command script
```

# To turn command into boot up command

```
cd /home

cd /etc/systemd/system/

sudo nano startup.service # Make your own service file, something that is run by the system
```

# Copy the following into the service file. Replace the ExecStart path with the path you have

```
[Unit]
Description=Run Startup Script
After=network.target

[Service]
Type=simple
ExecStart=/home/pi/startup_script.sh
Restart=on-failure
User=pi
StandardOutput=journal
StandardError=journal
```

```
Environment=DISPLAY=:0

[Install]
WantedBy=multi-user.target


# Save the service file and activate it

sudo systemctl daemon-reload

sudo systemctl enable startup.service

sudo systemctl start starup.service

sudo systemctl status startup.service # If this shows no fails or
errors, it should automatically run on bootup now!

# Make sure there is no rc.local file inside /home/etc/, sometimes
causes errors, though cannot be concluded to be true, mere hypothesis
```

Delaying Boot Up of Pi

```
# After setting up the Pi on a robot, ip address was changed. A new
problem arises: since the Pi automatically runs the detection command
on boot up, it requires a valid ip address right as it starts up (the
main.py script requires it). However, the FRC radio takes a while to
start, producing a valid ip address only after the pi runs the
script. This means that script crashes. The only solution is to delay
the execution of the command until the radio is sure to be fully
booted.

# Edit the service file

sudo nano startup.service

# update the ExecStart line to following - wait 60 seconds and run
your detection command after waking. The program should not crash.
NOTE, based on tests, 25 seconds after boot up is the minimum
required delay time. The radio officially wakes up at 45 seconds, at
around 50-60 seconds the wifi is available for connection. At 33
seconds, the rasp pi 4 finishes booting. 33+25=58, roughly the time
when the ip address is available. Different pi's might have different
times, this is something to test out!
```

```
ExecStart=/bin/bash -c "sleep 25 && /path/to/your/script.sh"


# Save and execute the following

sudo systemctl daemon-reload

sudo systemctl enable startup.service
```

Using Network Tables

```
# What are networktables?

# In main.py, change localhost to 10.75.20.2 or
roboRIO-7520-FRC.local in line 17,
NetworkTables.initialize(“localhost”)

NetworkTables.initialize(“10.75.20.2”)
```

# Great, so now your info is on network tables! Under the topic labeled noteTable/MaxConfObj, you can retrieve a string containing information about the note the tpu is most confident in in the robot code.

# To retrieve the information, create a Network Table Instance and get the topic under your wanted table. Since the information of a note passed is a string, create an algorithm to split the string into substrings to isolate the important information: x, y, heigh, width, and confidence. Use FRC’s example code: https://docs.wpilib.org/en/stable/docs/software/networktables/networktables-intro.html

# Something to think about is whether we want to use the Max Confidence Object or the largest/closest object - sometimes the AI might be more confident in a note farther away. Usually, the most confident object is often the largest, but not always. For smarter optimization, both size and confidence levels should be considered when determining the best note to go for. We’ll have to make weight factors for each property manually in the code - THIS IS ACTUALLY SOMETHING THE AI MODEL IS SUPPOSED TO CALCULATE FOR US (the whole reason why we wanted AI model) but since it is not heavily trained, we must deal with the factors ourselves.

To see implementation of this in autonomous programming, visit the team github repository for commit messages.

https://github.com/Team7520/FRC2024/tree/CNE-2024-AUTONOMOUS-PROJECT