THE UNIVERSITY OF TENNESSEE KNOXVILLE

National Science Foundation

# Building a Autonomous Vehicle with Jetson Orin

Subir Warren
Katie Cheung

Baruch COLLEGE

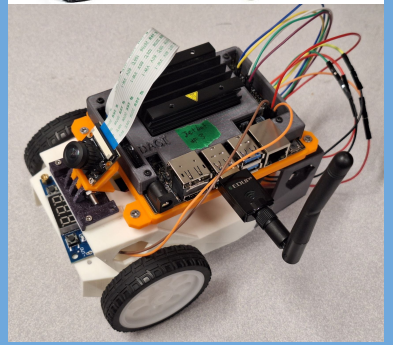
Stony Brook University

# Acknowledgements

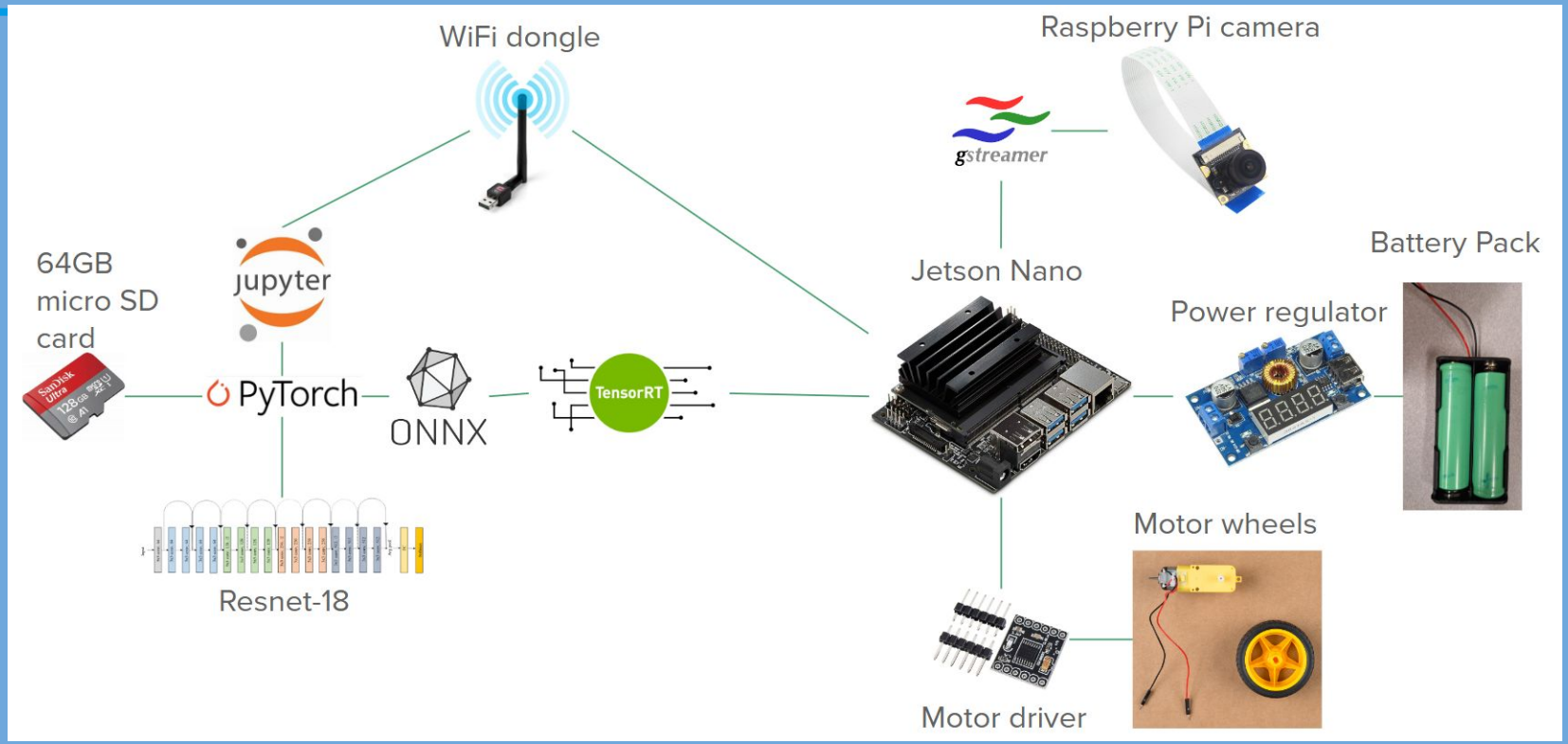
This project was sponsored by the National Science Foundation (NSF) through the Research Experiences for Undergraduates (REU) award and hosted by the University of Tennessee Knoxville. We also would like to thank the NSF for sponsoring this project as well as our mentors: Dr. Kwai Wong, Steven Qiu, and Franklin Zhang, for their contributions to this project.

# Grasping the Basics of Jetbot

- Starting with the Jetson Nano, we've recreated and tested preexisting Jetbots
- Version 1. Jetbot kit from NVIDIA guided us in understand the software and basic functions of the Jetbot
- Version 3. Jetbot kit made by Patrick Lau helped us comprehend how to assemble the hardware components to work together

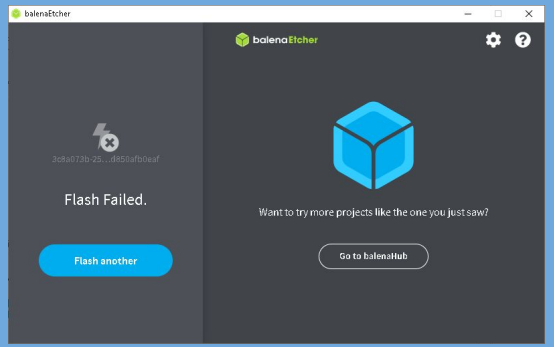


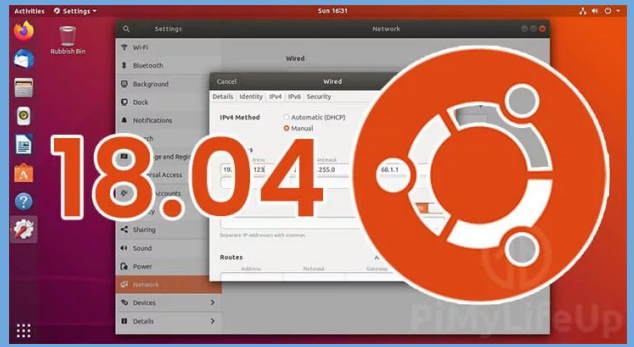
# Jetson Nano Jetbot Schematic



# Jetson Nano no longer supported

- Nvidia stopped supporting the Jetson Nano in 2023
- Recent software by Steven Qiu and Franklin Zhang showed us to how the Jetson Nano has incompatible and outdated updates and installations
- This led to relying on older versions of certain software and not being able to use upgrades



# Upgrading the Jetbot via Jetson Orin

- Transitioning software for the Jetbot on the Jetson Nano to the newer Jetson Orin
- Creating a new chassis to support the Orin
- Trying new techniques for object detection and autonomous movement with new software.
- Creating a code such that multiple cameras can be integrated simultaneously.

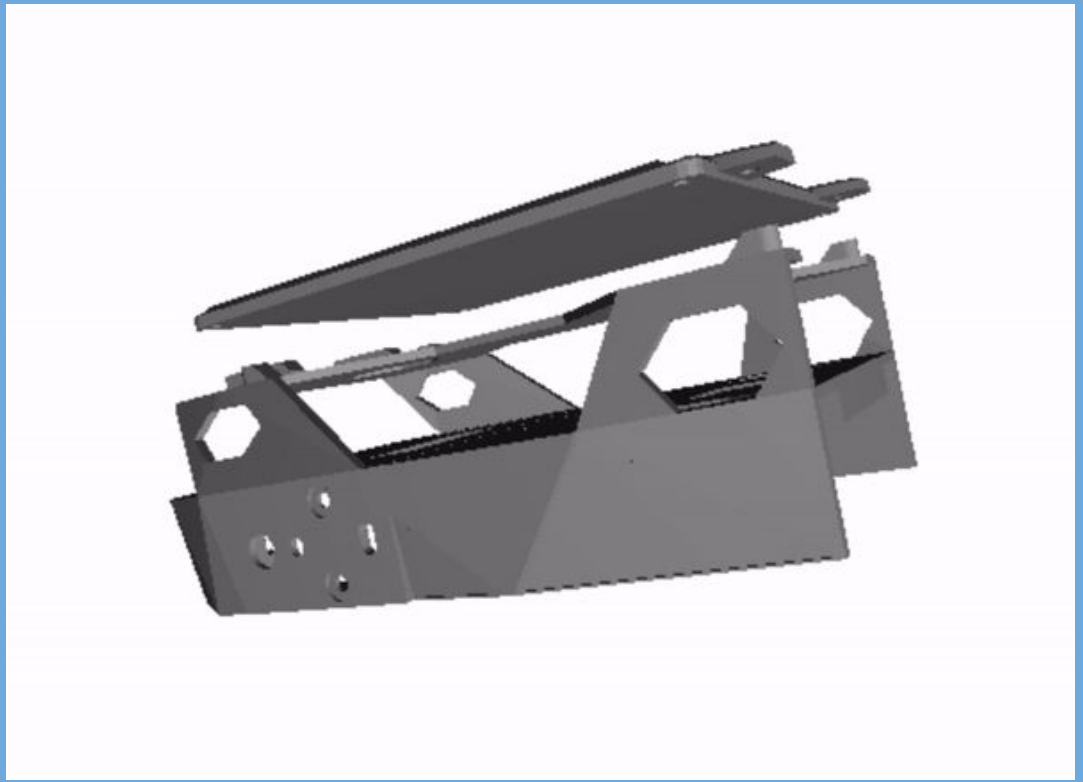
Why?

- Camera synchronization is better on the Jetson Orin
- 80x processing power than the Jetson Nano
- Software is being updated regularly

# Modified 3D models

Here is a 3D render of the STL file.

The chassis was modified to fit a four  battery pack. The base has added screw hole attachments. This aligns with the open source Jetson Orin case we choose. A double camera mount was also made for 2 cameras.



# Assembly Problems

Major Problems:

- There is no pre-existing code that is built for the Jetson Orin Nano and no guide for how it should be done.
- Different versions of software cause conflicts with already pre-trained models and causes the Orin to run into errors.
- Our cameras didn’t work with the newest version of Jetpack released by Nvidia.
- Hardware hase unidentifiable errors, which could be due to the quality of materials used.
- Batteries had a difficult time maintaining power on the Orin.

# Configuring Motors

Goals: Enabling the Jetson Orin to use motors.

What we did:

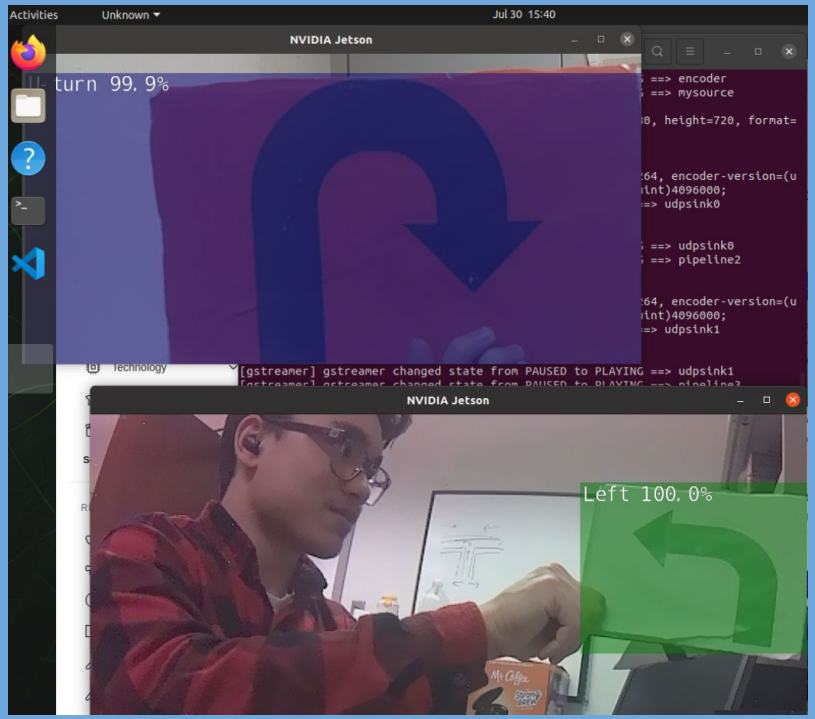
- Used the jetson-io interface to configure pins for PWM and GPIO function, so that the motors could move.
- Using python code to create a importable module, whereas on the Jetson Nano,  the Nano was already configured to run motor functions.

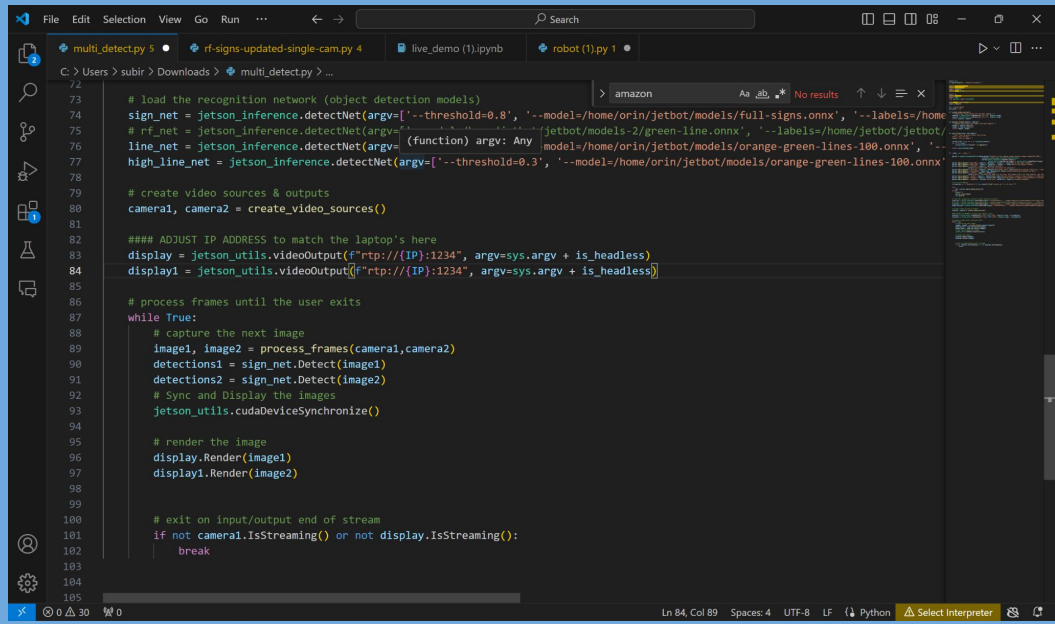
Results: Motors function and can move forward, right, and left. One motor is able to move backwards.

```python
class Robot():

    def set_motors(self, left_velocity, right_velocity):
        self.set_left_motor(left_velocity)
        self.set_right_motor(right_velocity)

    def stop(self):
        self.p_left.ChangeDutyCycle(0)
        self.p_right.ChangeDutyCycle(0)
        GPIO.output(self.IN1, GPIO.LOW)
        GPIO.output(self.IN3, GPIO.LOW)


    def forward(self, speed=1.0):
        self.set_motors(speed, speed)


    def backward(self, speed=1.0):
        self.set_motors(-1 * speed, -1 * speed)


    def left(self, speed=1.0):
        self.set_motors(-1 * speed, speed)


    def right(self, speed=1.0):
        self.set_motors(speed, -1 * speed)
```

# Using Cameras

On the Jetson Nano, using multiple cameras would cause the Nano to crash ,so we have created a code such that multiple cameras can be used at the same time on the Orin. The camera can do object detection on multiple models.



# NanoOWL

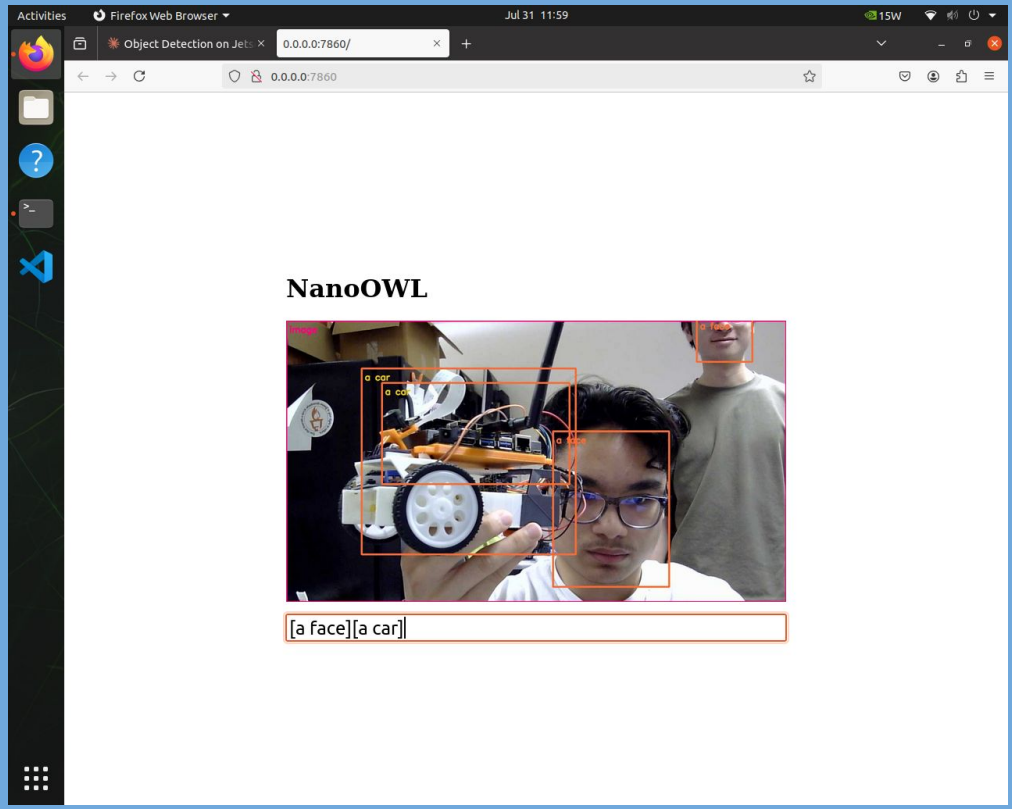
NanoOWL is a real-time object detection/image classification software, which saves time training new models.

Goal: To have NanoOWL detecting according to various prompts, such as “stop sign”, “person”, “ambulance”

Problems:

- Configuring which camera to use.
- Requires more time to create code for.

The following is a pilot code.



# NanoOWL Developments

```python
from jetbot import Robot
import jetson_utils

# NanoOWL
from nanoowl.owl_predictor import (OwlPredictor)
from nanoowl.owl_drawing import (draw_owl_output)

from PIL import Image  # Pillow import

import numpy as np

robot = Robot()

# Change CSI Source as needed
camera = jetson_utils.videoSource("csi://0", argv=sys.argv)
image = camera.Capture()
jetson_utils.cudaDeviceSynchronize()

# If you wish to change the model, you can, but this is the default upon creating a instance of class OwlPredictor
model = "google/owlvit-base-patch32"
image_encoder_engine = "/home/orin/nanoowl/data/owl_image_encoder_patch32.engine"
predictor = OwlPredictor(model, image_encoder_engine=image_encoder_engine)

# Gets prompt and then searches for detections
texts = "[a stop sign, a can]"
threshold = 0.1
texts = texts.strip("][()")
text = texts.split(',')
print(text)
images = Image.fromarray(np.uint8(image.value))
text_encodings = predictor.encode_text(text)
detections = predictor.predict(image=images, text=text, text_encodings=text_encodings, threshold=threshold, pad_square=False)
print("detections: ", detections)
```

# Road Following

With one camera

- Python code has already been established
- Can run on track with object detection when motors are fully working
- Only one model  is being run at one time.

While this can be implemented with multiple cameras, dealing with states of multiple cameras is complicated

```
###### ROAD FOLLOWING AND SIGN DETECTION #######
# At any given time, the robot will be focused on either
# road following or responding to signs.
# These are represented by different "states."
# Motor values are part hardcoded and part dynamically adjusted.

if state == "rf":
    # Look for road line.
    road_lines = high_line_net.Detect(image1, overlay=opt.overlay)
    road_lines.sort(key=attrgetter("Right"), reverse=True)
    appropriate_line = False
    for rl in road_lines:
        if not appropriate_line and rl_follow_dir(rl):
            appropriate_line = True
    if not appropriate_line:
        strikes += 1
        if strikes % 4 == 0:
            robot.set_motors(0.3 * max_speed, 0.3 * max_speed)
            time.sleep(0.2)
        elif strikes % 4 == 1:
            robot.set_motors(0.3 * max_speed, 0.3 * max_speed)
            time.sleep(0.2)
        robot.stop()
        time.sleep(0.2)
    if strikes == 10:
        # After the robot fails to detect any road lines many frames in a row,
        # stop and look for signs.
        state = "signs"
        strikes = 0
        robot.stop()
elif state == "straight":
    road_lines = line_net.Detect(image1, overlay=opt.overlay)

    best_green_line = None
    green_lines, orange_lines = sort_lines(road_lines, 2)

    final_green_candidates = []
    for gl in green_lines:
        above_orange = False
        far_right = False
        far_left = False
        for ol in orange_lines:
            # good green lines are not largely above an orange line
            if fractional_coord(gl, "x", 0.75) < ol.Right and fractional_coord(gl, "x", 0.25) > ol.Left:
                above_orange = True
            if ol.Left > image1.width/2 and gl.Left > ol.Left and fractional_coord(gl, "y", 0.6) > ol.Top:
                above_orange = True
            # if the above isn't disqualifying already,
            # good green lines do not have their center on the right half
            elif fractional_coord(gl, "x", 0.5) > image1.width/2 and gl.Right > image1.width * 0.9:
                far_right = True
            #elif fractional_coord(gl, "x", 0.3) < image1.width/4:
            elif gl.Left < image1.width/10:
                far_left = True
        if not above_orange and not far_right and not far_left:
            final_green_candidates.append(gl)
```

# Multi-Camera Autonomous Driving

Proposed Plan:

- Having a single camera detect for lines
- Having another camera detect for cars or other objects.
- For more cameras, more models can be run, which would require multiprocessing.

Problem: Requires much more time than available and testing.

```
if state == "classify":
    class_id, confidence = class_net.Classify(image1)
    class_desc = class_net.GetClassDesc(class_id)

    if confidence > 0.4:
        # overlay the result on the image
        font.OverlayText(image1, image1.width, image1.height, "{:05.2f}% {:s}".format(confidence * 100, class_desc), 5, 5, for

        # check if it is a return journey, reverse left and right
        if forward == False:
            if class_id == 0:
                class_id = 1
            elif class_id == 1:
                class_id = 0

        # When the Jetbot is facing straight, go straight
        if class_id == 2:
            print("Straight")
            robot.set_motors(0.60 * max_speed, 0.60 * max_speed)
            time.sleep(0.5)

        # When the Jetbot is facing left, turn right
        elif class_id == 0:
            print ("Facing Left")
            robot.set_motors(0.80 * max_speed, 0.40 * max_speed)
            time.sleep(0.5)

        # When the Jetbot is facing right, turn left
        elif class_id == 1:
            print ("Facing right")
            robot.set_motors(0.40 * max_speed, 0.80 * max_speed)
            time.sleep(0.5)
```

# Replicating the Jetbot

By replicating this project with the Jetson Orin, we are now able to leverage these new capabilities. Also, in the future, better models can be trained using ResNet50 and kept up to date with the newest versions of TensorRT. Due to a lack of time to further test we have done the following

- Created a documentation for those using the Jetson Orin that will allow for replication as well as provide debugging help.
- Updated Jetson Nano documentation as they can still be used to help train Orin Jetbots while also cheaper
- Uploaded all code to a Github with clear and detailed instructions, which with documentation can be replicated

# Future Work

To continue progress with the Jetson Orin Nano

- Python code should continue to be developed so that NanoOWL can identify many prompts and avoid them as best as possible.
- Use CVAT or another type of annotation tool for semantic segmentation, which could be used on multiple cameras.
- Create multiprocessing python scripts
- Add a ultrasonic sensor

# Thank you!

# References

*Tutorial - Nanoowl*. NanoOWL - NVIDIA Jetson AI Lab. (n.d.)
https://www.jetson-ai-lab.com/vit/tutorial_nanoowl.html

Zhang, Franklin. “Jetbot” *GitHub*, github.com/franklinzhang12/jetbot.

Lau, Patrick, “Jetbot Assembly Guide”,
https://docs.google.com/presentation/d/1MOtegvjN1XGcSuTHRs4KnJr0uxTNdZtAVcILk1ymLaA/edit#slide=id.g2bc93d7f113_0_4