Autonomous JetBot Project

Kwai Wong, Patrick Lau, Franklin Zhang, Eric Zhao
University of Tennessee, Knoxville

2022 - 2023

Contents

What : Goals of the Project

- ✔ Run a JetBot vehicle autonomously using a Jetson Nano GPU Card

Why : Purpose of the Project

- ✔ Setup a practical application for a class project
- ✔ Showcase the ability of the DNN functionalities
- ✔ Integrate a set of software tools from Nvidia onto the Jetson Nano Card

Modeling Specification of the Project

- ✔ Run a toy vehicle autonomously on a cross section of pathway

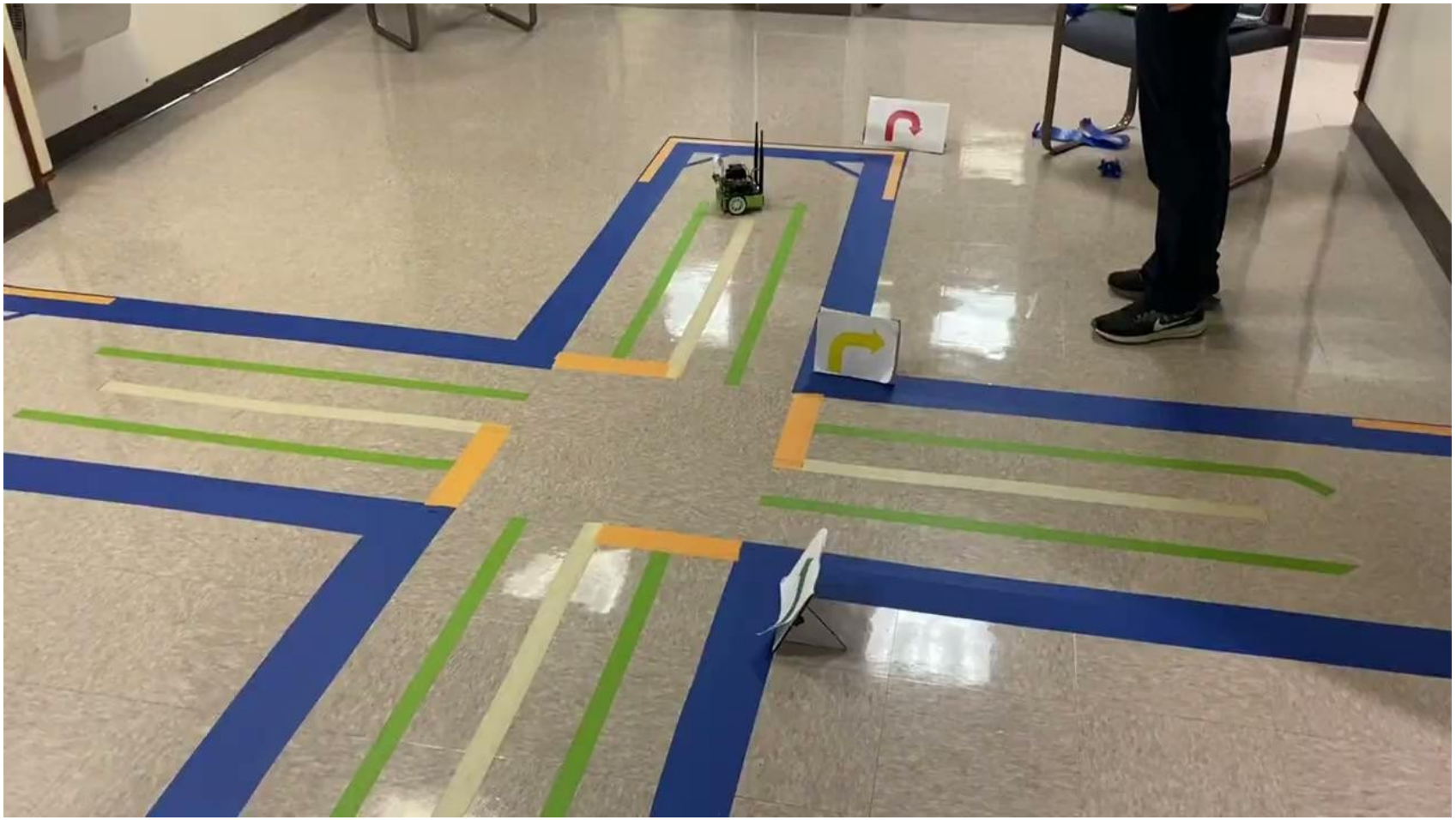
The robot should be able to:

1. Follow the straight parts of the road
2. Respond to U-Turn signs placed at each of the four ends by turning around
3. Respond to Turn Left/Turn Right signs placed at the intersection by turning accordingly

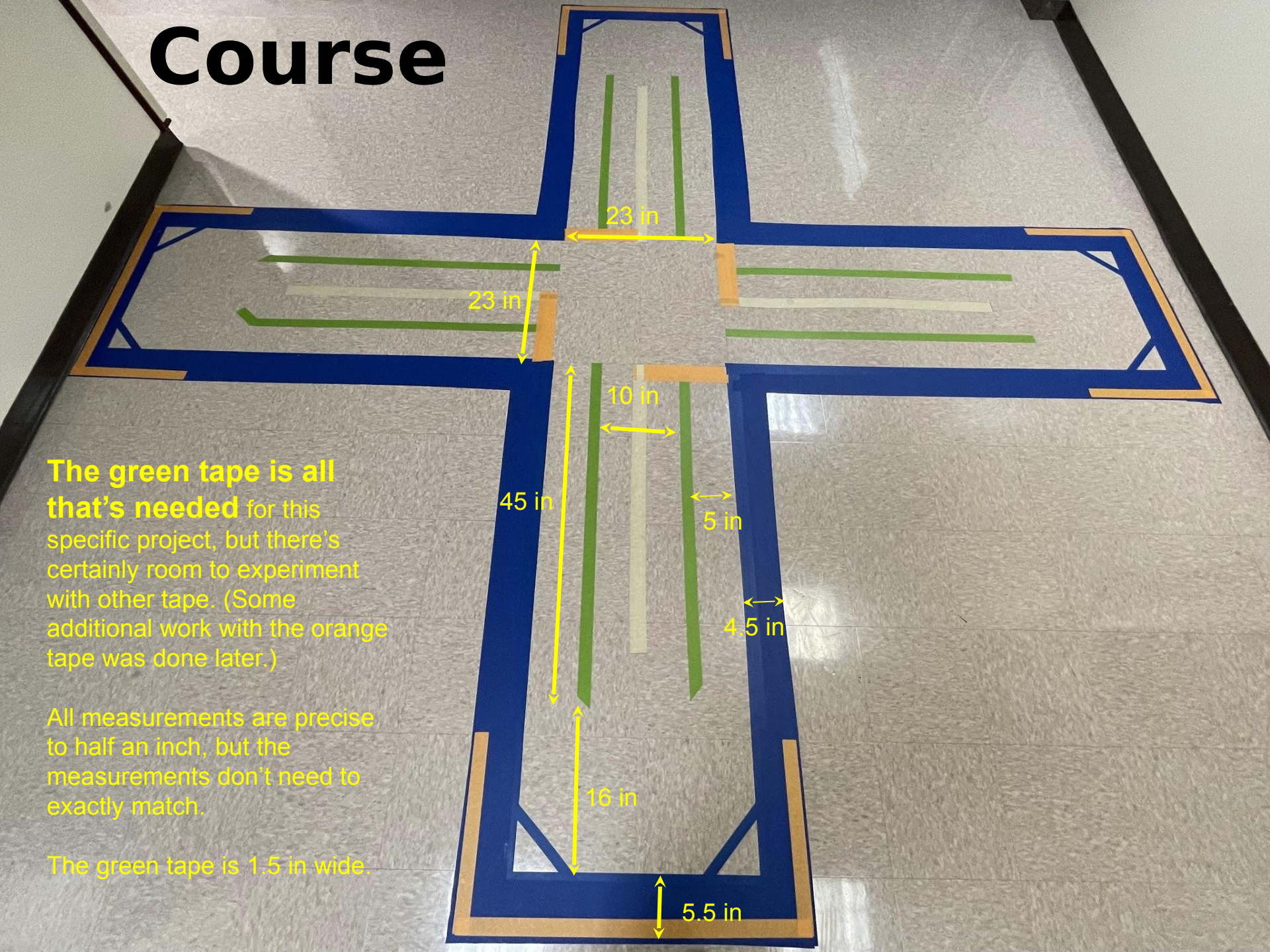
Signs are different colors on a white background:

- U-Turn = red, Left Turn = green, Right Turn = yellow

Demonstration Video



Course



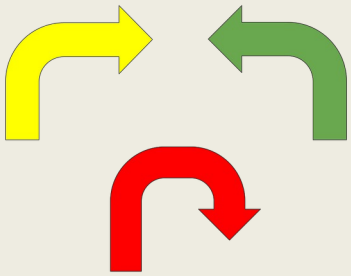
The green tape is all that's needed for this specific project, but there's certainly room to experiment with other tape. (Some additional work with the orange tape was done later.)

All measurements are precise to half an inch, but the measurements don't need to exactly match.

The green tape is 1.5 in wide.

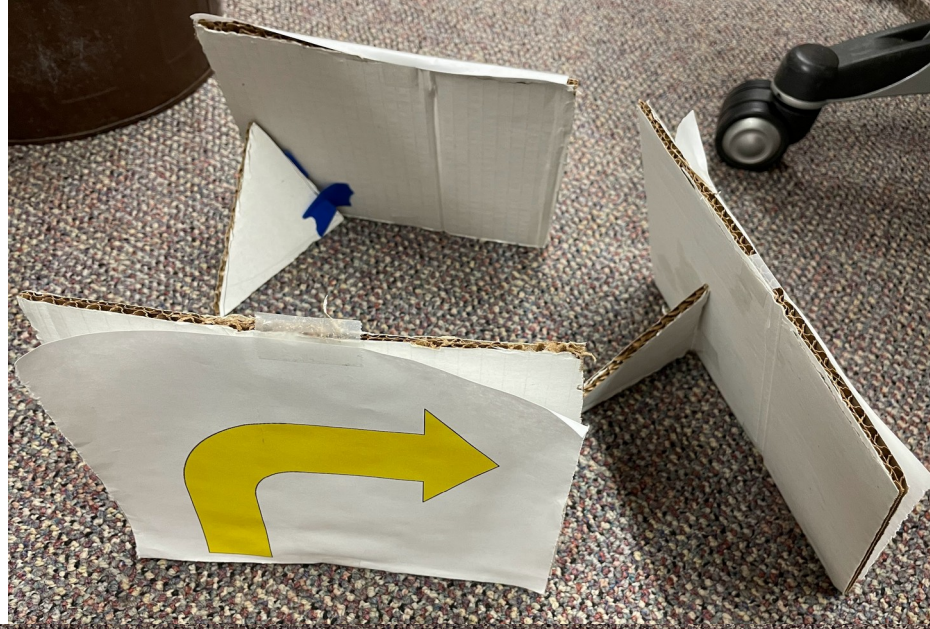
Signs

Drawing

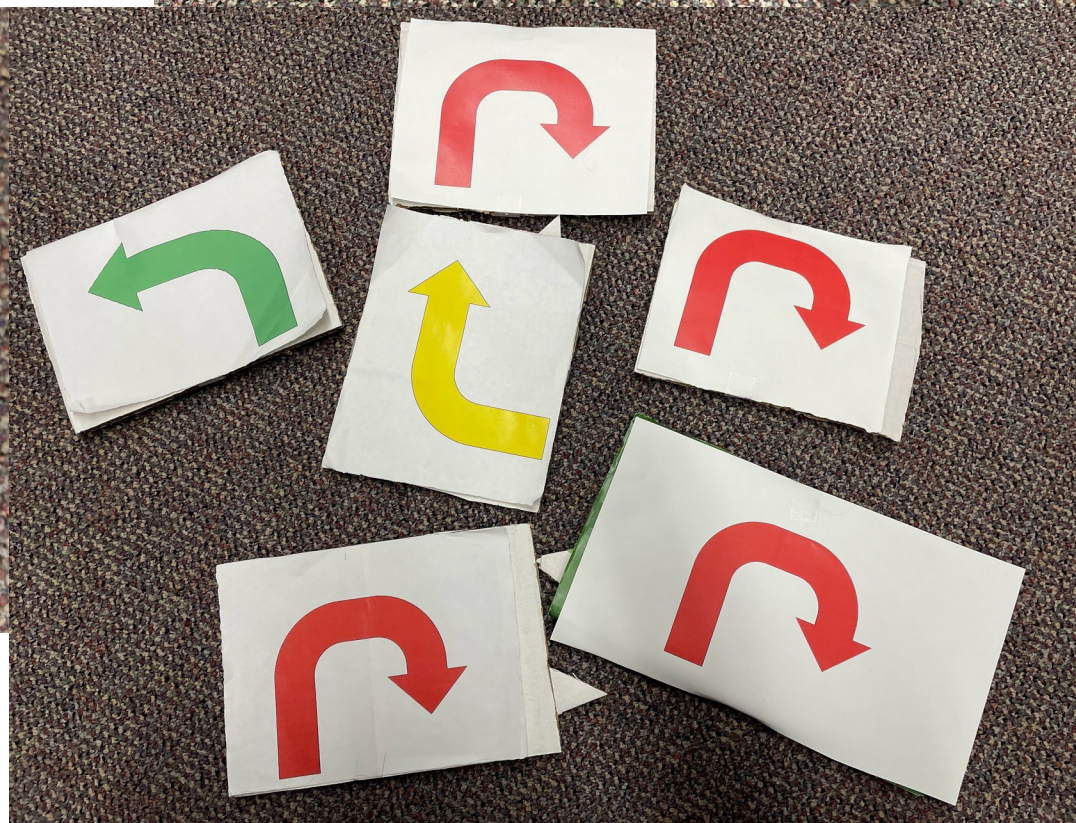


Printable

https://docs.google.com/document/d/1zxXWrhZquiRAq2lARWC_oSfjJx4mTcNcrNoBi8vthRA/edit?usp=sharing



Back view



The signs tend to be about 6 in by 8 in, though there is no need to be exact.

Hardware Specification of the Project

- ✔ Toy vehicle
  - ○ WiFi Antennas
  - ○ Camera
  - ○ Wheels, motors
- ✔ Jetson Nano GPU Card (a computer)
- ✔ JetBot Expansion Board
- ✔ Batteries
- ✔ Gamepad controller / dongle (for movement and image capture)
- ✔ Separate laptop/computer – used to remotely access and control the robot

Hardware Specification of the Project

Kit (image on next slide):
https://www.waveshare.com/catalog/product/view/id/3755 or https://www.amazon.com/Waveshare-JetBot-AI-Kit-Accessories/dp/B07V8JL4TF/?th=1

Info: https://www.waveshare.com/wiki/JetBot_AI_Kit

Batteries must be purchased separately. See the batteries slide (a few slides later) for more info. A laptop or other computer is also needed.

Ignoring the laptop, estimated total cost is around $400.

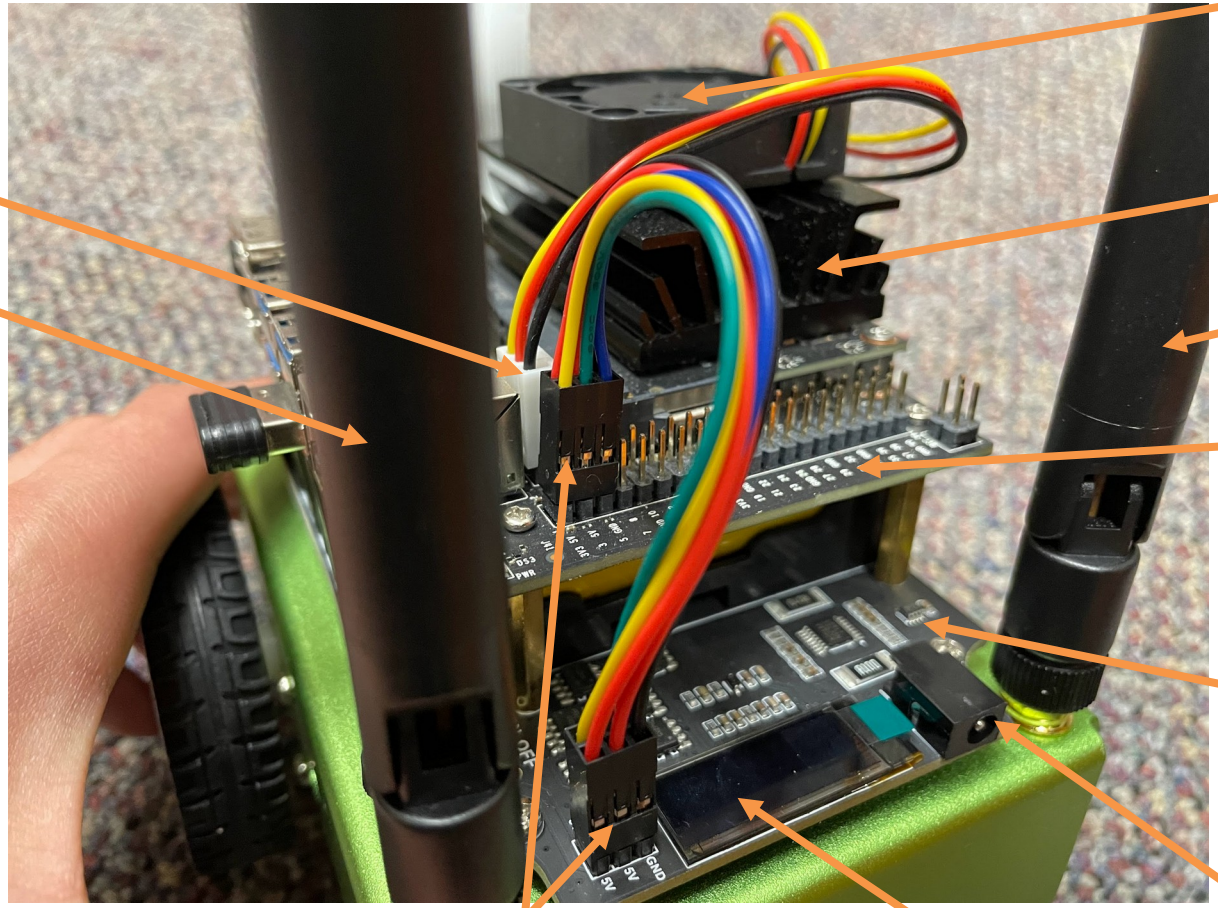
Useful Links

https://www.waveshare.com/wiki/JetBot_AI_Kit_Assemble_Manual
– Hardware assembly guide (not needed if robot already set up)

Note: images of assembly of a slightly different JetBot can be found in the jetbot/docs/images directory that is located on all JetBots or linked here: https://github.com/NVIDIA-AI-IOT/jetbot/tree/master/docs/i


JetBot
AI ROBOT KIT
Collision Avoidance
Object Following
Road Following
Teleoperation
Face Recognition
X002A47PDZ
JetBot AI Kit Acce
NEW
Made in China
WAVESHARE
SINCE 2006
JETSON NANO DEVELOPER KIT
JETSON NANO™ DEVELOPER KIT

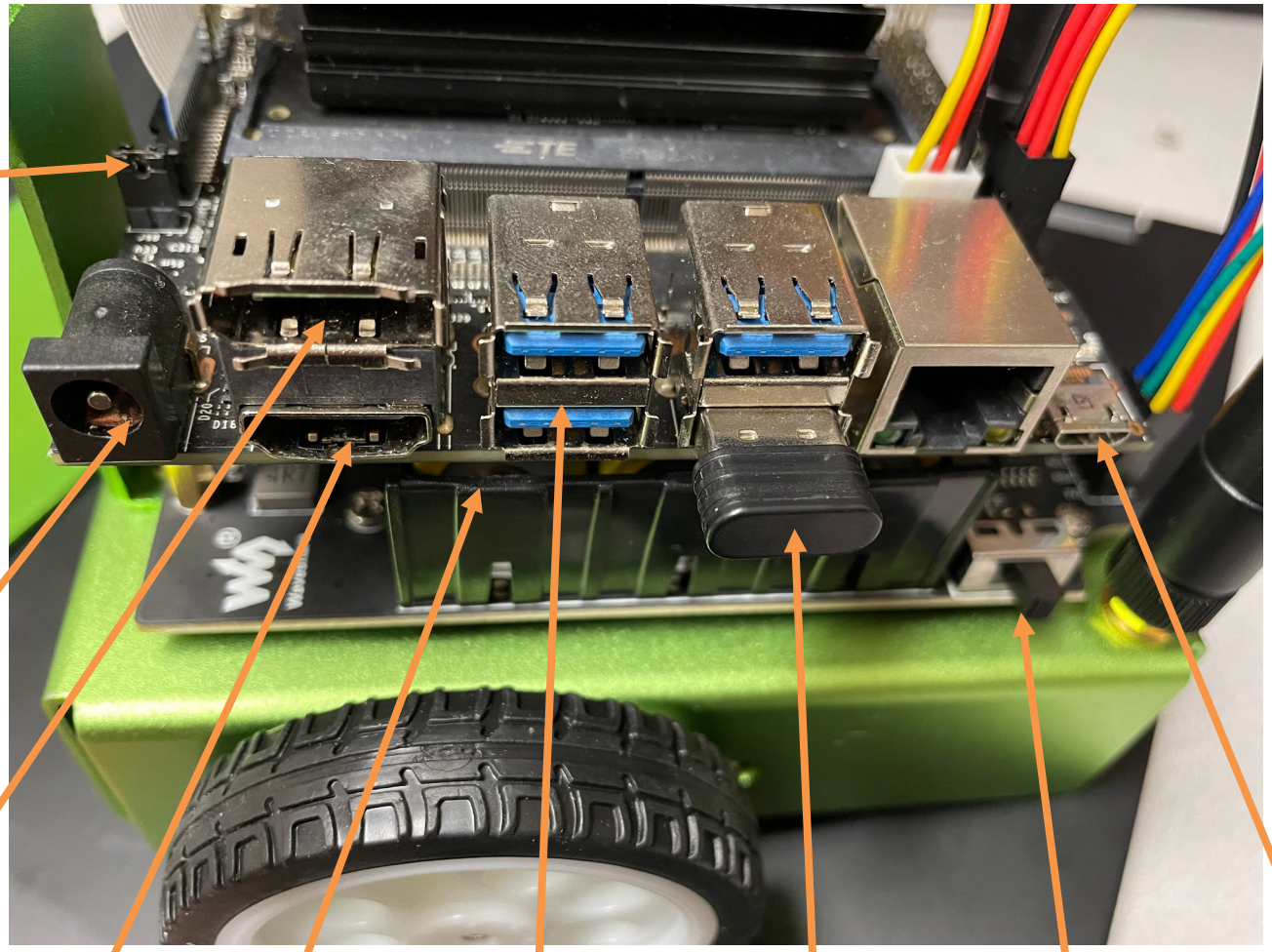
Wiring, Antennas, Charging (back view)



The fan cools the heat sink, which assists cooling as it absorbs heat from the Jetson Nano and has a greater area of contact with the cool air. The communication and power wire supplies power to the Jetson Nano from the batteries and allows it to communicate with the JetBot Expansion Board and motors.

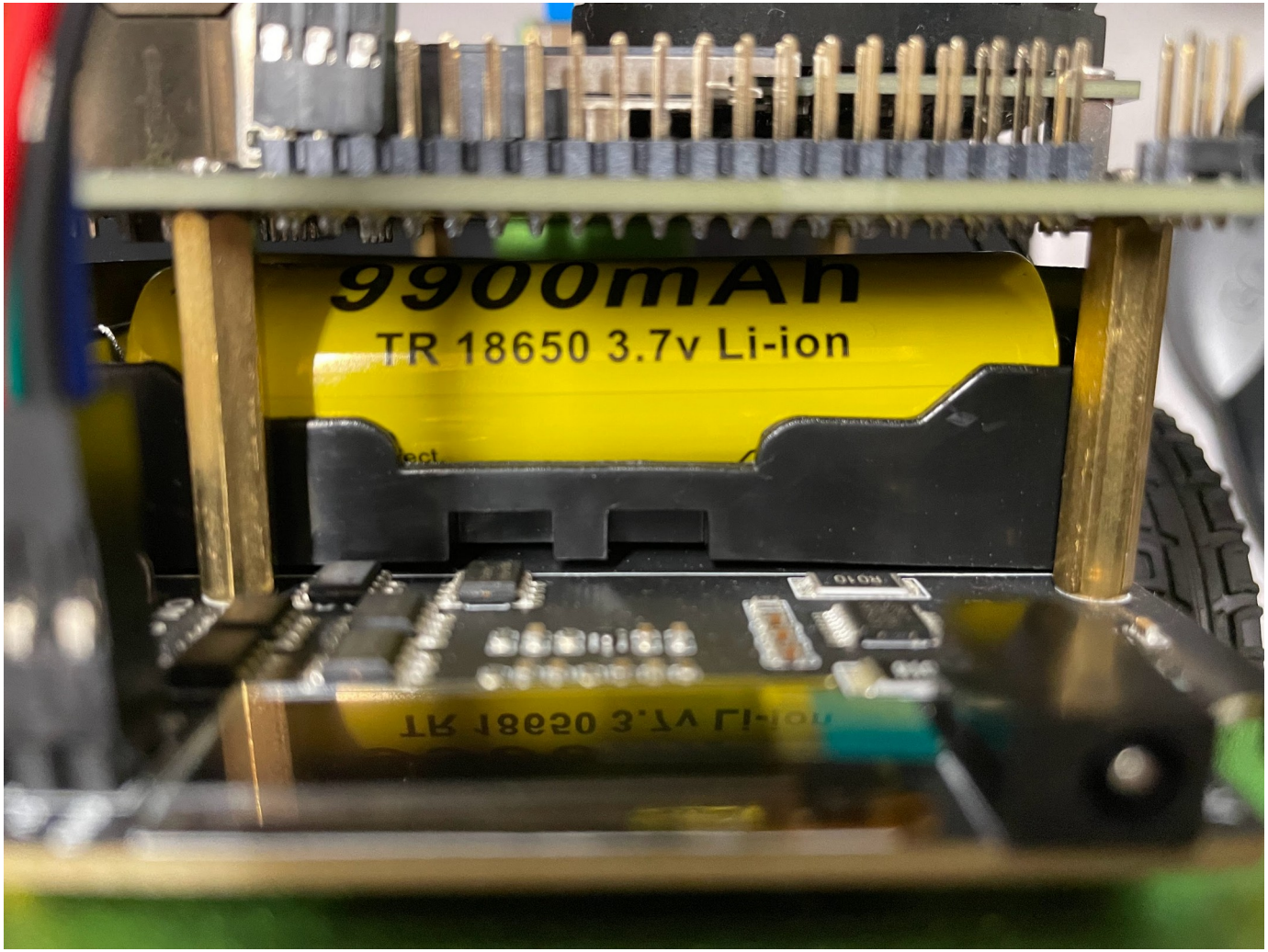
Ports and Power (left side view)

All labeled parts here except the power switch are part of the Jetson Nano.



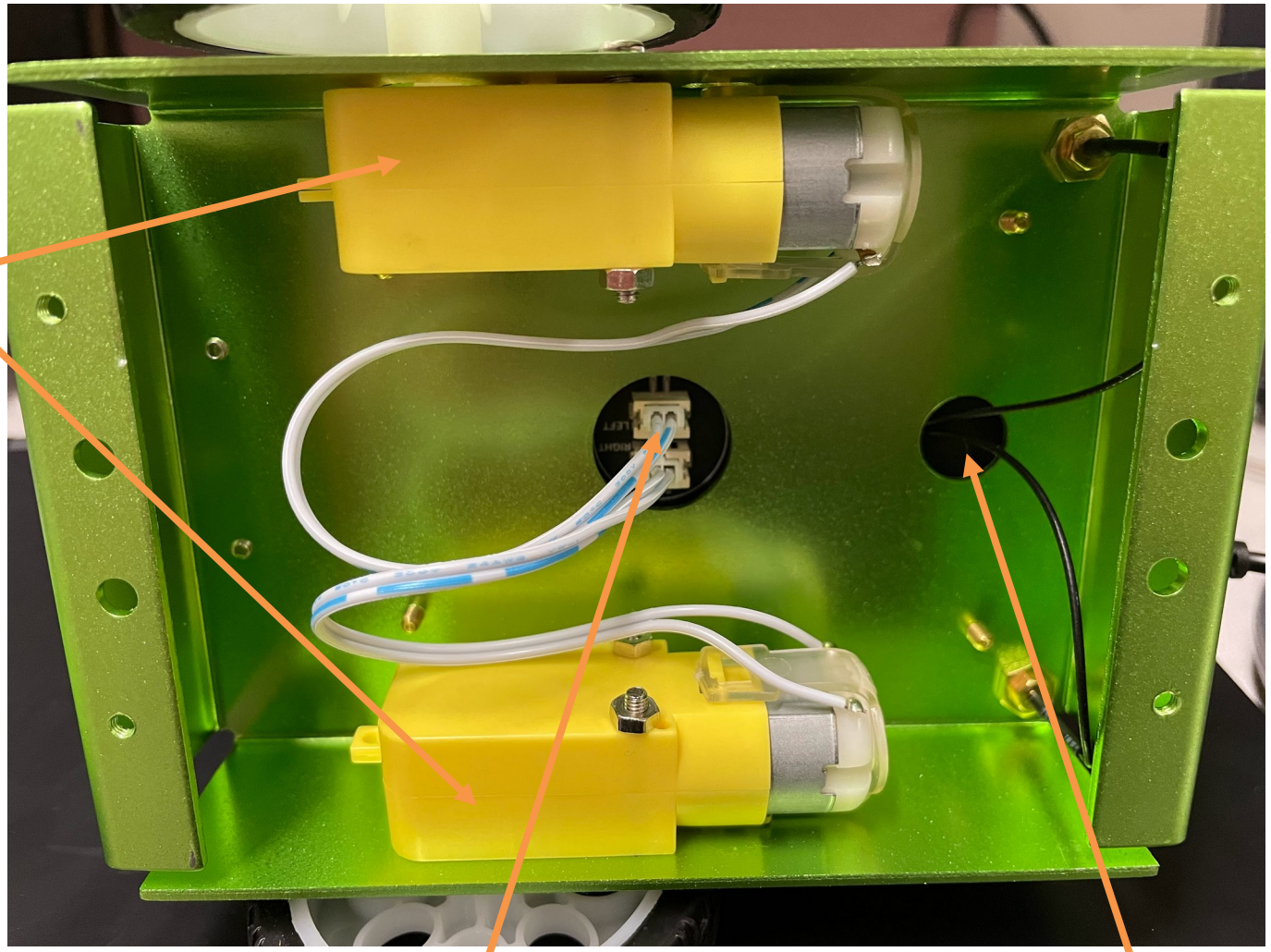
Neither of these charging ports actually charge the battery; they just provide a temporary power supply.

Batteries (back view)



There are three TR 18650 3.7v Lithium ion batteries (9900 mAh).
The batteries can only be charged via the charging port on the bottom right here (more clearly shown two slides ago)

Motors (bottom view, cover unscrewed)



Two independent motors inside the bottom of the robot control the wheels.

Turning occurs when one motor spins faster than the other

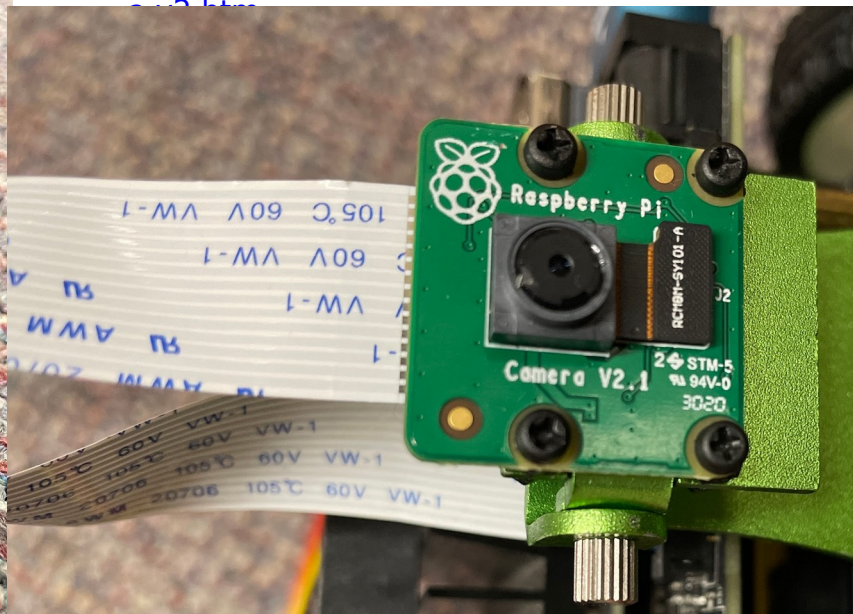
Front/Right View

Camera connection wire to Jetson Nano

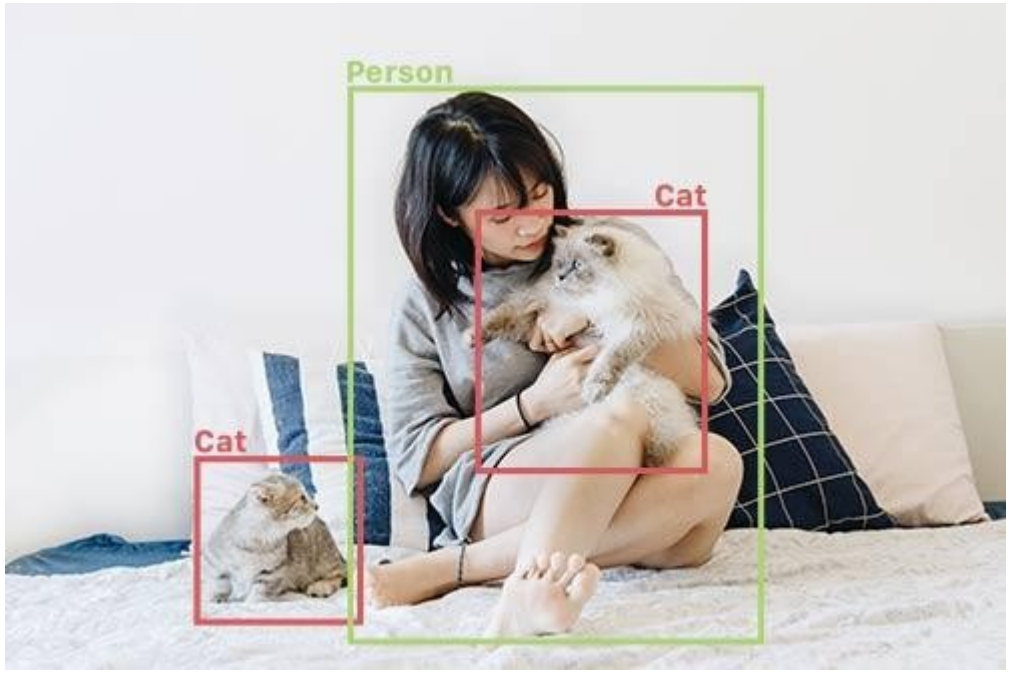
Came ra

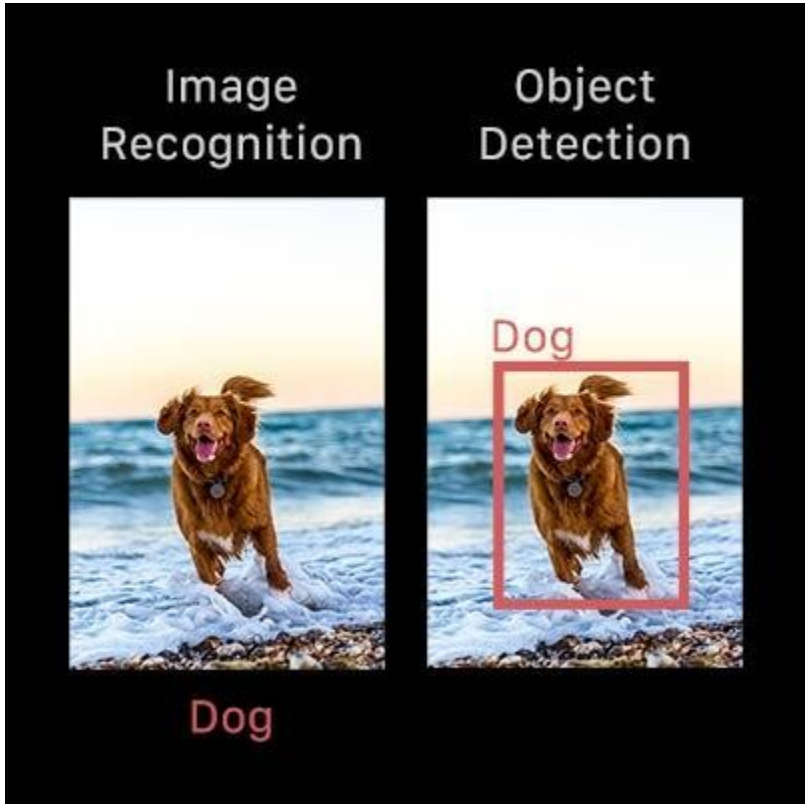
MicroSD Card

Camera Closeup: (https://www.waveshare.com/product/rpi-camer



Object Detection





This project uses **object detection**. This type of neural network can recognize not only what objects are in the image, but their location and size (represented by bounding boxes).
Also, this project uses transfer learning to train a neural network – taking a pretrained model and retraining it on new data.

The pretrained model is SSD-mobilenet-v1, trained on 91 classes.

https://github.com/dusty-nv/jetson-inference/blob/master/docs/detectnet-console-2

Schematic

Software

Training

rf-signs.py Inference code

.onnx file

labels.txt

TensorRT

model files

onnx_export.py

PyTorch

train_ssd.py

Data Collection

START of training process

image-capture-single.py

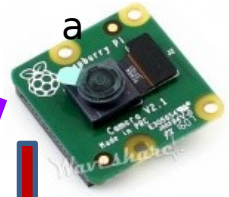
Images

Annotations

Hardware

GPU, Accessories

Camera



Gamepad Controller

MicroSD Card

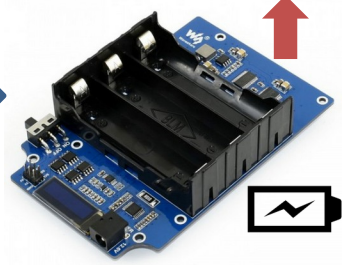
WiFi Antenna

Jetson Nano Card

JetBot Car

Wheels

Motors



JetBot Expansion Board

rf-signs.py Inference code

Network Service

START of inferenc

Work Computer

Annotation Tool

Video Streaming

Software Specification of the Project

- ✔ Run a toy vehicle autonomously using a Jetson Nano GPU Card
- ✔ Train an object detection neural network on images of road and signs
- ✔ The robot should navigate autonomously using the object detection model

The following devices are necessary, with appropriate software on each:

- work computer – ideally portable (i.e., a laptop)
- JetBot with Jetson Nano GPU card

The work computer can remotely access the JetBot computer, as detailed later.

<u>Work Computer</u>

- Uses **Linux operating system** (*Ubuntu 20.04 LTS*) in this project. However, a different OS should also be fine. (Note: some details in these slides likely will not align with other operating systems. Adjustments would probably be necessary.)
- *CVAT* – used to annotate data with bounding boxes
- *GStreamer* – allows the JetBot's camera feed to be streamed to another computer. This lets you see what the JetBot is seeing, which is very helpful.

Instructions for installing CVAT and GStreamer will be provided later in this document.

Software Specification of the Project

Jetson Nano

- *Ubuntu 18.04 LTS* – Linux operating system
- Artificial Intelligence tools:
  - *CUDA 10.2.89* – software for GPU processing
  - *CuDNN 8.0.0.180* – GPU-based library for deep neural networks
  - *TensorRT 7.1.3.0* – machine learning framework
  - *PyTorch* – machine learning framework
  - *Jetson-inference* (JetBot) – code for neural network deployment

https://developer.nvidia.com/embedded/jetpack-sdk-45-archive
Information about JetPack

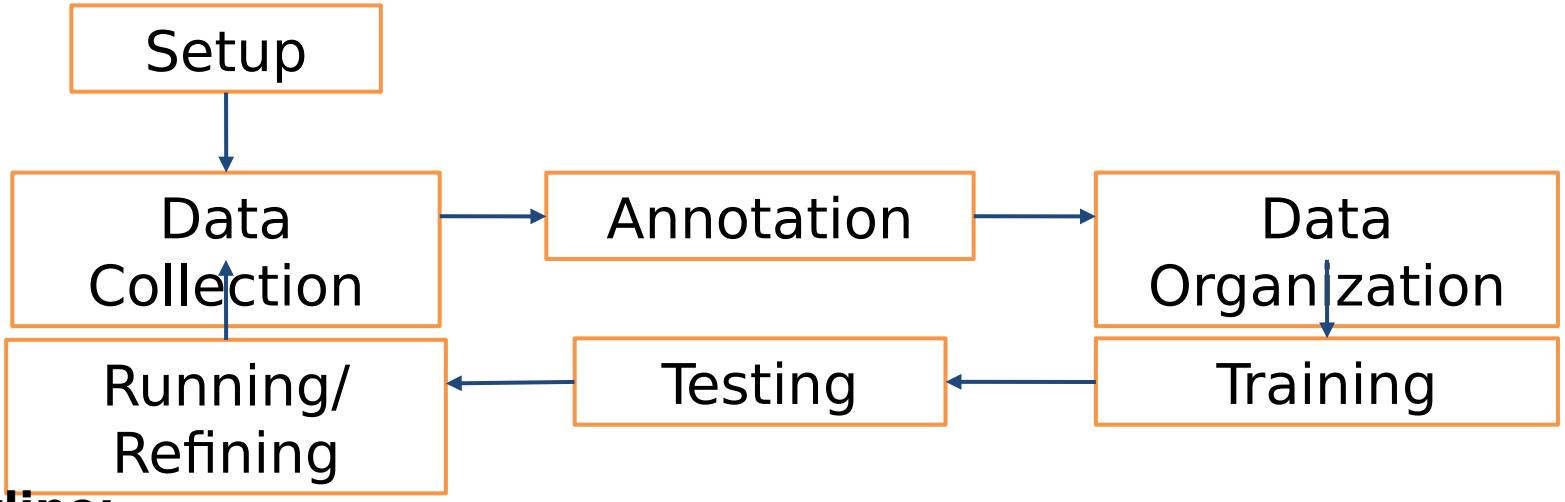
These can all be set up by following the presentation linked here:
https://docs.google.com/presentation/d/1cCeMHUpsSFOgPN5odmHXncWkYEUl-6fNVQ77BbnpAJ4/edit?usp=sharing

Useful reference – jetson-inference Github:
https://github.com/dusty-nv/jetson-inference

This completes the general setup of the JetBot for artificial intelligence! Now, the various models can be tried out – see the jetson-inference Github, links under Hello AI World. This project is based on the "Collecting your own Detection Datasets" guide.

More project-specific setup is still needed, as detailed on upcoming

Workflow



Outline:

1. *Setup* – prepare course, signs, JetBot, etc.
2. *Data Collection* – take images of the environment that the JetBot will be in, so that the JetBot can learn from those images to build a model that can interpret new images
3. *Annotation* – label and identify various objects of interest in the images by drawing bounding boxes
4. *Data Organization* – compile and prepare the data for training
5. *Training* – use the data to create an object detection model that can identify objects in new images
6. *Testing* – determine how accurate the model is
7. *Running/Refining* – Run the robot on the track autonomously using the model, making adjustments as needed

Repeat the process to train new models as necessary

1. Setup

- Print out and create signs as shown earlier. Place tape for a course.
- Setup software – guide on Slides [1] (if not already set up)

Note: It may be best to train on another computer, but this requires installing jetson-inference on it. Alternatively, the JetBot itself can be used to train, in which case no special action is needed.

Key points

a) Connecting to WiFi

Use the JetBot ports to connect it to a monitor, mouse, and keyboard. Then, login and connect to WiFi.
The JetBot's username and password are both "jetbot" by default.

b) GStreamer

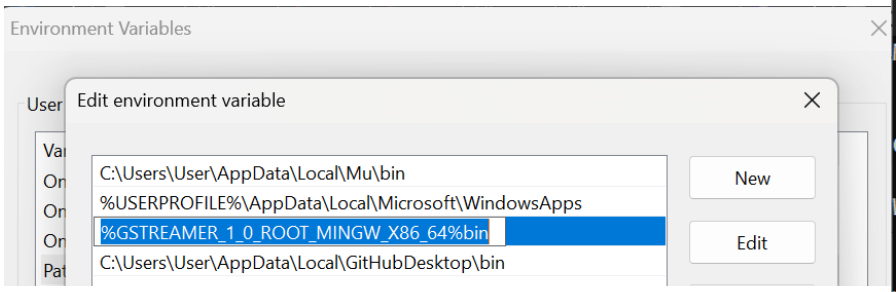
If not done already, install GStreamer [2] on the work computer that will receive the camera feed.

Whenever you use Gstreamer, you must first open a separate terminal window on the work computer and run a command [3] to start the pipeline (gst-launch-1.0 -v udpsrc port=1234 caps = "application/x-rtp, media=(string)video, clock-rate=(int)90000, encoding-name=(string)H264, payload=(int)96" ! rtph264depay ! decodebin ! videoconvert ! autovideosink).

[1]

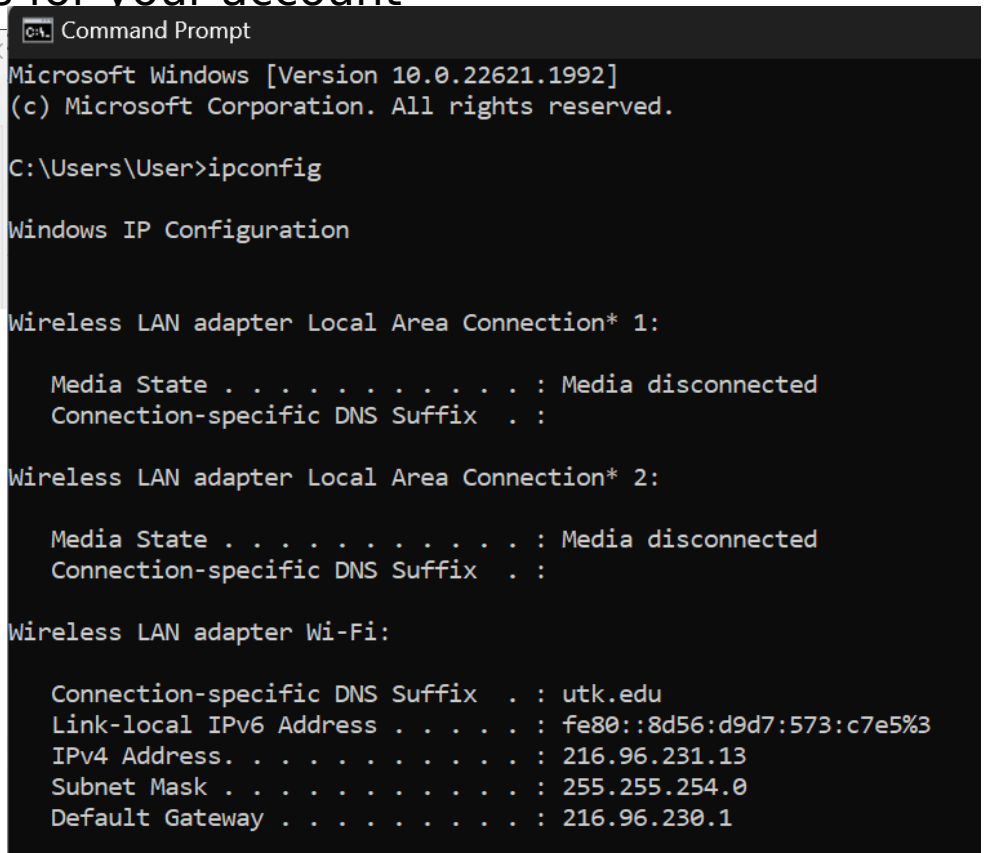
Setup on Windows

1. Go to [Download GStreamer]
2. Download both 64-bit (depend on your computer) runtime and development installer
3. Follow download instruction from installer
4. Allow the firewall
5. Open Edit environment variables for your account



Check IP Address:

1. Open Command Prompt (cmd)
2. Run $ ipconfig
3. Find your IP Address in line showing IPv4 Address



1. Setup

Remotely connecting to the JetBot

Very often, connecting a mouse, monitor, and keyboard to the JetBot is not practical, especially when the JetBot moves. Thus, we can remotely connect to the JetBot using **SSH** (secure shell). To do this, open a terminal window on the work computer, and type in the command `ssh jetbot@<IP>` to remotely access the JetBot computer.

- `<IP>` is the IP address, which can be found on the small JetBot screen (see Slide 9) once the robot is turned on, or by running `hostname -I` or `ifconfig` in the terminal of the JetBot computer (accessible by connecting the JetBot to a monitor, mouse and keyboard).
- A password prompt will appear – by default, the password is jetbot – same as before.

This will provide access to the command line on the JetBot, allowing control over the system.
Navigating the JetBot remotely requires some familiarity with the command line interface!
Some other useful commands and links to info:

- **Basic commands:** https://ubuntu.com/tutorials/command-line-for-beginners#1-overview
- **VI/VIM** (usage: `vim FILENAME`) is a text editor, allowing for creation and

1. Setup

The separate code for the project can be found on Github.
To follow this project, download the code and place the src folders in the jetbot directory on the JetBot. Also create a models directory inside the jetbot directory.

Note: The code in the src folder is still based on jetson-inference code, just modified. However, the rf-signs.py files' image interpreting and robot movement were added.

In general, to run a Python file, use `sudo python3 NAME.py [any command line args]` with NAME.py in the current directory.

- Add `--flip-method=rotate-180` to the end if the program produces images with Gstreamer, as the image is flipped the wrong way by default.
- Optionally, you can add `--headless` to the end of any program where you use flip-method, preventing the robot from attempting to stream through an HDMI port.

<u>Examples</u>
Use robot with gamepad controller: `sudo python3 gamepad-control.py`
Take images: `sudo python3 image-capture-single.py --flip-method=rotate-180`
Run the demo (video shown earlier):

JetBot File Tree – Key Parts

(home directory of Jetson Nano card – *this is how the file tree should look at this point)*

jetbot/

src/ *(downloaded from Github link)*

data-collection/

gamepad-control.py

image-capture-single.py

detectiontest.py

rf-signs.py

models/ *(created empty directory, or downloaded from Github)*

(empty directory unless downloaded from Github)

(other jetbot files and directories)

jetson-inference/ *(downloaded earlier in software setup process)*

(directories and files for jetson-inference)

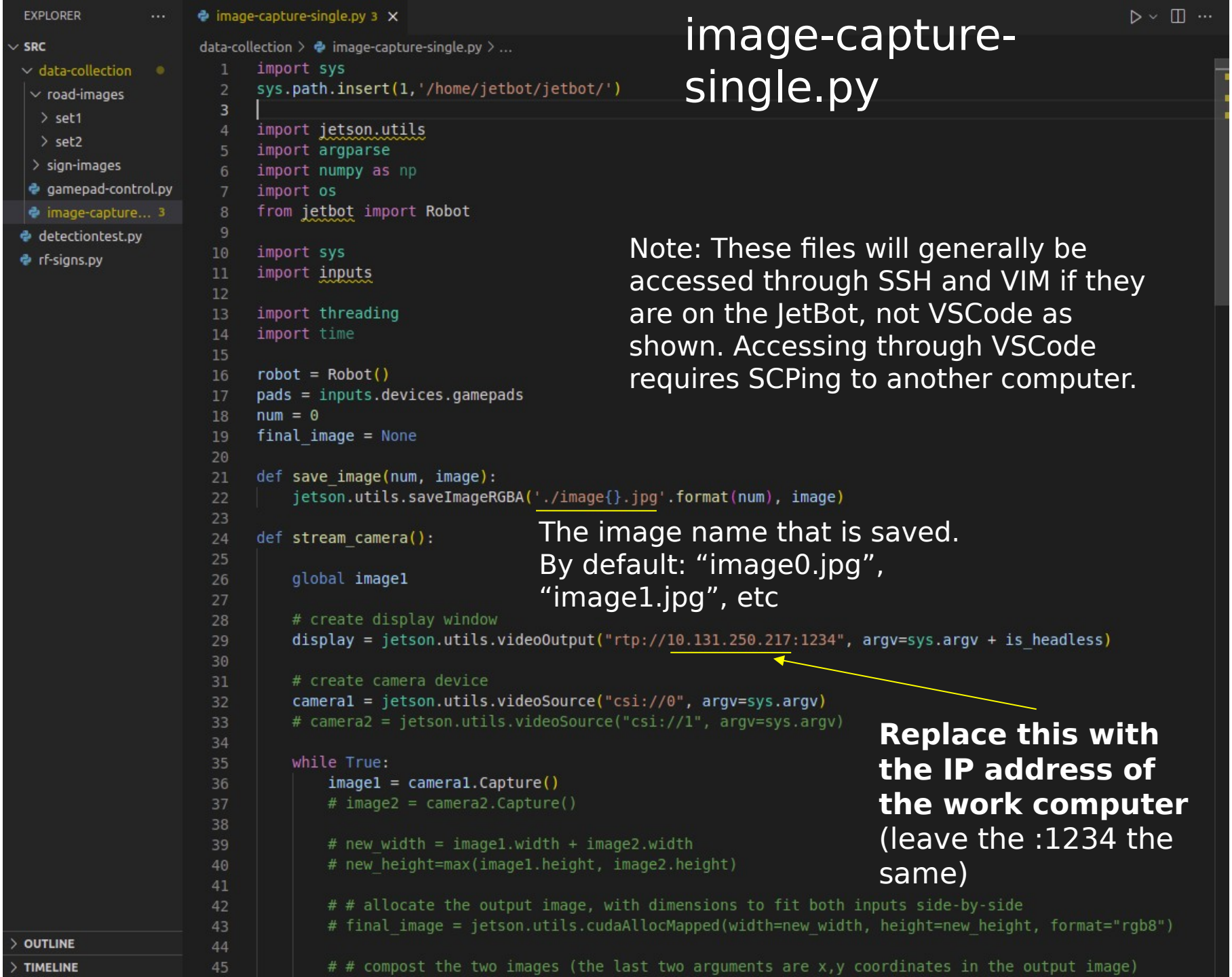
Desktop/

2. Data Collection

After connecting to the JetBot with SSH, run gamepad-control.py and image-capture-single.py in two separate terminals. These are in *jetbot/src/data-collection* (or src-2/data-collection), you must navigate there first.
Using the gamepad controller, take images of objects of interest at various locations. Adjust code as described below. Info about image-capture-single.py is also shown on subsequent slides.

- Set the correct IP address of the work computer in image-capture-single.py.
  - `display = jetson.utils.videoOutput("rtp://[YOUR IP HERE]:1234", argv=sys.argv + is_headless)`
  - This will broadcast camera feed to the correct work computer via gstreamer.
  - Most other files will require changing this as well.
  - Keep the “1234”!
  - You can check the IP address by opening a new terminal window on the work computer and typing the command `hostname -I`.
- At the top of main in image-capture-single.py, you’ll find `DATASET_PATH = ‘./[DIRECTORY]’`. This directory is where the images will be saved, you may wish to change it.
- Press the R1 button on the gamepad controller to take images.
- Use the joysticks to quickly move the robot between images, or simply adjust by hand.
- Take a couple hundred images or so, at varying angles and distances.
  - The greater quantity and variety of images taken, the better the

image-capture-single.py

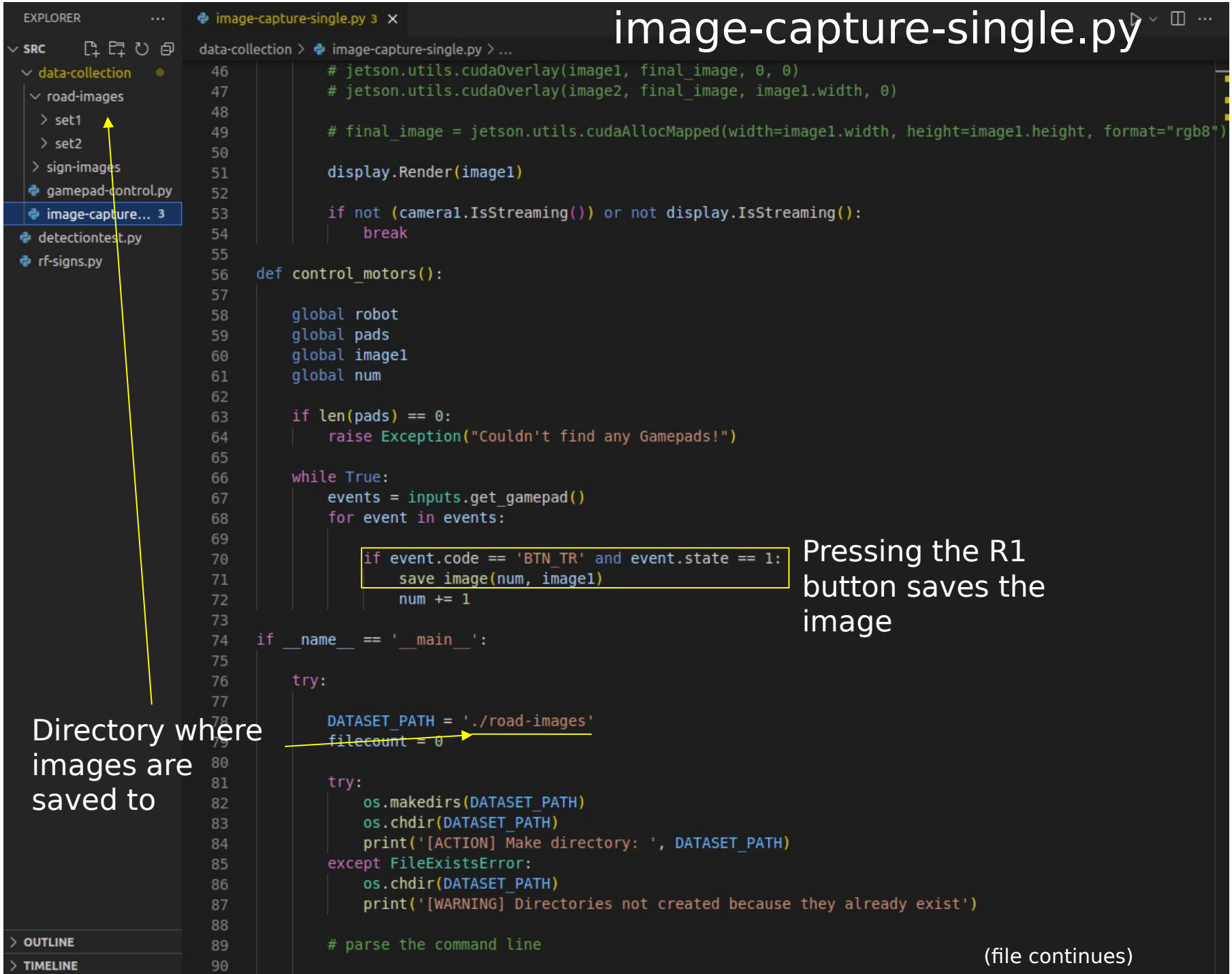


Note: These files will generally be accessed through SSH and VIM if they are on the JetBot, not VSCode as shown. Accessing through VSCode requires SCPing to another computer.

The image name that is saved. By default: “image0.jpg”, “image1.jpg”, etc

Replace this with the IP address of the work computer (leave the :1234 the same)

image-capture-single.py



Pressing the R1 button saves the image

Directory where images are saved to

(file continues)

3. Annotation

Use CVAT [1] to draw *bounding boxes* around signs.

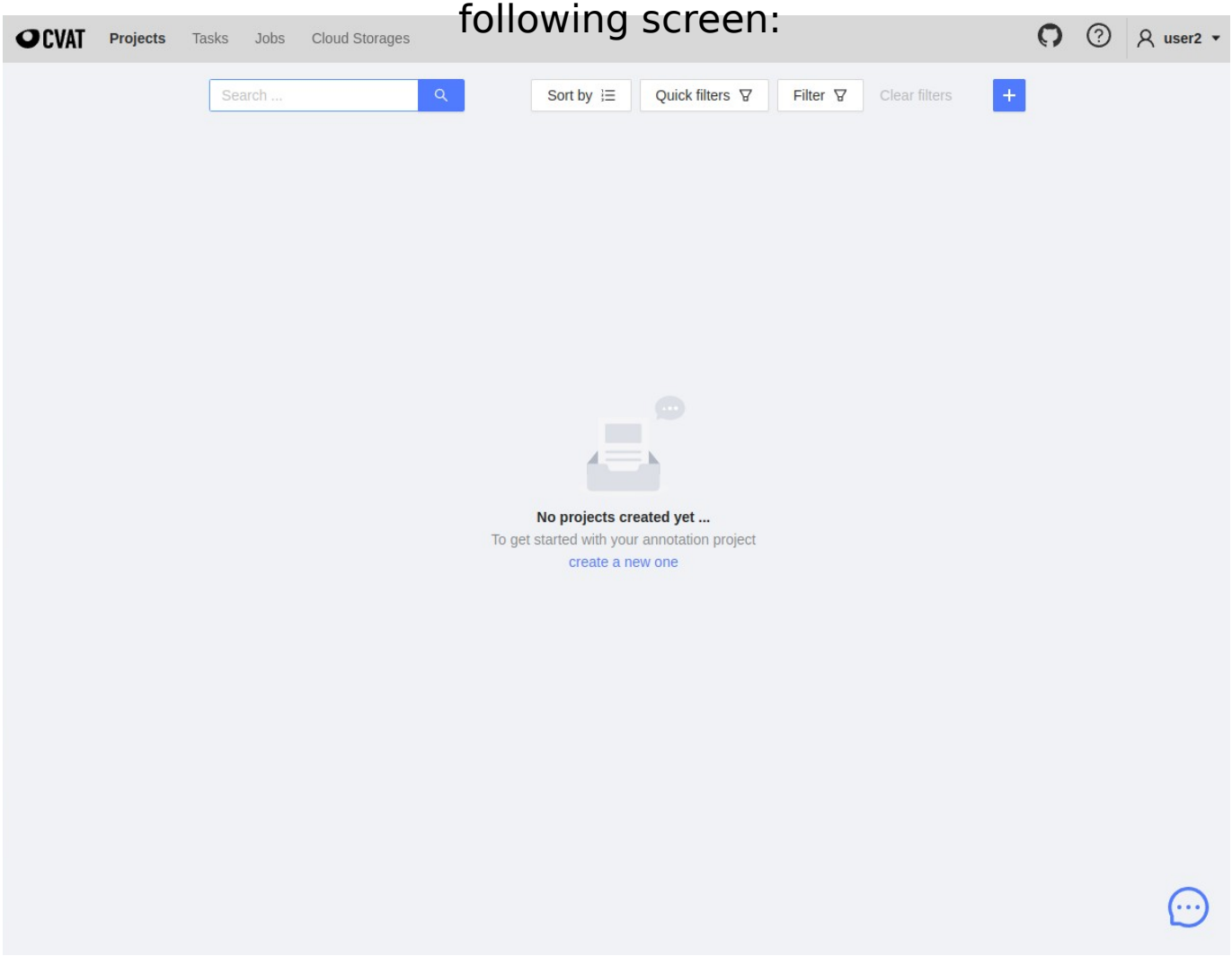
1) Setup: Follow the CVAT Installation Guide [2] and the CVAT User Guide [3]
2) When creating a project, set *labels* – the types of objects that the model should detect. For example, with signs, there is Left, Right, and U-turn.
3) When creating a task, upload images that were taken earlier.
4) Draw **rectangular** bounding boxes that cover the entirety of the object.
    a) Polygonal or other annotations will not work, apparently due to incompatibility with Pascal VOC.
5) When finished, export the task in Pascal VOC format. There is an option to export the images themselves, but there's no need.

More details and images of CVAT appear on subsequent slides.

[1] https://cvat.ai/
[2] https://github.com/maheriya/cvat/blob/master/cvat/apps/documentation/installation.md
[3] https://github.com/maheriya/cvat/blob/master/cvat/apps/documentation/user_guide.md

CVAT Home Page

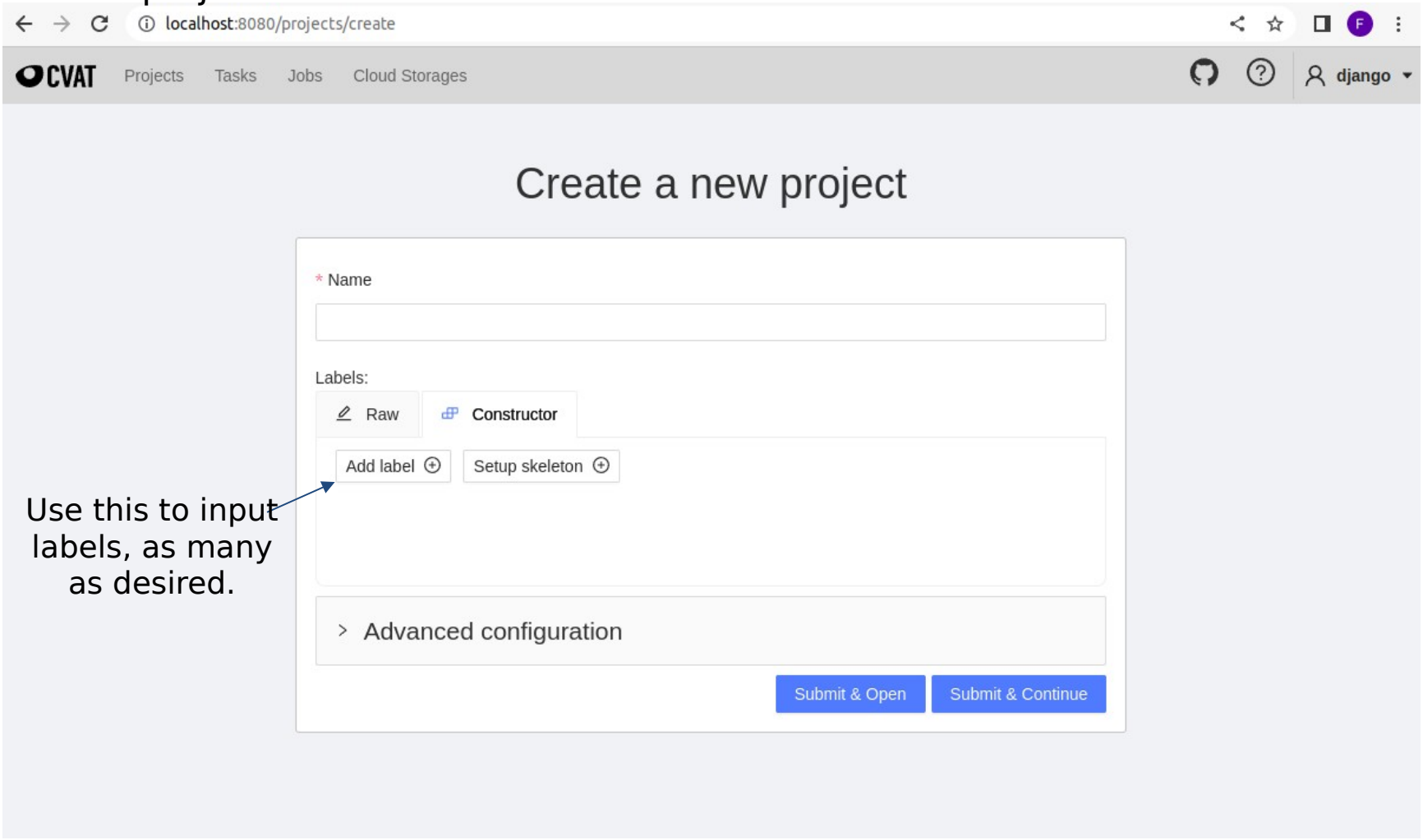
After going to localhost:8080 and setting up an account as described in the user guide, logging in gives the following screen:



Click the + button to create a new project.

Project Creation

From the home page, press the + button to create a new project.

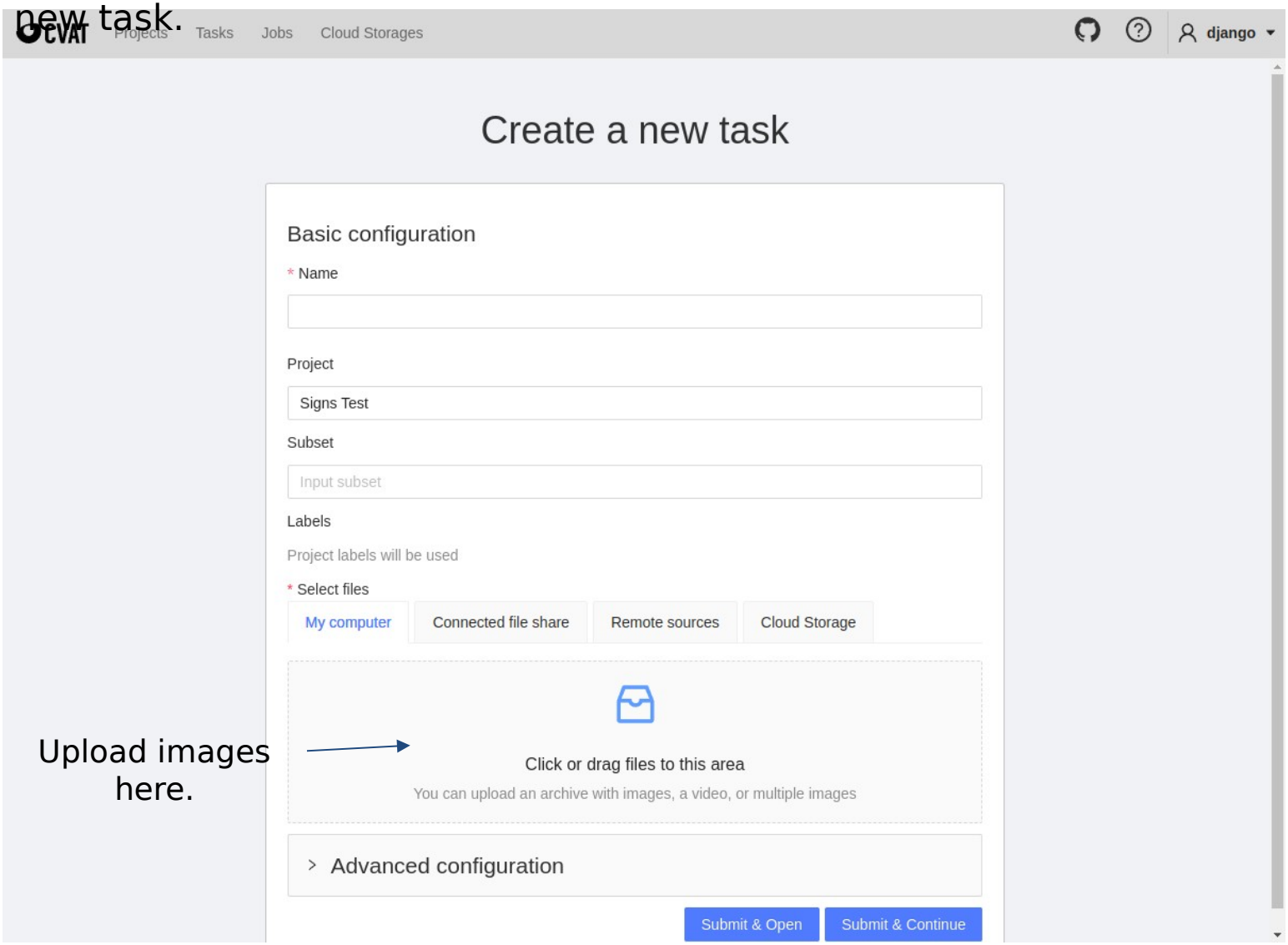


Use this to input labels, as many as desired.

After inputting a name and all labels, click “Submit & Open.” The names aren’t important but should be understandable.

Task Creation

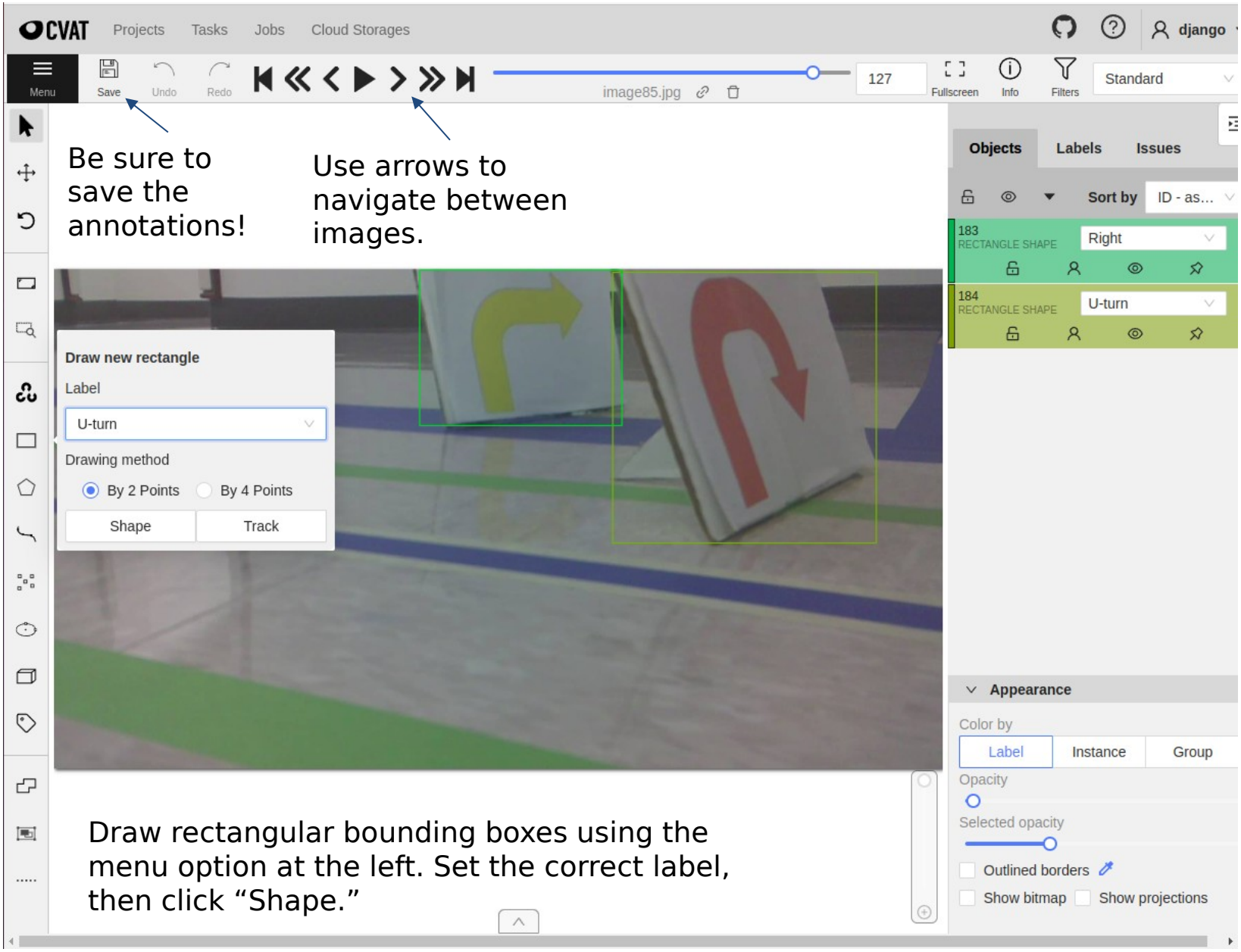
From the project page, press the + button and create a new task.



Upload images here.

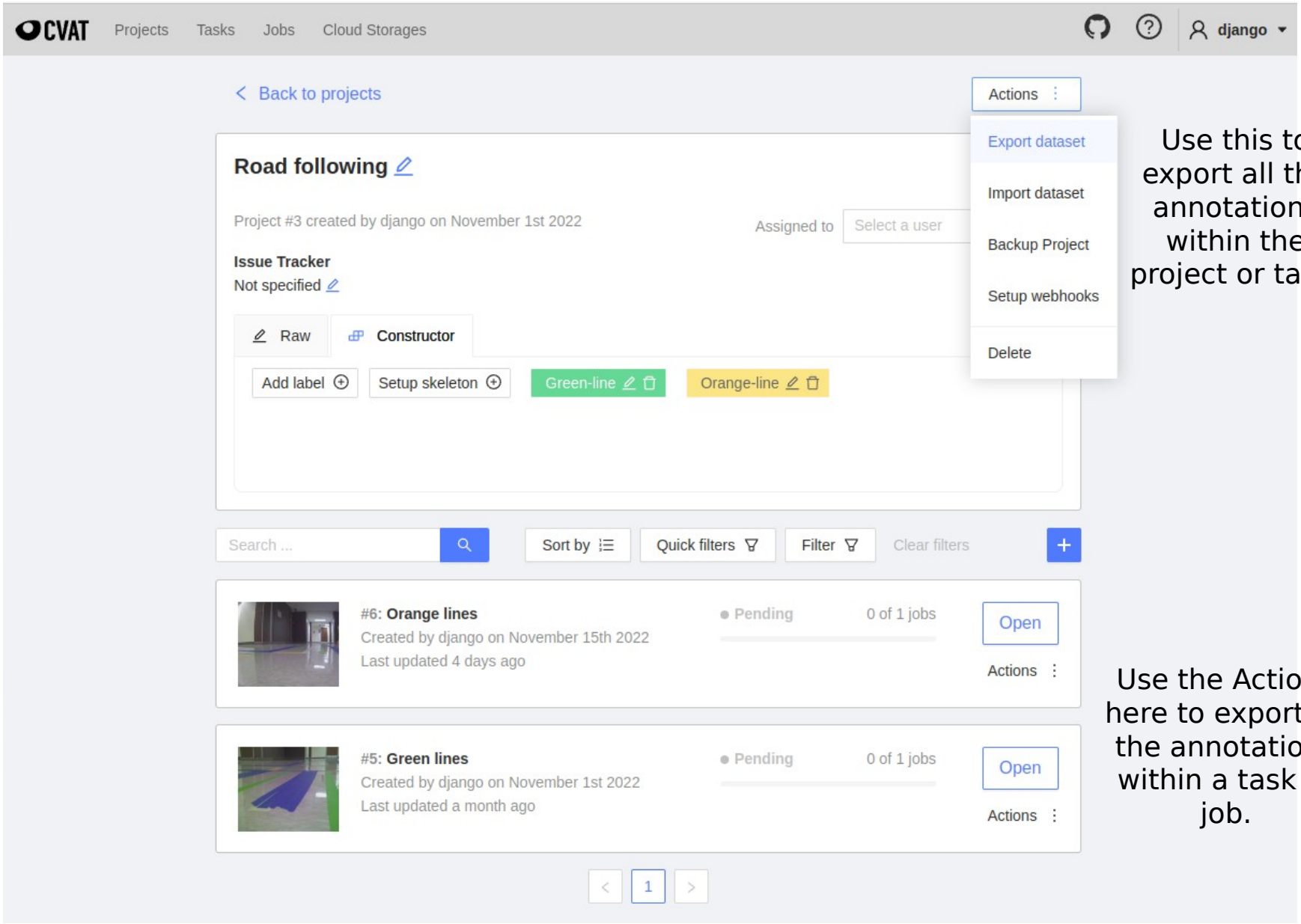
After inputting a name and uploading, click “Submit & Open.” Then, from the task page, click on the Job that appears toward the bottom.

Annotating with CVAT



Exporting Data

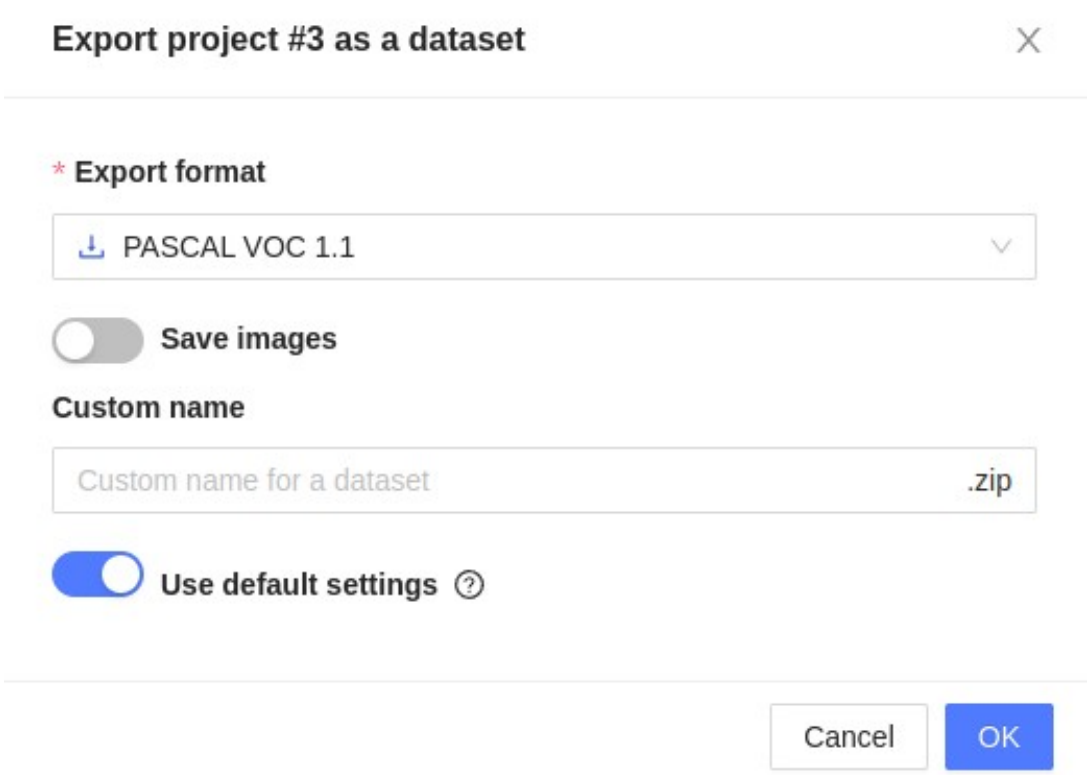
Return to a project or task page.



Use this to export all the annotations within the project or task.

Use the Actions here to export all the annotations within a task or job.

Exporting Data



When exporting, select PASCAL VOC 1.1 as the export format.

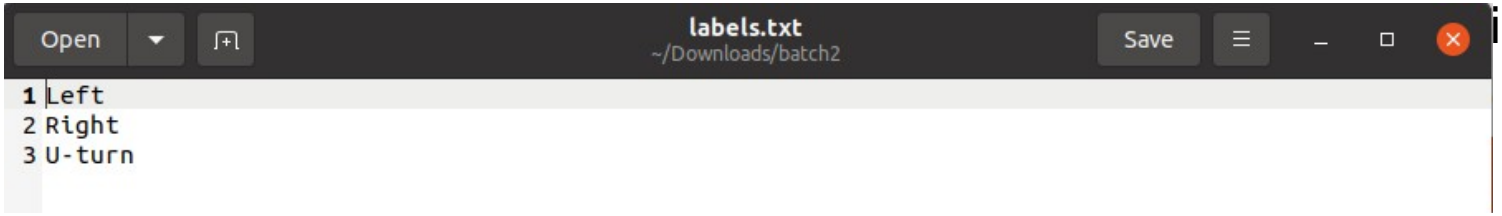
The rest can be ignored. Press “OK” to exit.

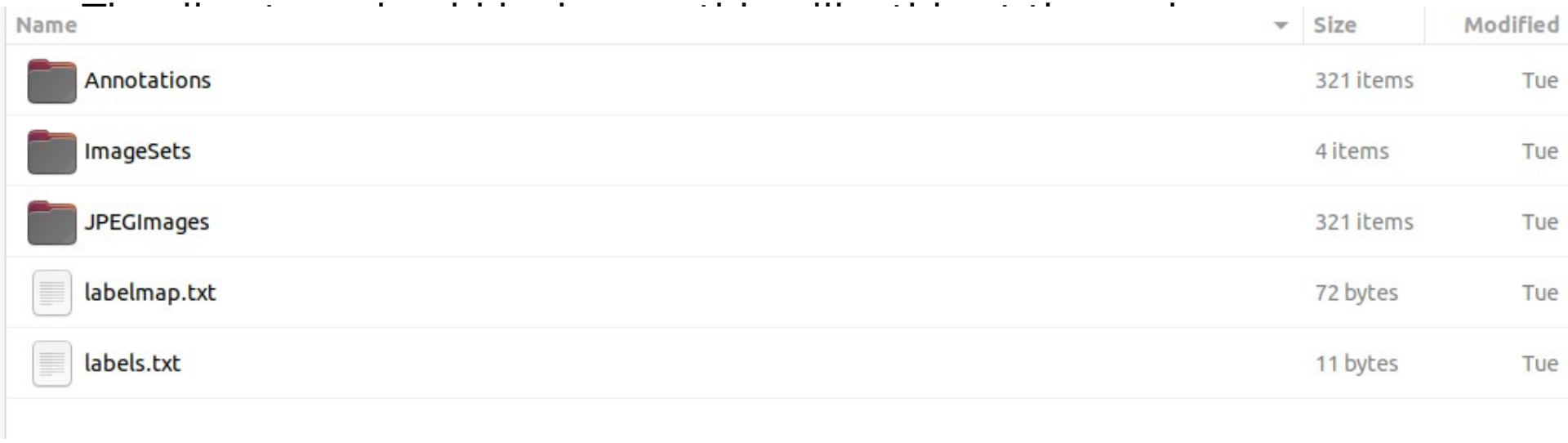
A zip file will be downloaded; extract it to a new folder. It should contain “Annotations,” “ImageSets,” and “labelmap.txt.” These contain the annotations!

4. Data Organization

Before training, add the images to the new folder from the CVAT export. In the directory (same as the folders called Annotations and ImageSets, add the following):

- A folder called “JPEGImages” containing all the images taken
- A text file called “labels.txt” containing the different labels (one line is:





5. Training

As outlined at https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-collect-detection.md#training-your-model (from the guide we are generally following for this project),

1) Put the folder created in the last slide into jetson-inference/python/training/detection/ssd/data.
2) While in the ssd folder (not data), use train_ssd.py to train an object detection model on the data collected:

```
python3 train_ssd.py --dataset-type=voc --data=data/<YOUR-DATASET> --model-dir=models/<YOUR-MODEL> --num-epochs=<NUM-EPOCHS>
```

<YOUR-DATASET> is the folder that was just put in ssd/data.
For <YOUR-MODEL>, create a folder in ssd/models to store the trained model.
<NUM-EPOCHS> is the number of training cycles.
See the next slide for a file tree before training.

- Sign model was trained for 300 epochs, road following was trained for 30 epochs.
- The more epochs, the better the fit, although too many epochs can lead to overfitting.

Training will take a while. When done,

File tree before training (on JetBot) – Key parts

jetson-inference/python/training/detection/ssd/

- data/
  - <YOUR-DATASET>/ *(arbitrary directory name)*
    - Annotations/ *(contains .xml files)*
    - ImageSets/ *(contains a directory called Main)*
    - JPEGImages/ *(contains all the images)*
    - labelmap.txt
    - labels.txt *(contains the labels)*
  - *(possibly other old datasets if this isn't the first time training)*
- models/
  - <YOUR-MODEL>/ *(arbitrary directory name)*
    - *(nothing here yet, empty directory)*
  - *(possibly other old models if this isn't the first time training)*
- train_ssd.py
- onnx_export.py

6. Testing

To test a trained model on live camera feed:

- Copy the onnx model file and the newly created labels text file to an appropriate directory, e.g., jetbot/models.
- Run detectiontest.py after editing the code to refer to the appropriate model and label files. See next slide for details.
- Set up the robot and environment and point the camera toward various objects of interest (the objects that were labeled earlier). Check the GStreamer feed and see if the bounding boxes are appearing as expected. (See the Improved Turning Camera View video toward the end for an idea of how it might look.)

Note: The label file is the one generated during training in the models folder. It is not the labels file made when setting up the directory before training.

Don’t forget to update the rtp IP address for viewing the camera feed through GStreamer.

detectiontest.py

```
import jetson.inference
import jetson.utils
import argparse
import sys
import keyboard

if __name__ == '__main__':

    parser = argparse.ArgumentParser(description="Classify a live camera stream using an image recognition DNN.",
                                     formatter_class=argparse.RawTextHelpFormatter,
                                     epilog=jetson.inference.imageNet.Usage() +
                                            jetson.utils.videoSource.Usage() + jetson.utils.videoOutput.Usage() + jetson.utils.l
ogUsage())
    parser.add_argument("input_URI", type=str, default="", nargs='?', help="URI of the input stream")
    parser.add_argument("output_URI", type=str, default="", nargs='?', help="URI of the output stream")
    parser.add_argument("--network", type=str, default="resnet18",
                        help="pre-trained model to load (see below for options)")
    parser.add_argument("--overlay", type=str, default="box,labels,conf", help="detection overlay flags (e.g. --overlay=box,labels,c
onf)\nvalid combinations are:  'box', 'labels', 'conf', 'none'")
    parser.add_argument("--threshold", type=float, default=0.5, help="minimum detection threshold to use")
    parser.add_argument("--camera", type=str, default="0",
                        help="index of the MIPI CSI camera to use (e.g. CSI camera 0)\nor for VL42 cameras, the /dev/video device to
 use.\nby default, MIPI CSI camera 0 will be used.")
    parser.add_argument("--width", type=int, default=640, help="desired width of camera stream (default is 1280 pixels)")
    parser.add_argument("--height", type=int, default=720, help="desired height of camera stream (default is 720 pixels)")
    parser.add_argument('--headless', action='store_true', default=(), help="run without display")

    # For rtp streaming
    is_headless = ["--headless"] if sys.argv[0].find('console.py') != -1 else [""]

    try:
        opt = parser.parse_known_args()[0]
    except:
        print("")
        parser.print_help()
        sys.exit(0)

    # load the recognition network
    #collision_net = jetson.inference.imageNet(argv=['--model=/home/jetbot/jetbot/models/intersection-resnet18.onnx', '--labels=/hom
e/jetbot/jetbot/models/intersection-labels.txt', '--input-blob=input_0', '--output-blob=output_0'])
    sign_net = jetson.inference.detectNet(argv=['--threshold=0.15','--model=/home/jetbot/jetbot/models-2/orange-green-lines.onnx', '
--labels=/home/jetbot/jetbot/models-2/orange-green-labels.txt', '--input-blob=input_0', '--output-cvg=scores', '--output-bbox=boxes'
])
    #detect_net = jetson.inference.detectNet(argv=['--model=/home/jetbot/jetbot/models/detection-ssd-mobilenet.onnx', '--labels=/hom
e/jetbot/jetbot/models/detection-labels.txt', '--input-blob=input_0', '--output-cvg=scores', '--output-bbox=boxes'] )
    # sign_net = jetson.inference.detectNet("ssd-mobilenet-v2", threshold=0.5)
    # create video sources & outputs
    camera1 = jetson.utils.videoSource("csi://0", argv=sys.argv)
    #camera2 = jetson.utils.videoSource("csi://1", argv=sys.argv)

    display = jetson.utils.videoOutput("rtp://10.131.124.154:1234", argv=sys.argv + is_headless)
```

Update these to be the correct path to the model and label files

As with image-capture-single.py, change the IP address to match that of the computer that receives the camera feed through GStreamer

7. Running the robot

Use rf-signs.py to run the robot. This combines road following and turning at signs.
The same changes must be made as with detectiontest.py:

- In the code, change the variables to match the model files generated in training.
- Don’t forget to update the IP address (rtp) for GStreamer!

Place the robot on an end of the course, facing the intersection, and run rf-signs.py while in the src directory:

```
sudo python3 rf-signs.py --flip-method=rotate-180
```

Useful link: detectNet class
https://github.com/dusty-nv/jetson-inference/blob/master/c/detectNet.h#L109

- Info about how to get the area, width, height, center, and other information about the bounding boxes

This may be useful for improving rf-signs.py.

Future Possibilities

- Improving the line following/turning
- Detection of other robots
- Segmentation

Updates

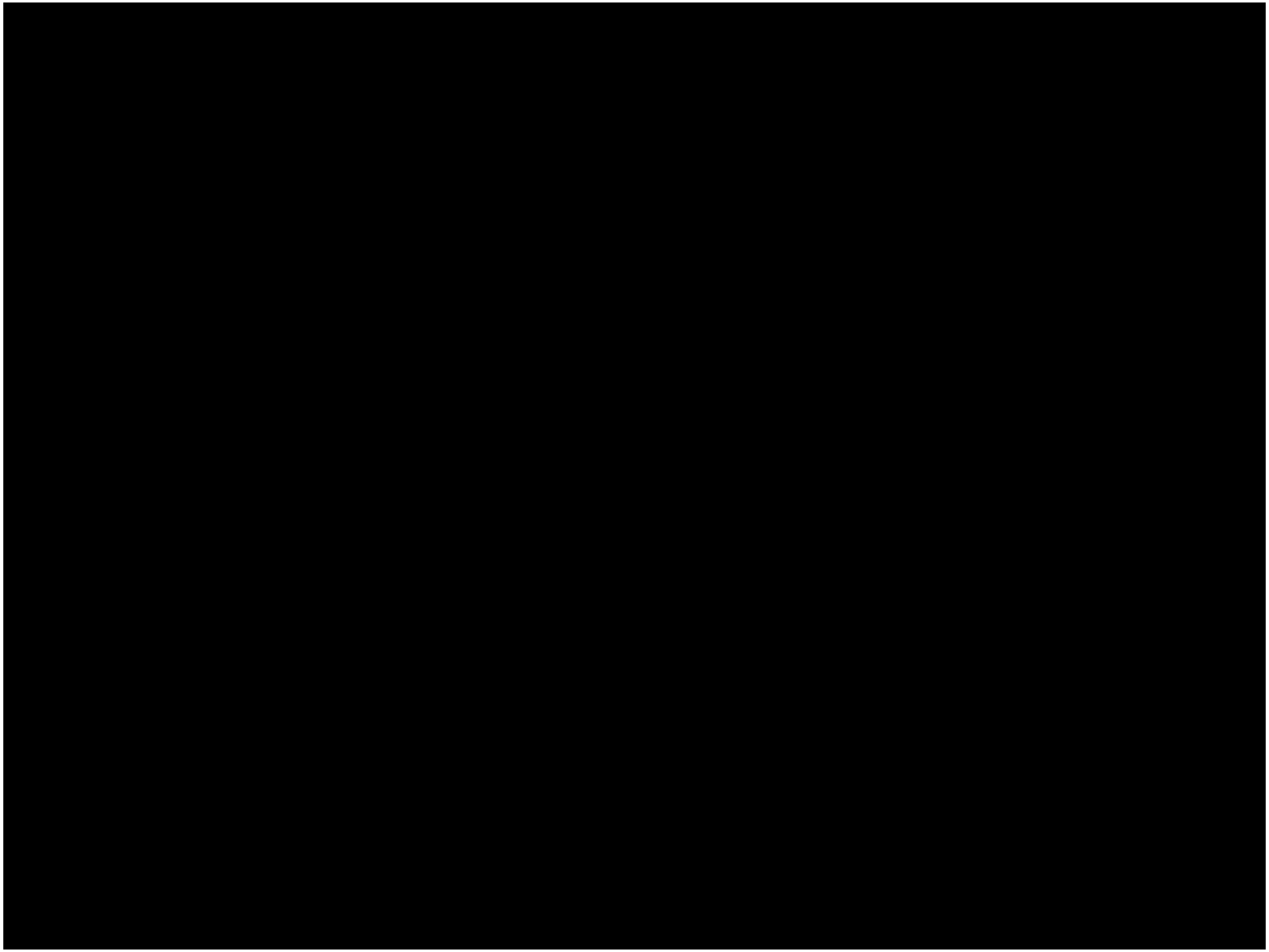
Improving turning

Using the same process but with more bounding boxes drawn, a new object detection model was trained to detect all green lines as well as the orange lines at the intersection.

This gives the robot the ability to turn more accurately by targeting a specific green line and dynamically adjusting to get there, rather than simply hard-coding the turn.

The resulting video, shown through the GStreamer point of view, is on the next slide.
(This is not perfect – some refinement is still needed.)

Object Detection Camera



Collision Avoidance

By following the example from jetbot.org (https://jetbot.org/master/examples/collision_avoidance.html), we also trained the robot to navigate a room without colliding into objects. This is a classification model – the robot is either “blocked” or “free.”

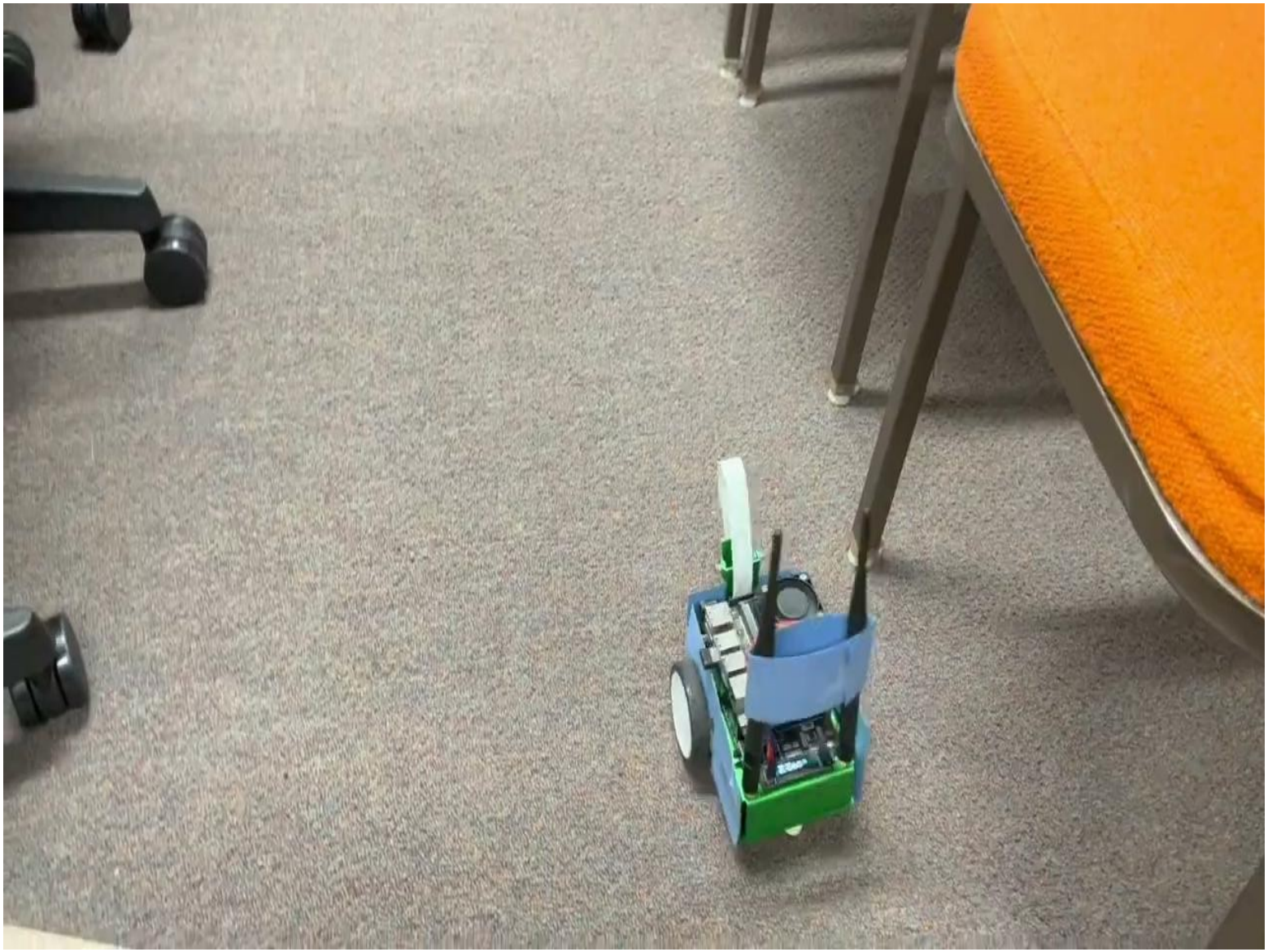
The process is similar:

1. Collect data on JetBot
    a. Capture various images of blocked vs free
    b. Label and organize the data
2. Train neural network on the data
    a. This uses transfer learning with ResNet-18
3. Optimize the model on Jetson Nano
    a. Using TensorRT
4. Run live demo on JetBot

More details can be found in the link above.

A video demo is shown on the next slide. This is, again, not perfect.

Collision Avoidance Demo



Some possible links of interest

https://www.nvidia.com/en-us/on-demand/session/gtcspring21-s31151/

https://www.nvidia.com/en-us/on-demand/session/gtcfall22-a41101/?playlistId=playList-5d17c33f-65f5-4fb1-9fed-bc5b7f4dec44

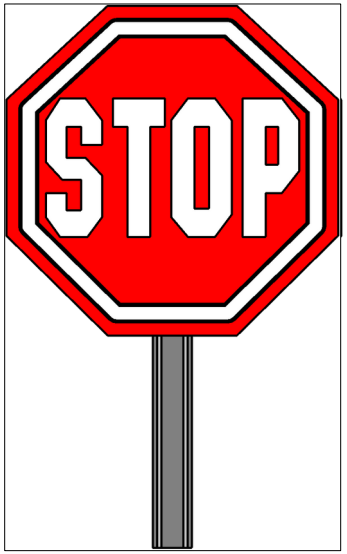
https://www.youtube.com/watch?v=4APkMJdiudU (7 parts)

https://www.nvidia.com/en-us/edge-computing/5g/

https://github.com/dusty-nv/jetson-inference/blob/master/docs/pytorch-collect-detection.md#training-your-model

https://gstreamer.freedesktop.org/documentation/installing/index.html?gi-language=c

The End



- The Er