



SPRINT 1

Chick Counting Project



JJ McCauley, Jason Holt,
Anye Forti, Logan Kelsch

01

Overview

01 OVERVIEW

02 HARDWARE

- Cameras
- Integration
- Hardware Configuration

03 BACKEND

- YOLO Implementation
- Ultralytics Object Counter
- Custom Counter Mount
- Clustering Options
- Non-YOLO Approaches

04 SPRINT GOALS

- Sprint 2 Goals
- Potential Constraints

05 CLOSING

- Questions
- Thank You

02

Hardware

Cameras

RGBs:

Arducam 5mp

1080p

30 fps

CoolPix 900

1080p

30 fps

Thermal:

Long-wave IR Thermal Imaging

80x62

25 fps

Current Camera-pi Configuration

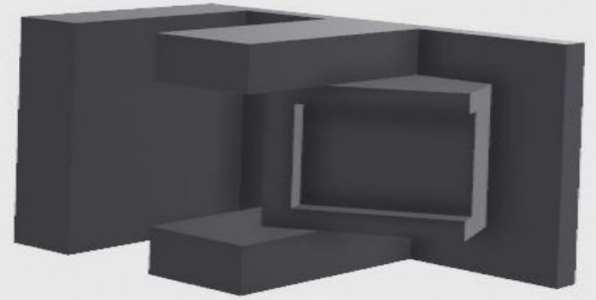


2.1

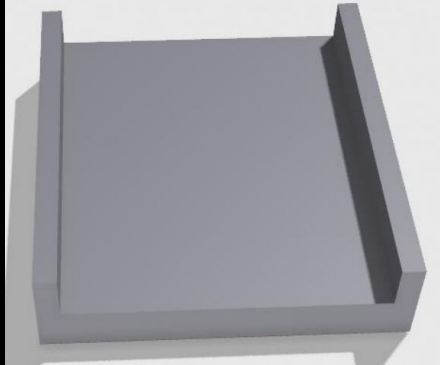
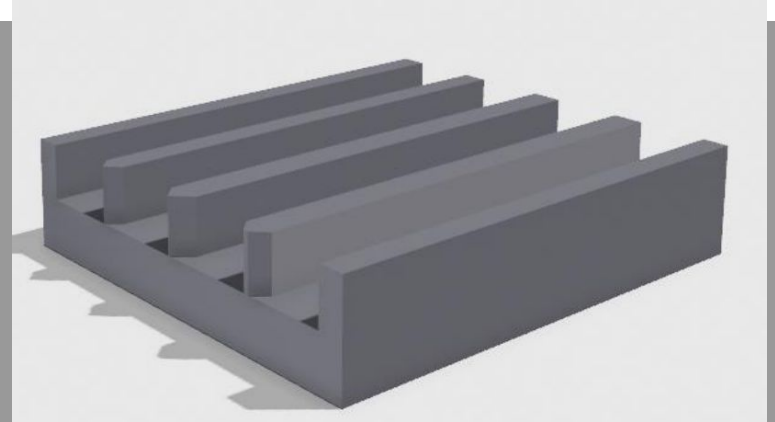
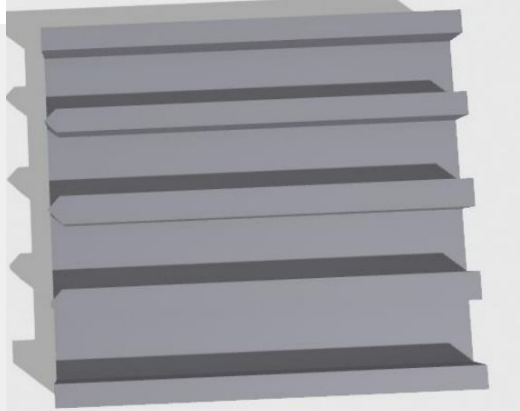
Hardware Configuration

Hardware Configuration

Pi RGB Camera Holder



Hardware Configuration Testing Track

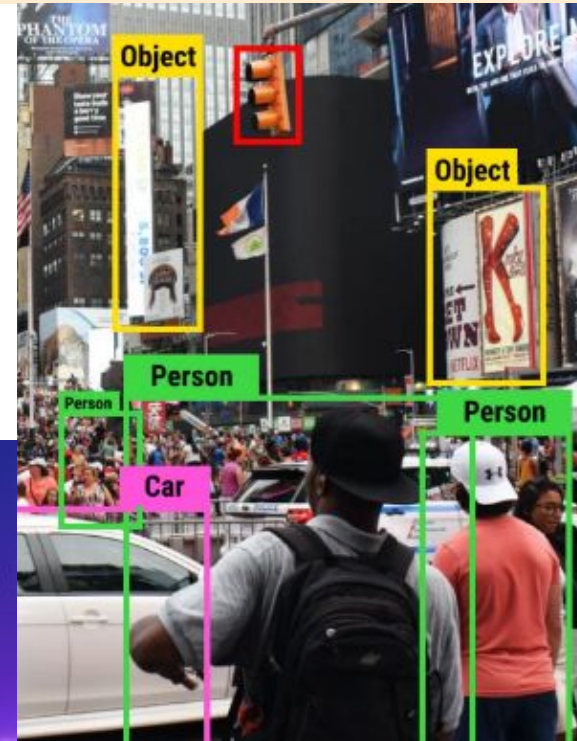


03

Backend

Client's Desired Path

Implementation of the pre-trained image classification model, "YOLOv11", by Ultralytics to identify and count chicks.



```
# Open the video file
cap = cv2.VideoCapture("data/fishies.mp4")
assert cap.isOpened(), "Error reading video file"

# Get video properties: width, height, and frames per second (fps)
w, h, fps = (int(cap.get(x)) for x in (cv2.CAP_PROP_FRAME_WIDTH, cv2.CAP_PROP_FRAME_HEIGHT, cv2.CAP_PROP_FPS))

# Define points for a line or region of interest in the video frame
line_points = [(400, 100), (400, 620)] # Line coordinates

# Initialize the video writer to save the output video
video_writer = cv2.VideoWriter("object_counting_output.avi", cv2.VideoWriter_fourcc(*"mp4v"), fps, (w, h))

model = "models/yolo11n.pt"

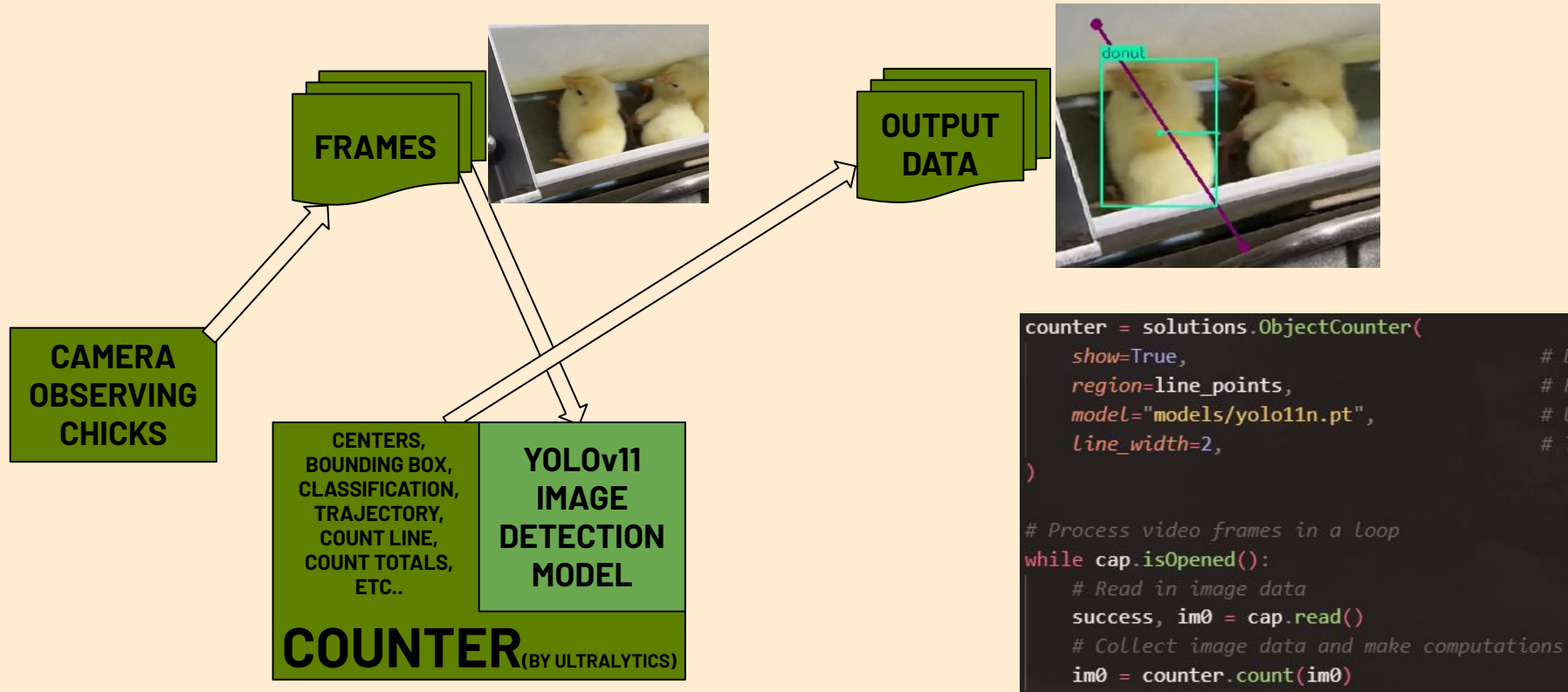
# Initialize the Object Counter with visualization options and other parameters
counter = solutions.ObjectCounter(
    show=True, # Display the image during processing
    region=line_points, # Region of interest points
    model="models/yolo11n.pt", # Ultralytics YOLO11 model file
    line_width=2, # Thickness of the lines and bounding boxes
)

# Process video frames in a loop
while cap.isOpened():
    success, im0 = cap.read()
    if not success:
        print("Video frame is empty or video processing has been successfully completed.")
        break

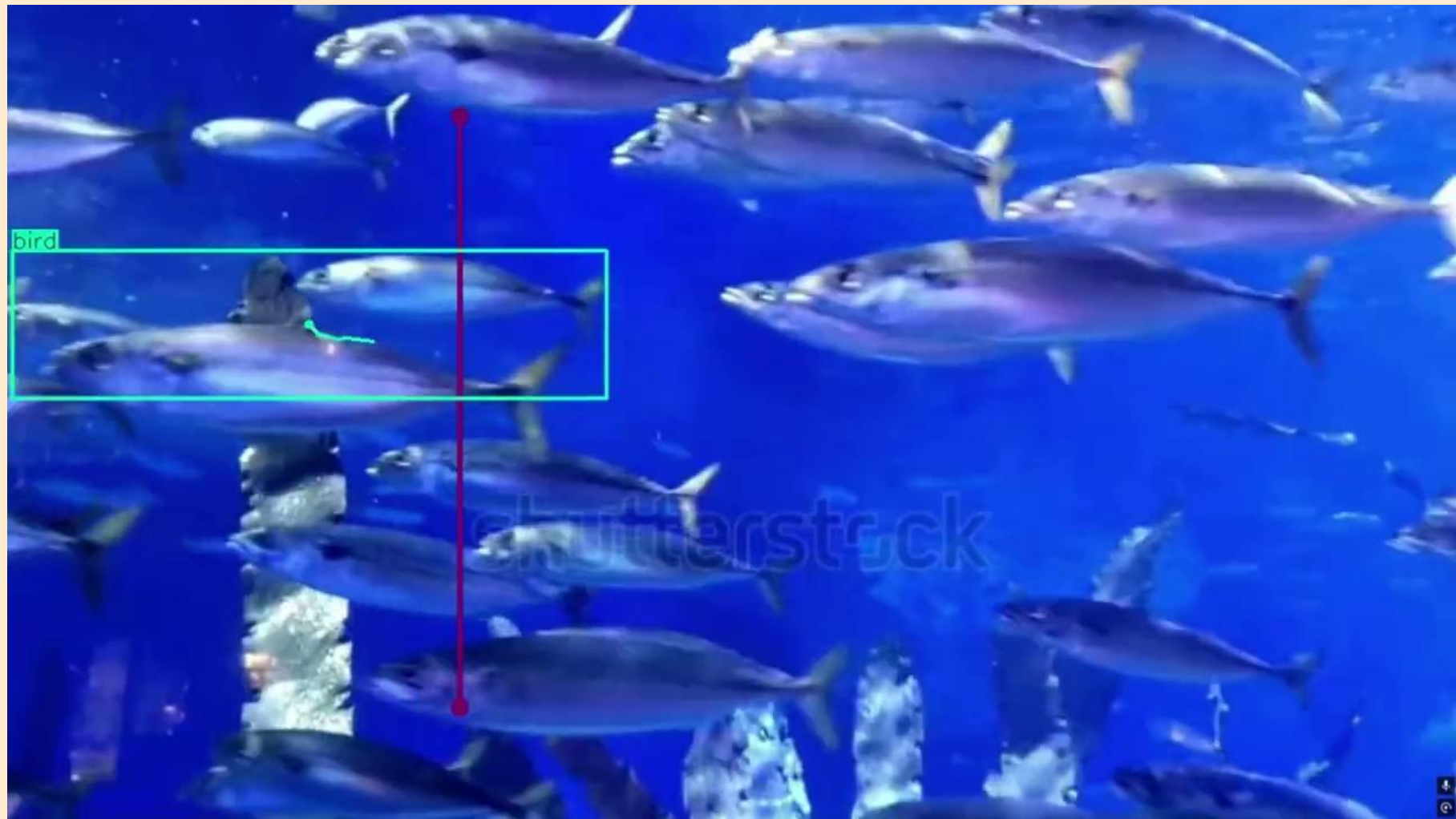
    # Use the Object Counter to count objects in the frame and get the annotated image
    im0 = counter.count(im0)

    # Write the annotated frame to the output video
    video_writer.write(im0)
```

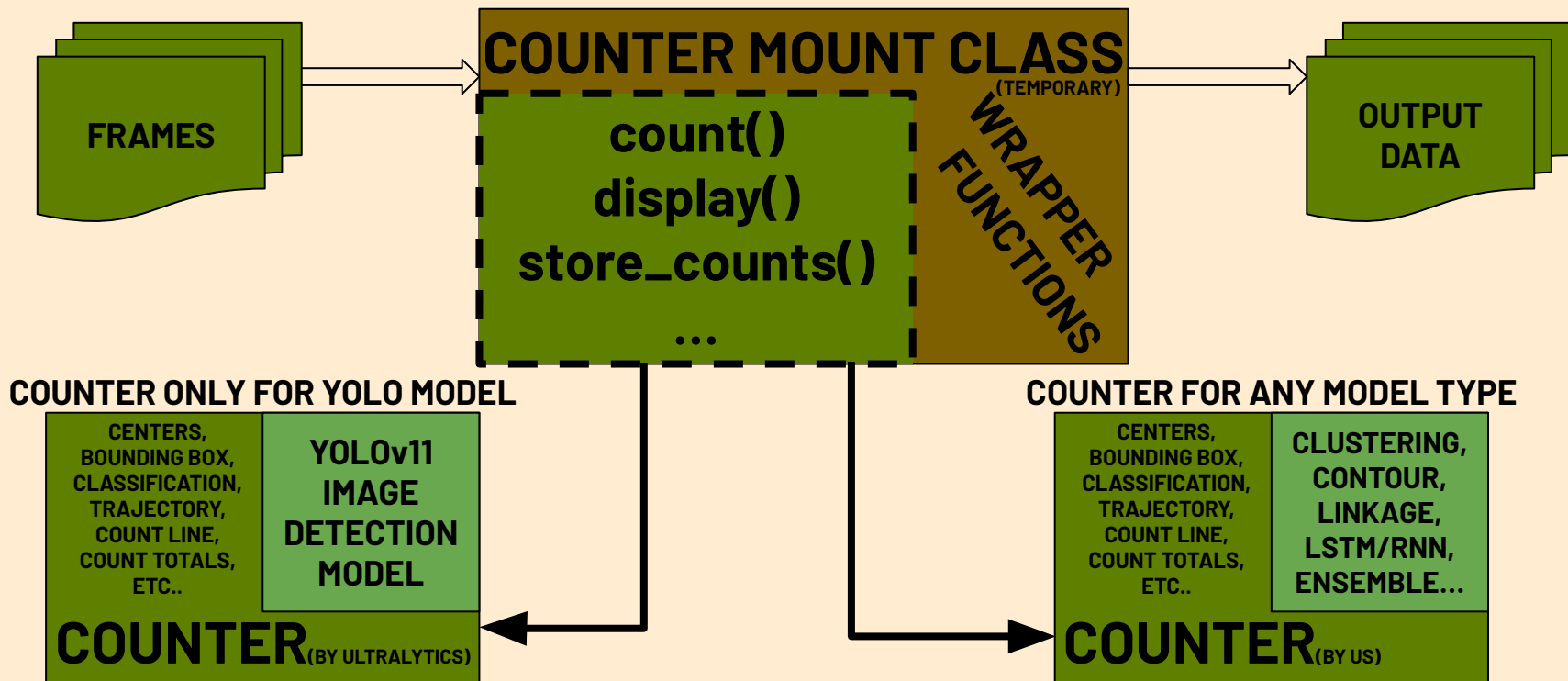
Ultralytics Live Object Counting







Expanding Capacity of Counter




```

10 #counter class acts as a mount for yolo or custom models
11 class Counter():
12     ...
13     ### info: ###
14     This class will load and mount a desired counter method.
15     ### params: ###
16     - counter-type:
17     - - Type of clustering model that will be used.
18     - counter-kwags:
19     - - Dict variable containing all parameters and values for the counter.
20     ...
21 def __init__(
22     self,
23     counter_type : Literal['YOLO','Clustering'] = None,
24     counter_kwags : dict = {}
25 ):
26     assert (counter_type != None), "Counter_type not defined for class 'Counter'. Please select a valid option."
27
28     #define the counter type based off of counter method used
29     match(counter_type):
30
31         #if we are working with a YOLO based counter, make this class act as a mount
32         case 'YOLO':
33             self._counter = yolo_implementation.YOLO_ObjectCounter(**counter_kwags)
34
35         #if we are working with a custom clustering model, NOTE DEV HERE END#NOTE
36         case 'Clustering':
37             raise NotImplementedError(f'Counter type {counter_type} has not yet been implemented.')
38
39     #this variable keeps the string variable of the type of counter, ex: YOLO
40     self._counter_type = counter_type
41
42     #this variable is a list of tuples, with each tuple being a coordinate of a center
43     self._detected_object_centers = []
44
45     #this variable is NOTE POSSIBLY TEMPORARY NOTE for tracking projection of centers
46     #this variable will be a tuple of projected X, Y offset for next frame
47     self._detected_object_vectors = []
48
49     #this variable will be a dictionary (str:int) of detection classifications and totals over time
50     self._detected_totals = {}

```

CLUSTERING APPROACHES

```
class contour():  
    ...  
class kmeans():  
    ...  
class complete_linkage():  
    ...  
class single_linkage():  
    ...  
class temporal_crf():  
    ...
```

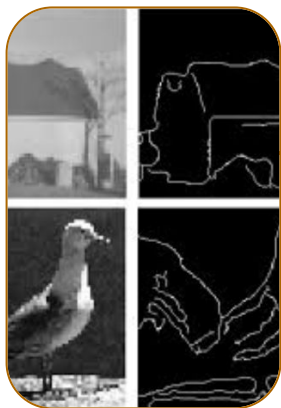
TRACKING APPROACHES

```
class DETR():  
class DeepSORT():
```

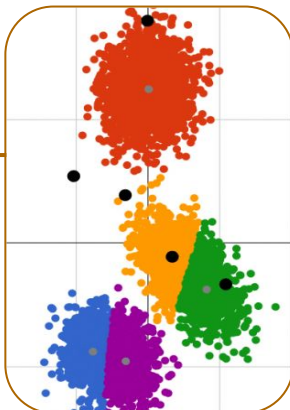
TEMPORAL / TIME-SERIES
NEURAL NETWORK (LSTM-RNN)

Clustering / Tracking Methods

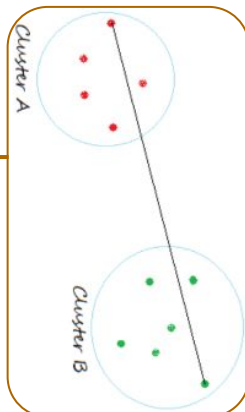
19



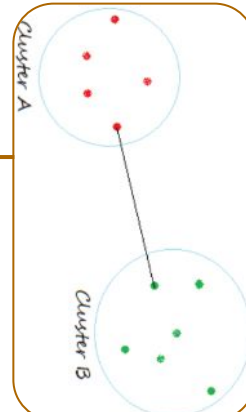
CONTOUR DETECTION
 $O(n) < t < O(n \log n)$
 COMPLEXITY



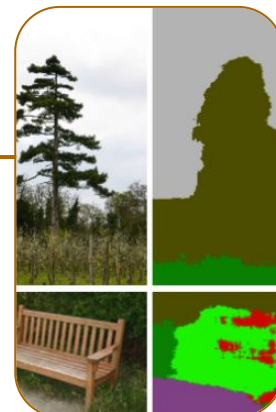
K-MEANS CLUSTERING
 $O(n * p * q * r)$
 COMPLEXITY



COMPLETE LINKAGE
 $O(n^2) < t < O(n^3)$
 COMPLEXITY



SINGLE LINKAGE
 $O(n^2) < t < O(n^3)$
 COMPLEXITY



CONDITIONAL RANDOM FIELD
 $O(n * k^2)$
 COMPLEXITY



RT-DETR $O(n) < t < O(n^2 * k^2)$ COMPLEXITY



DEEPSORT $O(n) < t < O(n \log n)$ COMPLEXITY

Tracking Loophole / Other Counting Approach

LSTM-RNN

```
def build_LSTM_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Input(shape=(X_train.shape[1],X_train.shape[2])),
        tf.keras.layers.LSTM(256, activation='tanh', recurrent_dropout=0.0, return_sequences=True),
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.LSTM(128, activation='tanh', recurrent_dropout=0.0, return_sequences=False),
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Dense(64),# kernel_regularizer=tf.keras.regularizers.l2(0.01)),
        tf.keras.layers.BatchNormalization(),
        tf.keras.layers.Activation('relu'),
        tf.keras.layers.Dropout(0.25),
        tf.keras.layers.Dense(params.target_neurons, activation=params.target_activation)
    ])
    model.compile(optimizer=opt6,
                  loss=params.loss_function,
                  metrics=params.performance_metrics)
    return model
```

04

Sprint Goals

Sprint 2 Goals

Configure All
Cameras



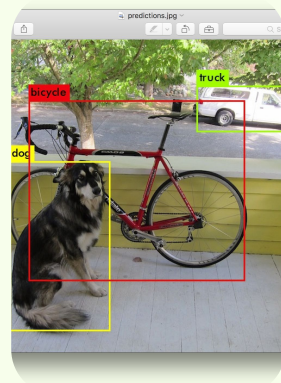
Attach, Write Scripts for
RGB & Thermal Cameras, testing
on live environments and
translating video to
model-usable format

Perform Tests
using Mock
Environment



Utilize the Cameras & YOLO Models
To run tests in the simulated
environment

Optimize YOLO &
Custom Models



Using simulated environments
Test and optimize YOLO &
Custom models to determine the
best approach looking forward

Updated Product/Sprint 2 Backlog

The image displays two views of a Jira backlog. The left view shows a 'Sprint Backlog' for 'Sprint 2' (Estimate: 0) with tasks categorized into four columns: 'To Do', 'In Process', and 'Awaiting Review'. The right view shows the 'Product Backlog' with tasks categorized into 'New Tasks', 'In Process', and 'Completed' columns.

Sprint Backlog (Sprint 2, Estimate: 0)

- To Do (Tasks/Spikes) (Estimate: 0)**
 - Check-Counting #24: Determine Methods to Transfer BASH Scripted-video to Python Models
 - Check-Counting #23: Configure & Assemble Simulated Environment
- In Process (Tasks/Spikes) (Estimate: 0)**
 - Check-Counting #7: Collect continuous data using Raspberry Pi
- Awaiting Review (Tasks/Spikes) (Estimate: 0)**
 - This item is awaiting peer review

Product Backlog (Increased items preview | Feedback)

Backlog | Team capacity | Current iteration | Roadmap | My items | + New view

Filter by keyword or by field

- New Tasks (3, Estimate: 0)**
 - Check-Counting #2: Collect Feature Data from Perdue
 - Check-Counting #8: Push data from Raspberry Pi into Database
 - Check-Counting #6: Optimize Vision Detection Model
- In Process (2, Estimate: 0)**
 - Check-Counting #4: Create Database Design
 - Check-Counting #5: Train Initial Vision Detection Model
- Completed (4, Estimate: 0)**
 - Check-Counting #1: Define Chick Input Methods
 - Check-Counting #7: Collect continuous data using Raspberry Pi
 - Check-Counting #9: Redefine Requirements (after Client Meeting)
 - Check-Counting #3: Set up Raspberry Pi with input methods

Questions?

THANK YOU

Chick Counting