Safe Navigation

Our Team

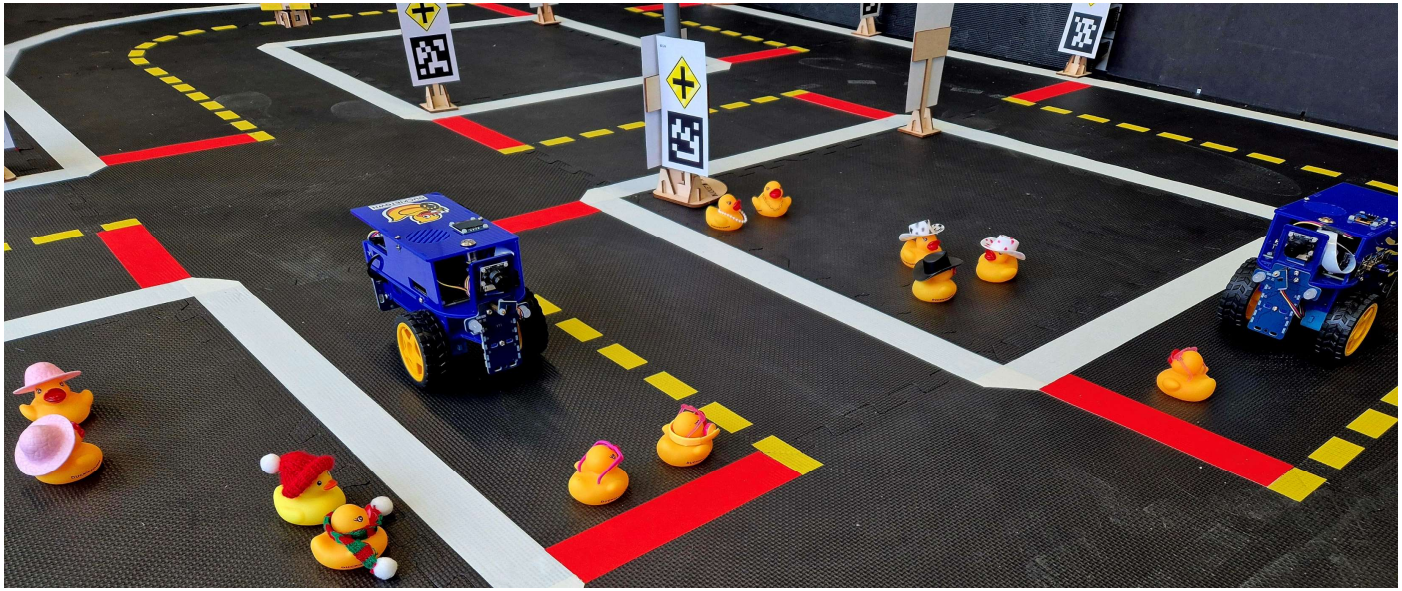
Ilia Panayotov

Esra Mehmedova

Esin Mehmedova

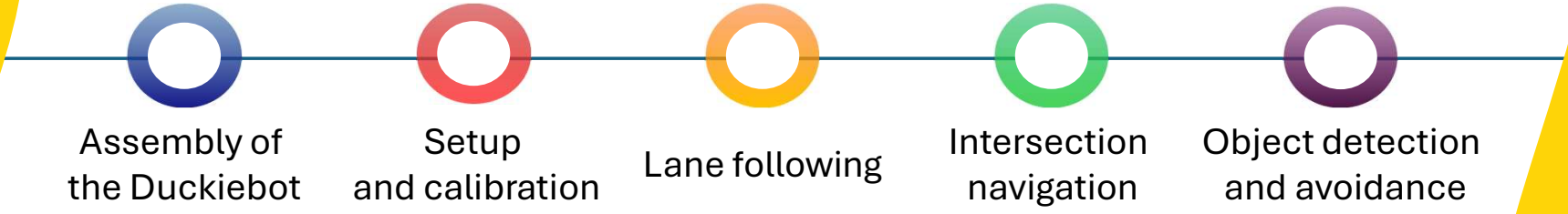
Our Goal

Develop a robot that can safely navigate a town, avoiding other Duckiebots and ducks



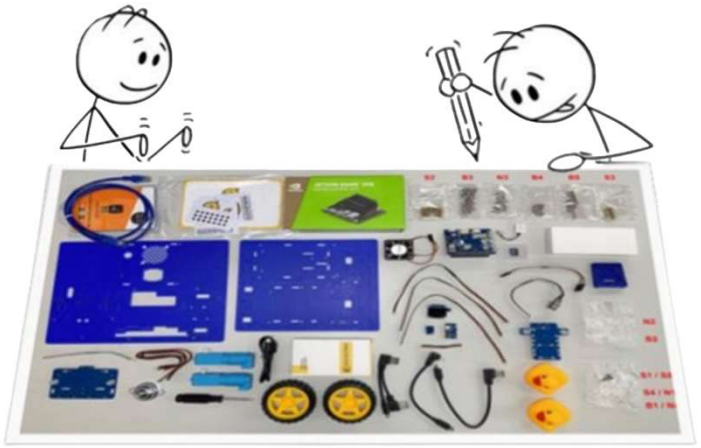
Milestones

Over the course of this project
we have successfully accomplished the following:



Assembly of the Duckiebot

We have assembled the Duckiebot, transforming individual components into a fully functional robot



Milestones

Over the course of this project
we have successfully accomplished the following:

Assembly of the Duckiebot

Setup and calibration

Lane following

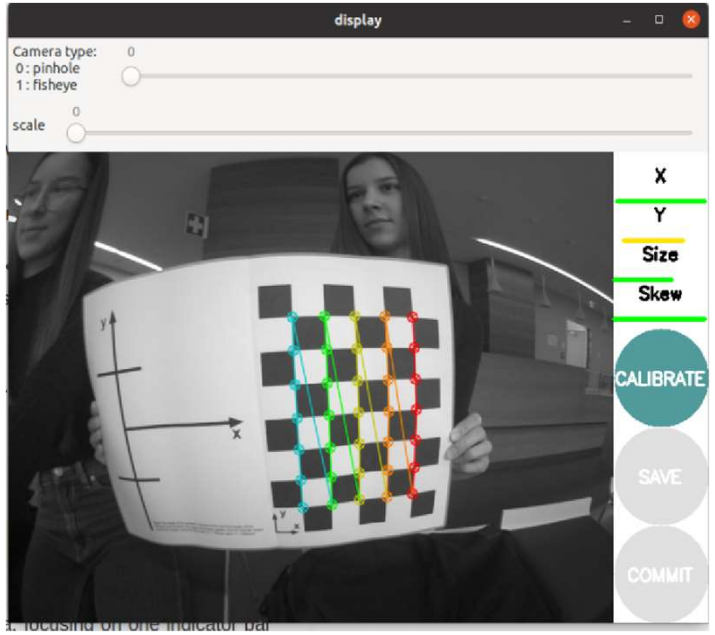
Intersection navigation

Object detection and avoidance

Setup and calibration

Following the tutorials, we have:

- set up our laptop and accounts
- flashed the SD card
- calibrated the Duckiebot's wheels and camera



Milestones

Over the course of this project
we have successfully accomplished the following:

Assembly of the Duckiebot

Setup and calibration

Lane following

Intersection navigation

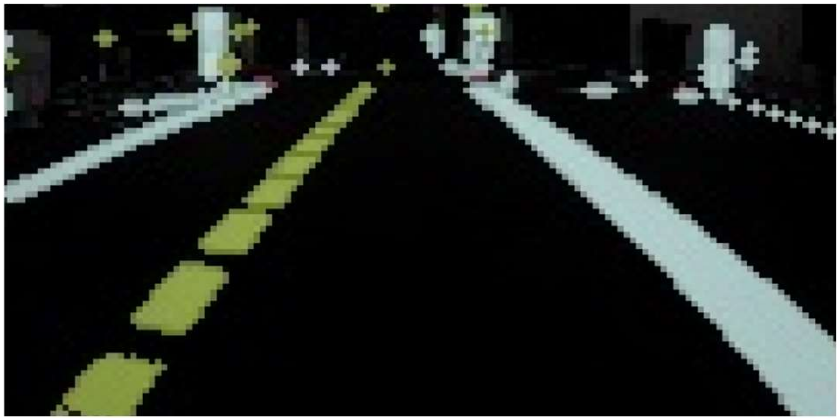
Object detection and avoidance

Lane following

We used the provided line-following demo

- The **camera** captures real-time images of the road and processes them to identify lane markings:

  Yellow lines mark the left boundary
  White lines mark the right boundary



- Once the lanes are detected, the system determines the **relative position of the Duckiebot** on the road

- The **PID controller** is used to adjust the Duckiebot's angle and direction based on the lane position error

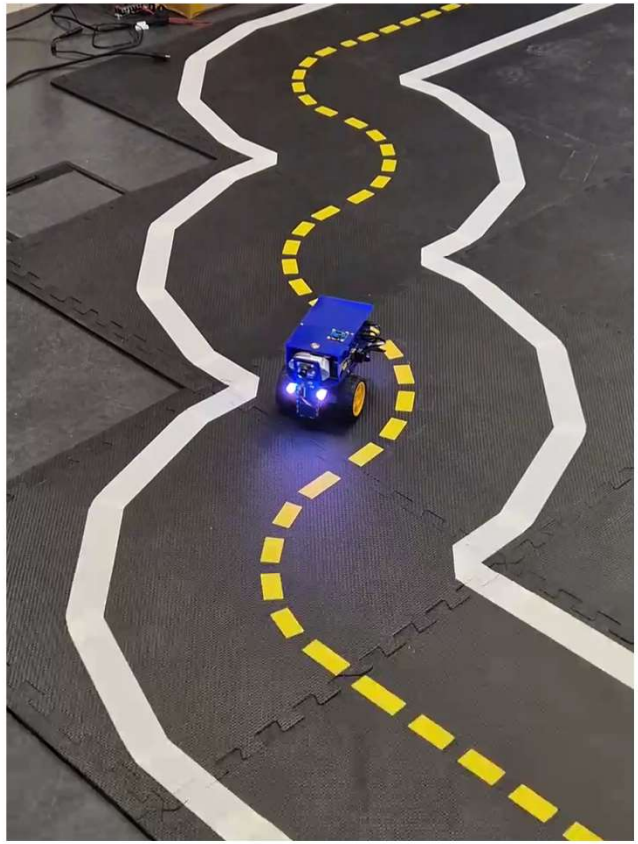
- The PID controller's output is converted into **motor** commands

Lane following

We modified the PID controller and color ranges

```
v_bar: 0.10       # The robot's normal speed.
k_d: -7.0         # Adjusts for how far off-center the robot is.
k_Id: -0.3        # Helps correct long-term drift off-center.
k_Dd: -0.3        # Helps smooth out sudden changes in position.
k_theta: -1.7     # Adjusts for the robot's angle.
k_Iphi: -0.0      # Helps correct long-term drift in angle.
k_Dphi: -0.0      # Helps smooth out sudden changes in angle.

theta_thres: 0.523  # The angle limit before the robot corrects itself.
d_thres: 0.2615     # The distance limit before the robot corrects itself.
```

```
colors:
  RED:
    low_1: [0,140,100]
    high_1: [20,255,255]
    low_2: [165,140,100]
    high_2: [180,255,255]
  WHITE:
    low: [0,0,150]
    high: [180,100,255]
  YELLOW:
    low: [23,120,80]
    high: [45,255,255]
```



Milestones

Over the course of this project
we have successfully accomplished the following:

Assembly of the Duckiebot

Setup and calibration

Lane following

Intersection navigation

Object detection and avoidance

Intersection navigation

1. Stopping at stop lines

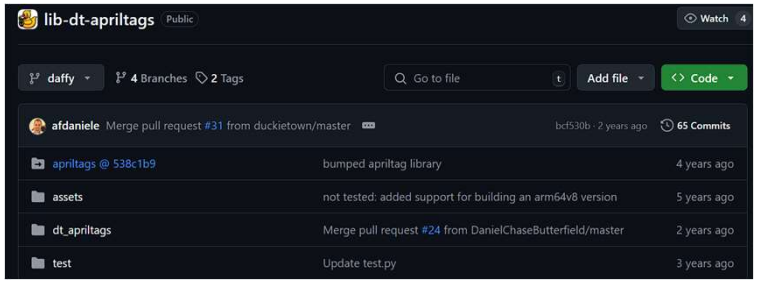
```yaml
colors:
  RED:
    low_1: [0,140,100]
    high_1: [20,255,255]
    low_2: [165,140,100]
    high_2: [180,255,255]
```

```yaml
stop_distance: 0.22  # Distance to stop line in meters
min_segs: 2          # Minimum number of segments to detect a stop line
off_time: 3          # Time in seconds to wait before rechecking stop line
max_y: 0.3           # Maximum y-coordinate value for stop line detection
```

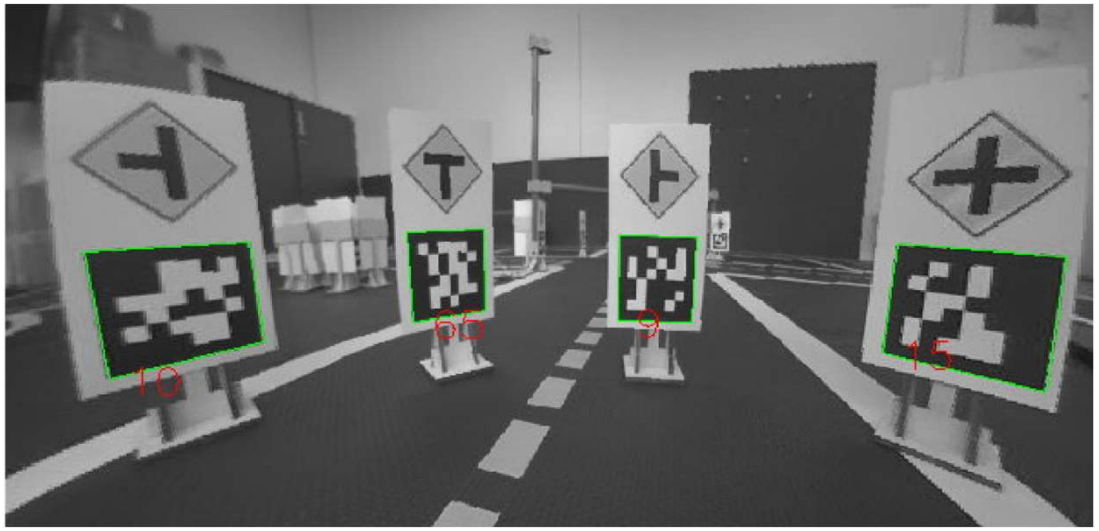
```python
# Wait at the stop line
sleep_sec = 3
rospy.loginfo(f"Sleeping for {sleep_sec} seconds")
rospy.sleep(sleep_sec)  # wait at stop line
```

Intersection navigation

2. Detecting signs




```
self.possible_turns = [0, 1, 2]  # left, straight, right
self.tags_to_turns = {
        67: [0, 2],
        65: [0, 2],
        11: [0, 2],
        63: [0, 1],
        10: [0, 1],
        61: [0, 1],
        59: [1, 2],
        9: [1, 2],
        57: [1, 2],
        13: [0, 1, 2],
        14: [0, 1, 2],
        15: [0, 1, 2],
        19: [0, 1, 2],
}
```



```
def _detect(self, detector_id, msg):
    # turn image message into grayscale image
    with self.profiler("/cb/image/decode"):
        img = self._jpeg.decode(msg.data, pixel_format=TJPF_GRAY)
    # run input image through the rectification map
    with self.profiler("/cb/image/rectify"):
        img = cv2.remap(img, self._mapx, self._mapy, cv2.INTER_NEAREST)
    # detect tags
    with self.profiler("/cb/image/detection"):
        tags = self._detectors[detector_id].detect(img, True, self._camera_parameters, self.tag_size)
```

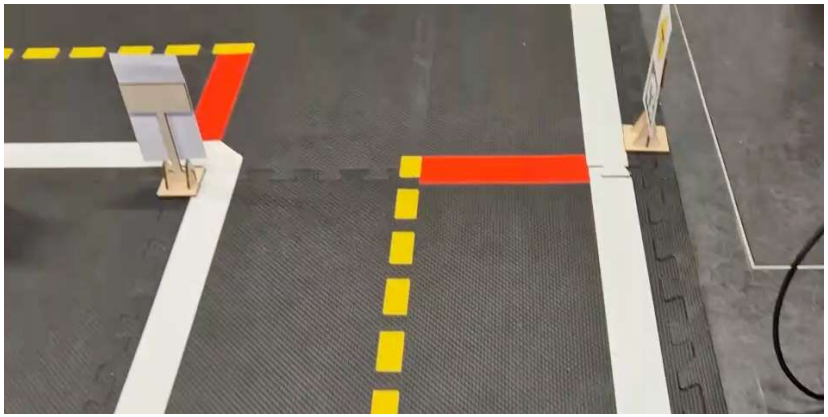
Intersection navigation

3. Activating the turn signal

```
colors: # All triplets should be ordered [R,G,B]
  switchedoff: [0,0,0]
  white: [0.5,0.5,0.5]
  yellow: [1,0.8,0]
  red: [1,0,0]
frequencies:
  f0: &f0 0.5
  f1: &f1 1.9

CAR_SIGNAL_RIGHT:
  color_mask: []
  color_list: ["white","yellow","white","red","yellow"]
  frequency_mask: [0,1,0,0,1]
  frequency: *f1

CAR_SIGNAL_LEFT:
  color_mask: []
  color_list: ["yellow","red","white","yellow","white"]
  frequency_mask: [1,0,0,1,0]
  frequency: *f1

CAR_SIGNAL_STRAIGHT:
  color_mask: []
  color_list: ["white","red","white","red","white"]
  frequency_mask: [0,0,0,0,0]
  frequency: *f1
```

```
self._led_signals = [
    String("CAR_SIGNAL_LEFT"),
    String("CAR_SIGNAL_STRAIGHT"),
    String("CAR_SIGNAL_RIGHT"),
]
```

```
# Set the LED signal lights
if self.params["~use_LEDs"].value:
    leds = self._led_signals[turn]
    rospy.loginfo(f"Setting leds: {leds}")
    msg = ChangePatternRequest(leds)
```



Intersection navigation

4. Executing the turn

```
self.l_turn_v = DTParam(
    "~l_turn_v", default = 0.1,
)
self.l_turn_omega = DTParam(
    "~l_turn_omega", default = 1.3,
)
self.l_turn_secs = DTParam(
    "~l_turn_secs", default = 4,
)

self.r_turn_v = DTParam(
    "~r_turn_v", default = 0.1,
)
self.r_turn_omega = DTParam(
    "~r_turn_omega", default = -1.75,
)
self.r_turn_secs = DTParam(
    "~r_turn_secs", default = 3,
)

self.s_turn_v = DTParam(
    "~s_turn_v", default = 0.1,
)
self.s_turn_omega = DTParam(
    "~s_turn_omega", default = 0.0,
)
self.s_turn_secs = DTParam(
    "~s_turn_secs", default = 3.0,
)
```

```
self.turn_params = [
    (self.l_turn_v, self.l_turn_omega, self.l_turn_secs),
    (self.s_turn_v, self.s_turn_omega, self.s_turn_secs),
    (self.r_turn_v, self.r_turn_omega, self.r_turn_secs),
]
```

```
def _available_turns(self):
    avail_turns = self.tags_to_turns.get(self.current_tag_id, [1])
    return avail_turns
```

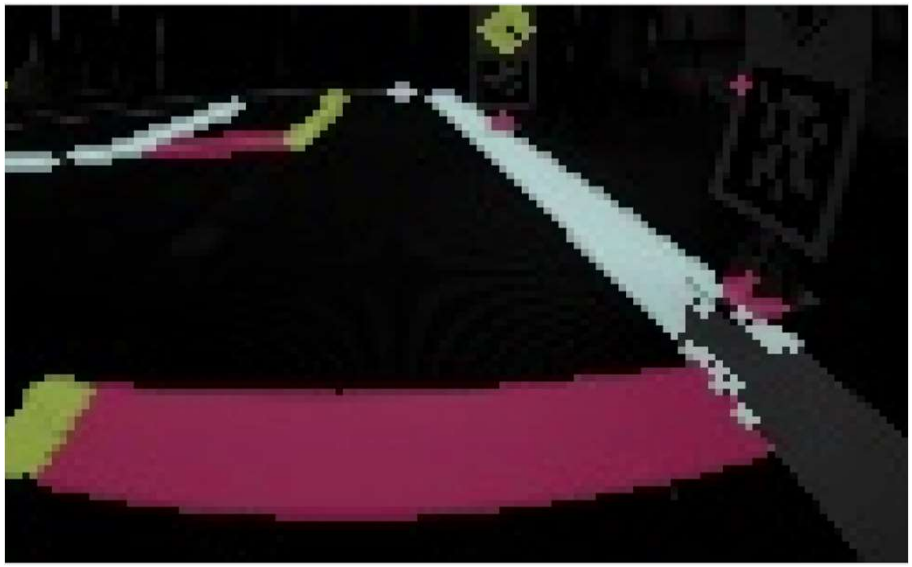
```
def selectTurn(self):
        # Take a random turn from the available turns (based on apriltag
        # detection)
        turn = self._available_turns()
        return int(np.random.choice(turn))
```

```
# Construct turning command
v, omega, sleep_time = self.turn_params[turn]
car_control_msg = Twist2DStamped()
car_control_msg.header.stamp = rospy.Time.now()
car_control_msg.v = v.value
car_control_msg.omega = omega.value

# Turn
rospy.loginfo(f"v: {v}, omega: {omega}, time: {sleep_time.value}")
self.publishCmd(car_control_msg)
rospy.loginfo("Turning now")
rospy.sleep(sleep_time.value)
```

Intersection navigation

Duckiebot's perspective

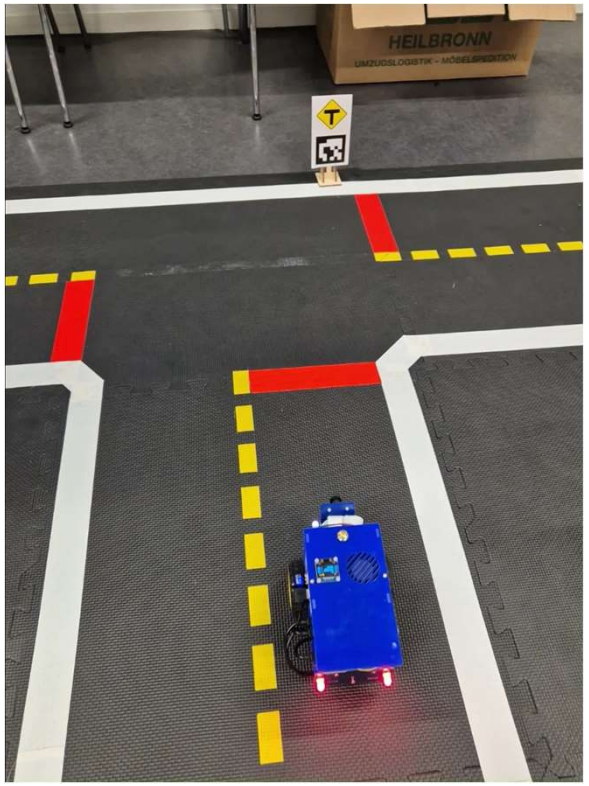
Line Detection

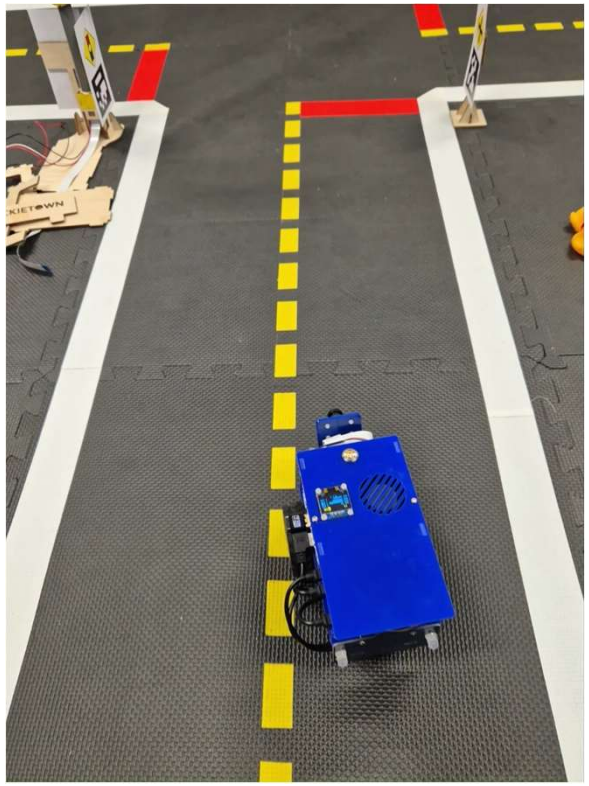


Sign Detection

Intersection navigation

Demo Videos





Milestones

Over the course of this project
we have successfully accomplished the following:

Assembly of the Duckiebot

Setup and calibration

Lane following

Intersection navigation

Object detection and avoidance

Object detection and avoidance

Duckietown's implementation

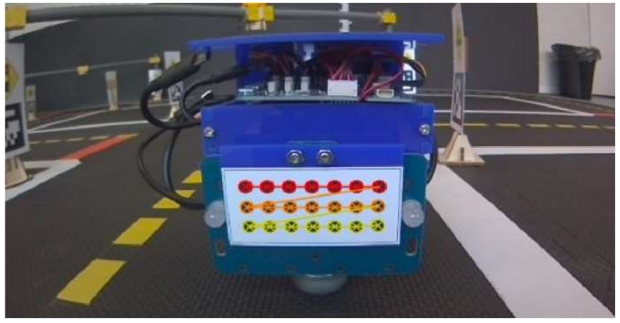
1. Detect if there is another Duckiebot by recognizing the pattern of circles

```
process_frequency: 2                    # Frequency at which to process the incoming images
circlepattern_dims: [7, 3]              # Number of dots in the pattern
blobdetector_min_area: 10               # Minimum area for the dot pattern
blobdetector_min_dist_between_blobs: 2  # Minimum distance between dots
```

2. Publish a stop line message if the other vehicle is too close

```
distance_between_centers: 0.0125            # Distance between the centers of the dots, in meters
max_reproj_pixelerror_pose_estimation: 1.5  # Maximum tolerable reprojection error for pose estimation
virtual_stop_line_offset: 0.5               # Distance before the stop, in meters
```

```
if self.at_obstacle_stop_line:
    rospy.loginfo("at obstacle stop line")
v = 0
omega = 0
car_control_msg = Twist2DStamped()
car_control_msg.header = pose_msg.header
self.publishCmd(car_control_msg)
```

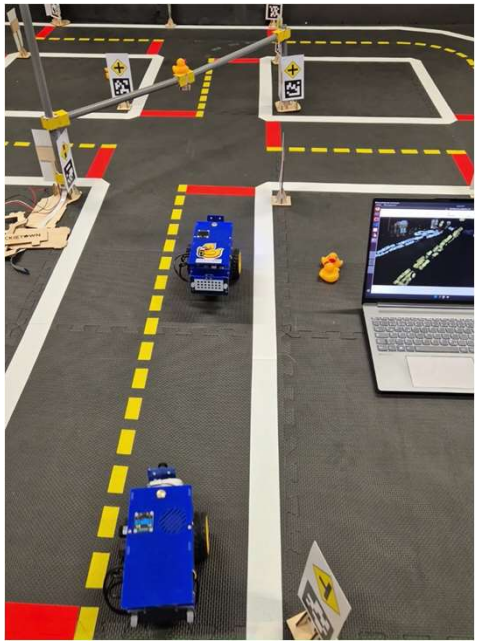


Object detection and avoidance

Duckietown’s implementation

Duckiebot's perspective

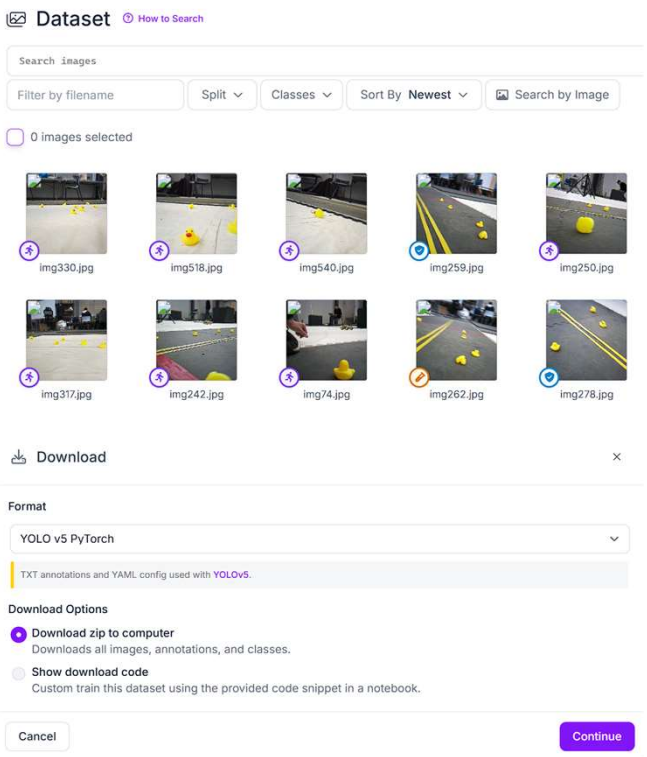
Vehicle detection

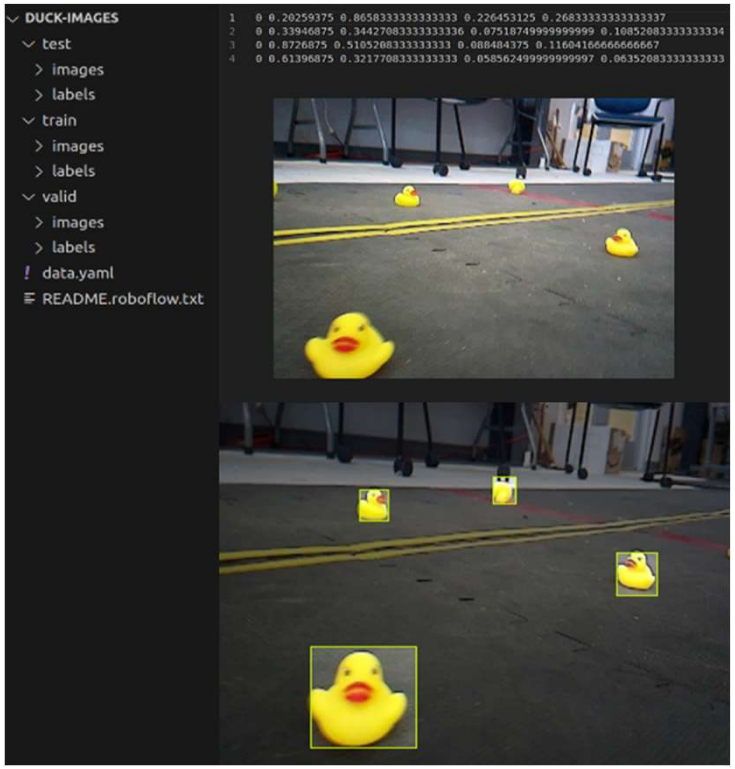


Object detection and avoidance

Our YOLOv5-Based Approach

1. Data Collection





Object detection and avoidance

Our YOLOv5-Based Approach

2. Training

dt_object_detection_training.ipynb

File Edit View Insert Runtime Tools Help

+ Code + Text Copy to Drive

And we're ready to train! This step will take about 5 minutes.

Notice that we're only training for 10 epochs. That's probably not enough!

```
[ ] !cd yolov5 && python3 train.py --cfg ./models/yolov5n.yaml --img 416 --batch 32 --epochs 10 --data duckietown.yaml
```

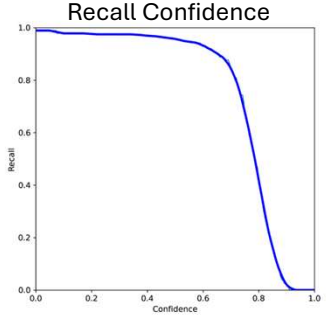
```
[ ] import numpy as np

    all_exps = os.listdir("yolov5/runs/train")
    all_exps_filtered = map(lambda x: int(x.replace("exp", "1")), filter(lambda x: x.startswith("exp"), all_exps))
    all_exps_filtered = np.array(list(all_exps))
    latest_exp_index = np.argmax(all_exps)
    latest_exp = all_exps[latest_exp_index]
    print(f"Latest exp is {latest_exp}")

    prun(f"cp yolov5/runs/train/{latest_exp}/weights/best.pt yolov5/best.pt")
    print(f"Marked the model from the latest run ({latest_exp}) as yolov5/best.pt.")
```

Next, we can upload your model to Duckietown's cloud!

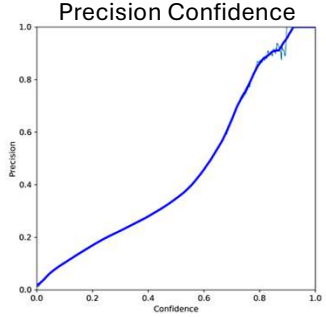
We will need our token to access our personal cloud space.

```
[ ] # TODO: Fill in the duckietown token here
    YOUR_DT_TOKEN = "YOUR_TOKEN_HERE"
```

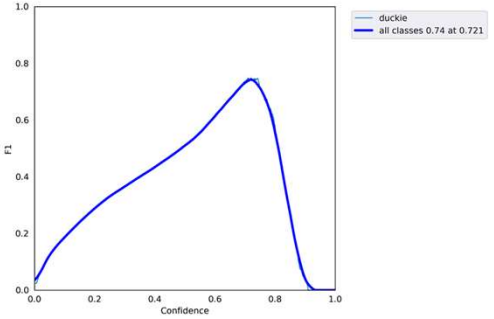
Recall Confidence



Precision Confidence



F1 Confidence



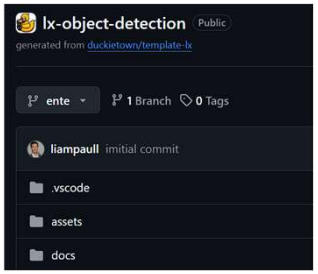
Object detection and avoidance

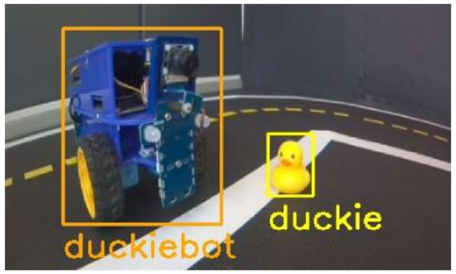
Our YOLOv5-Based Approach

3. Implementation

```python
def DT_TOKEN() -> str:
    # TODO: change this to your duckietown token
    dt_token = "dt1-3nT7FDbT7NLPrXykNJmqrVWv9QTz2jy2yYj
    return dt_token
```

```python
bboxes, classes, scores = self.model_wrapper.predict(rgb)

# Separate stop logic for ducks and duckiebots
stop_signal = False
large_duck = False
large_duckiebot = False

# Define the left and right boundaries of the center region
left_boundary = int(IMAGE_SIZE * 0.33)
right_boundary = int(IMAGE_SIZE * 0.75)

for cls, bbox, score in zip(classes, bboxes, scores):
    # Calculate center of bounding box
    center_x = (bbox[0] + bbox[2]) / 2

    if cls == 1 and score > 0.7:  # Duckiebot
        area = (bbox[2] - bbox[0]) * (bbox[3] - bbox[1])
        if area > 10000 and left_boundary < center_x < right_boundary:
            stop_signal = True
            large_duckiebot = True
            self.log(f"Duckiebot detected. Area: {area}")

    if cls == 0 and score > 0.7:  # Duck
        area = (bbox[2] - bbox[0]) * (bbox[3] - bbox[1])
        if area > 2000 and left_boundary < center_x < right_boundary:
            stop_signal = True
            large_duck = True
            self.log(f"Duck detected. Area: {area}")
```





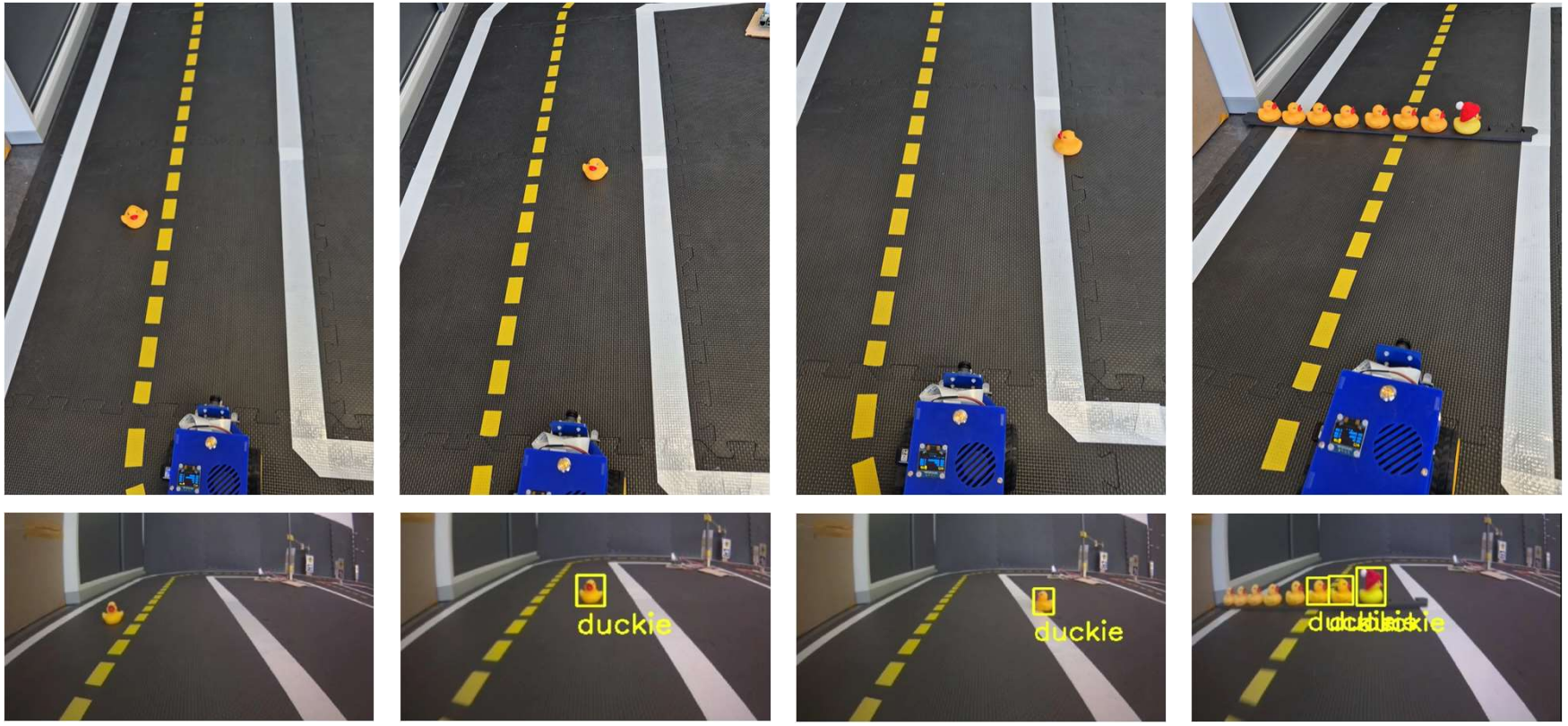

```python
# Create velocity command
vel_cmd = Twist2DStamped()
vel_cmd.header.stamp = rospy.Time.now()

if stop_signal:
    vel_cmd.v = 0.0
    vel_cmd.omega = 0.0
    if large_duck and not large_duckiebot:
        self.log("Stopping for Duck.")
    elif large_duckiebot and not large_duck:
        self.log("Stopping for Duckiebot.")
    else:
        self.log("Stopping for duck and duckiebot.")
else:
    vel_cmd.v = 0.2
    vel_cmd.omega = 0.0
    self.log("Driving...")

self.pub_vel.publish(vel_cmd)
```

Object detection and avoidance

Duckiebot Detection Demo

Object detection and avoidance

Duck Detection Demo



Milestones

Over the course of this project
we have successfully accomplished the following:

Assembly of the Duckiebot

Setup and calibration

Lane following

Intersection navigation

Object detection and avoidance

Challenges

1. Camera not turning on
2. LEDs showing unexpected colors

3. Storage full

```
  copyfile(src, dst, follow_symlinks=follow_symlinks)
 File "/usr/lib/python3.8/shutil.py", line 264, in copyfile
   with open(src, 'rb') as fsrc, open(dst, 'wb') as fdst:
SError: [Errno 28] No space left on device: '/code/catkin_ws/logs/lane_filter/build.cmak
.000.log'
build] Note: Workspace packages have changed, please re-source setup files to use them.
025-01-22 19:55:22 esra-VirtualBox dts[655876] ERROR The command '/bin/bash -c . /opt/ro
/${ROS_DISTRO}/setup.sh &&    catkin build      --workspace ${CATKIN_WS_DIR}/' returned a
on-zero code: 1
025-01-22 19:55:22 esra-VirtualBox dts[655876] ERROR An error occurred while building th
 project image.
```


Thank you
for the attention!
Questions?