Trailer Backer Upper

Boston Dynamics v2: Brayden Hambright, Christopher Nair, Joshua Bergthold, Manny Camacho, Farhad Taghizadeh

Problem Formulation

Backing up a trailer is hard. (Still)

Problem Formulation

- Learning to back up a trailer is an important skill for many occupations
- Very applicable to everyday life
  - Fishing boat, transporting any large machine (lawn mowers, snowmobiles, etc)
- Commercial Applications
  - Semi trucks, farm equipment

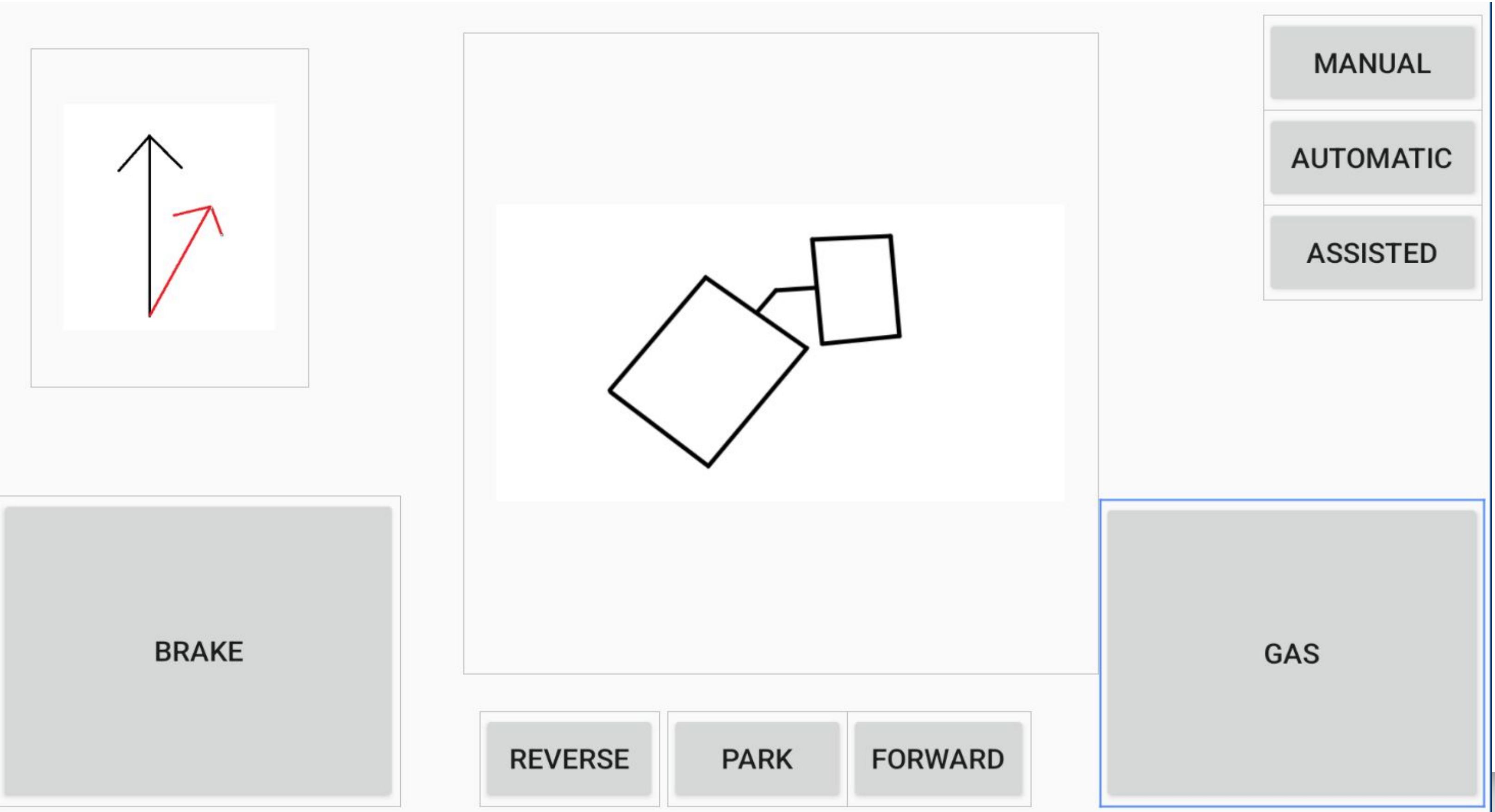
Approach

- Create an app that will:
  - Guide users to back up the vehicle optimally
  - Display vital information about the position of the trailer and vehicle
  - Take control of the vehicle and back it up for the user
- Create a system for the Model Trailer to:
  - Get information about its surroundings using a camera
  - Be able to back up on its own within lanes, or
  - Broadcast suggested steering angle from backing algorithm

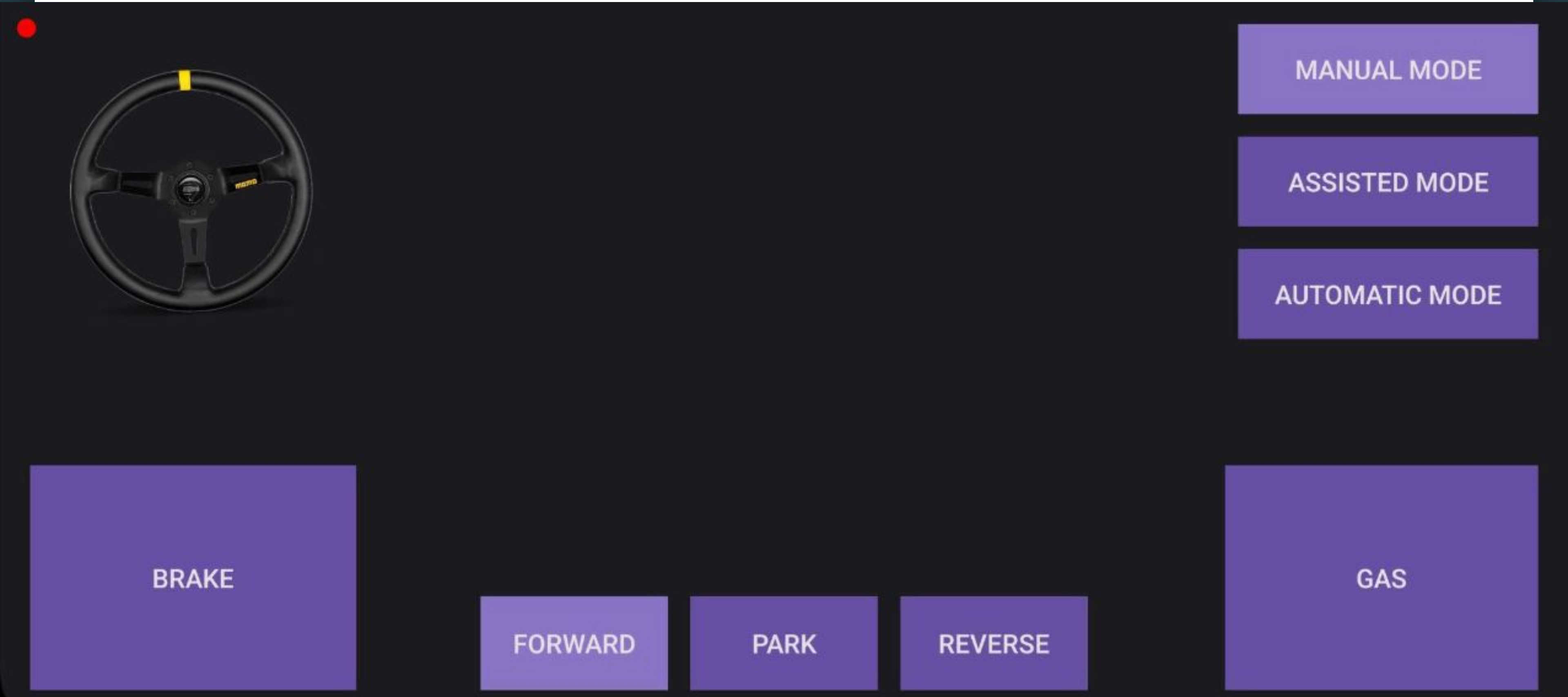
App Design

- Wanted to create an app that is easy to use, develop, and control the robot
  - Give the user feedback on inputs
  - Intuitive design
  - Give user several options on how to back up the trailer
  - Option to see more information about the system (a debug mode)
- Easy ways to transition state:
  - Gears: Forward, Park, Reverse
  - Control states: Manual, Assisted, Automatic

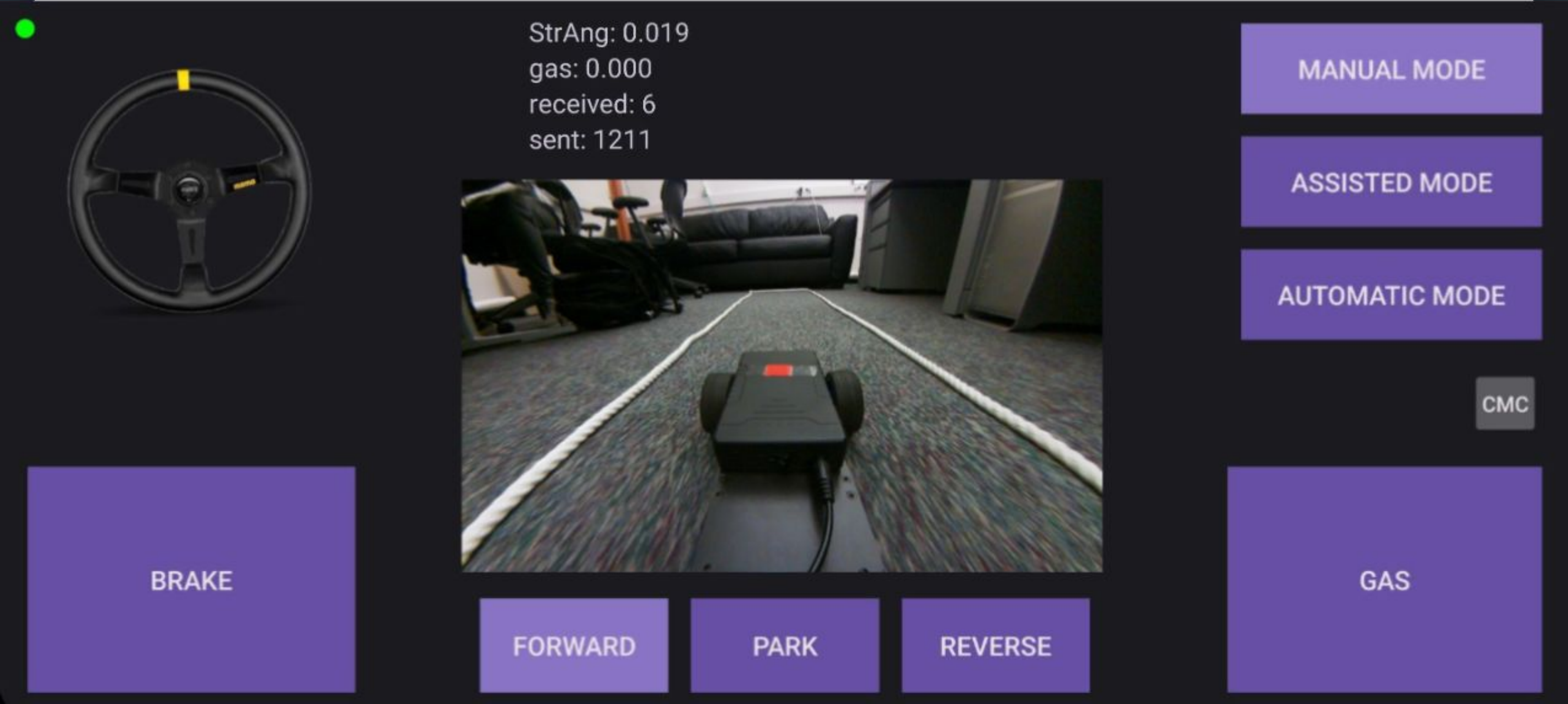
Initial design then:



Starting Screen

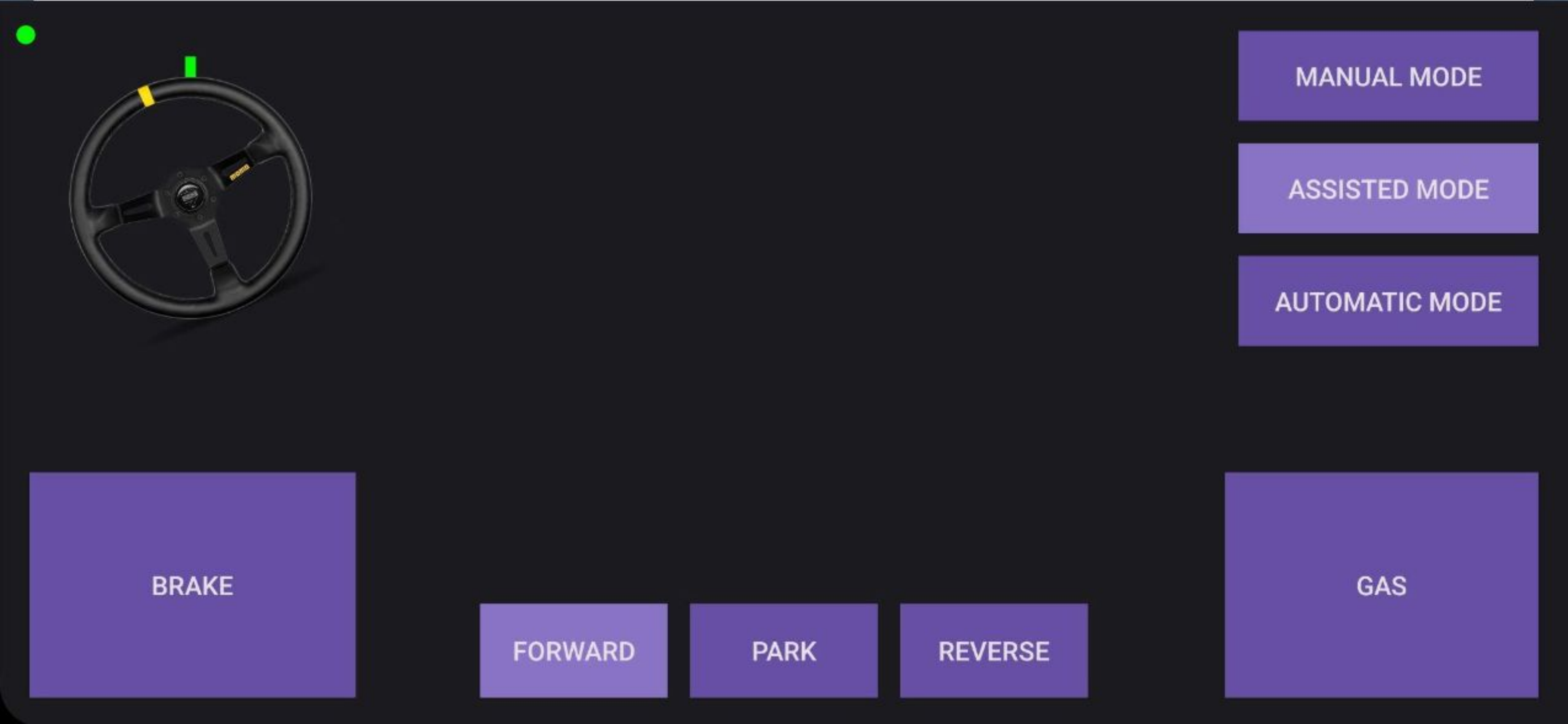


Debug Mode

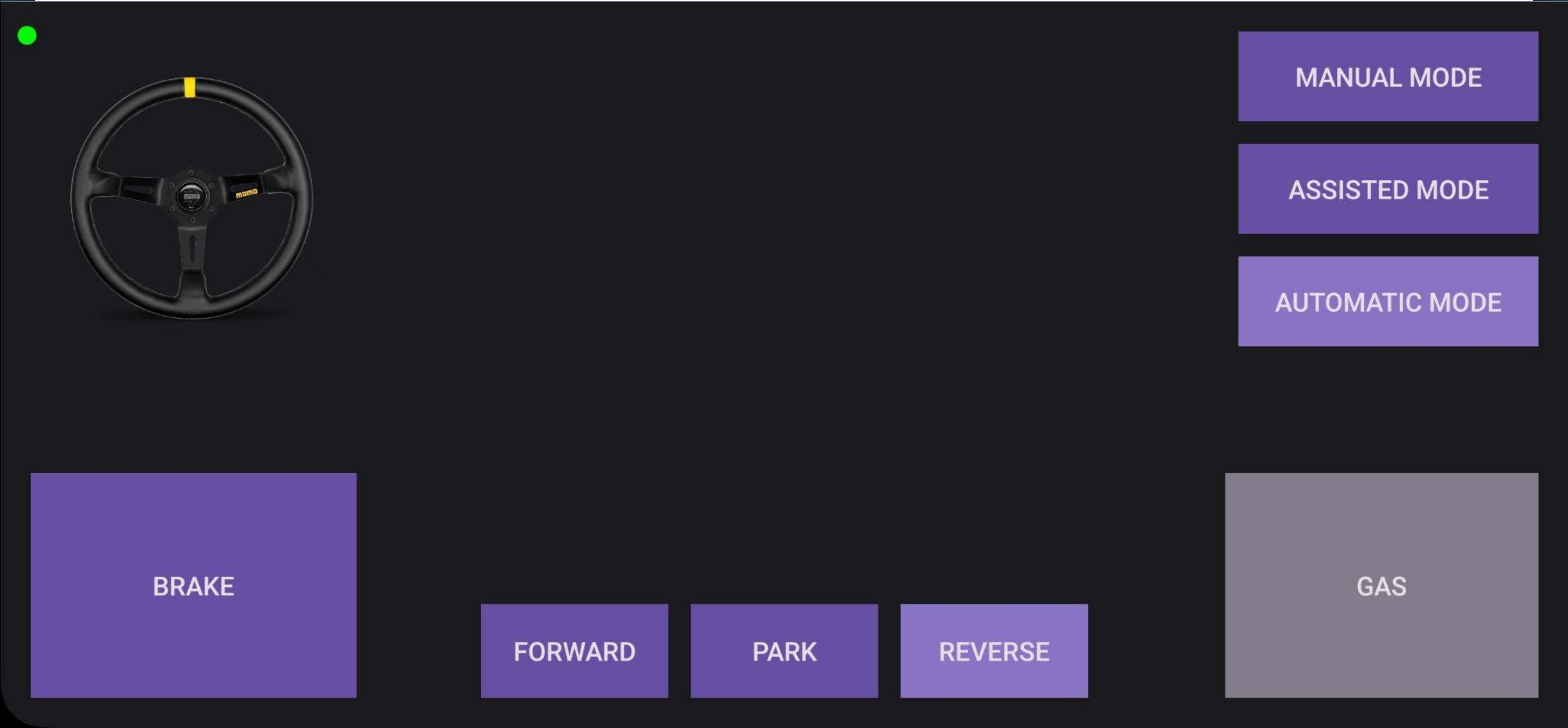


StrAng: 0.019
gas: 0.000
received: 6
sent: 1211

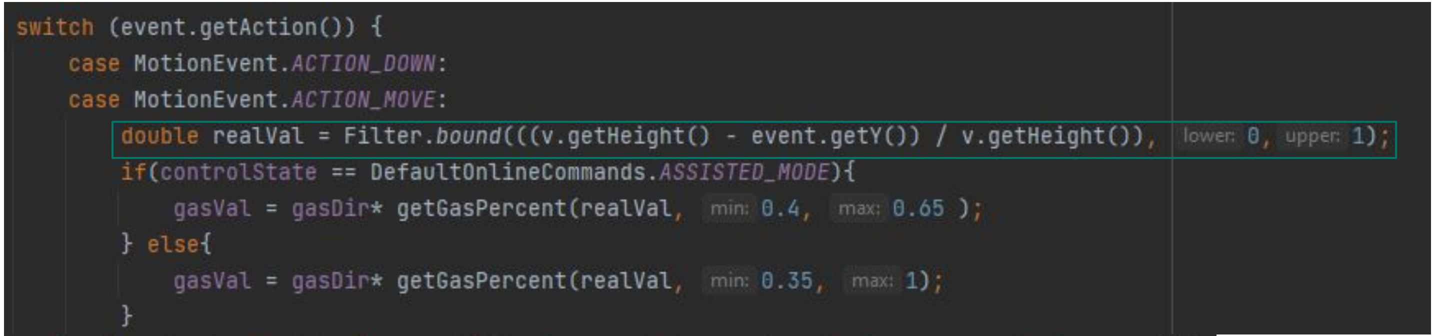
Assisted Mode



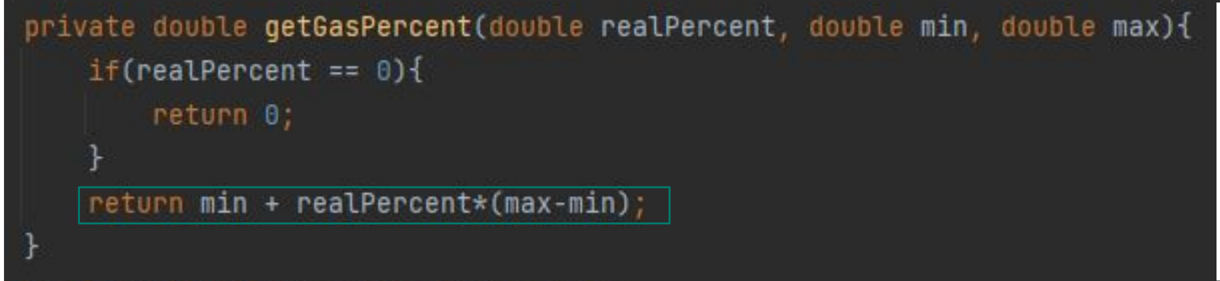
Automatic Mode



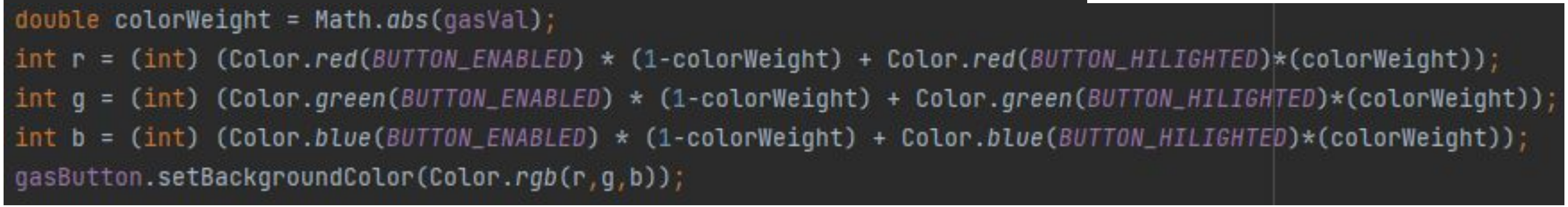
Important UI Features - Gas Feedback



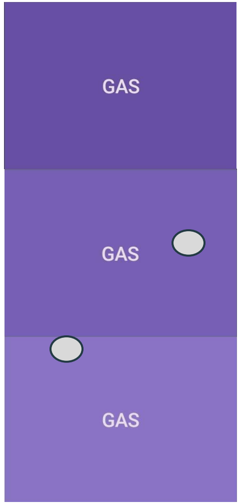
```
switch (event.getAction()) {
    case MotionEvent.ACTION_DOWN:
    case MotionEvent.ACTION_MOVE:
        double realVal = Filter.bound(((v.getHeight() - event.getY()) / v.getHeight()), lower: 0, upper: 1);
        if(controlState == DefaultOnlineCommands.ASSISTED_MODE){
            gasVal = gasDir* getGasPercent(realVal, min: 0.4, max: 0.65 );
        } else{
            gasVal = gasDir* getGasPercent(realVal, min: 0.35, max: 1);
        }
```



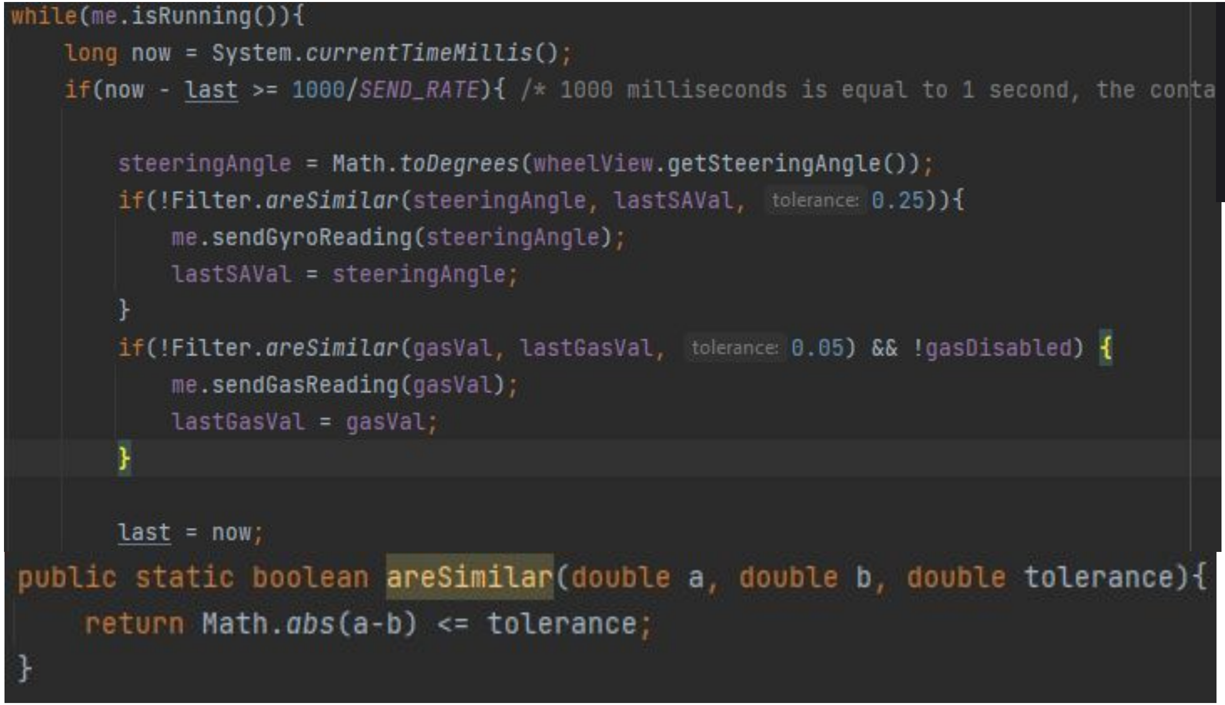
```
private double getGasPercent(double realPercent, double min, double max){
    if(realPercent == 0){
        return 0;
    }
    return min + realPercent*(max-min);
}
```



```
double colorWeight = Math.abs(gasVal);
int r = (int) (Color.red(BUTTON_ENABLED) * (1-colorWeight) + Color.red(BUTTON_HILIGHTED)*(colorWeight));
int g = (int) (Color.green(BUTTON_ENABLED) * (1-colorWeight) + Color.green(BUTTON_HILIGHTED)*(colorWeight));
int b = (int) (Color.blue(BUTTON_ENABLED) * (1-colorWeight) + Color.blue(BUTTON_HILIGHTED)*(colorWeight));
gasButton.setBackgroundColor(Color.rgb(r,g,b));
```

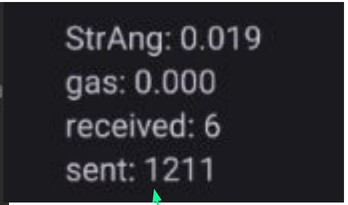


Optimization - Send Only Necessary Packets



```java
while(me.isRunning()){
    long now = System.currentTimeMillis();
    if(now - last >= 1000/SEND_RATE){ /* 1000 milliseconds is equal to 1 second, the conta

        steeringAngle = Math.toDegrees(wheelView.getSteeringAngle());
        if(!Filter.areSimilar(steeringAngle, lastSAVal, tolerance: 0.25)){
            me.sendGyroReading(steeringAngle);
            lastSAVal = steeringAngle;
        }
        if(!Filter.areSimilar(gasVal, lastGasVal, tolerance: 0.05) && !gasDisabled) {
            me.sendGasReading(gasVal);
            lastGasVal = gasVal;
        }

        last = now;
```

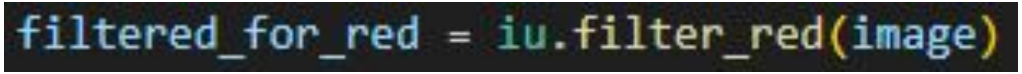
```java
public static boolean areSimilar(double a, double b, double tolerance){
    return Math.abs(a-b) <= tolerance;
}
```



```
StrAng: 0.019
gas: 0.000
received: 6
sent: 1211
```

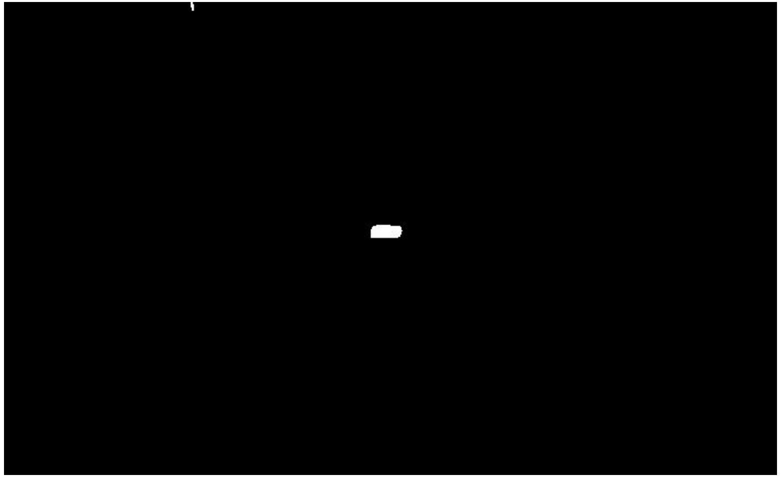
Want to keep the rate at which this number grows as small as possible

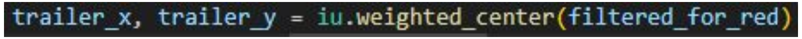
Image Processing Pipeline



```
filtered_for_red = iu.filter_red(image)
```

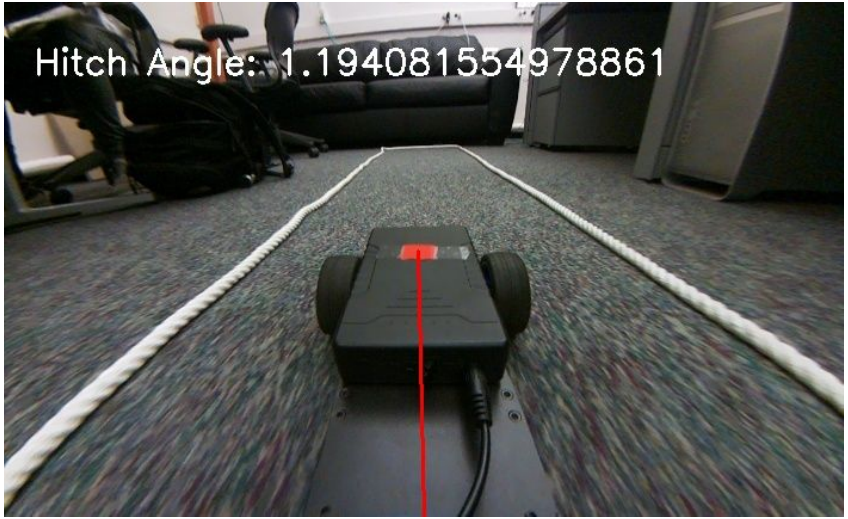
Image Processing Pipeline

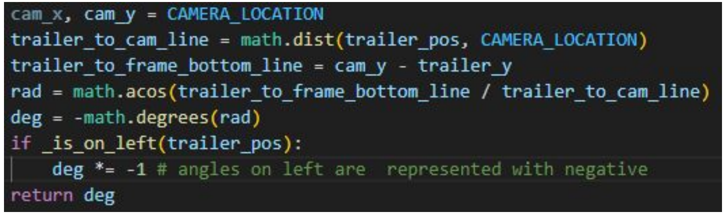




```
trailer_x, trailer_y = iu.weighted_center(filtered_for_red)
```

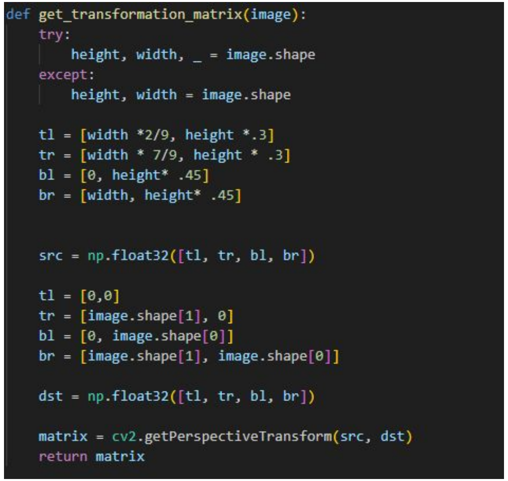
Image Processing Pipeline



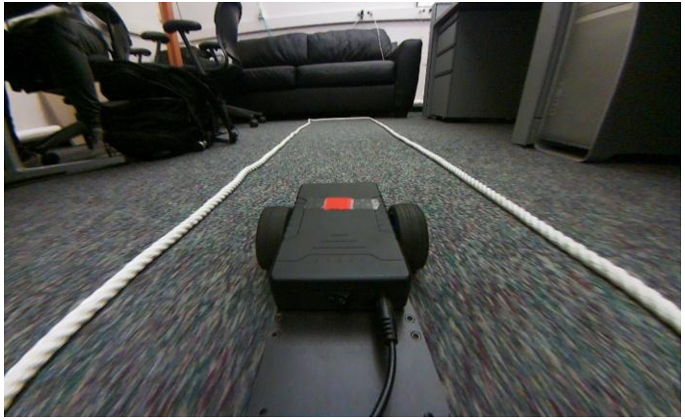


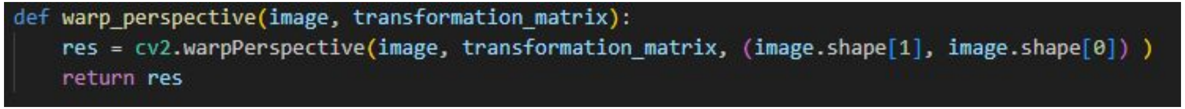
```
cam_x, cam_y = CAMERA_LOCATION
trailer_to_cam_line = math.dist(trailer_pos, CAMERA_LOCATION)
trailer_to_frame_bottom_line = cam_y - trailer_y
rad = math.acos(trailer_to_frame_bottom_line / trailer_to_cam_line)
deg = -math.degrees(rad)
if _is_on_left(trailer_pos):
    deg *= -1 # angles on left are  represented with negative
return deg
```

Image Processing Pipeline



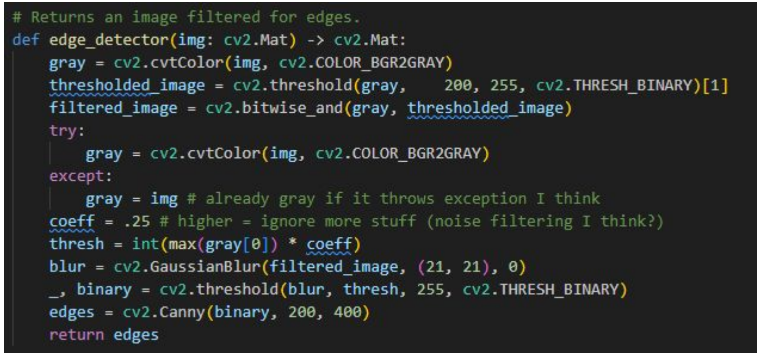
```python
def get_transformation_matrix(image):
    try:
        height, width, _ = image.shape
    except:
        height, width = image.shape

    tl = [width *2/9, height *.3]
    tr = [width * 7/9, height * .3]
    bl = [0, height* .45]
    br = [width, height* .45]


    src = np.float32([tl, tr, bl, br])

    tl = [0,0]
    tr = [image.shape[1], 0]
    bl = [0, image.shape[0]]
    br = [image.shape[1], image.shape[0]]

    dst = np.float32([tl, tr, bl, br])

    matrix = cv2.getPerspectiveTransform(src, dst)
    return matrix
```





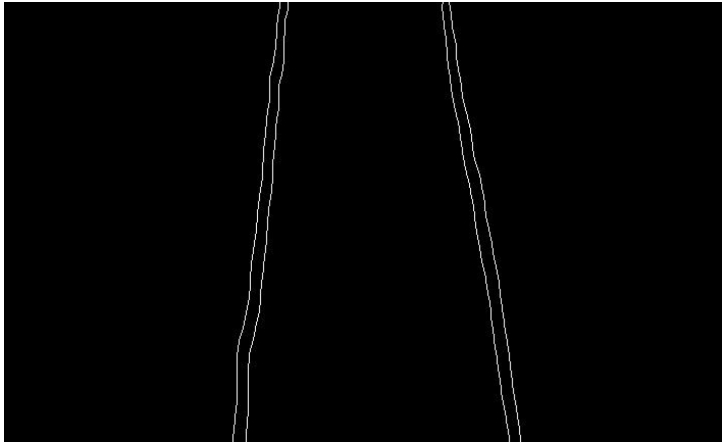
```python
def warp_perspective(image, transformation_matrix):
    res = cv2.warpPerspective(image, transformation_matrix, (image.shape[1], image.shape[0]) )
    return res
```

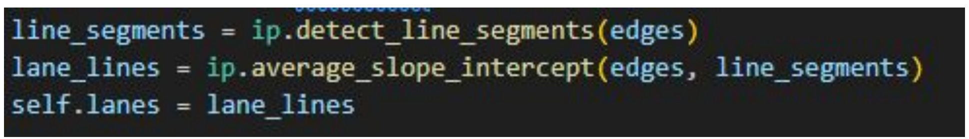
Image Processing Pipeline



```
# Returns an image filtered for edges.
def edge_detector(img: cv2.Mat) -> cv2.Mat:
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    thresholded_image = cv2.threshold(gray,    200, 255, cv2.THRESH_BINARY)[1]
    filtered_image = cv2.bitwise_and(gray, thresholded_image)
    try:
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    except:
        gray = img # already gray if it throws exception I think
    coeff = .25 # higher = ignore more stuff (noise filtering I think?)
    thresh = int(max(gray[0]) * coeff)
    blur = cv2.GaussianBlur(filtered_image, (21, 21), 0)
    _, binary = cv2.threshold(blur, thresh, 255, cv2.THRESH_BINARY)
    edges = cv2.Canny(binary, 200, 400)
    return edges
```

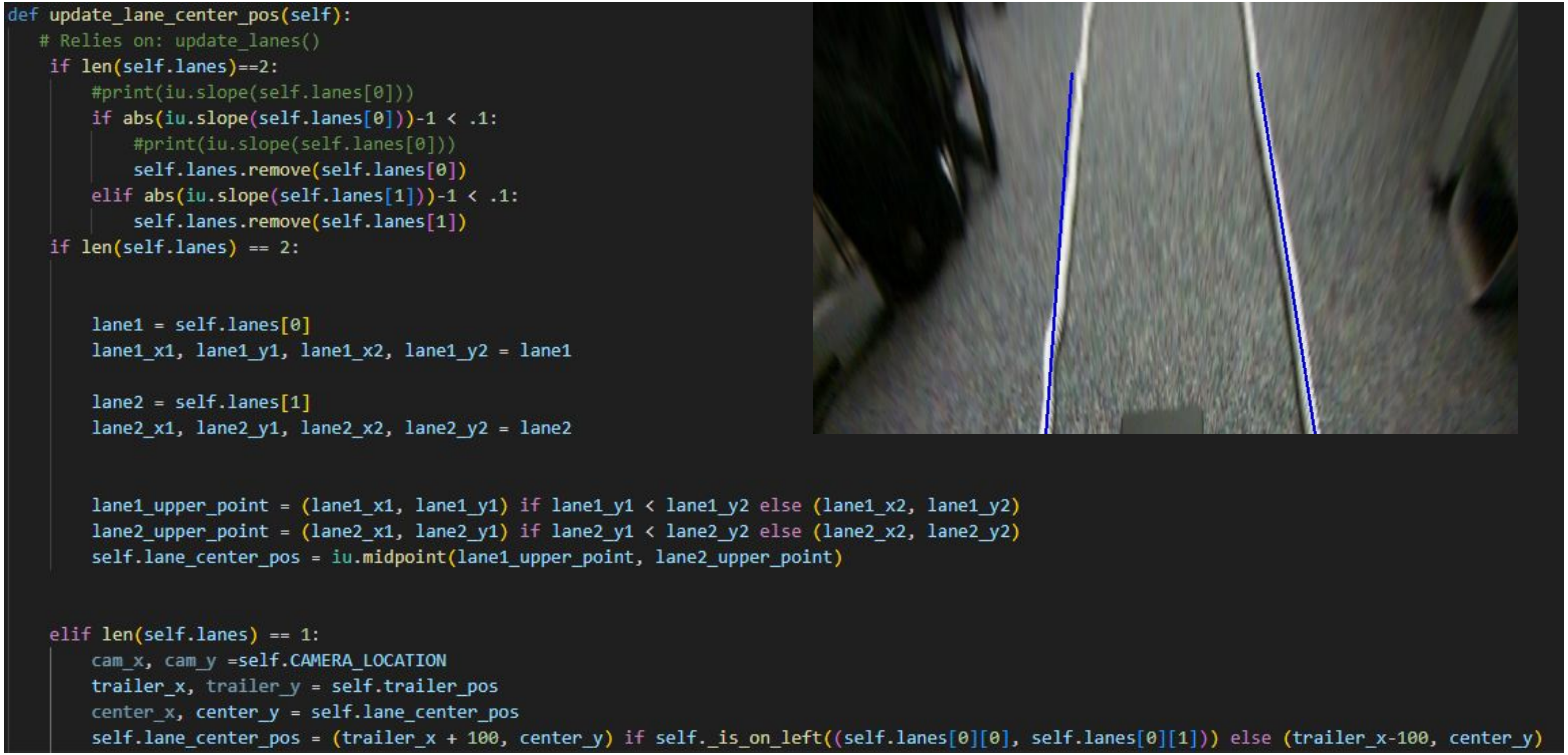
Image Processing Pipeline





```
line_segments = ip.detect_line_segments(edges)
lane_lines = ip.average_slope_intercept(edges, line_segments)
self.lanes = lane_lines
```

Image Processing Pipeline



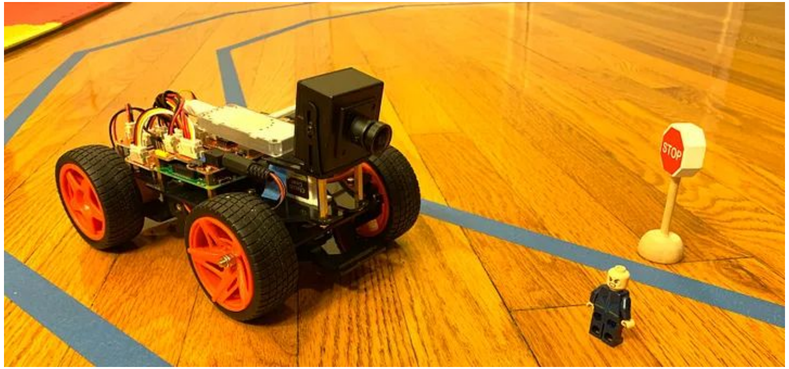
```
def update_lane_center_pos(self):
    # Relies on: update_lanes()
    if len(self.lanes)==2:
        #print(iu.slope(self.lanes[0]))
        if abs(iu.slope(self.lanes[0]))-1 < .1:
            #print(iu.slope(self.lanes[0]))
            self.lanes.remove(self.lanes[0])
        elif abs(iu.slope(self.lanes[1]))-1 < .1:
            self.lanes.remove(self.lanes[1])
    if len(self.lanes) == 2:


        lane1 = self.lanes[0]
        lane1_x1, lane1_y1, lane1_x2, lane1_y2 = lane1

        lane2 = self.lanes[1]
        lane2_x1, lane2_y1, lane2_x2, lane2_y2 = lane2


        lane1_upper_point = (lane1_x1, lane1_y1) if lane1_y1 < lane1_y2 else (lane1_x2, lane1_y2)
        lane2_upper_point = (lane2_x1, lane2_y1) if lane2_y1 < lane2_y2 else (lane2_x2, lane2_y2)
        self.lane_center_pos = iu.midpoint(lane1_upper_point, lane2_upper_point)


    elif len(self.lanes) == 1:
        cam_x, cam_y =self.CAMERA_LOCATION
        trailer_x, trailer_y = self.trailer_pos
        center_x, center_y = self.lane_center_pos
        self.lane_center_pos = (trailer_x + 100, center_y) if self._is_on_left((self.lanes[0][0], self.lanes[0][1])) else (trailer_x-100, center_y)
```

Experimentation

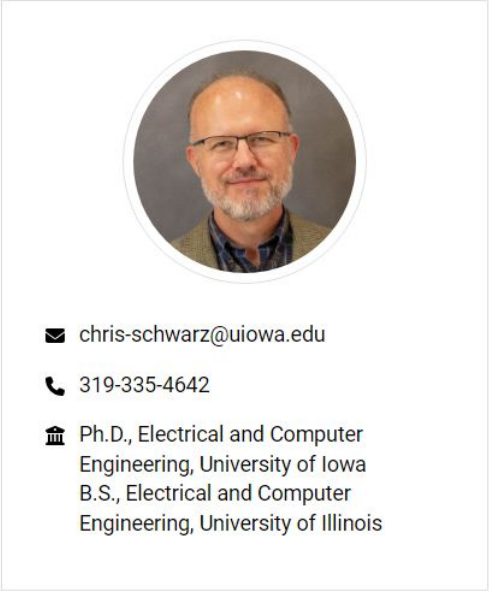
- Inspect image for lanes
- Minimize time from image taken to driving angle received
- Lots of trial and error
- Lighting conditions important
- Brief attempt at deep learning

Related Works

DeepPiCar



Initial Model Predictive Control algorithm provided by Chris Schwarz

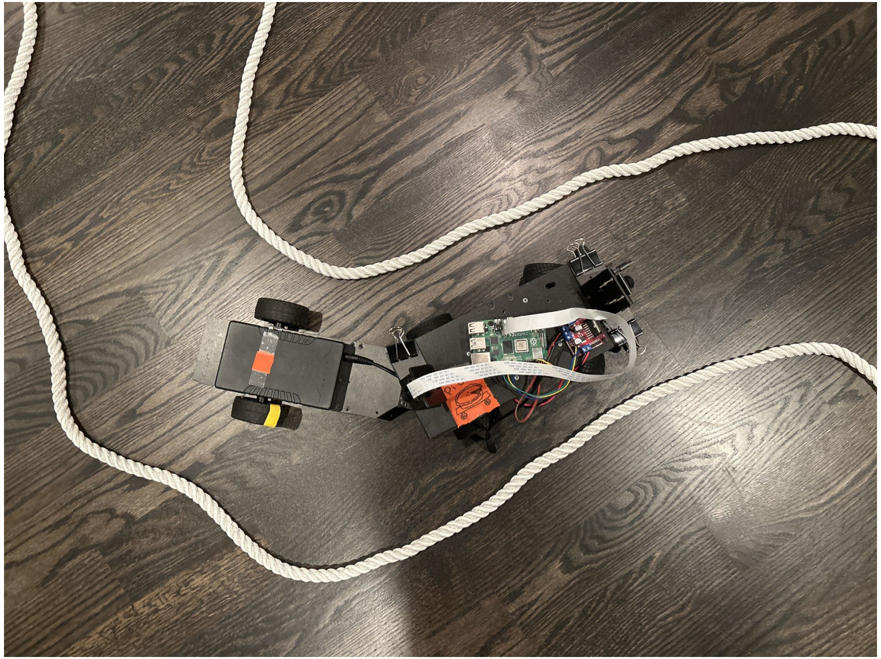


✉ chris-schwarz@uiowa.edu

☎ 319-335-4642

🏛 Ph.D., Electrical and Computer Engineering, University of Iowa
B.S., Electrical and Computer Engineering, University of Illinois

Demo!



Whoops!