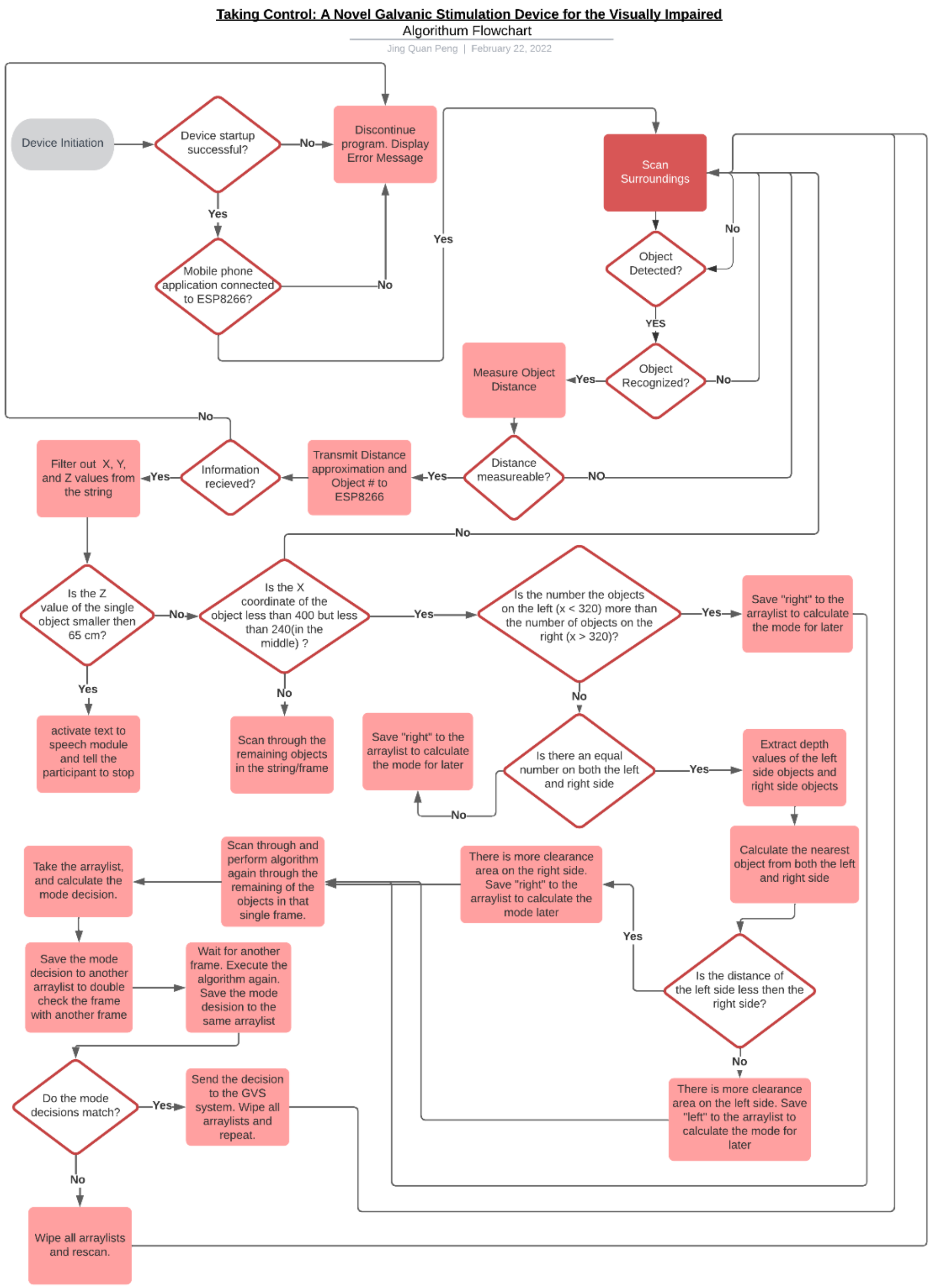
Taking Control: A Novel Galvanic Stimulation Device for the Visually Impaired

Justin Peng

Python Algorithm Breakdown/Explanation



```python
1   import pyttsx3
2   engine = pyttsx3.init()
3   import os, sys
4   import cv2
5   import pyrealsense2 as rs
6   import numpy as np
7   import time
8   from pathlib import Path
9   import torch
10  import torch.backends.cudnn as cudnn
11  import random
12
13  FILE = Path(__file__).absolute()
14  sys.path.append(FILE.parents[0].as_posix())
15
16  import socket
17  UDP_IP = "192.168.1.205"
18  UDP_PORT = 10000
19  cond = True
20  sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)  # UDP
21
22  from http.server import BaseHTTPRequestHandler, HTTPServer
23  from models.experimental import attempt_load
24  from utils.datasets import LoadStreams, LoadImages
25  from utils.augmentations import Albumentations, augment_hsv, copy_paste, letterbox, mixup, random_perspective
26  from utils.general import check_img_size, check_requirements, check_imshow, colorstr, non_max_suppression, \
27      apply_classifier, scale_coords, xyxy2xywh, strip_optimizer, set_logging, increment_path, save_one_box
28  from utils.plots import Annotator, colors
29  from utils.torch_utils import load_classifier, select_device, time_sync
30
```

Line 1-12: library imports (e.g., intel Realsense library, time library, cv2, etc…)

Line 16-20: UDP library setup (so that the program can communicate with the ESP8266 chip)

Line 22-29: function imports for YoloV5

```python
32  def run():
33      engine.setProperty("rate", 500)
34      lastoutput = ' '
35      global thread1, thread2
36      weights = 'yolov5s.pt'  # model.pt path(s)
37      imgsz = 640  # inference size (pixels)
38      conf_thres = 0.25  # confidence threshold
39      iou_thres = 0.45  # NMS IOU threshold
40      max_det = 10  # maximum detections per image
41      classes = None  # filter by class: --class 0, or --class 0 2 3
42      agnostic_nms = False  # class-agnostic NMS
43      augment = False  # augmented inference
44      visualize = False  # visualize features
45      line_thickness = 3  # bounding box thickness (pixels)
46      hide_labels = False  # hide labels
47      hide_conf = False  # hide confidences
48      half = False  # use FP16 half-precision inference
49      stride = 32
50      device_num = ''  # cuda device, i.e. 0 or 0,1,2,3 or cpu
51      view_img = False  # show results
52      save_crop = False  # save cropped prediction boxes
53      nosave = False  # do not save images/videos
54      update = False  # update all models
55      name = 'exp'  # save results to project/name
56
57      set_logging()
```

Line 32-55: some pre-made and coded configurations for YoloV5

```
57      set_logging()
58      device = select_device(device_num)
59      half &= device.type != 'cpu'
60
61      model = attempt_load(weights, map_location=device)
62      stride = int(model.stride.max())
63      imgsz = check_img_size(imgsz, s=stride)
64      names = model.module.names if hasattr(model, 'module') else model.names
65      if half:
66          model.half()
67
68      classify = False
69      if classify:
70          modelc = load_classifier(name='resnet50', n=2)
71          modelc.load_state_dict(torch.load('resnet50.pt', map_location=device)['model']).to(device).eval()
72
73      view_img = check_imshow()
74      cudnn.benchmark = True
75
76      if device.type != 'cpu':
77          model(torch.zeros(1, 3, imgsz, imgsz).to(device).type_as(next(model.parameters())))
```

Line 57-58: Initializes the device, check if half precision is supported. In my case, I have a Radeon GPU, so it does not have CUDA cores.

Line 61-66: Loads model, if half is supported, it will load half of it.

Line 68-71: Initialize second-stage classifier

Line 73-74: Data loader

Line 76-77: In my case, I will be using the CPU. These two lines do not apply to my situation

```
79      config = rs.config()
80      config.enable_stream(rs.stream.color, 640, 480, rs.format.bgr8, 30)
81      config.enable_stream(rs.stream.depth, 640, 480, rs.format.z16, 30)
82
83      pipeline = rs.pipeline()
84      profile = pipeline.start(config)
85
86      align_to = rs.stream.color
87      align = rs.align(align_to)
88      while (True):
```

Line 79-81: configure image size for the D435 sensor

Line 83-84: initializes the pipeline class

Line 86-87: align depth

```
118     while (True):
119         frames = pipeline.wait_for_frames()
120         aligned_frames = align.process(frames)
121         color_frame = aligned_frames.get_color_frame()
122         depth_frame = aligned_frames.get_depth_frame()
123         if not depth_frame or not color_frame:
124             continue
125 
126         img = np.asanyarray(color_frame.get_data())
127         depth_image = np.asanyarray(depth_frame.get_data())
128 
129         s = np.stack([letterbox(x, imgsz, stride=stride)[0].shape for x in img], 0)
130         rect = np.unique(s, axis=0).shape[0] == 1
131         if not rect:
132             print(
133                 'WARNING: Different stream shapes detected. For optimal performance supply similarly-shaped streams.')
134 
135         # Letterbox
136         img0 = img.copy()
137         img = img[np.newaxis, :, :, :]
138 
139         # Stack
140         img = np.stack(img, 0)
```

Line 118: start loop for detection

Line 119-123: configure some settings for the D435 camera

Line 126-127: save data into a variable for future use

Line 129-133: check for common shapes

Line 136-137: letterbox

Line 139-140: stack

```
143         img = img[..., ::-1].transpose((0, 3, 1, 2))
144         img = np.ascontiguousarray(img)
145 
146         img = torch.from_numpy(img).to(device)
147         img = img.half() if half else img.float()
148         img /= 255.0
149         if img.ndimension() == 3:
150             img = img.unsqueeze(0)
151 
152         t1 = time_sync()
153         pred = model(img, augment=augment,
154                      visualize=increment_path(save_dir / 'features', mkdir=True) if visualize else False)[0]
155 
156         pred = non_max_suppression(pred, conf_thres, iou_thres, classes, agnostic_nms, max_det=max_det)
157         t2 = time_sync()
158 
159         if classify:
160             pred = apply_classifier(pred, modelc, img, img0)
161 
```

Line 143-144: changes BGR to RGB, and BHWC to BCHW

Line 146-147: uint8 to fp16/32

Line 148-150: 0-255 to 0-1

Line 152-154: Inference

Line 156-157: Apply NMS

Line 159-160: Apply Classifier

```
165         for i, det in enumerate(pred):
166             s = f'{i}: '
167
168             obj = []
169             annotator = Annotator(img0, line_width=line_thickness, example=str(names))
170             if len(det):
171                 det[:, :4] = scale_coords(img.shape[2:], det[:, :4], img0.shape).round()
172                 for *xyxy, conf, cls in reversed(det):
173                     c1, c2 = (int(xyxy[0]), int(xyxy[1])), (int(xyxy[2]), int(xyxy[3]))
174                     center1 = round((c1[0] + c2[0]) / 2)
175                     center2 = round((c1[1] + c2[1]) / 2)
176                     center_point = center1, center2
177                     circle = cv2.circle(img0, center_point, 5, (0, 255, 0), 2)
178                     test_coord = cv2.putText(img0, str(center_point), center_point, cv2.FONT_HERSHEY_PLAIN, 2,
179                                              (0, 0, 255))
180                     obj.append(center_point)
181
182                 numofObjects = 0
183                 counter = 0
184                 for c in det[:, -1]:
185                     n = (det[:, -1] == c).sum()
186                     dist = depth_frame.get_distance(obj[numofObjects][0], obj[numofObjects][1])
187                     dist = int(dist * 100)
188                     s += f"{obj[numofObjects]}  {dist}. "
189                     numofObjects = numofObjects + 1
190
```

Line 165-166: Process detections, detections per image

Line 168-169: create annotator and a “obj” array

Line 171: Rescales boxes from “img_size” to “im0” size

Line 173-178: Calculates center point of bounding box

Line 180: save “center_point” variable for future use

Line 182-189: save all the objects detected in the frame to one single string, may be easier to access in the future. Also created an object counter variable to calculate the number of objects that are in that single frame.

```
194
195                 c = int(cls)  # integer class
196                 label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
197                 annotator.box_label(xyxy, label, color=colors(c, True))
198
199         cv2.imshow("IMAGE", img0)
200         depth_colormap = cv2.applyColorMap(cv2.convertScaleAbs(depth_image, alpha=0.08), cv2.COLORMAP_JET)
201         cv2.imshow("DEPTH", depth_colormap)
202         global stop_threads
203         if cv2.waitKey(1) & 0xFF == ord('q'):
204             break
205
206
207 if __name__ == '__main__':
208     run()
```

Line 195: call integer class and transfer into int

Line 196-197: label the bounding box with the object names

Line 199-201: show image and depth map on screen for the user to see

Line 202-203: if the user presses the “q” key, the program will stop

Line 207-208: run the program

```
178 for y in range(numofObjects):
179     # find depth value by finding : and .
180     str1 = cutstring[y]
181     d1 = str1.find("d") + 1
182     d2 = str1.find(".")
183     ObjDis = str1[d1:d2]
184     ObjDis = int(ObjDis)
185     # slice off brackets
186     b1 = str1.find("(") + len("(")
187     b2 = str1.find(")")
188     strxy = str1[b1:b2]
189     # slice off comma and y value
190     y1 = strxy.find(",")
191     strx = strxy[:y1]
192     xvalue = int(strx)
```

Line 180-184: Filters out the depth value by finding the “:” and “.” to substring it. It then converts it to an integer so it can be used to compare with other values in the future

Line 186-192: Finds brackets “(“and “)” and then “,” to substring it to find the x value. It is then turned into a integer variable.

```
193     if (ObjDis < 65 and ObjDis != 0):
194         direction.append("stop")
195         break
196     else:
197         if (xvalue < 400 and xvalue > 240):
198             leftcounter = 0
199             rightcounter = 0
200             for z in range(numofObjects):
201                 # compare x values
202                 str1 = cutstring[z]
203                 b3 = str1.find("(") + len(")")
204                 b4 = str1.find(")")
205                 strxy1 = str1[b3:b4]
206                 # slice off comma and y value
207                 y2 = strxy1.find(",")
208                 strx1 = strxy1[:y2]
209                 xvalue = int(strx1)
210                 # find depth value of the substring
211                 d3 = str1.find("d") + 1
212                 d4 = str1.find(".")
213                 depthvalue = str1[d3:d4]
214                 depthvalue = int(depthvalue)
215                 if (xvalue < 320):
216                     leftcounter = leftcounter + 1
217                     depthleftcounter.append(int(depthvalue))
218                 if (xvalue > 320):
219                     rightcounter = rightcounter + 1
220                     depthrightcounter.append(int(depthvalue))
221
```

Line 193-195: If the object distance is less then 65 meters away, or does not equal to zero, it saves “stop” to the first array list called “direction”

Line 196-220: Similar to line 180-192, it filters everything else except the x value. As seen on the for loop on line 200, it goes through every single one of the objects in the frame and records them down. If the object it is calculating right now in this frame is on the left side (x < 320), it will add one to the left counter and one to an array list (reason is explained later). If the object it is calculating right now in this frame is on the right side (x > 320), it will add one to the right counter, and add the value to an array list.

```
222                 if (leftcounter > rightcounter):
223                     direction.append("right")
224                 if (rightcounter > leftcounter):
225                     direction.append("left")
226                 if (leftcounter == rightcounter):
227                     # generate random
228                     minleft = depthleftcounter[0]
229                     minright = depthrightcounter[0]
230                     for r in range(0, len(depthleftcounter)):
231                         if (depthleftcounter[r] < minleft):
232                             minleft = depthleftcounter[r]
233                     for p in range(0, len(depthrightcounter)):
234                         if (depthrightcounter[p] < minright):
235                             minright = depthrightcounter[p]
236                     if (minleft < minright):
237                         direction.append("right")
238                     if (minright < minleft):
239                         direction.append("left")
240             else:
241                 direction.append("forward")
242         h = h + 1
243
```

Line 222-223: If the left counter is greater then the right counter "right" will be added to an array list called "direction. This is because if there are more objects on the left side, it would be reasonable to go to the opposite direction where there are less objects. If the right counter is greater then the left counter, "left" will be added to the same array list.

Line 226-239: This is where the array list "depthleftcounter" and "depthrightcounter" comes in. If there is an equal number of objects on both the left and right side, it will perform the following. From depthleftcounter, it will find the nearest object on the left. From depthrightcounter, it will find the nearest object on the right. If the object on the left, is nearer then the object on the right, "right" will be added to yet another array list (to double check in the future) because there is more space on the right. If it is the opposite, then it is on the right.

Line 241: This "else" is attached to the previous "if" statement where it calculates whether there is any objects in the middle. If the object is outside the range, it will add "forward" to the array list.

```
for u in range(len(direction)):
    if (direction[num] == 'stop'):
        stop += 1
    else:
        if (direction[num] == 'left'):
            left += 1
        else:
            if (direction[num] == 'right'):
                right += 1
            else:
                forward += 1
    num += 1
```

Line 250-271: After scanning through the entire frame, it will calculate the mode direction it can go. This is done by a simple counter algorithm.

```
262                 if (stop > 0):
263                     doublecheck.append("stop")
264                 else:
265                     if (left > 0):
266                         doublecheck.append("left")
267                     else:
268                         if (right > 0):
269                             doublecheck.append("right")
270                         else:
271                             doublecheck.append("forward")
272                 c = int(cls)  # integer class
273                 label = None if hide_labels else (names[c] if hide_conf else f'{names[c]} {conf:.2f}')
274                 annotator.box_label(xyxy, label, color=colors(c, True))
275                 x += 1
276
```

Afterwards, it will perform a second check on another frame.

```
277         num1 = 0
278         stop1 = 0
279         left1 = 0
280         right1 = 0
281         forward1 = 0
282         for p in range(len(doublecheck)):
283             if (doublecheck[num1] == 'stop'):
284                 stop1 += 1
285             else:
286                 if (doublecheck[num1] == 'left'):
287                     left1 += 1
288                 else:
289                     if (doublecheck[num1] == 'right'):
290                         right1 += 1
291                     else:
292                         forward1 += 1
293             num1 += 1
294
```

Line 277-294: After storing two values from two separate frames, it checks if the results are identical. Generally, the time it takes to calculate per frame, is about 0.05seconds.

```
295         if (stop1 == 2):
296             print("stop")
297             engine.say("stop")
298         else:
299             if (left1 == 2):
300                 print("left")
301                 bstop = bytes("Y0:X-75", encoding='utf-8')
302                 sock.sendto(bstop, (UDP_IP, UDP_PORT))
303             else:
304                 if (right1 == 2):
305                     print("right")
306                     bstop = bytes("Y0:X75", encoding='utf-8')
307                     sock.sendto(bstop, (UDP_IP, UDP_PORT))
308                 else:
309                     if (forward1 == 2):
310                         print("clear")
311
```

Line 295-310: If the two results match (counter is equal to two), it will finally send the result through the packets module to the ESP8266.