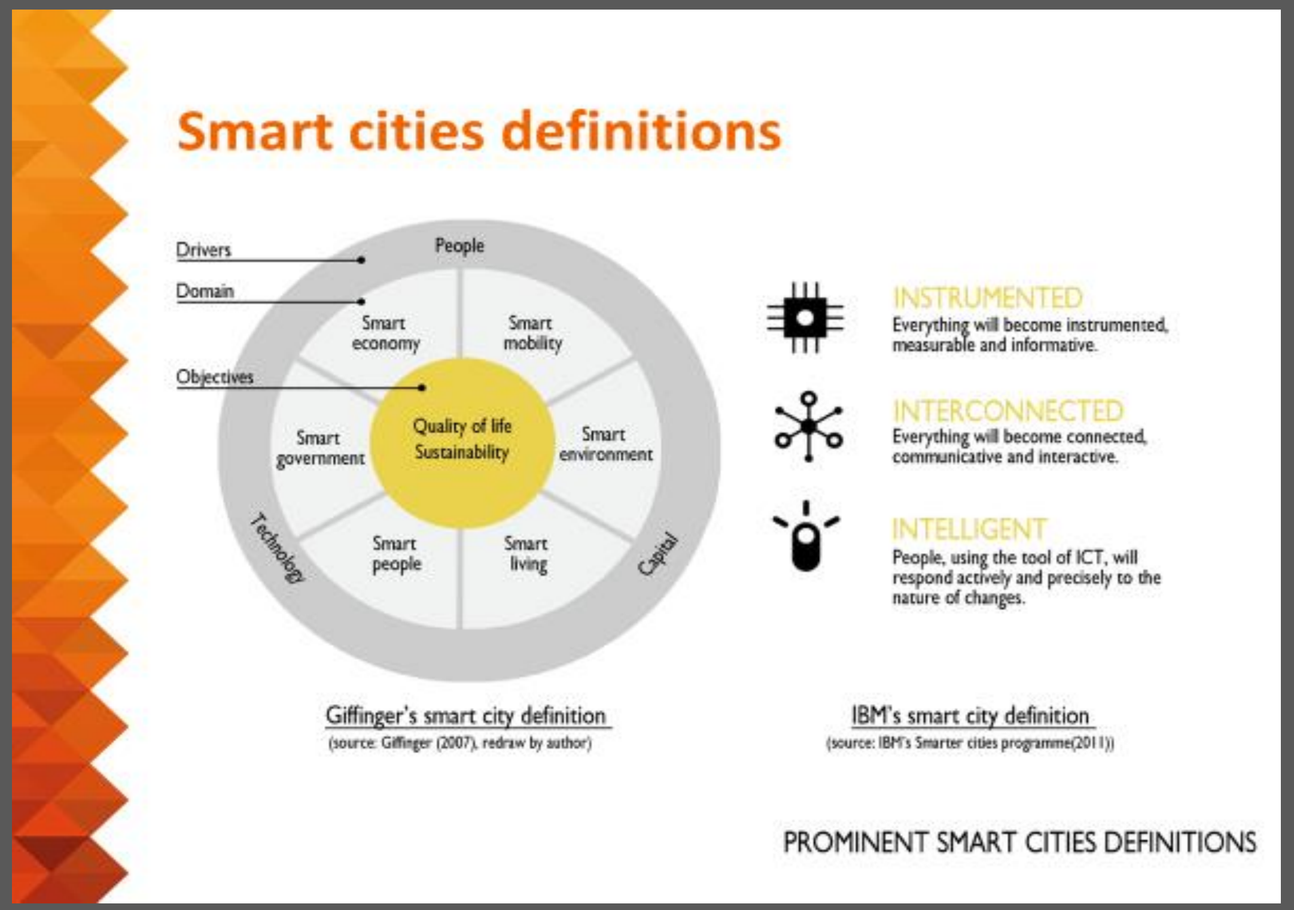
Using Machine Learning Algorithms and Triangulation to Map and Track Entities from Live Video Footage for AUT

Purpose

A joint data science research project between Singapore and New Zealand is currently being conducted. This research is centred around the use of Artificial Intelligence and the Internet of Things to enhance citizen security, lighting, and parking and traffic control, in order to create smart, green, energy efficient cities. Much research has already been completed informing that smart cities produce a higher quality of living. To the left; IBM and Griffinger define what makes a smart city.


Smart cities definitions
Drivers
Domain
Objectives
People
Smart economy
Smart mobility
Smart government
Quality of life
Sustainability
Smart environment
Smart people
Smart living
Technology
Capital
INSTRUMENTED
Everything will become instrumented, measurable and informative.
INTERCONNECTED
Everything will become connected, communicative and interactive.
INTELLIGENT
People, using the tool of ICT, will respond actively and precisely to the nature of changes.
Giffinger's smart city definition
(source: Giffinger (2007), redraw by author)
IBM's smart city definition
(source: IBM's Smarter cities programme(2011))
PROMINENT SMART CITIES DEFINITIONS

However, there is a misalignment between top-down smart city development and citizen's needs. Public participation is essential in smart city development, but it is usually neglected, leading to a dislocation of council and society within a city. Therefore, technologies must be employed to connect citizens and corporations more closely to the improvement of the smart cities' initiative.

The goals set by our client were divided into two parts:

Part 1, consisting of mapping individuals on local area maps, will let any corporations, who possess large areas with many points of interest and many users of these points of interest, to provide locations in relation to their facilities to all who require it. The technology could eventually be purchasable by any party as it will become very profitable for user experience in any area.

Part 2, consisting of tracking vehicles and pedestrians using live camera footage, will allow governments and transport agencies to gather extremely useful data surrounding the locations of objects, types of objects (car, truck, person), and their speed and direction. Then the data can be transferred to semi-AI systems to control parts of a city without need for human interaction making city maintainability and living easier.

Authored by:

Axel Couldrey - 14880202
Yiluo Huang - 16929381
JieQiong Yu - 18017677

Mentor: Parma Nand

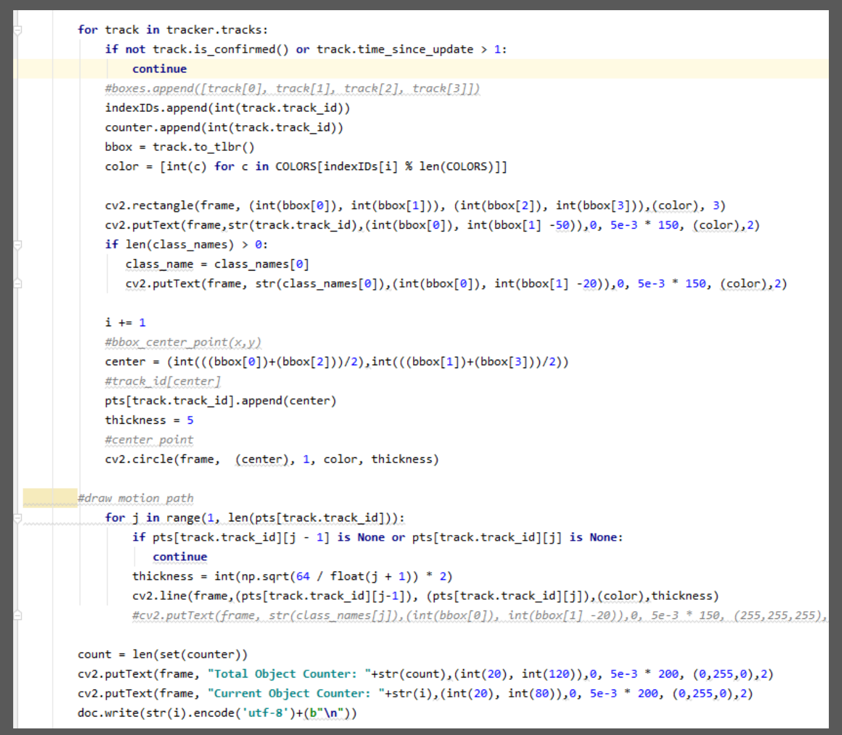
Rationale

The rationale for this project stems from the need expressed above for citizens, government, and service providers to be connected and aware of the progress made in technology to ensure more habitable cities.

Citizens should be able to know in real time, without losing their privacy; where they are, what is around them, any opportunities around them, and any potential hazards. Technology can achieve this and so must be concisely provided to all citizens. It follows then that smart cities will track citizens and other objects in the city, all while not breaking privacy barriers. The eventuality of a smart city knowing where citizens are and identifying hazards will allow automatic control over the improvement of environment and the notification of opportunities and hazards to citizens.

The data gathered by this project will permit for these progressions to be made and lead to much more liveable cities and more productive communities.

No prior work had been completed for the client, so our task was to involve research from a base level of little understanding.



```
for track in tracker.tracks:
    if not track.is_confirmed() or track.time_since_update > 1:
        continue
    #boxes.append([track[0], track[1], track[2], track[3]])
    indexIDs.append(int(track.track_id))
    counter.append(int(track.track_id))
    bbox = track.to_tlbr()
    color = [int(c) for c in COLORS[indexIDs[i] % len(COLORS)]]

    cv2.rectangle(frame, (int(bbox[0]), int(bbox[1])), (int(bbox[2]), int(bbox[3])),(color), 3)
    cv2.putText(frame,str(track.track_id),(int(bbox[0]), int(bbox[1] -50)),0, 5e-3 * 150, (color),2)
    if len(class_names) > 0:
       class_name = class_names[0]
       cv2.putText(frame, str(class_names[0]),(int(bbox[0]), int(bbox[1] -20)),0, 5e-3 * 150, (color),2)

    i += 1
    #bbox_center_point(x,y)
    center = (int(((bbox[0])+(bbox[2]))/2),int(((bbox[1])+(bbox[3]))/2))
    #track_id[center]
    pts[track.track_id].append(center)
    thickness = 5
    #center point
    cv2.circle(frame,  (center), 1, color, thickness)

#draw motion path
    for j in range(1, len(pts[track.track_id])):
        if pts[track.track_id][j - 1] is None or pts[track.track_id][j] is None:
           continue
        thickness = int(np.sqrt(64 / float(j + 1)) * 2)
        cv2.line(frame,(pts[track.track_id][j-1]), (pts[track.track_id][j]),(color),thickness)
        #cv2.putText(frame, str(class_names[j]),(int(bbox[0]), int(bbox[1] -20)),0, 5e-3 * 150, (255,255,255),

count = len(set(counter))
cv2.putText(frame, "Total Object Counter: "+str(count),(int(20), int(120)),0, 5e-3 * 200, (0,255,0),2)
cv2.putText(frame, "Current Object Counter: "+str(i),(int(20), int(80)),0, 5e-3 * 200, (0,255,0),2)
doc.write(str(i).encode('utf-8')+(b"\n"))
```

Code to provide different colour rectangles for each entity showing that two people are not the same.

Method

As a team we began this project rather blind. Two of the three members were currently studying image processing and the third had no experience in the field. We began by addressing the issue of a technology stack. For part one, what were we trying to achieve was a final mobile application product which could be distributed to users who needed guidance in local area maps such as zoos or universities. For part two, we just needed a proof of concept, therefore, any information storage from an MLA we felt would do.

This meant choosing a platform for mobile development which we decided should be iOS due a clean IDE and integration with our goals of obtaining user location and triangulating position. We used this on the provided Apple computers in AUT. This would alter prove troublesome as we had to move systems because of the global pandemic. We decided to switch to QPython and use an android simulator to rebuild the GPS code.

Then for part 2 of the project we used a collection of software and languages to achieve our goal.

Kivy is a module for developing applications on Python which we imported and used for multiple sections of our project. Then we implemented the combination of Yolov3 and deep sort. These are modules used for object tracking and detection.

QPython is an application for writing python on an android device which we used for getting our Android GPS results. This allowed us to record user locations very easily. Then we only had to implement and Android emulator to test the functionality.

The programming languages we used to achieve success within these software frameworks were; initially swift, and then Python. As stated previously due to COVID-19 we couldn't continue to use swift so we changed to Python which was used for all of our project and provided little resistance.

Product artefacts

We delivered most of what we set out to deliver during this project. High-level requirements were followed by all team members.

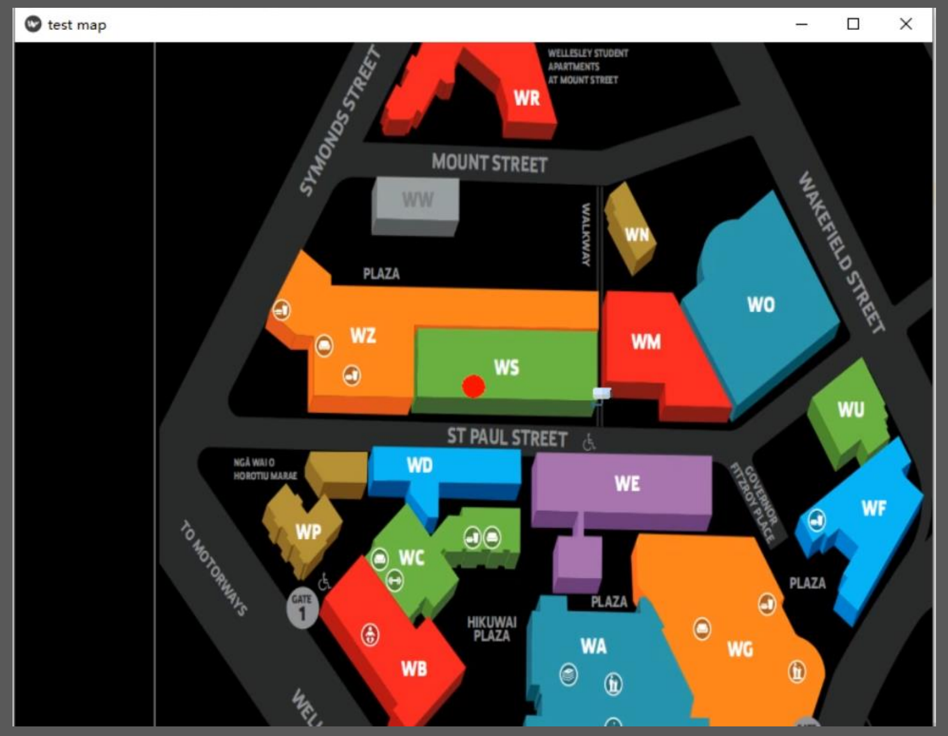
Our high-level requirements outlined in our proposal were as follows:

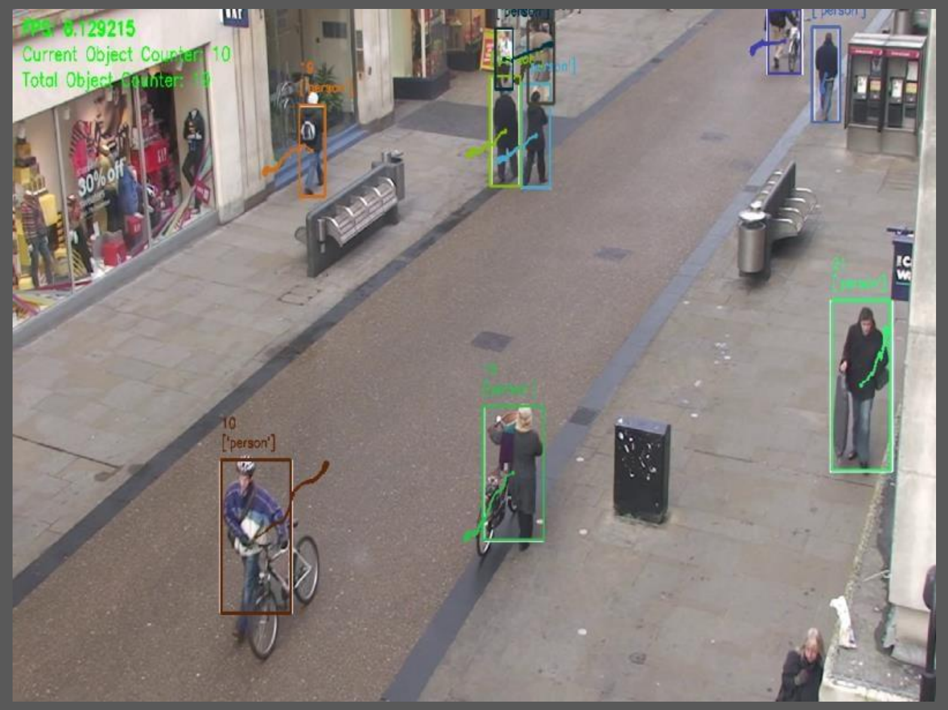
Part 1 Product Requirements:

- The local area map will be viewable in an application.
- Triangulation points will be retrieved from Google Maps and mapped onto the local area map.
- The application created will show the user's current position using the triangulation points in reference to the user's GPS position.
- The location of the user will be accurate to within 1 meter against the local map
- The application will update in real time with movement of a user
- The source of the user's location will not break privacy.

Part 2 Product requirements:

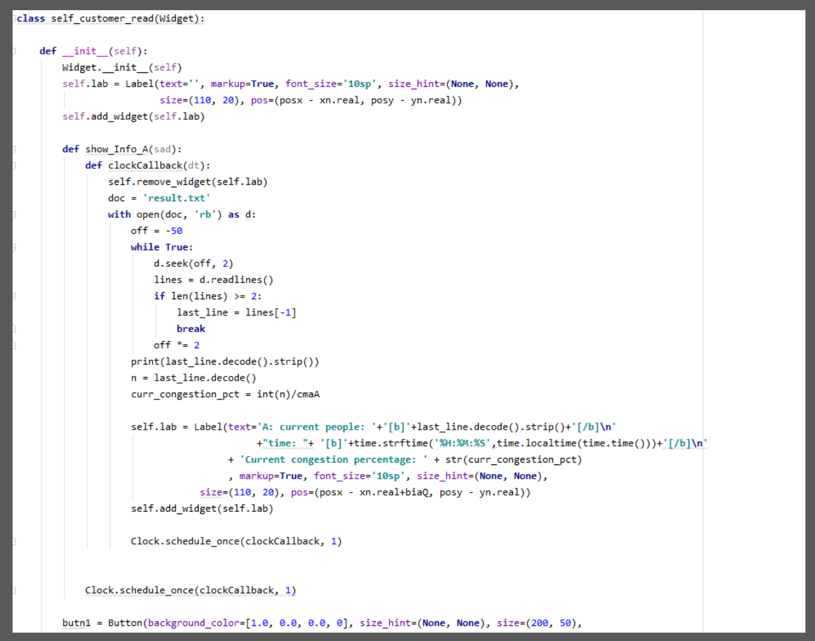
- Cars pedestrians and trucks will be differentiated by the image processing algorithm.
- Location information about the objects processed will be gathered using the same technique from project one with triangulation against Google Maps.
- Locations, timestamps, and type of object will be recorded by a linked database in order for the data to be used later to map or provide statistics of traffic flow.
- Information gathered will be displayed on an application in a table format.
- The data will be mapped back onto a local area map viewable in an app analogous to the one created in part 1.





We delivered a GPS application using android, Kivy, and QPython. This is shown to the top right in the map of AUT, where a red dot represents the location of the user. This user application updates every second allowing for an accurate location using a triangulation method with three arbitrary points. This simple application works well as it has little room for error. As long as the coordinates provided by the user are accurate and the three arbitrary points are consistent, the location is very accurate.

We also delivered a tracking a mapping portion of the project using a combination of yolov3 (a machine learning algorithm for image detection) and an mp4 video file. Originally it was planned to use a live video feed, but we struggled connecting the two and found it just as profitable to connect a recorded video as it was a proof of concept. The data pf all passing entities was recorded using different coloured rectangles which the algorithm could distinguish. Then the total number of entities at any moment could be recorded. A working screenshot of this algorithm is shown below the map.



```
class self_customer_read(Widget):

    def __init__(self):
        Widget.__init__(self)
        self.lab = Label(text='', markup=True, font_size='10sp', size_hint=(None, None),
                         size=(110, 20), pos=(posx - xn.real, posy - yn.real))
        self.add_widget(self.lab)

        def show_Info_A(sad):
            def clockCallback(dt):
                self.remove_widget(self.lab)
                doc = 'result.txt'
                with open(doc, 'rb') as d:
                    off = -50
                    while True:
                        d.seek(off, 2)
                        lines = d.readlines()
                        if len(lines) >= 2:
                            last_line = lines[-1]
                            break
                        off *= 2
                    print(last_line.decode().strip())
                    n = last_line.decode()
                    curr_congestion_pct = int(n)/cmaA

                    self.lab = Label(text='A: current people: '+'[b]'+last_line.decode().strip()+'[/b]\n'
                                          +"time: "+ '[b]'+time.strftime('%H:%M:%S',time.localtime(time.time()))+'[/b]\n'
                                     + 'Current congestion percentage: ' + str(curr_congestion_pct)
                                     , markup=True, font_size='10sp', size_hint=(None, None),
                                 size=(110, 20), pos=(posx - xn.real+biaQ, posy - yn.real))
                    self.add_widget(self.lab)

                    Clock.schedule_once(clockCallback, 1)


            Clock.schedule_once(clockCallback, 1)

        butn1 = Button(background_color=[1.0, 0.0, 0.0, 0], size_hint=(None, None), size=(200, 50),
```

Class in which the map system gets the current number of entities and prints it to the screen every second

Difficulties

We had five notable technical difficulties during this project:

1. Because of the nature of our project in having to change programming language from swift to Qpython, an issue arose where we could not connect Qpython to Kivy. Qpython's latest update didn't allow for Kivy support meaning we couldn't directly update the user location automatically to the GPS map. This couldn't be fixed because of location constraints for our group members and insufficient access to certain machines, but it could be an easy fix in the future through a further update of Qpython or Kivy.

2. We had trouble getting the API for a live video feed from YouTube to run alongside our deep sort model. This was easily overcome with an mp4 video which worked seamlessly and was sufficient for proof of concept.

3. Along with the group separation we wanted all parts of our code to eventually work together. This wasn't achieved in the project but to ensure that it could be in future, we decided to change from swift to python as it was easier to connect components. This wasn't so much of a difficulty as just time consuming. But necessary.

4. We could not print the positions of entities gathered by the machine learning algorithm on to an updating map. This was one of the last requirements posed by our client and was not a complete necessity, but we still hoped we could complete it. All we achieved was to show the coordinates of the actual camera on the map and provided the number of entities passing. This was a sufficient workaround and could be fixed in the future.

5. Our original swift triangulation method was very accurate when the user was positioned in the center of the map (around 10 meters), but as we changed platform, we had to move this method to python. Python proved to be less accurate (around 50-100 meters). This was annoying but as we didn't have access to our previous code, we could not combat it.