AN INTELLIGENT MOBILE APPLICATION TO MONITOR THE AVAILABILITY OF TENNIS COURTS USING MACHINE LEARNING AND OBJECT DETECTION

Harry Su[1], John Morris[2]

[1]Cate School, 1960 Cate Mesa Road, Carpinteria, CA 93013
[2]Computer Science Department, California State Polytechnic University, Pomona, CA 91768

ABSTRACT

Acemind aims to address the lack of real-time tennis court availability information, enhacing player experience, promoting community well-being, and making sports easily accessible to not only the wealthy [1]. The technology utilizes Raspberry Pi computers with Pi cameras to record live footage, Firestore Database to store information regarding court status, and a front-end mobile application made with flutter to display information to users [2]. Key challenges include running object detection model, YOLOv5, on the computer seamlessly with the camera, which was solved by adjusting libraries' versions appropriately and ensuring the proper installation of all packages [3]. The mobile application also struggled to display the correct court's information, but the problem was fixed with a setState function that updates the bottom popup widget using a variable. During experimentation, YOLOv5 consistently identified humans among distractions commonly found near tennis courts even under suboptimal conditions, proving its resilience to unwanted challenges in inputs [4]. Although Acemind has limitations such as the need of a cellular connection and government permission, it is useful as it presents valuable court information regarding availability without the shortcomings of smart tennis courts and tennis maps, bridging the inequality gap by providing everybody a change to play.

KEYWORDS

Tennis, AI, Community, Convenience

1. INTRODUCTION

The absence of real-time information regarding court availability introduces a significant element of uncertainty, causing individuals to invest time and effort traveling to a tennis facility only to discover that all courts are occupied. This not only diminishes the joy of engaging in the sport but also leads to a suboptimal utilization of tennis facilities, frustrating players eager to participate. However, the challenge of locating available tennis courts transcends mere inconvenience as the impact of limited court availability extends to the well-being of individuals and communities. Tennis, recognized for its physical demands, promotes fitness andserves as a platform for forging new connections and friendships. The sport provides an opportunity for people to connect easily, meet up on a regular basis, and build communication skills in a game of doubles. The inability to access courts in a timely manner hinders the social aspect of the sport and may also dissuade individuals from regular play, compromising their physical and mental health. This is critical considering that excercise is not only “associated with reduced risk of 13 different types of cancer, including breast, colon, liver and myeloid leukemia,” but people who are physically active also “reported more excitement, happiness, and motivation” [5]. Thus, in addressing the challenge of locating open tennis courts, communities prioritize not only the convenience of individuals but also the cultivation of an active and socially connected environment.

First, the traditional two-stage object detection pipeline, involving a Region Proposal Network, is computationally complex for real-time applications. Conversely, YOLOv5's one-stage architecture processes the entire image in a single pass, offering a flexible trade-off between speed and accuracy. Employing anchor boxes and different model sizes enables YOLOv5 to match the Raspberry Pi's image capture rate effectively.

Second, smart tennis courts, incorporating technology like sensors and cameras, enhance the tennis experience by tracking player movements and providing data-driven features. However, even though smart courts can detect whether they are in use, widespread adoption is hindered by high costs.

The last solution, Tennis Maps, addresses the lack of facilities by mapping and documenting courts, offering users court details. Unlike smart courts, it is more feasible for the general public, but it does not offer information regarding a court’s live availability. Acemind, by adopting YOLOv5, provides confidence about tennis court availability in real-time, overcoming the limitations of smart courts and other mapping solutions.

Acemind court tracker is a solution to this problem by monitoring courts with cameras that can detect people and reflect the availability on a mobile application. By addressing this challenge, tennis enthusiasts can enjoy a more seamless experience, fostering a greater sense of accessibility and convenience within the tennis community. This is more effective than any reservation system because saving a court hours or days before playing disrupts the spontaneity of tennis—not everyone plans out when they would play. In addition, nobody can enforce the use of a reservation system at many public courts. The system’s success depends on individuals’ kind cooperation, which is often unreliable. There is also no reservation system that apply to all tennis courts in an area, so people must use different applications to reserve courts at different places, causing more inconvenience. Acemind court tracker solves these issues by informing people courts’ statuses anytime and anywhere all in one application. It is also great because in the future, Acemind can implement more features such as displaying weather conditions, court usage traffic analytics, and expanding this technology to other sports to encourage staying active, which are all feats that a reservation system can never accomplish.

The computer vision component's accurate functionality is crucial for the project's success because the failure to accurately detect people undermines the technology’s ability to show courts’ availability. The first experiment strategically placed objects resembling humans, such as closed umbrellas and tall trash cans, at a distance to test the program's ability to distinguish people from distractions. Despite varying conditions, the YOLOv5 model consistently identified individuals among objects, demonstrating reliable object detection algorithms. Realistic scenarios, including different court locations, angles, and lighting, were tested in the second experiment to ensure the program's adaptability. The AI effectively recognized people even in less-than-ideal conditions [6]. The experiment's comprehensive setup, with cameras positioned at diverse court locations, confirmed the model's consistent accuracy in reflecting tennis court status. YOLOv5's resilience to changes in input conditions, while maintaining reliable assessments, is a reason for such consistent results and establishes its effectiveness in providing trustworthy outputs across diverse environments.

2. CHALLENGES

In order to build the project, a few challenges have been identified as follows.

2.1. Raspberry Pi

The Raspberry Pi computer cannot recognize the attached Pi camera when a script attempts to use it. The legacy camera was enabled in the computer's configuration, and the camera appears as supported and detected after running the command "vcgencmd get_camera" in the terminal. It can also take pictures with the command "libcamera_still -o test.jpg," but the camera cannot be detected when accessed through python code. To resolve this issue, I must make sure that all the packages are properly installed by running all necessary commands in the terminal, which should link the code with the hardware.

2.2. The Libraries

Two libraries, torchaudio and torchvision, cannot be found installed on the Raspberry Pi, which prevents YOLOv5 from analyzing the picture that was taken by the camera. Failure of the computer vision module would hinder the program from knowing a court's availability. However, the commands "pip show torchvision" and "pip show torchaudio" prove that both models are saved locally with all their information, including their version, summary, location, and author. To resolve this issue, I could research which versions of the libraries function with my Raspberry Pi and downgrade torchaudio and torchvision correspondingly.

2.3. Popup Information

For the application, the information that appears in the bottom popup should reflect the court that is selected on Google Maps [8]. However, after the user has already selected a court and decide to get information about a different court, the information in the popup would still show the court that was selected first because everything has been initialized. The program does not know that the selected court has changed, but to reinitialize all content would be inefficient. To resolve this issue, I could use a setState function and make the bottom popup a variable storing the widget.

3. SOLUTION

The three major components are the hardware, the backend database, and the front-end mobile application. For the hardware, a raspberry pi computer was used with a pi camera, which captures an image of its tennis court once every six seconds. This image is analyzed by an computer vision model, YOLOv5, locally, which determines a court's availability by detecting people on the court. This information is sent to Google's Firestore Database, the second major component that enables the mobile application to get constant updates regarding a court's status [7]. The front-end program is made using flutter and Google maps. It has a main screen that displays the user's current location and the tennis courts nearby. Their statuses are reflected by the color of a court's icon: green represents vacancy, and red represents unavailability. If the user wants more information regarding any court, they can click on the icon to see pictures. By clicking the Google Maps icon above the pop-up window, the user can get instructions to get to this court.

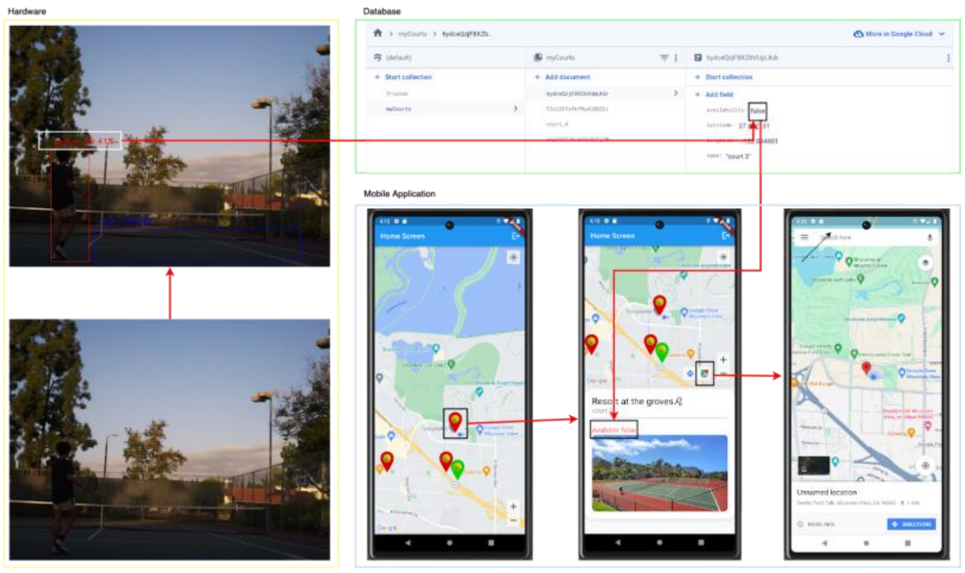


Figure 1. Overview of the solution

The purpose of the pi camera connected to a raspberry pi computer is to monitor the courts and know their availability. YOLOv5, or “You Only Look Once” version 5, is an object detection algorithm designed for realtime object detection tasks, and it is used to quickly detect people on tennis courts by making predictions in a single pass through the neural network and employing anchor boxes for bounding box predictions [9].

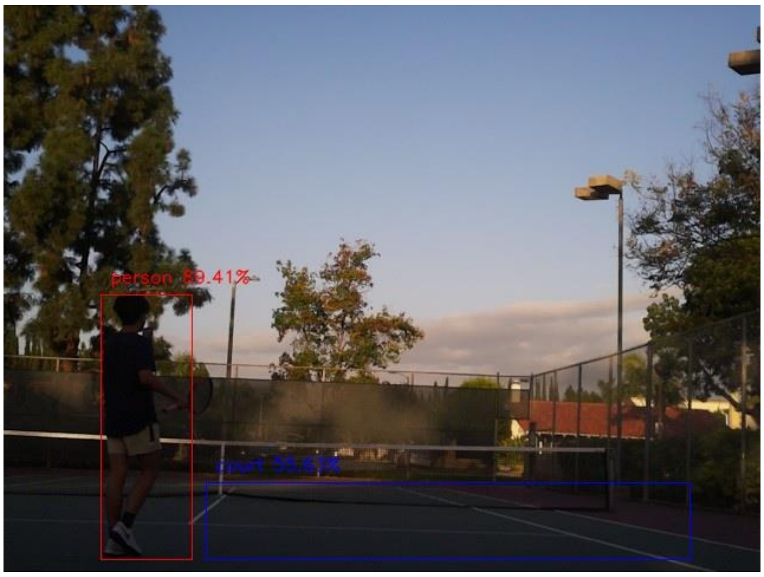
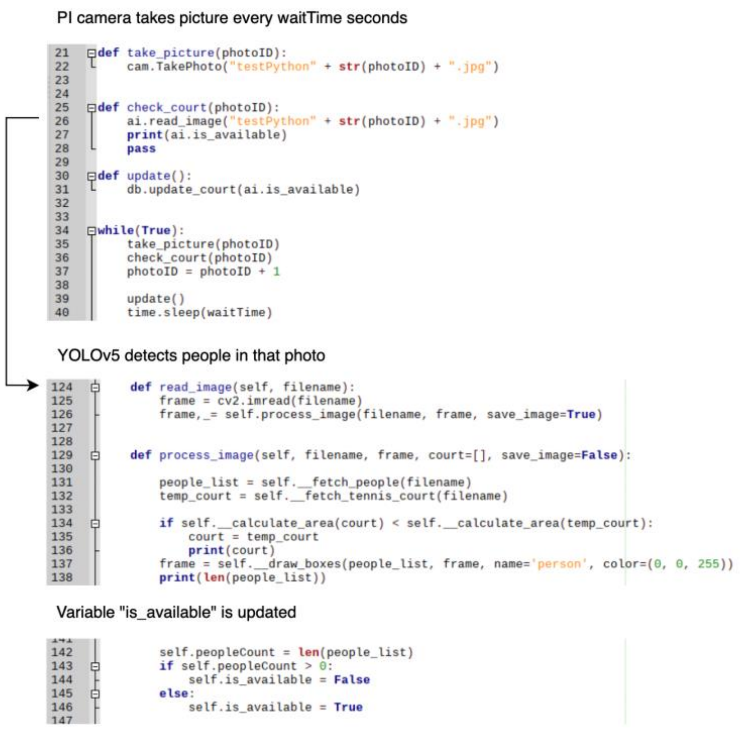


Figure 2. Camera detect picture


PI camera takes picture every waitTime seconds
YOLOv5 detects people in that photo
Variable "is_available" is updated

```
21  def take_picture(photoID):
22      cam.TakePhoto("testPython" + str(photoID) + ".jpg")
23
24
25  def check_court(photoID):
26      ai.read_image("testPython" + str(photoID) + ".jpg")
27      print(ai.is_available)
28      pass
29
30  def update():
31      db.update_court(ai.is_available)
32
33
34  while(True):
35      take_picture(photoID)
36      check_court(photoID)
37      photoID = photoID + 1
38
39      update()
40      time.sleep(waitTime)
```

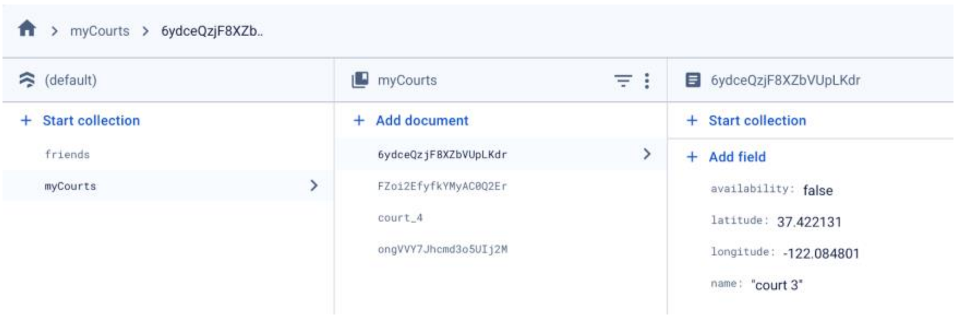
```
124     def read_image(self, filename):
125         frame = cv2.imread(filename)
126         frame,_= self.process_image(filename, frame, save_image=True)
127
128
129     def process_image(self, filename, frame, court=[], save_image=False):
130
131         people_list = self.__fetch_people(filename)
132         temp_court = self.__fetch_tennis_court(filename)
133
134         if self.__calculate_area(court) < self.__calculate_area(temp_court):
135             court = temp_court
136             print(court)
137         frame = self.__draw_boxes(people_list, frame, name='person', color=(0, 0, 255))
138         print(len(people_list))
```

```
142         self.peopleCount = len(people_list)
143         if self.peopleCount > 0:
144             self.is_available = False
145         else:
146             self.is_available = True
147
```

Figure 3. Screenshot of code 1

From line 34, the Pi camera begins an infinite while loop to take a picture of its court every waitTime seconds. In this case, waitTime is set to six. For every iteration of the loop, the camera runs the "take_picture" function, and the photoID parameter keeps track of the most recently taken photo, which helps with testing purposes [10]. Then, the "check_court" function takes the photoID and puts the corresponding picture through the AI class' "read_image" function. This function, along with the "process_image" function, finds a list of people (figures that the engine identifies as human) in the image, and the length of the list corresponds with the number of people. This information is used to update the "is_available" variable using a simple "if-statement."

The back-end, using Google's Firestore Database, is used to store information regarding the statuses of all the courts. It serves as a bridge between the two other components as the database can be updated by the raspberry pi computer and accessed by the mobile application.

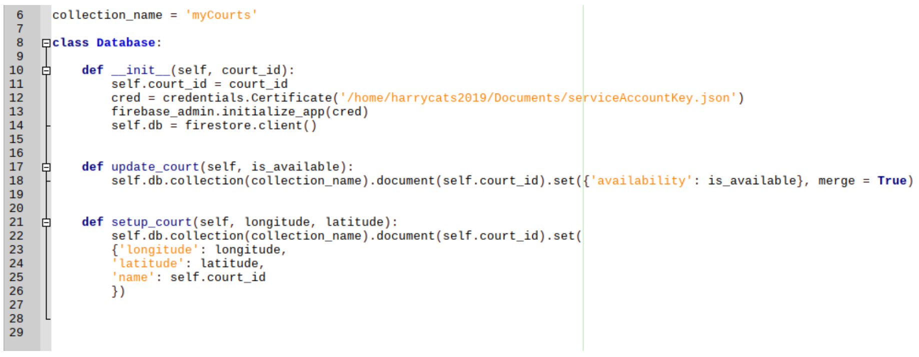


friends
myCourts

6ydceQzjF8XZbVUpLKdr
FZoi2EfyfkYMyAC0Q2Er
court_4
ongVVY7Jhcmd3o5UIj2M

availability: false
latitude: 37.422131
longitude: -122.084801
name: "court 3"

Figure 4. Screenshot of database



```
collection_name = 'myCourts'

class Database:

    def __init__(self, court_id):
        self.court_id = court_id
        cred = credentials.Certificate('/home/harrycats2019/Documents/serviceAccountKey.json')
        firebase_admin.initialize_app(cred)
        self.db = firestore.client()


    def update_court(self, is_available):
        self.db.collection(collection_name).document(self.court_id).set({'availability': is_available}, merge = True)


    def setup_court(self, longitude, latitude):
        self.db.collection(collection_name).document(self.court_id).set(
        {'longitude': longitude,
        'latitude': latitude,
        'name': self.court_id
        })
```

Figure 5. Screenshot of code 2

The credentials key was saved locally and accessed via the __init__ function, which allows the Raspberry Pi computer to access and modify data saved in the Firestore Database. The function also initializes variables such as "court_id" to represent which court is being accessed. "db" is used as an instance of of firestore's client used later to locate documents and change values. The database class only contains two public methods: update_court and setup_court. The former accepts a boolean parameter to represent a court's status and changes the availability of the court based on the passed in value. The other function establishes a court in Firestore Database if it is not registered by accepting the court's location. These functions are called after YOLOv5 finishes analyzing a picture and discerns a court's new status.

Lastly, the front-end mobile application is where the information about courts is displayed so that anyone with the app can view. A combination of Flutter and Google Maps are used to create the user interface, and the availability is retrieved via an async function that waits for Firestore Database's document, saved in a custom datatype in flutter.

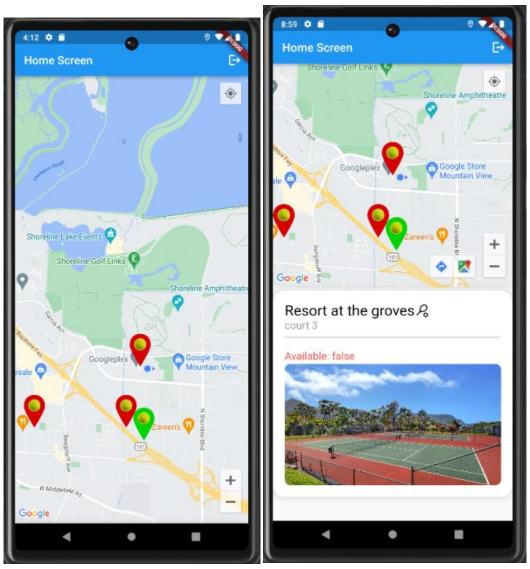
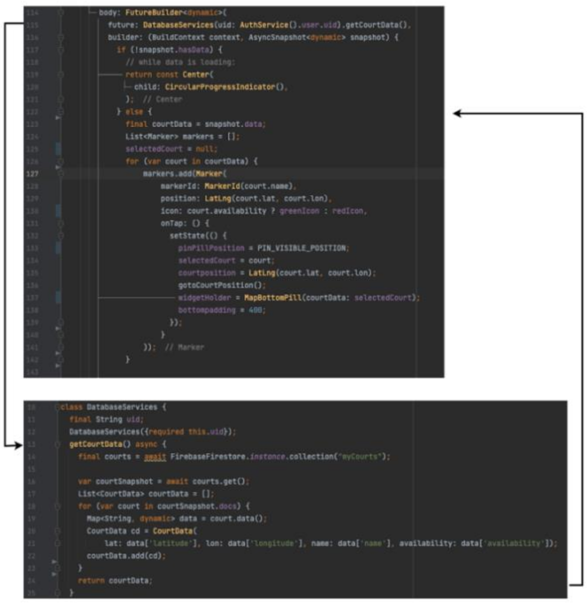


Figure 6. Screenshot of map



```
body: FutureBuilder<dynamic>(
  future: DatabaseServices(uid: AuthService().user.uid).getCourtData(),
  builder: (BuildContext context, AsyncSnapshot<dynamic> snapshot) {
    if (!snapshot.hasData) {
      // while data is loading:
      return const Center(
        child: CircularProgressIndicator(),
      ); // Center
    } else {
      final courtData = snapshot.data;
      List<Marker> markers = [];
      selectedCourt = null;
      for (var court in courtData) {
          markers.add(Marker(
              markerId: MarkerId(court.name),
              position: LatLng(court.lat, court.lon),
              icon: court.availability ? greenIcon : redIcon,
              onTap: () {
                setState(() {
                  pinPillPosition = PIN_VISIBLE_POSITION;
                  selectedCourt = court;
                  courtposition = LatLng(court.lat, court.lon);
                  gotoCourtPosition();
                  widgetHolder = MapBottomPill(courtData: selectedCourt);
                  bottompadding = 400;
                });
              }
          )); // Marker
      }
```

```
class DatabaseServices {
  final String uid;
  DatabaseServices({required this.uid});
  getCourtData() async {
    final courts = await FirebaseFirestore.instance.collection("myCourts");

    var courtSnapshot = await courts.get();
    List<CourtData> courtData = [];
    for (var court in courtSnapshot.docs) {
      Map<String, dynamic> data = court.data();
      CourtData cd = CourtData(
          lat: data['latitude'], lon: data['longitude'], name: data['name'], availability: data['availability']);
      courtData.add(cd);
    }
    return courtData;
}
```

Figure 7. Screenshot of code 3

The futurebuilder waits for the getCourtData function to complete and return information about all courts in the form of a list. Until the function completes, the module would only display a circular progress indicator to represent loading. Inside getCourtData, the system waits for Firestore Database to initialize and converts the raw json file given by the backend into a list of “CourtDatas,” a custom data type used in the application that has three strings and a boolean. The strings are for a court’s longitude, latitude, and name; the boolean represents a court’s availability. Then, for every court in the list, a marker is created based on the item’s location and availability. The markers also have programming for when they are pressed, which brings up a popup from the bottom, pushes Google Maps to only fill only the top half, and makes the court the selected court.

4. EXPERIMENT

4.1. Experiment 1

It is important that the computer vision component functions properly because its failure to identify people accurately would defeat the purpose of the project.

The experiment was set up such that objects that can resemble humans were placed in the far distance. For example, there were closed umbrellas, tall trash cans, big water jugs, and a big basket of tennis balls, which are all objects that are commonly found near tennis courts. The program must be able to distinguish them from people in order to be trusted. To test whether the program can detect a real human among these objects, I stood in the distance as well. Control data is a court where such distractions do not exist.

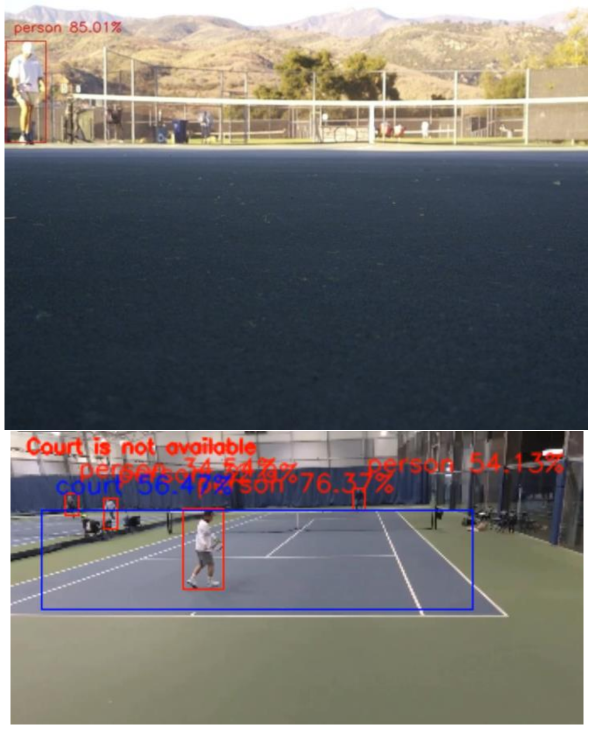


Figure 8. Figure of experiment 1

In both the control and the case with distractions, the model accurately distinguished people from objects. It is surprising that the AI did not recognize the objects as potential people with really low confidence. It turned out this way because YOLOv5 has excellent object detection algorithms, and the photos had enough clarity for successful results. In summary, under realistic circumstances, no objects were able to affect the program negatively or err its outputs in the experiment.

4.2. Experiment 2

It is also essential that the artificial intelligence can recognize people when set up at different places with different angles on a court because the camera cannot always be set at one place.

The experiment was set up such that cameras were set up at different courts, at different locations of a court, and at different angles. The control is the ideal location and angle, with the camera having a complete view of a court—the position used for streaming matches. Other places involve having the camera on the side of a court, on the ground, and in environments where light is not abundant. The test is constructed this way to see if the program can recognize people when it does not have the most ideal images as inputs.

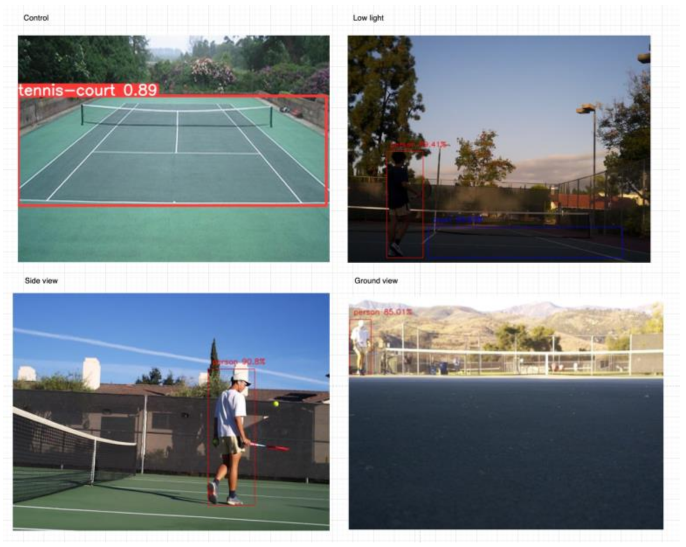


Figure 9. Figure of experiment 2

In all cases with different angles, locations, and lighting, the model accurately reflected the status of a tennis court. Given the results from the first experiment, it is no longer surprising that the artificial intelligence can recognize people even when they are not the clearest. It turned out this way because even though the conditions under which the photos were taken changed, the features of a person stayed the same, so YOLOv5 is not affected by the variance and continue to provide accurate assessments of a court's status. Thus, different angles, locations, and lighting do not influence the model's performance given that the camera's view is not obstructed.

5. Related Work

Traditional object detection typically follows a two-stage pipeline, involving a Region Proposal Network (RPN) and subsequent object classification and refinement [11]. This approach, while accurate, is computationally complex and slower, particularly for real-time applications. In contrast, YOLOv5, which is used for this project, adopts a one-stage architecture, processing the entire image in a single pass and predicting bounding boxes along with class probabilities directly. YOLOv5's anchor box mechanism, coupled with various model sizes, allows for a flexible trade-off between speed and accuracy, therefore being able to accurately keep up with the Raspberry PI's rate of capturing an image every six seconds.

Smart tennis courts integrate technology like sensors, cameras, and wearables to enhance the tennis experience [12]. These courts track player movements, analyze performance data, and offer features such as automated line calling, lighting control, and mobile app connectivity. The technology facilitates performance improvement, remote coaching, and a more interactive and data-driven approach to playing and enjoying tennis. However, although they can detect whether a court is in use, it is impractically to equip every court with such advanced technology due to its high price. Conversely, my solution only involves one camera, which although doesn't tell users as much information as a smart court, is more feasible for the general public.

Tennis maps attempts to solve the problem of a lack of facilities by having every court documented on a map [13]. People can see all tennis complexes in any area and decide on which one they want to go to. The courts are represented by rectangles, and the number of courts total is represented by the size of the rectangles. Additional information such as whether a facility is private or public, indoors or outdoors, lighted or not is also displayed. This solution is not as effective as desirable because it not only fails to reflect courts' live status but also fails to have all courts documented. Some newer courts and even major complexes in my town are not shown. Additionally, people still have to face uncertainty regarding a court's availability. My solution can provide users with confidence about whether they can play tennis at any time.

6. CONCLUSIONS

Limitations of Acemind include its need of a cellular connection for the devices to communicate with the Firestore Database. As a result, currently, there must be wifi-routers near the courts, which is an inconvenience given that a lot of public sports facilities do not offer this service [15]. To solve this problem, each device can include a cellular card that acts as a source of wifi connection. Since only a boolean value is uploaded to the backend, the cellular connection doesn't have to be robust. Additionally, due to privacy concerns, communities might be reluctant to implement this technology for its decently high-quality cameras. However, this problem can also be resolved by communicating with community officials and enforcing strict protection on the footage captured by the cameras. If the issue continues, the camera sensor on the devices can also be changed to be auditory, but this last resort is unlikely to be necessary given that parking lots have already implemented similar cameras to detect vacancy.

The lack of real-time information on tennis court availability affects players' joy, discourages excercizing, and negatively impacts social connections. Acemind court tracker, using camera monitoring and a mobile app, offers a seamless solution, not only prioritizing accessibility, spontaneity, and community well-being, but it also has potential for expansion and endless possibilities, overcoming reservation systems' limitations [14].

REFERENCES

[1] Starbuck, Chelsea, et al. "The influence of tennis court surfaces on player perceptions and biomechanical response." Journal of sports sciences 34.17 (2016): 1627-1636.

[2] Jolles, Jolle W. "Broad-scale applications of the Raspberry Pi: A review and guide for biologists." Methods in Ecology and Evolution 12.9 (2021): 1562-1579.

[3] Wu, Wentong, et al. "Application of local fully Convolutional Neural Network combined with YOLO v5 algorithm in small target detection of remote sensing image." PloS one 16.10 (2021): e0259283.

[4] Fang, Yiming, et al. "Accurate and automated detection of surface knots on sawn timbers using YOLO-V5 model." BioResources 16.3 (2021): 5390.

[5] Lowenberg, Bob, James R. Downing, and Alan Burnett. "Acute myeloid leukemia." New England Journal of Medicine 341.14 (1999): 1051-1062.

[6] Shi, Yuanming, et al. "Communication-efficient edge AI: Algorithms and systems." IEEE Communications Surveys & Tutorials 22.4 (2020): 2167-2191.

[7] Kesavan, Ram, et al. "Firestore: The nosql serverless database for the application developer." 2023 IEEE 39th International Conference on Data Engineering (ICDE). IEEE, 2023.

[8] Mehta, Heeket, Pratik Kanani, and Priya Lande. "Google maps." International Journal of Computer Applications 178.8 (2019): 41-46.

[9] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[10] Megreya, Ahmed M., Adam Sandford, and A. Mike Burton. "Matching face images taken on the same day or months apart: The limitations of photo ID." Applied Cognitive Psychology 27.6 (2013): 700-706.

[11] Zhou, Yimiao, and Heng Shen. "Wireless Sensor Network Technology-Based Design and Realization of Intelligent Tennis Sports System." Wireless Communications and Mobile Computing 2022 (2022).

[12] Karthi, M., et al. "Evolution of yolo-v5 algorithm for object detection: automated detection of library books and performace validation of dataset." 2021 International Conference on Innovative Computing, Intelligent Communication and Smart Electrical Systems (ICSES). IEEE, 2021.
[13] Solomon, Richard, et al. "PLAY Project Home Consultation intervention program for young children with autism spectrum disorders: A randomized controlled trial." Journal of Developmental and Behavioral Pediatrics 35.8 (2014): 475.
[14] Hoehle, Hartmut, and Viswanath Venkatesh. "Mobile application usability." MIS quarterly 39.2 (2015): 435-472.
[15] Hu, Peng Fei, and Kwok Wa Leung. "Polarization Diversity Glass Antenna for Consumer WiFi Routers." IEEE Transactions on Consumer Electronics (2023).