CSE 593 Independent Study, Summer 2022: Indoor Pedestrian Detection by combining Visual and Sensor Tracking

Vanshika Sharma

August 16, 2022

1 Abstract

Computer Vision and Deep Learning has come a long way with its algorithms to track pedestrians and objects visually using just camera and visual feeds. However, vision based pedestrian tracking becomes challenging when the pedes- trian temporarily gets occluded by an object in the scene, to an extent that visual tracking can no longer track him. In this project, a persistent indoor pedestrian tracking system is implemented which combines visual signal from surveil- lance cameras and sensor signals from Inertial Measurement Unit (IMU) carried by pedestrians themselves. IMU tracking performs Dead Reckoning (DR) utilizing accelerometer, gy- roscope and magnetometer. IMU tracking works independent of visual tracking and does not stop working even during visual occlusion. The experimental results show that the IMU and visual tracking complement each other and their combination performs robust pedestrian tracking in many challenging scenarios

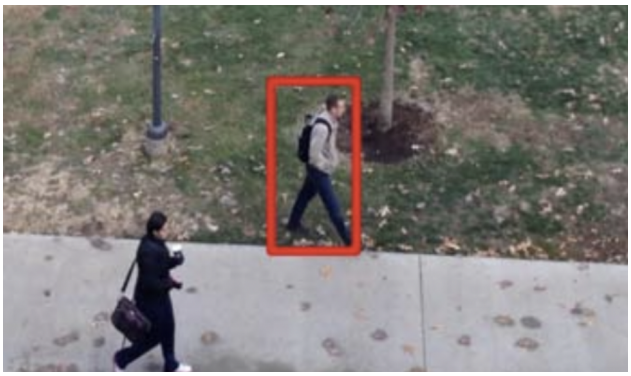


Figure 1: Visual Tracking

Figure 2: Aim: Tracking to Continue while Occlusion

2 Problem Introduction

Pedestrian Tracking has a wide range of applications, such as activity recognition, behaviour analysis, surveillance analysis, etc. This problem is not just limited to pedestrians, but also mobile robot navigation and other autonomous systems. Therefore, in order to perform the tracking and analysis robustly, it is crucial that the person's actions are monitored and analysed in an accurate and persistent manner. There can be two types of tracking involved to achieve this task: "Passive" and "Active" Tracking. A person can be passively tracked when he is not carrying or wearing any device, for example, tracking a person in a video stream from a surveillance camera using computer vision. Whereas, active tracking is performed where a person carries an electronic device, based on the data of which, person's movement, location or activity can be tracked. These devices include GPS systems, WiFi, base station signal receiver and Inertial Measurement Unit (IMU) sensors. The main task to be performed is to deal with the visual occlusion problem and combine these two types of trackings and develop an integrated system where sensor tracking can complement and overcome the challenges of vision based tracking, when required. The solution implemented in this project can also be used to track cooperative targets (e.g., children, teens, the elderly, patients with autism/alzheimers/dementia) by combining passive and active tracking. In this project, visual tracking is performed on the basis of an object detection algorithm called YOLOv3, whereas active tracking uses the Dead Reckoning (DR) Approach implemented using the IMU sensor: accelerometer, gyroscope and magnetometer data from the pedestrian's phone.

3 Literature Review and Related Work

3.1 Passive Tracking

This project work is deeply inspired from [1], it solves this problem using a similar methodology, one which has been used in this project. [1] first implements passive tracking by implementing Histogram of Gradients for feature extraction from images and videos of pedestrians and then uses these features to train a Support Vector Machine as a classifier. This classifier then classifies and detects people present in a fresh image or video or a real-time surveillance footage. This work also highlights some key facts and issues with the current tracking algorithms and approaches. It refers to the passive vision based pedestrian detection and tracking which has been studied and implemented for a long time [4, 5, 6, 7, 8, 9, 10, 11]. However, it is observed that when there is heavy occlusion, nearby clutter or pedestrians temporarily moving out of the field of view as shown in Fig.1, it is still difficult for a vision based tracking algorithm to persistently track the pedestrian without failure. As soon as one of the scenarios take place, the visual tracking stops. No matter how much the system and models are honed, it is still integral for the person to be visible in the video stream at all times and in the clearest way possible. Most of the previous efforts on pedestrian tracking in videos are made in the following two categories.

When the pedestrians are partially occluded, previous work mostly relies 2 on tracking the body parts that are visible. For example, Wu et al. [12] combine body part detectors which are trained by boosting edgelet feature based weak classifiers and whole pedestrian detectors. Merad et al. [13] classify pedestrian's appearances into front and back poses in addition to segmenting detected individuals into several parts such as head and torso. However, part-based tracking algorithms can only overcome the challenges unless the target is not totally occluded or has disappeared for a long time. Thus, more researchers focus on predicting the positions of occluded pedestrians. For realtime prediction, Li et al. [14] improves the traditional mean shift tracking algorithm by proposing "occlusion layers" to represent the pedestrian occlusion relation, and the non-occlusion part of the pedestrian is used for the mean shift tracking. Leykin and Riad [6] introduces a pedestrian tracker designed as a particle filter using a combined inputs from color and thermal cameras. These algorithms assume the pedestrians do not change their forwarding direction and speed when occluded, which may be not true in practice. Besides the aforementioned realtime tracking methods, Zhang et al. [15] proposes a unified algorithm that automatically learns the trajectory models from the local and global information to obtain an optimal assignment. Sherrah [16] tracks a pedestrian over short time-frames to form tracklets. Then an optimal path finding problem is posed in the generalized Hough space and can be solved using the Viterbi algorithm. Trajectory as- signment depends on the similarity of pedestrians' appearances. However, the appearance of the target may change a lot before and after the occlu- sion because of the illumination and viewpoint, which results in inaccurate matches. In [], Histogram of Gradients for feature extraction and Support Vector Machine classifier is used to detect pedestrians visually. This approach is efficient but comes at a cost of computation and time consumption. In the recent times, there have been a lot of upgrades through deep learning and neural networks and a lot of alternatives for this algorithm

have been developed which have proved to be more optimal and efficient in terms of time and computation cost.

3.2 Active Tracking

[1] performs active tracking by using four key steps to implement pedestrian dead reckoning: Step Detection, Pedestrian Velocity Estimation, Forward Movement Direction Detection and Coordinate Transformation. This project is again highly inspired by these steps and this work acted as a guide to implement this project. Highlighting the works been referred to in this paper, it discusses about tracking pedestrians with GPS since it is already widely used in vehicle navigation. However, it is observed that the accuracy of a common GPS module is not high enough, for example, Garmin reports its GPS receivers are accurate to within 15 meters on average [17]. Compared with common moving vehicles, time delay on locating pedestrians slower than vehicles is highly possible when using GPS. When a pedestrian starts moving, the position obtained from GPS may maintain as a constant for 2 to 3 seconds. When the pedestrian stops, the position reported from GPS may still move for a while. Such asynchrony leads tracking failures or frequent mistakes when analyzing a pedestrian's behaviour. Furthermore, GPS is based on the direct signal transmission between a GPS receiver and satellites. Obstructions such as city canyons or tall trees outdoors and walls/ceilings indoors cut off or weaken the signals, making the GPSbased localization and tracking unre liable. The author performed a GPSbased pedestrian tracking using Google Map on a Samsung Galaxy s4, as shown in Fig.2, when the pedestrian walked along a straight line indoors from red points A to B (50 meters away from each other), the GPS-based trajectory is from blue points C to D which are almost stationary at the same location, deviating from the true pedestrian trajectory. WiFi is another type of ubiquitous signal in daily lives [18, 19, 20]. However, WiFi hotspots are available mostly in indoor environments and the coverage area of most WiFi hotspots is less than 50 meters, limiting its application in pedestrian tracking outdoors. Base stations in mobile telephony are high power and can cover a radius as large as 35km, with which we can make a call and send messages. However, base station is only suitable for coarse localization and tracking [21]. Inertial Measurement Unit (IMU) is a good choice for active pedestrian tracking, which consists of Gyroscope, Magnetometer and Accelerometer. In an ideal environment without any noise or movement, Magnetometer measures the direction of the earth's geomagnetic north pole and Accelerometer measures the gravity. The two signals can be used to set up a world coordinate system, which is available everywhere on the earth only except the north and south poles. So, unlike GPS that is inaccurate and unreliable indoors or WiFi that needs to artificially install WiFi hotspots, IMU-based active pedestrian tracking systems can be applied almost every where except the north and south poles.

4 Proposed Solution

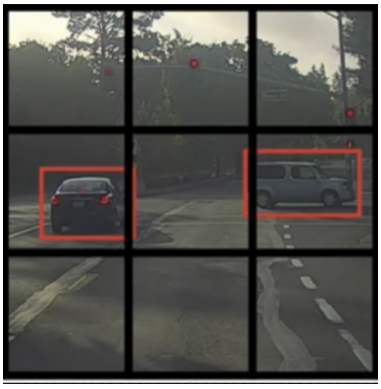
The solution proposed to solve the problem is by combining Active and Passive Tracking and making them complement each other, when required. The first step towards implementing the aforementioned efficiently is by implementing active and passive tracking separately and then combining them. In this project, I have first implemented passive tracking using the YOLO V3 algorithm, and then followed a certain set of steps to implement Dead Reckoning for Active Tracking. The scope of this project report is till the implementation of Dead Reckoning and the analysis of its results. The steps, algorithms and procedures to be followed after this to implement the last step of integration of active and passive tracking will be discussed as future work and extension of the work which has been implemented till now.

4.1 YOLOv3 for Visual Person Detection and Tracking

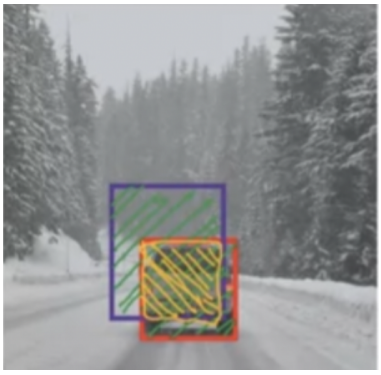
YOLOv3 (You Only Look Once, Version 3) is a real-time object detection algorithm that identifies specific objects in videos, live feeds, or images. The YOLO machine learning algorithm uses features learned by a deep convolutional neural network to detect an object. Versions 1-3 of YOLO were created by Joseph Redmon and Ali Farhadi, and the third version of the YOLO machine learning algorithm is a more accurate version of the original ML algorithm.

The first version of YOLO was created in 2016, and version 3, which is discussed and implemented extensively in this report and project respectively, was made two years later in 2018. YOLOv3 is an improved version of YOLO and YOLOv2. YOLO is implemented using the Keras or OpenCV deep learning libraries. Object classification systems are used by Artificial Intelligence (AI) programs to perceive specific objects in a class as subjects of interest. The systems sort objects in images into groups where objects with similar characteristics are placed together, while others are neglected unless programmed to do otherwise. For creating training set, the input image is divided into grid cells of size (nxn) cells, called anchor boxes. These cells then are assigned a 16 dimensional vector, based on the number of classes to be identified. The dimensions of the vector are as follows:

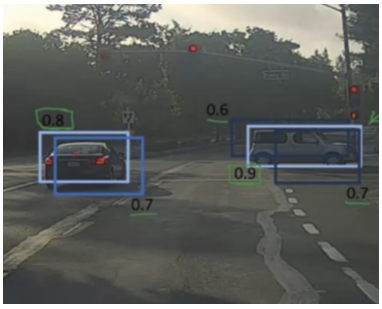
1. If there is an object or not in the first anchor box: 0 or 1? [Boolean]
2. Top left x coordinate of bounding box (Bx) [float]
3. Top left y coordinate of bounding box (By) [float]
4. Height of bounding box (Bh) [float]
5. Width of bounding box (Bw) [float]
6. Class 1 object: 0 or 1? [Boolean]
7. Class 2 object: 0 or 1? [Boolean]
8. Class 3 object: 0 or 1?... [Boolean]



(a) Anchor Boxes



(b) Intersection Over Union



(c) Non-Maximum Suppression

Figure 3: YOLO V3

9. If there is an object or not in the current bounding box?.
10. Top left x coordinate of bounding box (Bx)
11. Top left y coordinate of bounding box (By)
12. Height of bounding box (Bh)
13. Width of bounding box (Bw)
14. Class 1 object: 0 or 1?
15. Class 2 object: 0 or 1?
16. Class 3 object: 0 or 1?...

The above vector is computed using the CNN network shown in figureThe last 3 dimensions of this vector can vary according to the number of classes we will be be using to build the model. The first dimension identifies if an object is present in the anchor box, the second to fourth vectors identify the dimension of the anchor box having the object, fifth dimension onward, the vector identifies the which class does the object belong to. Once the object is identified, Intersection Over Union is used to get the exact bounding coordinates of the objects of interest in the image. intersection Over Union is the technique to combine all the anchor boxes where the object is present in and store the result. Then, Non-Maximum Suppression is applied to this result to form a rectangular bounding box over the object. As typical for object detectors, the features learned by the convolutional layers are passed onto a classifier which makes the detection prediction. In YOLO, the prediction is based on a convolutional layer that uses 1×1 convolutions. The algorithm YOLO is named "you only look once" because its prediction uses 1×1 convolutions; the size of the prediction map is exactly the size of the feature map before it. YOLO is a Convolutional Neural Network (CNN) for performing object detection in real-time. CNNs are classifier-based systems that can process input images as structured arrays of data and identify patterns between them (view image below). YOLO has the

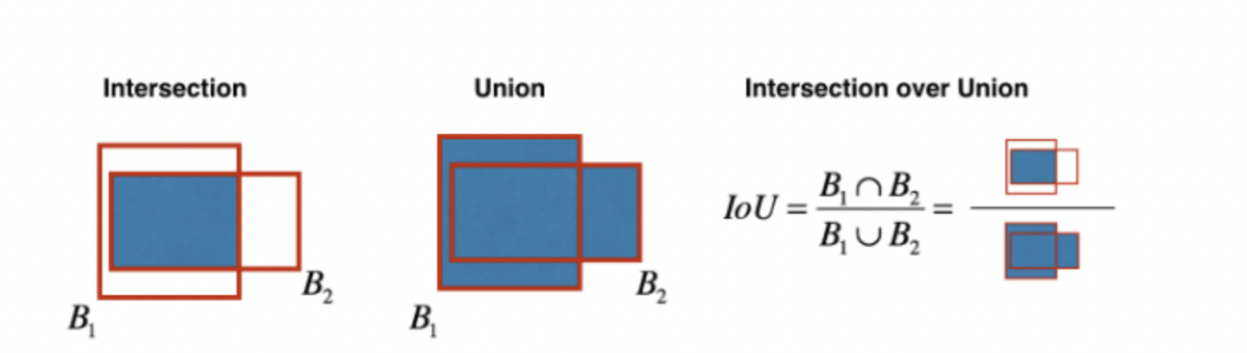


Figure 4: YOLO CNN Architecture

advantage of being much faster than other networks and still maintains accuracy. It allows the model to look at the whole image at test time, so its predictions are informed by the global context in the image. YOLO and other convolutional neural network algorithms "score" regions based on their similarities to predefined classes. High-scoring regions are noted as positive detections of whatever class they most closely identify with. For example, in a live feed of traffic, YOLO can be used to detect different kinds of people depending on which regions of the video score highly in comparison to predefined classes.

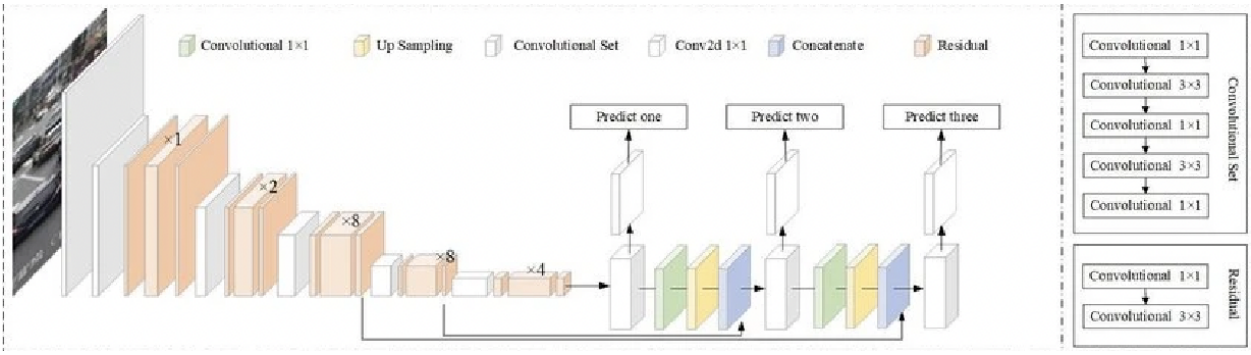


Figure 5: YOLO CNN Architecture

4.1.1 Advantages of YOLO V3

- Computationally efficient
- Uses convolutional neural networks
- 20x Faster, video processed at 65 fps
- No sliding window required
- Processes the image only once

4.2 Dead Reckoning for Active Tracking

Dead reckoning is a technique used in navigation to calculate current position of some moving object by using a previously determined position, or fix, and then incorporating estimates of speed, heading direction, and course over elapsed time. In this project the steps to perform dead reckoning involves:

1. Determining the number of steps taken by the pedestrian
2. Calculating linear velocity for every sample and step over elapsed time based on the accelerometer data
3. Calculating the linear distance covered with every step: the step length
4. Adding all the individual step distances to get the total distance covered during the elapsed time
5. Calculating angular displacement for every step over the elapsed time using the gyroscope data for every time step t (Also used Yaw for this)
6. Plotting the total linear distance along the change in angles for every sample and step over the elapsed time to get the movement trajectory

4.2.1 Determining number of Steps

For indoor pedestrian movement tracking, the data obtained from the phone IMU sensor can be susceptible to a lot of noise. This noise can be caused due to how the user is holding the phone, how much is he moving it while walking, the natural human gait might not always be consistent and can involve some noisy extra movement. Since IMU sensor is a very sensitive sensor in itself, these factors make it more prone to accumulating extra errors. Therefore, the approach to solve this problem, involves this method of determining the number of steps taken by an individual during the elapsed time, and get individual step lengths for all the steps, and eventually adding all these step lengths to calculate the final total distance walked by the pedestrian. The reason to do this stems from the method we are using to calculate the lengths and after that distances, which brings us to calculate the step-wise velocity and then step-length. The number of steps are determined by taking a threshold value in the gyroscope data along the axis which varies the most during walking, this can depend on positioning of the phone with respect to the ground and also the user. This threshold value represents a heel-strike, which can cause the gyroscope data to clearly capture it and eventually show patterns while movement. These patterns are shown in figure 5, in this example, the most movement pattern is shown along Y axes and it can be inferred that every sample peaking above 0.1G or below -0.1G represents the completion of one step, where G=9.8m/s2. Therefore, if we count these peaks, the number of steps should be close to 12.

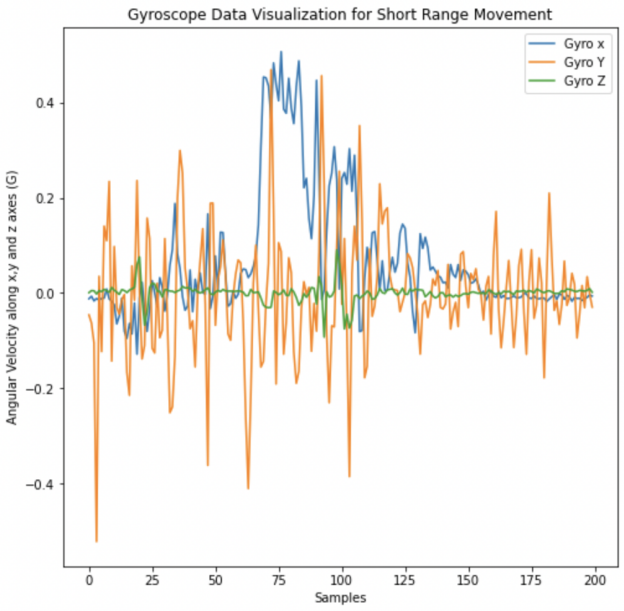


Figure 6: Gyroscope Data

4.2.2 Calculating Linear Velocity

Once we have divided all the raw samples into steps by calculating the number of steps and samples that represent the completion of those steps at timestep t. We can now go ahead to integrating all the linear acceleration samples obtained from the accelerometer between two consecutive steps and get the velocity at all the samples between two consecutive steps. The velocity at the end of each step is reset to 0, which represents the stance position in the human gait while walking, according to [2], this reduces the possibility of accumulating error. The method used to integrate the acceleration values is the trapezoidal rule which is depicted in equation: The velocity of each

$$v_k = v_{k-1} + \frac{a_{k-1} + a_k}{2} \Delta t$$

sample between a step is obtained by taking the velocity of the last sample at timestep t, initially 0, and adding it to the product of current sample's acceleration and elapsed time dt (difference in time between the two consecutive samples). Once, the sample-wise velocity is obtained, we can now go ahead to calculating the linear distance of each sample and eventually the whole step length.

4.2.3 Calculating Linear Distance

The linear distance is calculated by double integration of the acceleration values obtained from the accelerometer, which means single integration of the velocity obtained in the previous section. To perform this integration, the velocity of each sample between two consecutive steps is integrated, which is the distance covered at each sample during elapsed time dt at time t, added to the distance

covered at time (t-1). These cummulative sample distances as a whole make one step length and these step lengths then cummulatively added together gives us the total linear distance travelled by the pedestrian.

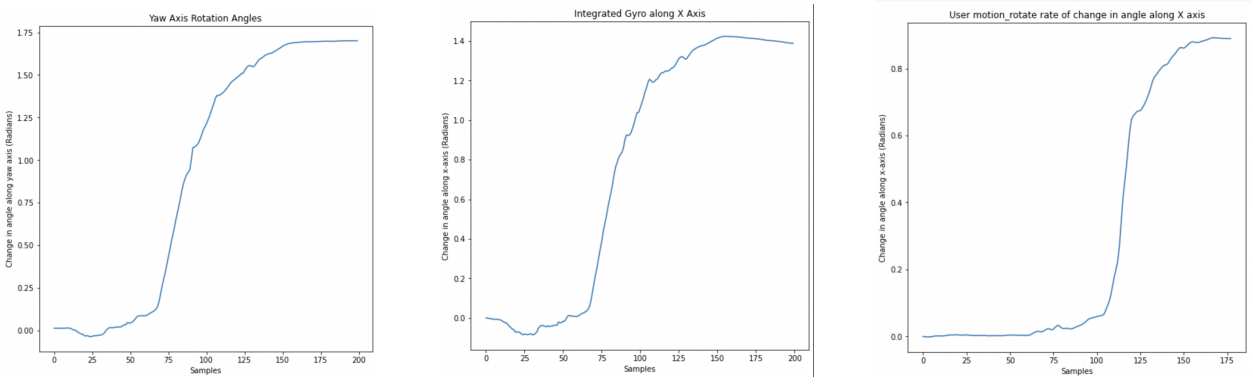
$$p_k = p_{k-1} + \frac{v_{k-1} + v_k}{2}\Delta t$$

4.2.4 Calculating Angular Displacement

Angular Displacement is calculated using the gyroscope, which gives us angular velocity of each sample at time t. We can use this data to calculate the angular displacement for every sample at timestep t, by single integration of the gyroscope samples along the relevant rotation axis. The integrated gyroscope data here, will give us change in angle for every sample during elapsed time dt at timesetep t. The change in angular orientation can also be obtained from the yaw data, which is also the change in angle measured along the yaw axis of the phone. The yaw angles are calculated using the accelerometer, gyroscope as well as the measurements from the magnetometer. According to how the phone is positioned with respect to the user, it is crucial to know that along which axis, will the phone's sensor be experiencing rotation while moving.

4.3 Plotting angular displacement with Linear Displacement: Trajectory

For this, we take the euclidean distance in the plane where the movement takes place, and plot this distance along the rotation axis. This is done by taking a distance between two consecutive points of samples or steps and plotting this distance at the changed angle corresponding to this sample or step. This is achieved by taking the starting point as the first sample and end point is computed by finding the cos and sin components of this point with the corresponding angle. These cos and sin components correspond to end (X,Y) coordinates for the current sample or step. When looped



(a) Yaw Angles (b) Integrated Gyro along X (c) User Motion Rotate X

Figure 7: Angular Orientation

over all the samples, it updates the start and end coordinates for each two successive samples till the end sample is reached. This trajectory is then plotted using matplotlib in python.

1. endy = y + length * sin(angle)
2. endx = x + length * cos(angle)

5 Experiments and Implementation

5.1 Passive: Visual Tracking

5.1.1 Experiment Setup

The visual tracking was performed in a home setting depicted in figure 8. The person I considered for tracking was myself. The video of me walking on a plane surface is the input to the YOLOv3 network.

5.1.2 Implementation

The YOLOv3 object detector pretrained model files on the coco dataset are taken, these were trained by Darknet. This experiment takes in video files as input and gives the processed video in real-time using YOLO, with annotated bounding boxes around the person and class name: person as the output video. Once, the model files and input videos are ready, all the labels of the trained classes are loaded and random colours are assigned to each one of them, however, here we are only concerned with the class 'Person', and the ID number of this class in the database would be 1, so only this class will be detected for this implementation. Then, the YOLO weights and configuration files are derived followed by loading YOLO from the disk. This is done by using OpenCV's DNN function called cv2.dnn.readNetFromDarknet. This function requires both a Config file path as well as the weights file path. Once YOLO, weights and config files are loaded and ready, now a file pointer to the video file is opened for reading frames in a loop, along with inititializing the video writer and frame dimensions. I then try to determine the number of frames in the video file, so that the time taken to be processing the video can be determined. Once this is done, the video frames are ready to processed one by one. The first frame is grabbed and subsequently others are looped over to be grabbed until the last frame, along with frame dimensions. The next step after this is to perform a forawrd pass of YOLO, using the current frame as input. For this, a blob is constructed and passed through the network, obtaining predictions. The forward pass operation is surrounded with time stamps so the elapsed time to make predictions on one frame can be calculated, and hence, the entire video. Three list data structures are initialized here: 1. boxes: the bounding boxes over the object, 2: confidences: The confidence value that YOLO assigns to an object. Lower confidence values indicate that the object might not be what the network thinks it is, 3: classIDs : The detected object's class label, which is going to be 1(Person) always for this project. Then these lists are populated with data from the YOLO output layers. To do this, the output layers and detections are looped over and the class IDs and confidence is extracted. The confidence is then used to filter out the weak detections, these confidences are obtained from Intersection Over Union step after detections. Now that the unwanted detections are filtered out, the bounding box coordinates are scaled so that they can be applied properly on the original video frames one by one. After scaling, the bounding box coordinates and dimensions are extracted in the form: (centerX, centerY, width, and height) and this information is used to derive the top-left (x, y)-coordinates of the bounding box, and these are added to the 'boxes' list. This is how the lists are populated. Now that we

have all the data in lists, this is used to apply Non-Maximum Suppression. YOLO does not apply NMS explicitly, so it is applied externally using OpenCV's built-in DNN module implementation of NMS. Applying non-maxima suppression suppresses significantly overlapping bounding boxes, keeping only the most confident ones. NMS also ensures that there are no redundant or extraneous bounding boxes. Then, looping over idxs obtained, the final bounding boxes from NMS are drawn with text on the video frames using random class colors (yellow for Person here). To wrap up, the video writer is now initialized on the first iteration of the loop over frames, print out the estimated time to be taken to process the video, write the frames to the output video file, cleanup and release the pointers. The visual tracking of these bounding boxes is done by recording the centre coordinates of these final bounding boxes in a list of lists(containing the coordinates). The coordinate points are then looped over to take two consecutive points progressively from the first till the last point, and draw a line between them using the Line Contour in OpenCV. This gives the moving trajectory of the box, and hence of the person in the video. The video file with the output is saved in the same directory and can be accessed to look at the results.

5.1.3 Results

The output video gives the trajectory of the pedestrian moving in the input video as well as the bounding box around the pedestrian which determines his position in every frame over elapsed time. This is visually depicted in figure 7.

Figure 8: YOLO Visual Tracking

5.2 Active: IMU Sensor Tracking

5.2.1 Experiment 1: Moving the Phone on the Table

This experiment was a simulation of long range movement and orientation while walking represented as a short range movement of the phone on the table in the same orientation. This was performed to validate the methodology to derive the dead reckoning solution. This experiment was limited to short range movement of the phone to gather least noise and accumulate minimum errors, and an attempt to get as close to the actual movement trajectory as possible.

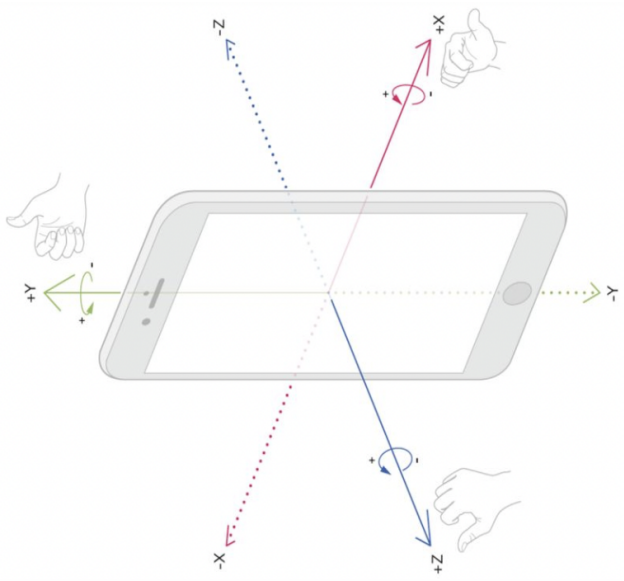


Figure 9: Axes of IMU Sensor of the Phone

5.2.2 Experiment Setup

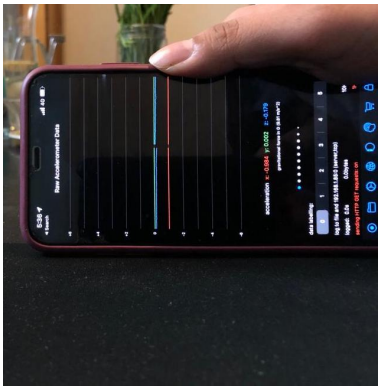
The phone was held in the position as shown in figure 10(a),(b) and (c), shown from different angles. The IMU sensor axes of the phone is shown in figure 9. The phone was moved straight and turned by an angle of ninety degrees. The IMU data of this movement was stored in the phone as a CSV file by using the Sensorlog App on iOS. The data which was of main focus was from the accelerometer, accelerometer timestamp, gyroscope, yaw orientation angles and User Motion Rotation Rate along X-axis. The format of data captured by these components is as follows:

- Accelerometer gives linear acceleration along X,Y and Z axes of the IMU sensor of the phone (Accel X, Accel Y, Accel Z). This data is used to compute the linear velocity and linear distance along each of these axes by double integrating this data individually with respect to change in time (dt)

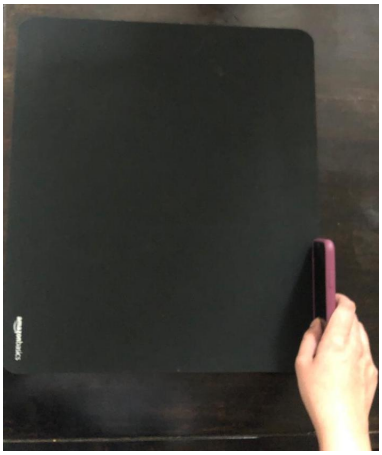
- Gyroscope gives angular velocity along X,Y and Z axes (Gyro X, Gyro Y, Gyro Z). This data is used to compute angular displacement along each of these axes by single integrating it with respect to change in time. The gyroscope data is also used to determine the number of steps using the threshold value which represents a heel strike and hence the step.

- The accelerometer timestamp gives the time elapsed with every sample, the sampling frequency of this data is 30 samples per second. This data is used to determine dt (change in time with every sample which is always close to 0.3 seconds for every sample, given the sampling frequency). This dt is used in integration of the accelerometer and gyroscope data, as explained the previous two points.

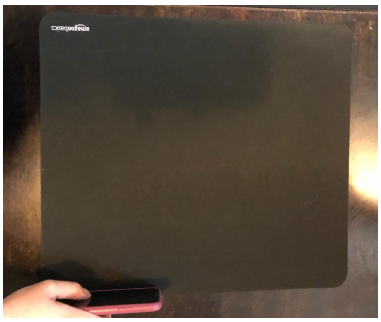
- Yaw Orientation Angles data are used to get the change in angular orientation at timestep t i.e. during time dt for every sample. Yaw angle computation is based on the accelerometer,



(a) Phone Orientation - 1



(b) Phone Orientation - 2



(c) Phone Orientation - 3

Figure 10: Figure 9(a) shows closely the readings of the IMU sensor when the phone is at rest, 9(b) shows the starting position of the phone from the vertical persepctive and 9(c) shows the starting position of the phone from the horizontal perspective

gyroscope and magnetometer data. The yaw angles are measured with respect to initial orientation of the phone, which is 0 radians.

- This data also corresponds to change angles at timestep t, for individual axes X,Y and Z.

In figure 7(a), since the phone's IMU sensor's negative X-Axis is pointing towards the ground. we can see that the linear acceleration along X axis is negative 0.98 (G), which corresponds to acceleration due to gravity along negative X axis of the phone. To calibrate this data according to the actual movement of the phone, this acceleration is is brought up to its actual linear acceleration along by adding 0.98 G to this, and bringing this acceleration to a net value of 0 G. This helps to reduce any errors which could be caused due to acceleration due to gravity. Once this step is achieved, it can then be used for integration and next steps.

5.2.3 Implementation

Once the experiment is setup and data from the CSV file of the app is ready for further mathematical computations and functions, the next step is to detect the number of steps (this is to induce similarity in the simulation with the actual movement and to perform computations in the similar manner) using a threshold value for the gyroscope data, it is taken to 0.1 (Radian/sec) along X axis, because the phone is rotated along the X axis. After steps, step lengths are calculated for each step by performing double integration of the acceleromenter data followed by adding these individual step lengths to get the total distance travelled along each axis. Since the phone is translating here in Y-Z plane, the euclidean distance is computed in this plane using the following equation... to compute the distance travelled in this plane. This is the linear distance, now this linear distance needs to be oriented accoring to angular displacement for every sample or every step. The angular displacement is plotted with linear displacement to get the final trajectory. The angles are plotted

in three using the three sources of getting angles explained earlier. These angle plots are also smoothed out using the PolyFit function in the scipy library in python as an attempt to get the trajectory with least amount of noise caused by the angles.

5.2.4 Results

The step-wise distances are shown in figure 10.

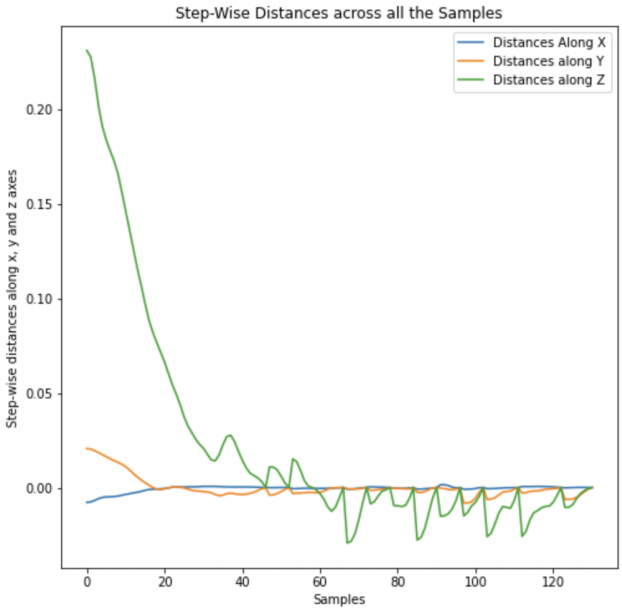


Figure 11: Step-wise Distances along X-Y-Z Axes

This figure shows the lengths covered by each step and they are reset to zero in the stance position because the velocity is reset to zero at stance, as explained in the solution section. This shows that the most acceleration is for the z axis, because this is the axis which is translated significantly according to the movement shown in the experiments section.

This is followed by the movement detected in Y axis, which points the straight direction of the phone, and since the phone is moving in a straight direction itself the whole time, it does not show much movement, but still this axis has to be considered while taking distance as it lies in the plane of movement. There is no translational movement in x axis since we are just rotating the phone along this axis.

The total distances covered along all 3 axes are shown in Figure 11. The Z Axis shows the most distance travelled because the Z axis is getting translated significantly in a straight line movement. The Y axis shows the second highest distance travelled since while taking the turn and moving the phone, the Y axis also experiences translation, and this denotes and validates that the phone is moving in a Y-Z plane. The huge jump of the Z axis in the starting is because of the distance which covered in the first set of samples when the phone just started to move. The X axis shows the least covered distance, since it is the rotation axis, and this will only show any acceleration results only when the phone is rotated, therefore even the slightest movement shown here is noise

and drift if it is along the X axis.

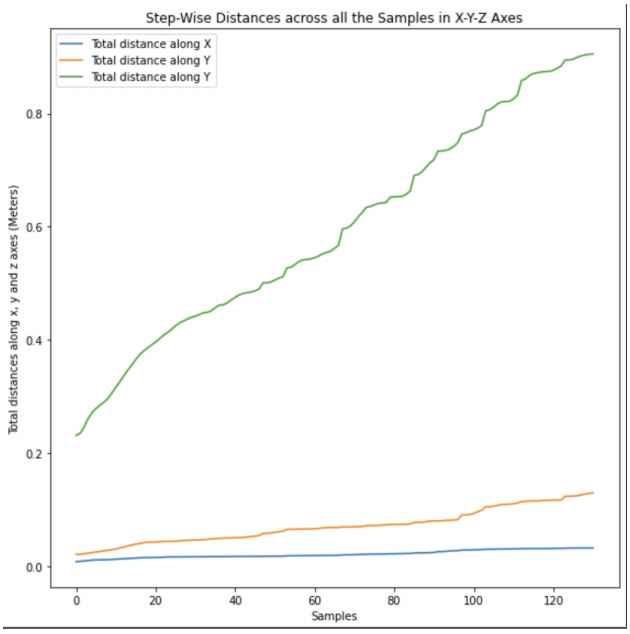


Figure 12: Total Distances along X-Y-Z Axes

The total Euclidean Distance covered in Y-Z plane are shown in Figure 12. The euclidean distance in a plane is calculated by taking the square root of the sum of squares of differences of two consecutive coordinates of Z and Y axes. This is shown in the equation. The euclidean distance is taken here because, as previously inferred, the phone experiences translation in the Y-Z plane. This euclidean distance gives us the final total distance covered while moving the phone in the specific way and orientation. The euclidean distance here, is almost close to the distance travelled along Z Axis since the most translation is along the Z axis as shown in figure 10. The distance calculated here is around 0.9 meters which is close to the ground truth perimeter of the moving trajectory of the phone. This validates the method.

Once the total distances were computed, the next step was to plot this distance along the change in angles at elapsed times. This plot gives us the direction with distance, which is the actual movement trajectory of translation in Y-Z plane and angular displacement (rotation) along X-Axis.

The figure shows the ground truth movement compared with the trajectory obtained while plotting the angles with total distance i.e. Direction with Distance. The trajectory here seems close to the actual movement. Figure 13 shows the ground truth movement compared with the trajectory obtained after plotting the total distance covered with the change in angles got from integrating the gyroscope data along X axis. This trajectory, however shows some drift from the actual path and we observe that the turn is assumed to be taken much before the the actual turn was taken. The reason for this could be accumulated noise and errors in the accelerometer and gyroscope data which could have led to extending the trajectory beyond the actual point. When the accelerometer data is double integrated, not just the data itself is getting integrated but also

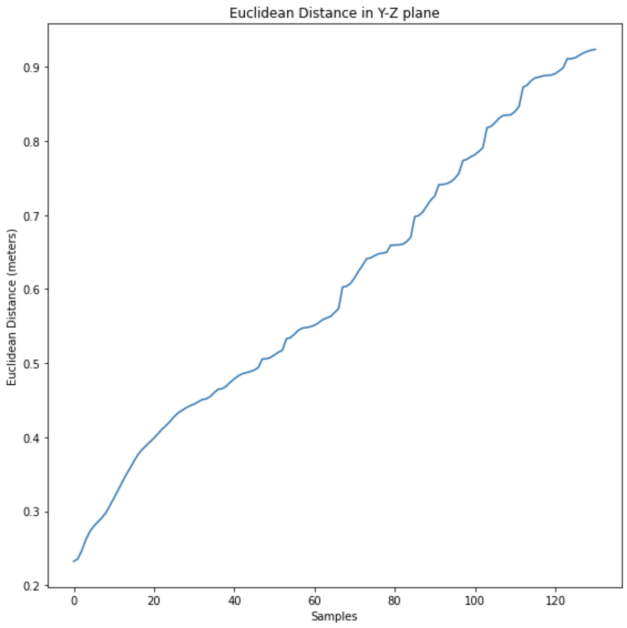
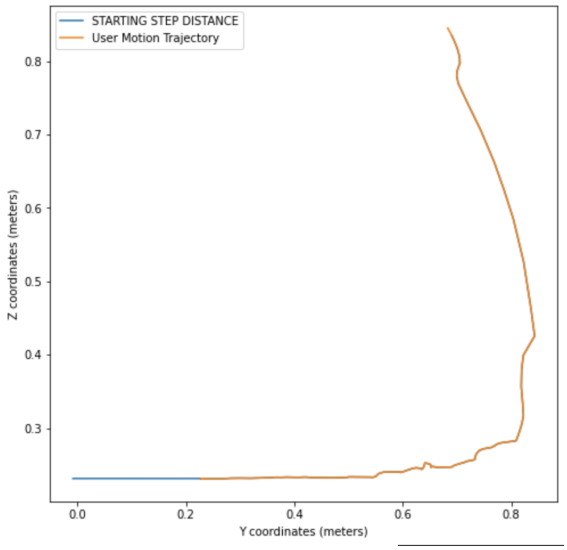
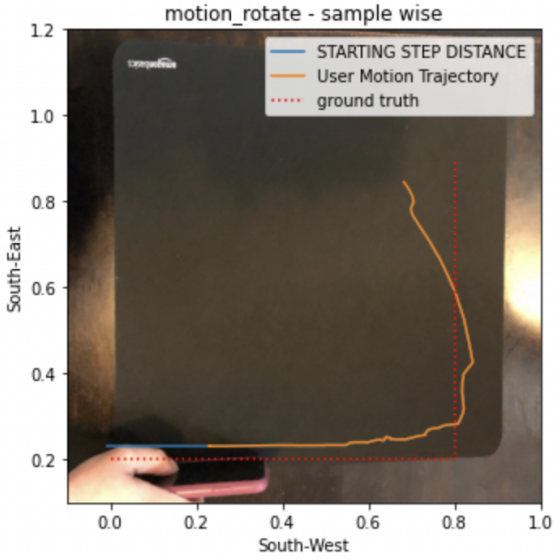


Figure 13: Euclidean Distance in Y-Z plane

the noise present, which results in adding some extra values which are unwanted while getting the correct trajectory. There are several ways to hone this and will be tried and tested in the future extension of this project. Next, there is a plot of the trajectory with the yaw angles which also shows similar drift as shown in figure 15. As we look closely at the angle plots in figure 6, the yaw angles and gyro integrated gyro take the jump from 0 radians to approximately 1.5 radians (0 degree to 90 degrees), earlier than the MotionRotationRate along X Axis, due to which the turn is assumed to occur earlier too in the former plots. To make this trajectory cleaner, Figure 16 and 17 also depict the plots of step-wise orientation plots and figure 17 and 18 smoothed angle curve plots and plotted angles with distance for them respectively. This was an attempt to make the trajectory smoother and more defined, which was achieved, however the drift still has to be solved for and honed. The possible ways to do this are discussed in the subsequent sections. Due to space constraints, all the remaining results are shown in the Google Colaboratory Notebook itself.

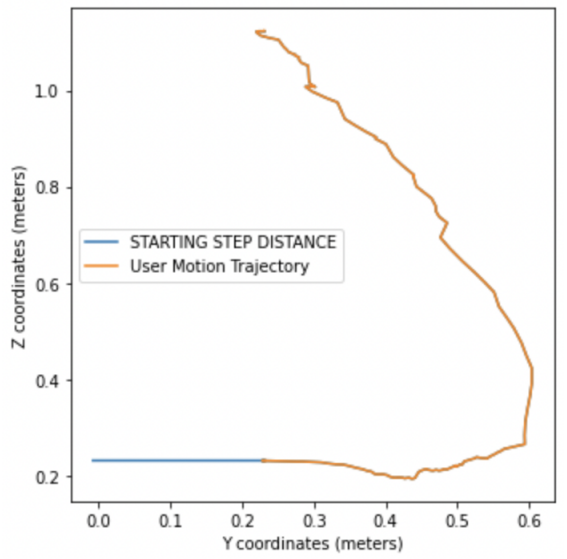


(a) MotionRotateRate X

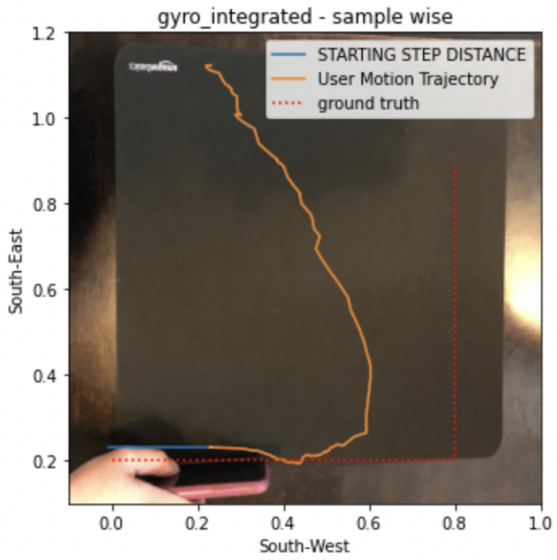


(b) Movement wrt GT

Figure 14: Distance with Direction: Uisng MotionRotationRateX Angles Data

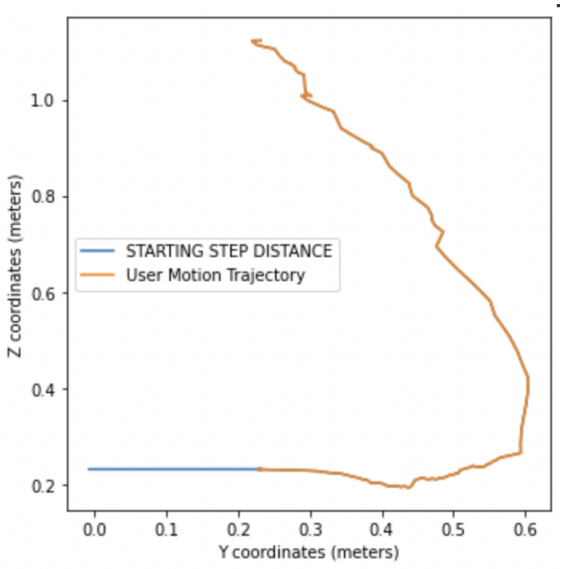


(a) Gyro Integrated along X

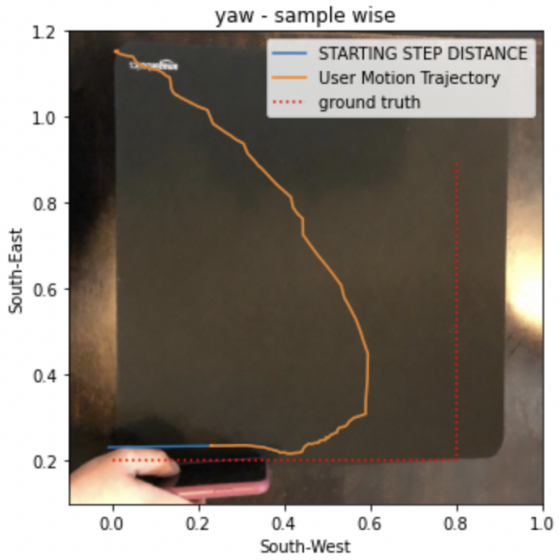


(b) Gyro Integrated along X wrt GT

Figure 15: Distance with Direction: Uisng Gyro Integrated Angles Data

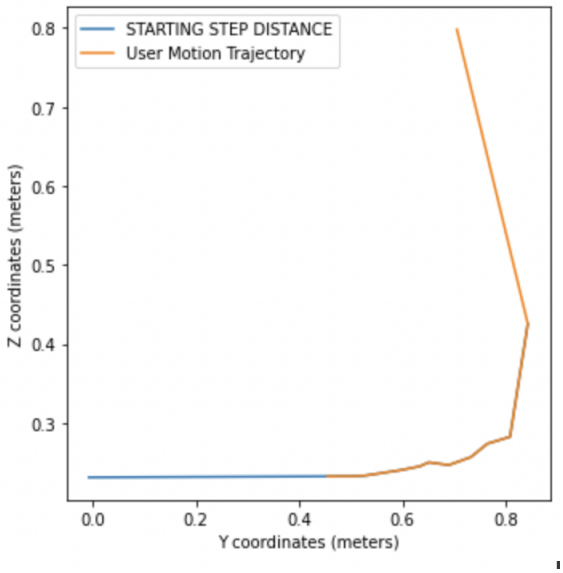


(a) Gyro Integrated along X

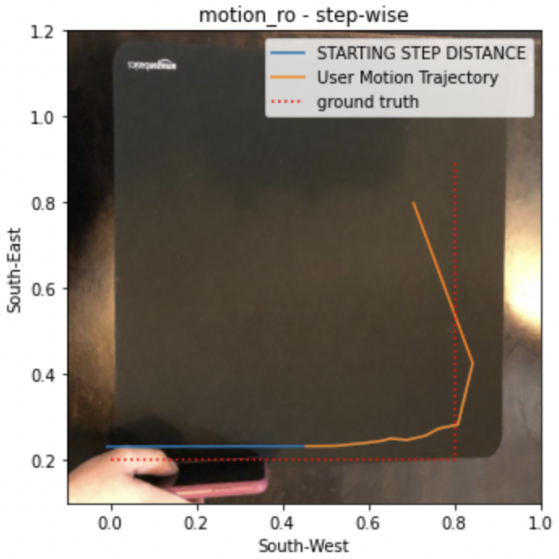


(b) Gyro Integrated along X wrt GT

Figure 16: Distance with Direction: Uisng Yaw Angles Data

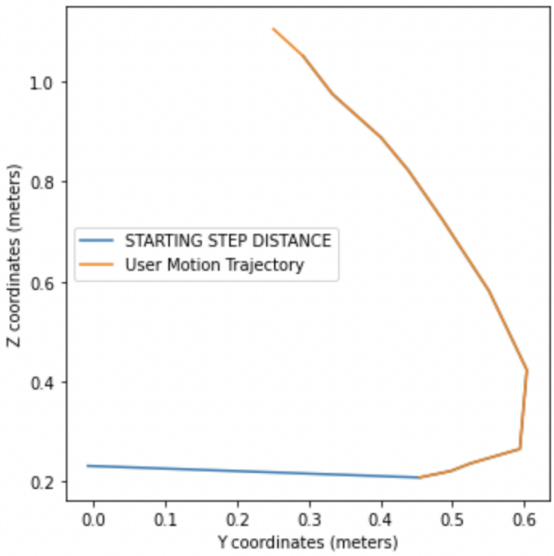


(a) MotionRotateRate along X

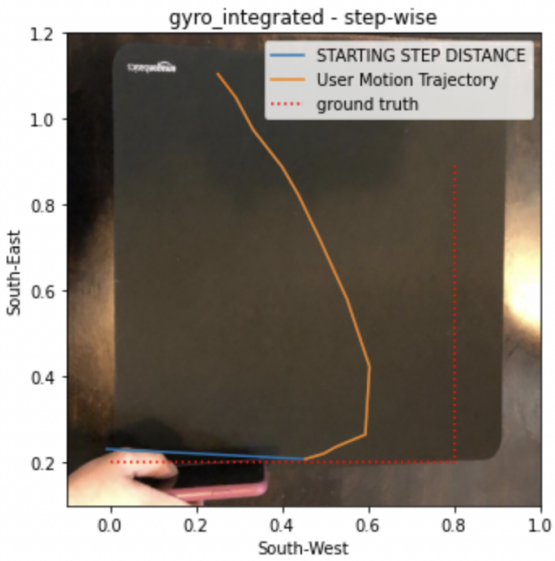


(b) MotionRotateRate along X wrt GT

Figure 17: Distance with Direction: Uisng Step-Wise Samples

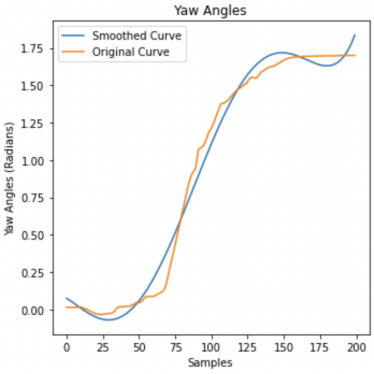


(a) Gyro Integrated along X

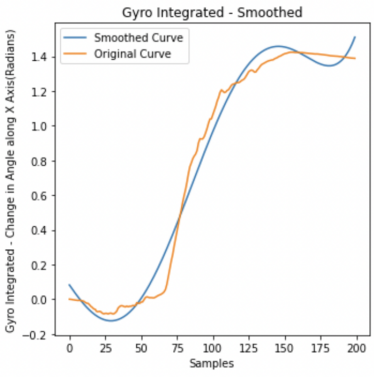


(b) Gyro Integrated along X wrt GT

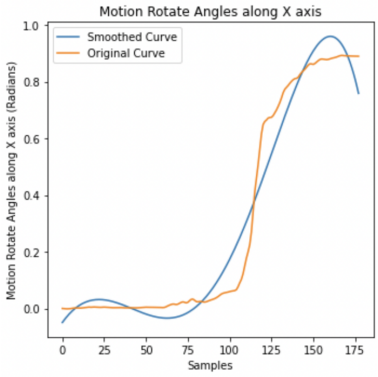
Figure 18: Distance with Direction: Uisng Step-Wise Samples



(a) Yaw Angles - Smoothed

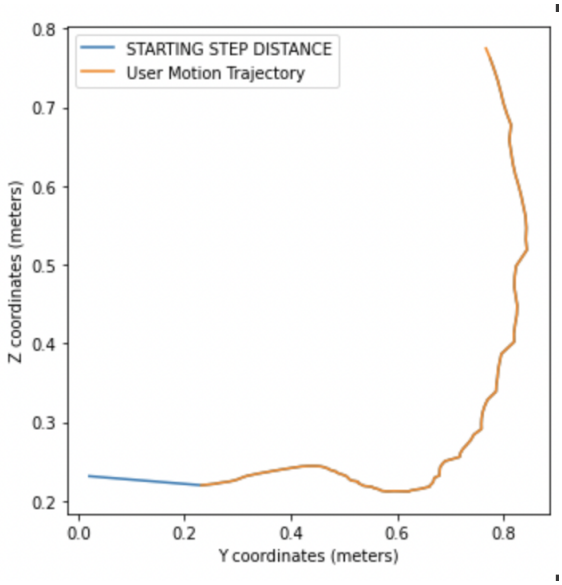


(b) Integrated Gyro along X - Smoothed

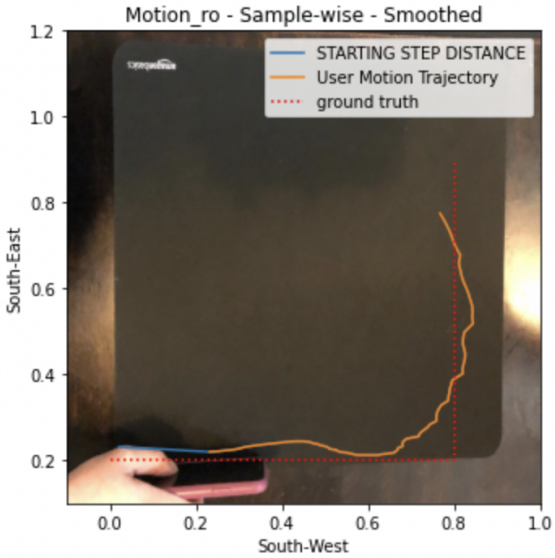


(c) User Motion Rotate X - Smoothed

Figure 19: Angular Orientation



(a) MotionRotateRate along X



(b) MotionRotateRate along X wrt GT

Figure 20: Distance with Direction: Uisng Angle Curve Smoothed Samples

5.2.5 Experiment 2: Pedestrian Walking

This experiment is the actual long range movement, which was performed with a pedestrian carrying a phone with IMU sensor in hand, oriented in the same way and along the same axes and coordinate system as experiment 1.

5.2.6 Experiment Setup

The setup, orientation, sensors involved to captured and everything is the same for this experiment except that now the phone is carried by a pedestrian and he performs a long range movement. The pedestrian moved and took a number of steps in the direction: straight-turn (90 degrees) - straight.

5.2.7 Implementation

Once the experiment is setup and data from the CSV file of the app is ready for further mathematical computations and functions, the next step is to detect the number of steps (this is to induce similarity in the simulation with the actual movement and to perform computations in the similar manner) using a threshold value for the gyroscope data, it is taken to 0.1 (Radian/sec) along X axis, because the phone is rotated along the X axis. After steps, step lengths are calculated for each step by performing double integration of the accelerometer data followed by adding these individual step lengths to get the total distance travelled along each axis. Since the phone is translating here in Y-Z plane, the euclidean distance is computed in this plane to compute the distance travelled in this plane. This is the linear distance, now this linear distance needs to be oriented according to angular displacement for every sample or every step. The angular displacement is plotted with linear displacement to get the final trajectory. The angles are plotted in three using the three sources of getting angles explained earlier. These angle plots are also smoothed out using the PolyFit function in the scipy library in python as an attempt to get the trajectory with least amount of noise caused by the angles.

5.2.8 Results

The step-wise distances are shown in figure 20.

This figure shows the lengths covered by each step and they are reset to zero in the stance position because the velocity is reset to zero at stance, as explained in the solution section. This shows that the most acceleration is for the z axis, because this is the axis which is translated significantly according to the movement shown in the experiments section.

This is followed by the movement detected in Y axis, which points the straight direction of the phone, and since the phone is moving in a straight direction itself the whole time, it does not show much movement, but still this axis has to be considered while taking distance as it lies in the plane of movement. There is no translational movement in x axis since we are only rotating the phone along this axis and not translating. However, since the movement is long-ranged, the possibility of accumulating noise due to the sensitivity to even the slightest movements of IMU sensor rises, and this is seen in the distance along Y and X axes. In the ideal case, there should be no movement in

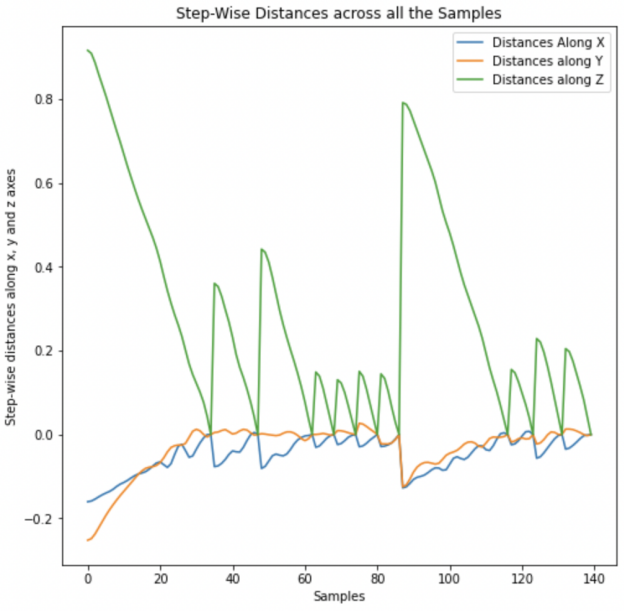


Figure 21: Step-wise Distances along X-Y-Z Axes

X axis, but due to accumulation of noise and then double integrated the unwanted values, it has overestimated the movements.

The total distances covered along all 3 axes are shown in Figure 21. The Z Axis shows the most distance travelled because the Z axis is getting translated significantly in a straight line movement. The Y axis shows the second highest distance travelled since while taking the turn and moving the phone, the Y axis also experiences translation, and this denotes and validates that the phone is moving in a Y-Z plane. The huge jump of the Z axis in the starting is because of the distance which covered in the first set of samples when the phone just started to move. The X axis shows the least covered distance, since it is the rotation axis, and this will only show any acceleration results only when the phone is rotated, therefore even the slightest movement shown here is noise and drift if it is along the X axis.

The total Euclidean Distance covered in Y-Z plane are shown in Figure 23. The euclidean distance in a plane is calculated by taking the square root of the sum of squares of differences of two consecutive coordinates of Z and Y axes. This is shown in the equation. The euclidean distance is taken here because, as previously inferred, the phone experiences translation in the Y-Z plane. This euclidean distance gives us the final total distance covered while moving the phone in the specific way and orientation. The euclidean distance here, is almost close to the distance travelled along Z Axis since the most translation is along the Z axis as shown in figure 10. Once the total distances were computed, the next step was to plot this distance along the change in angles at elapsed times. This plot gives us the direction with distance, which is the actual movement trajectory of translation in Y-Z plane and angular displacement (rotation) along X-Axis. The trajectory of the pedestrian while walking starts from the distance covered in the first step highlighted in blue and goes up till the end of orange line plot. Since this is a long range

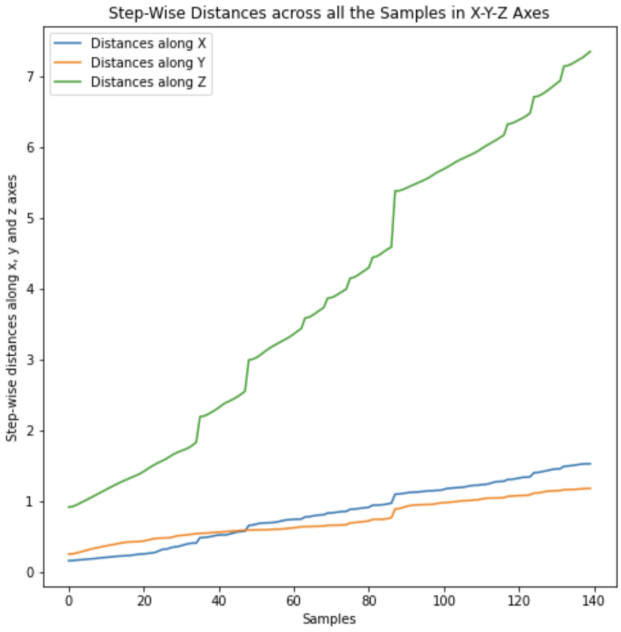


Figure 22: Total Distances along X-Y-Z Axes

movement, it accumulates more errors till the end of time of the movement, which again results in underestimating the turning point from the actual Ground Truth turning highlighted in red in Figure 24 (b). There is also some overestimation of the curved path, which was supposed to be almost a straight line, but due to accumulation of errors in the end of the trajectory and towards the end of the steps, it led to this drift. However, the noise in trajectory seen in Figure 24 has been corrected by smoothing the angles using the PolyFit curve function in SciPy, shown in figure 25. These are the results for gyroscope integrated data (normal and smoothed) and the rest of the results and components discussed in experiment 1 are shown in the google colaboratory file of experiment 2 due to the space consraints. The google colab file for the third experiment is also shared.

$$d = \sqrt{|y_2 - y_1|^2 + |x_2 - x_1|^2}$$

Figure 23: Euclidean Distance Formula: where x corresponds to y and y corresponds to z, in the Y-Z plane

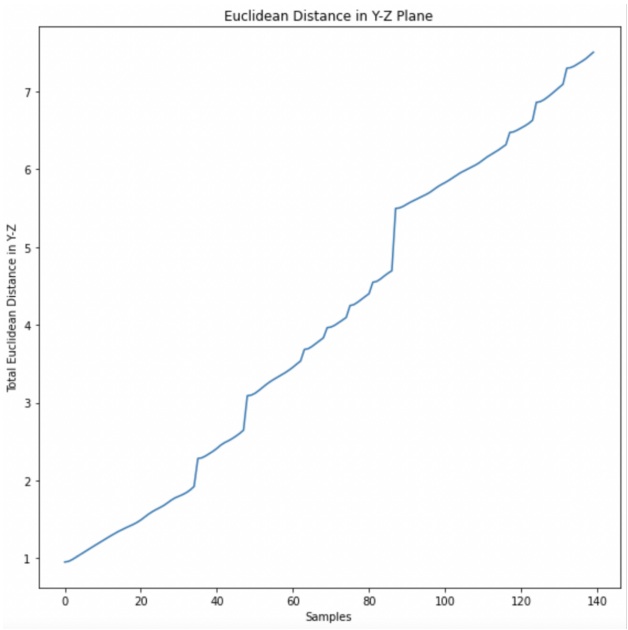
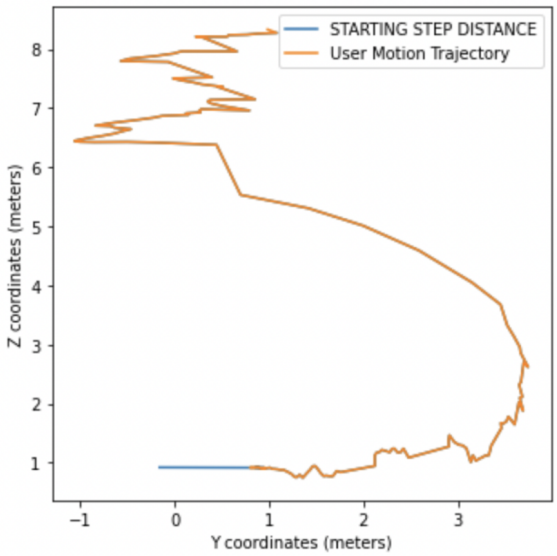
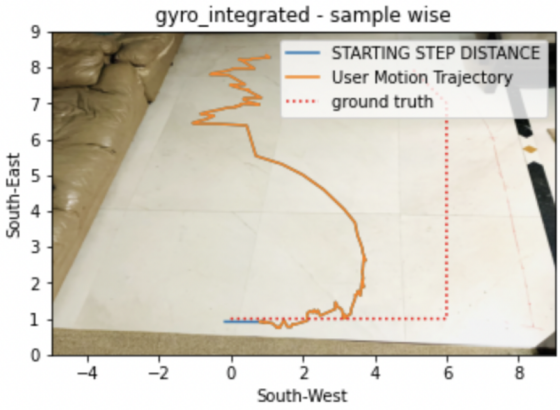


Figure 24: Euclidean Distance in Y-Z plane

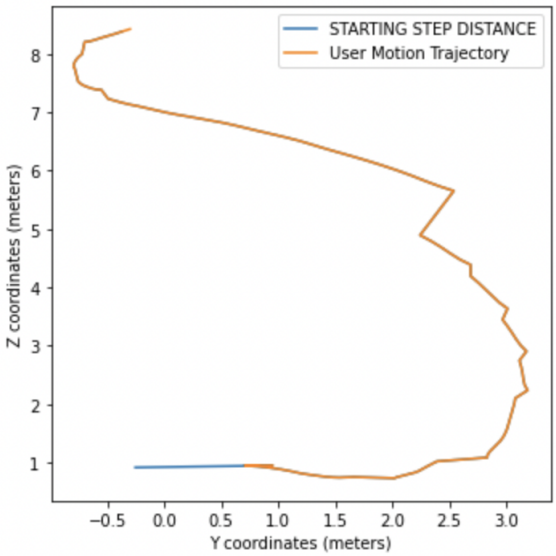


(a) Gyro Integrated along X

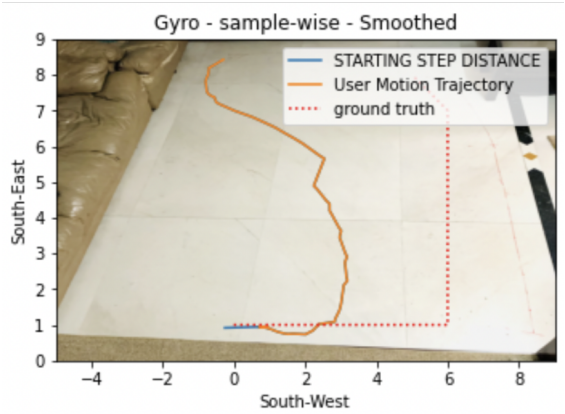


(b) Gyro Integrated along X wrt GT

Figure 25: Distance with Direction: Using Gyro Integrated Angles Data



(a) Gyro Integrated along X trajectory



(b) Gyro Integrated trajectory wrt GT

Figure 26: Distance with Direction: Smoothed Integrated Gyro Curve Angles

6 Comparative Analysis of all Results

For Experiment 1, 2 and 3 respectively:

Number Of Steps	GroundTruth Distance	Number of Steps Detected	Distance Calculated
NA	0.95m	13	0.92381m
10	5.2m	12	7.50630m
10	4m	13	3.8181m

For the second and third experiments, although the same number of steps were taken, the step lengths for each of these and the velocity with which the pedestrian was walking were different. Experiment 2 had an average stride of length 0.5 and for second, it was 0.3m approximately, which is also seen in the distance covered column.

7 Future Work

The future work and extension of the current work from here includes firstly, the reduction of drift in the trajectory obtained by plotting distance with direction. This drift reduction can be achieved by trying out the following techniques:

1. Filtering out acceleration values while integratiom which are not useful and causes unwanted noise
2. EKF for dead reackoning []
3. IMU based drift reduction methods: Altitude Correction []
4. IMU based drift reduction methods: Magnetometer heading Correction []
5. Step Length Calculation optimization using Discrete Fourier Transform, converting frequency domain to time domain and get the principle frequencies corresponding to the steps (Result in figure ..)

The integration of visual trajectory shown in sections 5.1 and active tracking using IMU sensor in section 5.2 can begin once this trajectory is optimal and as close to the actual trajectory as possible. The steps to be followed for this integration are as follows:

1. Scale, translate and rotate the active tracking trajectory using the sensor with the frames of the surveillance feed video used to track the pedestrian
2. Complement the visual and active trajectory and make them work together, i.e. although both work together, the active tracking results should take over the visual tracking once the person has been occluded and visual tracking stops
3. This can be achieved by updating the bounding box centroids with the dead reckoning trajectory coordinates to create the bounding box while occlusion

8 Conclusion

All the conducted experiments show that the task that was set out to be achieved, is optimal approach to address the problem. Once the drift in the dead reckoning results is handled, it can be efficiently integrated with Visual Tracking. It was also observed that for long-ranged data, it is feasible and efficient to use the PolyFit Function in order to smooth the angle curve and then plot with the distance, which efficiently reduces the noise in the trajectory towards the end of elapsed time. However, the accumulated noise which leads to accumulated overestimation of the curves, angles and distances can be solved by following one of the steps given in the Future Work section. The YOLOv3 results are also promising, since it gives the exact bounding boxes around the person in the surveillance feed and also tracks him or her over the elapsed time. These bounding boxes will be constructed by taking the dead reckoning coordinates as the mid point when the pedestrian will be occluded.

9 References

1 Combining Passive Visual Cameras and Active IMU Sensors for Persistent Pedestrian Tracking - Wenchao Jiang and Zhaozheng Yin - Missouri University of Science and Technology, USA

2 Indoor Location Estimation using a Pair of Wearable Devices - Anastasios Petropoulos and Theodore A. Antonakopoulos - Department of Electrical and Computer Engineering University of Patras, Patras 26504, Greece

3 Comparison of Drift Reduction Methods for Pedestrian Dead Reckoning Based on a Shoe-Mounted IMU - Woo Chang Jung and Jung Keun Lee+

4 https://www.youtube.com/watch?v=8jJZRG9wx2ct=1575s

5 M. Danelljan, G. H ager, F. S. Khan, M. Felsberg, Adaptive decontami- nation of the training set: A unified formulation for discriminative visual tracking, in: Computer Vision and Pattern Recognition, 2016.

6 F. Xiao, Y. J. Lee, Track and segment: An iterative unsupervised ap- proach for video object proposals, in: Computer Vision and Pattern Recognition, 2016.

7 A. Leykin, R. Hammoud, Robust multi-pedestrian tracking in thermal- visible surveillance videos, in: IEEE Conference on Computer Vision and Pattern Recognition Workshop, 2006, pp. 136–136.

8 P. Dollar, C. Wojek, B. Schiele, P. Perona, Pedestrian detection: An evaluation of the state of the art, IEEE transactions on pattern analysis and machine intelligence 34 (4) (2012) 743–761. 28

9 M. Enzweiler, D. M. Gavrila, Monocular pedestrian detection: Survey and experiments, IEEE transactions on pattern analysis and machine intelligence 31 (12) (2009) 2179–2195.

10 D. Geronimo, A. M. Lopez, A. D. Sappa, T. Graf, Survey of pedestrian detection for advanced driver assistance systems, IEEE transactions on pattern analysis and machine intelligence 32 (7) (2010) 1239–1258.

11 A. Roy, P. Chattopadhyay, S. Sural, J. Mukherjee, G. Rigoll, Modelling, synthesis and characterisation of occlusion in videos, IET Computer Vision 9 (6) (2015) 821–830.

12 W. Nie, A. Liu, Y. Su, H. Luan, Z. Yang, L. Cao, R. Ji, Single/cross- camera multiple-person tracking by graph matching, Neurocomputing 139 (2014) 220–232.

13 B. Wu, R. Nevatia, Detection and tracking of multiple, partially oc- cluded humans by bayesian combination of edgelet based part detectors, International Journal of Computer Vision 75 (2) (2007) 247–266.

13 D. Merad, K.-E. Aziz, R. Iguernaissi, B. Fertil, P. Drap, Tracking mul- tiple persons under partial and global occlusions: Application to cus- tomers behavior analysis, Pattern Recognition Letters, 2016.

14 Z. Li, Q. L. Tang, N. Sang, Improved mean shift algorithm for occlusion pedestrian tracking, Electronics Letters 44 (10) (2008) 622–623.

15 S. Zhang, J. Wang, Z. Wang, Y. Gong, Y. Liu, Multi-target tracking by learning local-to-global trajectory models, Pattern Recognition 48 (2) (2015) 580–590.

16 J. Sherrah, Occluded pedestrian tracking using body-part tracklets, in: International Conference on Digital Image Computing: Techniques and Applications, 2010, pp. 314–319.

17 http://www8.garmin.com/aboutGPS/.

18 H. Wang, H. Lenz, A. Szabo, J. Bamberger, U. D. Hanebeck, Wlan- based pedestrian tracking using particle filters and low-cost mems sen- sors, in: Workshop on Positioning, Navigation and Communication, 2007, pp. 1–7. 29

19 S. Woo, S. Jeong, E. Mok, L. Xia, C. Choi, M. Pyeon, J. Heo, Ap- plication of wifi-based indoor positioning system for labor tracking at construction sites: A case study in guangzhou mtr, Automation in Con- struction 20 (1) (2011) 3–13.

20 P. Falcone, F. Colone, A. Macera, P. Lombardo, Localization and track- ing of moving targets with wifi-based passive radar, in: IEEE Radar Conference, 2012, pp. 0705–0709.

21 J.-P. Dubois, J. S. Daba, M. Nader, C. El Ferkh, Gsm position track- ing using a kalman filter, Wolrd Academy of Science, Engineering and Technology 68 (2012) 1610–1619.