دانشکده فنی و مهندسی
گروه مهندسی برق

پایان نامه برای اخذ مدرک کارشناسی مهندسی برق

سیستم ردیابی داخلی عابر به روش ناوبری کور

دانشجو:
آراد آرنگ

استاد راهنما:
دکتر مهراد بابازاده

[بهمن 1402]

گزارش دفاع از پایان‌نامه کارشناسی

به حول و قوه الهی پایان نامه آقای/ خانم **آراد آرنگ** به شماره دانشجویی 96442105 رشته مهندسی برق با عنوان **سیستم ردیابی داخلی عابر به روش ناوبری کور** در تاریخ 1402/11/16 دفاع و با نمره ..... ارزیابی گردید.

اعضای هیات داوری پایان نامه

استاد راهنما: دکتر مهراد بابازاده	امضاء
داور : دکتر رضا امیدی	امضاء
داور : دکتر اباذر عرب عامری	امضاء
مدیر گروه: دکتر فرهاد بیات	امضاء

باسمه تعالی

تعهد نامه

اینجانب آراد آرنگ متعهد می‌شوم که مطالب مندرج در این پایان نامه حاصل کار پژوهشی اینجانب تحت نظارت و راهنمایی اساتید گروه مهندسی صنایع دانشگاه زنجان بوده و به دستاوردهای دیگران که در این پژوهش از آن ها استفاده شده است مطابق مقررات و روال متعارف ارجاع و در فهرست منابع و مآخذ ذکر گردیده است. این پایان نامه قبلاً برای احراز هیچ مدرک هم‌سطح یا بالاتر ارایه نگردیده است. در صورت اثبات تخلف در هر زمان، مدرک تحصیلی صادر شده توسط دانشگاه از درجه اعتبار ساقط بوده و دانشگاه حق پیگیری قانونی خواهد داشت.

تمامی نتایج و حقوق حاصل از این پایان نامه متعلق به دانشگاه زنجان می‌باشد. هرگونه استفاده از نتایج علمی و عملی، واگذاری اطلاعات به دیگران یا چاپ و تکثیر، نسخه‌برداری، ترجمه و اقتباس از این پایان نامه بدون موافقت کتبی دانشگاه زنجان ممنوع است.

نقل مطالب با ذکر ماخذ بلامانع است.

آراد آرنگ

امضاء

چکیده

ناوبری کور عابر (PDR) یک جنبه حیاتی در سیستم‌های موقعیت‌یابی داخلی است که بر ادغام داده حسگرهای شتاب‌سنج برای تخمین موقعیت و جهت کاربر استوار است. در این مقاله، یک مطالعه مقایسه‌ای از دو رویکرد متفاوت به PDR ارائه می‌دهیم: یک راه‌حل مبتنی بر شبکه عصبی و یک روش به‌روزرسانی پتانسیل سرعت صفر (ZUPT).

رویکرد شبکه عصبی شامل استفاده از تکنیک‌های یادگیری عمیق برای پردازش داده حسگرهای شتاب خطی و دورانی (IMU) و حسگرهای مغناطیسی (MARG) است. شبکه این اطلاعات را به نمایش‌های موقعیت و چهارگان تبدیل می‌کند و سپس موقعیت نسبی نهایی را محاسبه می‌کند.

به‌عنوان مقابل، روش ZUPT از تکنیک‌های گیتینگ برای جدا کردن گام‌ها استفاده می‌کند، داده‌های شتاب‌سنج را در این بازه‌های سرعت صفر تجمیع می‌کند تا تغییرات موقعیت نسبی را تعیین کند. این روش به‌طور تلفیقی از سیستم‌های ارجاع ارتفاع و جهت (AHRS) برای ثبت اطلاعات جهت استفاده می‌کند و در نهایت مسیر و موقعیت نسبی را محاسبه می‌کند.

تحلیل ما میزان دقت، انعطاف‌پذیری و کارایی محاسباتی این دو رویکرد را ارزیابی می‌کند. یافته‌ها به تفاوت‌ها و مزایا و معایب هر روش می‌افزاید و مبنایی برای بهینه‌سازی راه‌حل‌های PDR در محیط‌های داخلی فراهم می‌کند. در نهایت، هدف این تحقیق بهبود درک تکنیک‌های PDR و هدایت توسعه سیستم‌های موقعیت‌یابی داخلی می‌باشد.

واژگان کلیدی:
Pedestrian Dead Reckoning (PDR), Attitude and Heading Reference System (AHRS), Zero Velocity Potential Update (ZUPT), Inertial Measurement Unit (IMU), Deep Neural Networks (DNN), Quaternions, Indoor Positioning System (IPS)

فهرست عناوین

فهرست شکل ها

فهرست جداول

فصل اول

کلیات پژوهش

1-1- مقدمه

یکی از چالش ها و معضلات سیستم های جهت یابی از دست دادن سیگنال جهت‌یابی یا کاهش کیفیت و در نتیجه خطای بالا آن در محیط حامل عوامل مخرب سیگنال از جمله سد فیزیکی و نویز الکترومغناطیسی می‌باشد. در نتیجه بازه‌های بلندی در این محیط‌ها وجود دارد که تخمین مکان به وسیله روش‌های ماهواره‌ای یا ایستگاهی غیرقابل اعتماد است.

اهمیت تخمین صحیح در شرایط بحرانی مانند حوادث طبیعی (حریق، آوار، طوفان و غیره) ، در محیطی صنعتی (با نویز الکترومغناطیسی بالا) یا پرجمعیت (بیمارستان‌ها) که در آن موقعیت یابی از سیستم های ماهواره‌ای مانند GPS امکان‌پذیز نیست، دوچندان می‌شود.

برای مرتفع کردن این مشکل می توان با استفاده از آخرین مکان دقیق و خروجی های شتاب خطی، دورانی و قطب نما و تکنیک های کاهش خطا از طریق هوش مصنوعی کیفیت و دقت این تخمین را در ادوات قابل حمل هوشند [مخصوص عابر] افزایش داد.

1-2- بیان مساله و اهداف

با توجه به افزایش تقاضا برای سیستم‌های مکان‌یابی قابل حمل (wearable) و افزایش ادغام سیستم‌های اینترنت اشیا در زندگی روزمره افراد و حضور سنسورهای یاد شده در اکثر تلفن‌های هوشمند، نتایج و روش‌های بدست آمده از تحقیق در این زمینه می‌تواند سیستم‌های مکان‌یابی محلی را به کالایی ارزان و قابل دسترس برای مصرف کنندگان تبدیل کند.

می‌توان از عواقب مثبت این دسترسی آسان موارد ذیل را یاد نمود:

- ردیابی و نظارت آنلاین بیماران با شرایط خاص در محیط‌های سربسته برای تسریع رسیدگی
- تسریع یافتن مصدومین زیر آوار یا در حریق (در نبود دید کافی یا دقت پایین GPS)
- ردیابی و نظارت ادوات صنعتی یا قیمتی با تعداد بالا در محیط‌های پویا و وسیع

فصل دوم

ادبیات پژوهش

2-1- مقدمه

تاکنون سیستم‌های ناوبری مختلفی برای محیط‌ها و شیوه‌های حرکتی متفاوت گسترش داده شده‌اند. از جمله سیستم‌هایی که هم‌اکنون ناوبری کور به صورت ترکیبی، تکمیلی یا کاملا وابسته در آنها استفاده می‌شود، می‌توان به ناوبری فضایی، ناوبری دریایی، ناوبری هوایی و ناوبری وسایل نقلیه زمینی اشاره کرد.

این تحقیق برروی ناوبری عابر در محیط‌های داخلی متمرکز شده است، در در ذیل سیستم‌هایی که متداولا در این زمینه استفاده می‌شوند به همراه قیمت و دقت برای مقایسه آورده شده است.

جدول1 مقایسه دقت و قیمت سیستم‌های مکان‌یابی داخلی

Indoor Positioning Technology	Accuracy	Cost
Mobile Telecommunication Networks	1-2 m	Low
RFID	1 m	Medium / High
Bluetooth	1 m	Low
Visual Light Communication (VLC)	1 m	Low / Medium
Ultra-Wide Band (UWB)	10 cm	High
Inertial Measurement Unit (IMU)	1 m	Low
Infrared, Ultrasound, Pressure Sensors	1-5 m	High
Computer Vision	1mm-1m	Low / Medium / High
Simultaneous Localization and Mapping (SLAM)	1 m	Medium / High

2-2- ناوبری کور عابر

به فرایند محاسبه موقعت محلی یک عابر با استفاده از حسگرهای میکرومکانیکال (MEMS) که در این زمینه به آنها ادوات سنجش لختی (IMU) نام دارند، اتلاق می‌شود. اکثر IMU ها حداقل دو حسگر شتاب خطی (Acceleration) و شتاب دورانی (Gyroscope) با سه درجه آزادی تجهیز شده‌اند گاهی قطب نما (Magnetometer) سه محور و فشارسنج (Barometer) نیز برای یافتن جهت شمال و اندازه گیری دقیق‌تر ارتفاع در این محصولات تعبیه می‌شوند.

اکثر این سیستم‌های ناوبری با پیدا کردن جهت حرکت در فضای 3 بعدی و میزان شتاب خطی در آن جهت و دو انتگرالگیری، سرعت و مکان نسبی عابر را می‌یابند. لازم به ذکر است که این انتگرال دوگانه موجب افزایش خطا بصورت غیرخطی (مربعی) با زمان می‌شود و این خطای انباشتی در صورت نبود منبعی

به عنوان حقیقت مبنا، برای کاهش آن پس از زمان کوتاهی خروجی بدست آمده را غیرقابل استفاده می‌کند.

برای میسر کردن این امر تکنیک‌هایی برای کاهش خطا از جمله تلفیق سنسور (Sensor Fusion)، انواع فیلتر کالمن (Kalman Filter)، پردازش سیگنال (DSP)، گیتینگ ورودی (Input Gaiting)، بدست آوردن نوع حرکت و استفاده از تعداد و زمان قدم و هوش مصنوعی می‌توان اشاره کرد.

2-3- سیستم مرجع نگاشت و جهت‌یابی (AHRS)

این سیستم با استفاده از بردارهای دکارتی 6 یا 9 محوره ورودی و الگوریتم های مختلف جهت محاسبه شده را در یکی از قالب‌های زیر ارایه می‌شود:

- زوایای اویلری (Euler Angles)
- چهارگان (Quaternion)
- ماتریس دورانی (Cosine Matrix)
- دکارتی (Cartesian)

عموما زوایای اویلری یا چهارگان برای این امر استفاده می‌شوند. اما زوایای اویلری برای وجود مشکل «قفل گیمبال» (Gimbal Lock) کمتر مرسوم می‌باشند. به همین دلیل در تمامی قسمت‌های پروژه از چهارگان‌ها استفاده شده است.

در این زمینه الگوریتم‌های متفاوتی و متعددی وجود دارند، در این پروژه چهار الگوریتم مورد بررسی قرار گرفته است که در ذیل ذکر شده‌اند:

- Madgwick: Gradient Descent Algorithm
- Mahoney: PI controller with earth acceleration reference + Dead Reckoning
- Extended Kalman Filter (EKF)
- Madgwick with Complementary Filter: renewed algorithm for faster convergence and self-adaptive parameters

4-2-چهارگان (Quaternion)

چهارگان یک نوع سیستم مختصات است که برای توصیف چرخش و جابجایی در فضای سه بعدی مورد استفاده قرار می‌گیرد. برخلاف زوایای اویلری یا ماتریس‌های چرخش، چهارگان‌ها به صورت یک چهارتایی اعداد مختلف نمایش داده می‌شوند. هرچهارگان از یک قسمت حقیقی و سه قسمت موهومی تشکیل شده است.

یک کواترنیون به صورت $q = w + x + y + z$ نمایش داده می‌شود. در اینجا w حقیقی و x و y و z فرمولهای چرخش و جابه‌جایی با استفاده از چهارگان‌ها پیچیده‌تر از زوایای یولر به نظر می‌آیند، اما از آنجا که مشکل قفل گیمبال را حل می‌کنند و به نحوی کمتر از مشکلات ناشی از اعداد شناور (Floating point operations) نسبت به ماتریس‌های چرخش مواجه می‌شوند، در بسیاری از حوزه‌هایی مانند گرافیک کامپیوتری و کنترل موقعیت و جابه‌جایی در رباتیک مورد استفاده قرار می‌گیرند.

ضرب کواترنیون با نماد $\otimes$ نشان داده می‌شود و برای دو چهارگان Q_1 و Q_2 به صورت زیر است:

$$Q_1 = [a_1 \quad b_1 \quad c_1 \quad d_1]$$

$$Q_2 = [a_2 \quad b_2 \quad c_2 \quad d_2]$$

$$Q_1 \otimes Q_2 = \begin{bmatrix} a_1a_2 - b_1b_2 - c_1c_2 - d_1d_2 \\ a_1b_2 + b_1a_2 + c_1d_2 - d_1c_2 \\ a_1c_2 - b_1d_2 + c_1a_2 + d_1b_2 \\ a_1d_2 + b_1c_2 - c_1b_2 + d_1a_2 \end{bmatrix}^T$$

ضرب چهارگان جابه‌جایی پذیر نیست: $Q_1 \otimes Q_2 \neq Q_2 \otimes Q_1$

مزدوج چهارگان: $Q^* = |w \quad -x \quad -y \quad -z|$

خطای چهارگان SLAM یا $\mathcal{L}_{QME}$[1]: $\mathcal{L}_{QME} = 2 \cdot \left\| imag(\hat{Q} \otimes Q^*) \right\|_1$

[1] این خطا در ادامه قسمت شبکه عصبی و یادگیری ماشین به عنوان بخشی از تابع خطا استفاده می‌شود

2-5-به‌روز رسانی سرعت صفر (ZUPT)

یکی از روش‌های کاهش خطا برای سیستم‌های مکان‌یابی می‌باشد. در این الگوریتم با توجه به ورودی‌های سیستم و تشخیص یا اطلاع از نوع حرکت در صورت وجود لحضاتی در حرکت که عابر ثابت و بدون سرعت باشد، پتانسیل یا احتمال برای زمان حال محاسبه می‌شود و با توجه به این احتمال در صورت تشخیص بدون سرعت بودن یک بازه، سرعت به‌روزرسانی می‌شود و در مدت بی‌حرکتی ورودی‌های سیستم در وضعیت سیستم تغییری ایجاد نمی‌کنند تا نوع حرکت شتاب‌دار دوباره تشخیص داده شود.

عموما به انواع روش‌هایی که در شرایط خاصی از ورودی یا خروجی، قسمت یا کل داده‌های ورودی را نادیده بگیرد، کنترل ورودی (Input Gaiting) گفته می‌شود که ZUPT یکی از این روش‌ها است.

2-6-یادگیری ماشین (ML)

تلفیق هوش مصنوعی بخصوص در حوضه برق و نرم افزار رشد چشگیری داشته و به طور گسترده در صنعت و تکنولوژی روز استفاده می‌شود. در زمینه PDR چندین الگوریتم و مدل شبکه عصبی پیشنهاد یا پیاده‌سازی شده است که در ادامه به چند مثال آنها پرداخته می‌شود.

- Supervised regression (Deng et al., 2020)
    - random forest regressor (RFR)
    - k-nearest neighbor regressor (KNNR)
    - support vector regressor (SVR)
- ZUPT detector (Kone et al., 2020)
    - histogram-based gradient boosting (HGB)
    - random forest (RF)
- Supervised Odometry (Silva do Monte Lima, 2019)
    - Convolutional Neural Networks (CNN)
    - Long Short-Term Memory (LSTM)

برخی بخشی از PDR مانند پیدا کردن طول قدم یا جهت حرکت یا (ZUPT) را تکمیل می‌کنند و مدل‌های بزرگ‌تر با معماری عمیق (DNN) به تبدیل مستقیم داده‌های خام سیستم به چهارگان و مکان می‌پردازند.

فصل سوم

مراحل انجام پژوهش

3-1- مقدمه

در این پروژه برای دسترسی ساده به داده‌ها و تغییر برنامه در حال اجرا از ساختار تعبیه شده (embedded) تاحدی خارج شده و تنها برای قرایت و ارسال اطلاعات از یک میکروکنترلر ESP32 استفاده شده است.

در نتیجه اکثر پردازش داخل سرور محلی انجام می‌شود. این امر در سرعت تغییر برنامه (compile time) و افزایش قدرت پردازش سیستم کامل کمک می‌کند.

یک ESP32 به شبکه محلی مشترک با سرور متصل است و داده های سنسور MPU9250 که یک سنسور 9 درجه آزادی (شتاب خطی، دورانی و قطب نما) را با پروتکل UDP به سرور ارسال می‌کند..

برروی سرور نیز یک پلتفرم نرم‌افزاری یکپارچه در python نوشته شده است که به کاربر قابلیت های ذیل را ارایه می‌دهد:

- اخذ و نمایش داده های خام
- تغییرات بازه ورودی داده‌ها برروی حسگر به صورت زنده
- فیلتر پایین-گذر با fs و مرتبه و فرکانس نمونه برداری قابل تغییر برای پیش‌پردازش داده‌ها
- رابط کاربری برای کالیبر هر سه سنسور به روش تطبیق بیضوی (Ellipsoid fitting)
- نمایش جهت سه بعدی و مقادیر چهارگان که توسط یکی از چهار الگوریتم قابل انتخاب Madgwick، Mahoney، EKF و Complementary EKF استخراج شده‌است
- ثبت داده در فرمت CSV بعد از همگرایی الگوریتم‌های AHRS
- پردازش داده‌های ثبت شده در انتهای ثبت توسط سه الگوریتم مکان‌یابی پیشنهادی و در نهایت مقایسه خروجی سه الگوریتم

3-2- الگوریتم‌های موقعیت‌یابی کور

در این پروژه سه الگوریتم default، AHRS+ZUPT و شبکه عصبی استفاده شده که در ادامه توضیحات هرکدام ذکر شده است.

3-2-1- الگوریتم Default

ساده‌ترین شیوه محاسبه لحظه‌ای مکان محاسبه چهارگان از یکی از سه الگوریتم (Madgwick, Mahoney, EKF) محاسبه می‌شود و از آن ماتریس چرخشی بدست آورده و با ضرب آن در ورودی خام شتاب خطی (Acceleration) شتاب در سه محور (x, y, z) یا NED (North, East, Down) بستگی به نوع الگوریتم جهت‌یابی و تنظیمات آن می‌دهد.

با محاسبه انتگرال دوگانه از شتاب در محورهای واقعی به مکان نسبی می‌توان دست یافت. بدیهی است که در این روش هیچ روش کاهش خطایی در آن استفاده نشده و شاهد خطای تجمعی خواهیم بود. این روش بیشتر جنبه مقایسه دارد تا یک راه حل به این مسئله.

3-2-2- الگوریتم ZUPT + AHRS

این شیوه پس از محاسبه چهارگان از از حالت قبلی با توجه به اندازه بردار شتاب که جاذبه کالیبر شده زمین از آن حذف شده، درصورت عبور اندازه این بردار از یک آستانه عددی (در بعضی موارد تکنیک‌های دیگری برای تشخیص آستانه یا شروع حرکت ZUPT استفاده شده در این پروژه برای سادگی از آستانه گیری عددی ساده استفاده شده است).

حال از شتاب، سرعت را با انتگرال در زمان‌های متحرک مشخص شده از ZUPT محاسبه کرده و سپس اتنگرال ثانویه از سرعت گرفته می‌شود تا مکان نسبی نهایی بدست آید. در این حالت با تکنیک استفاده شده خطای بدست آمده از مکان نسبت به شتاب ورودی رابطه خطی خواهد داشت.

3-2-3- شبکه عصبی

اکثر شبکه‌های عصبی یادگیری شده در حوضه PDR در حالت «یادگیری تحت نظارت» (Supervised Learning) آموزش داده شده‌اند. روش یادگیری ذکر شده به معنای حضور داده‌های ورودی و داده‌های متناظر خروجی آن در حین یادگیری می‌باشد.

در این پروژه به علت نبود تکنولوژی‌های استفاده شده و همچنین قیمت بالای آنها تصمیم به استفاده از داده‌های موجود قابل استفاده از مجموعه داده Oxford Inertial Odometry Dataset (OXiOD) گرفته شد. در نتیجه این امر یک لایه کالیبر و معیارسازی و بازگردانی مقیاس (normalize, denormalization) در مدل تعبیه شده است تا اختلاف داده‌های دو مدل سنسور استفاده شده در این پروژه با سنسور استفاده شده در OXiOD را حداقل کند.

لازم به ذکر است OXiOD حاوی داده‌ها و حرکت‌های متنوعی است اما هیچ دسته از داده‌ها شامل مجموعه‌ای برای محل نصب برروی پاشنه پا نبود. درنتیجه بررسی انجام شده مقایسه‌ای متناظر بین این الگوریتم و نسخه ZUPT نیست. (در این مجموعه داده 10 مدل حرکتی مورد بررسی قرار گرفته که به چند مورد : داخل کیف (handbag)، جیب (pocket)، دویدن (running)، دردست (handheld) و دویدن (running) می‌توان اشاره کرد که در برای این مدل نزدیک‌ترین و قابل مقایسه‌ترین مجموعه‌ها از نظر نوع و فرکانس حرکتی به سنسور نصب شده برروی پا، به مجموعه handheld می‌توان اشاره کرد چراکه المان‌های حرکت تناوبی که به وضوح در داده‌های پاشنه پا دیده می‌شود در این داده‌ها نیز قابل مشاهده است.

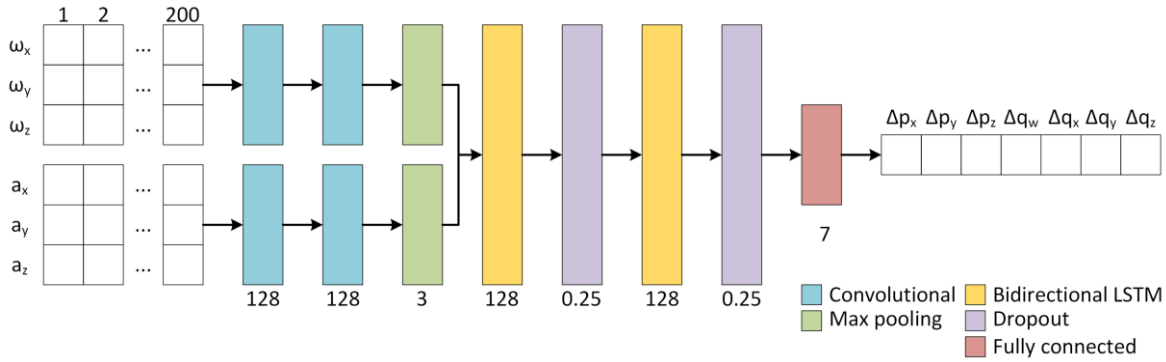
3-2-3-1- معماری

این شبکه مطابق شبکه ذکر شده در مقاله (Silva do Monte Lima, 2019) با معماری دولایه Convolutional به ازای هر سه نوع حسگر و یک لایه Max Pooling و دو لایه Bi-LSTM با drop out برابر با 25٪ می‌باشد.

انواع لایه‌های بازگشتی (Recurrent Layers) در اکثر مقاله‌های PDR با رویکرد شبکه عصبی مشاهده می‌شود که خاصیت حافظه کوتاه و بلند مدت را به علت ورودی از جنس زمانی داده‌ها و پویا (dynamic) بودن ورودی و سیستم ارایه می‌دهد.

ورودی شبکه از داده‌های خام سه سنسور و خروجی آن تغییرات بردار مکان و چهارگان می‌باشد که از داده‌های خروجی OXiOD استخراج می‌شود.

شکل 1معماری شبکه برای حالت 6-درجه آزادی و تعداد ویژگی‌ها (features) زیر هر لایه مشخص شده است خروجی بردار تغییر مکان در سه بعد و بردار تغییر جهت در فالب چهارگان می‌باشد



3-2-3-2- تابع خطا (Loss Function)

در مقاله منبع از چندین تابع خطای متفاوت بهره‌گیری شده که تابع با بهتری عملکرد در اعتبارسنجی نهایی مورد استفاده قرار گرفته است. این تابع خطا را با بهره‌گیری از نرم 1 بردار مکان و نرم 1 موهومی خطای ضرب چهارگان در مزدوج خروجی آن $\mathcal{L}_{QME}$ محاسبه شده است.

$$\begin{cases} \mathcal{L}_{TMAE} = \left\|(\hat{P} - \mathrm{P})\right\|_1 \\ \mathcal{L}_{QME} = 2 \cdot \left\|imag(\hat{Q} \otimes Q^*)\right\|_1 \end{cases}$$

فصل چهارم

یافته های پژوهش

4-1- مقدمه

با استفاده از پلتفرم توسعه داده شده که در فصل 2 ذکر شد، 8 دسته داده اخذ شده است که نصف آن ادوات اندازه‌گیری ساخته شده در دست (handheld) نگه داشته می‌شود (مناسب شبکه عصبی) و در نیمی دیگر دستگاه برروی پاشنه پای اصلی (مناسب ZUPT) نصب شده است. در هر دو حالت ذکر شده حرکت کننده در دو مسیر به شکل بیضی و مربع حرکت می‌کند.

شایان به ذکر است این آزمایش برای جنسیت و قدهای متفاوت نیز تکرار شد ولی تنها عامل قد در مقیاس اندازه (scaling factor) تاثیر گذاشته است که با مقالات تخمین اندازه قدم بهمراه ZUPT همخوانی دارد.

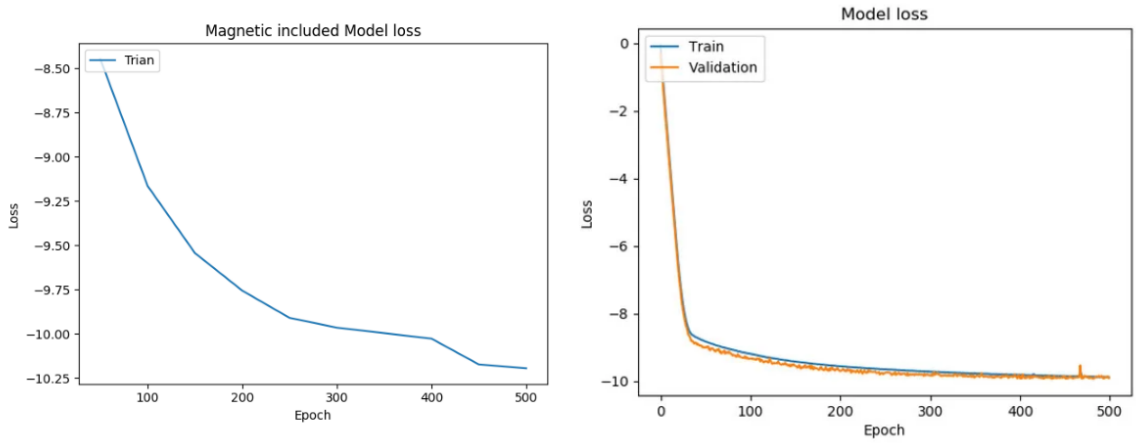
هرکدام از این حرکت‌ها توسط الگوریتم default و الگوریتم متناسب با آن (ZUPT یا NN) مورد بررسی قرار گرفته است. از بعد عمودی (z) در این آزمایش به علت خطای بالا در مقاله شبکه عصبی ذکر شده و خطای بالای تخمین توسط ZUPT صرف نظر شده است.

4-2- ساختار آموزش شبکه های عصبی

با پیاده سازی مدل و معماری مشابه مقاله ذکر شده در دو چهارچوب یادگیری ماشینی Pytorch و Keras و با استفاده از داده‌های handheld برگرفته شده از OXiOD ، با بازه 200 فریم ، از داده‌های خام ورودی (accel, gyro, magneto) ، سایز دسته 32 (batch size)، طول گام 10 (stride length)، بهینه‌ساز Adam با نرخ یادگیری 0.0001 و فرکانس ورودی داده 200Hz.

همچنین باتوجه به معماری استفاده شده در مقاله اصلی، تنها داده‌های شتاب مورد استفاده بود و از داده‌های قطب‌نما به علت نویز محیطی بالا صرف نظر شده است اما برای مقایسه، دو مدل آموزیش داده شده که اولین مطابق مقاله ذکر شده و آخرین به همراه ورودی قطب‌نما آموزش داده شده است. داده‌ها به تقسیم 10% اعتبارسنجی (Validation) و برای 500 epoch آموزش دیده است.

شکل 2نمودار کاهش loss با تعداد epoch. سمت راست مدل اولیه و سمت چپ مدل آموزش داده شده همراه با داده‌های قطب‌نما



هر دو مدل در Pytorch و Keras و با کارت گرافیک RTX2070 Super به مدت 8 الی 10 ساعت به ازای هر آموزش مدل زمان برده است. در انتهای آموزش مدل‌های محیط Keras از دقت و loss پایین‌تری در برخوردار بودند، درنتیجه مدل‌های محیط ذکر شده در ادامه آزمایشات استفاده شده است.

4-3-مقایسه سه الگویتم

4-3-1- داده‌های اخذ شده

12 داده زیر در قالب csv و در پوشه capture موجود است. این داده‌ها در فضای 13m x 9m ثبت شده و برابر با اضلاع مستطیل و دو قطر بیضی و ضلع بزرگتر مثلث می‌باشد.

جدول2 تعداد قدم و دسته‌بندی داده‌های اخذ شده

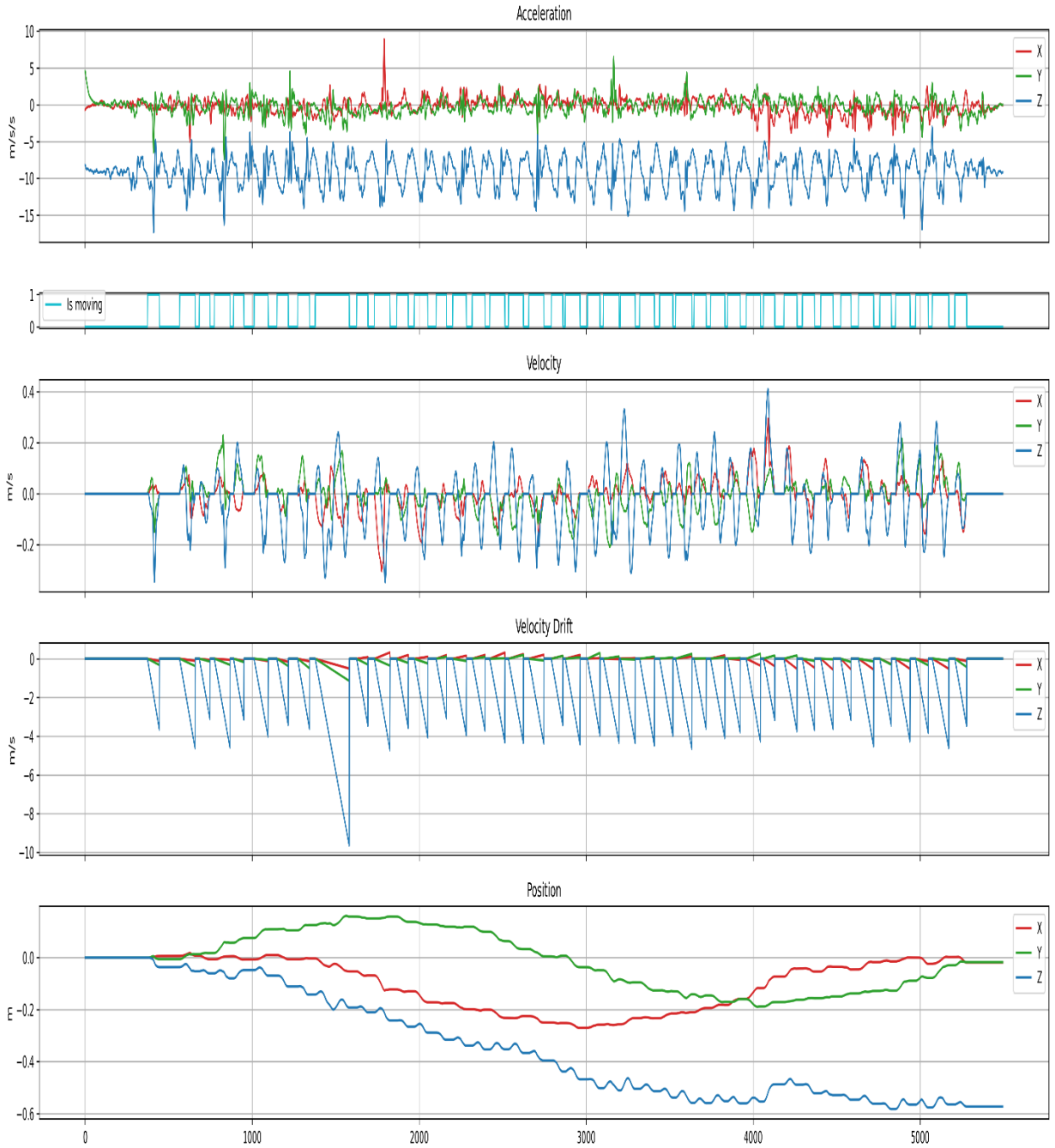
Height/Gender	Handheld			Foot mounted		
	Rectangular	Circular	Triangular	Rectangular	Circular	Triangular
Male	65	48	46	68	46	38
Female	68	48	44	70	53	47

4-3-2- خروجی ZUPT

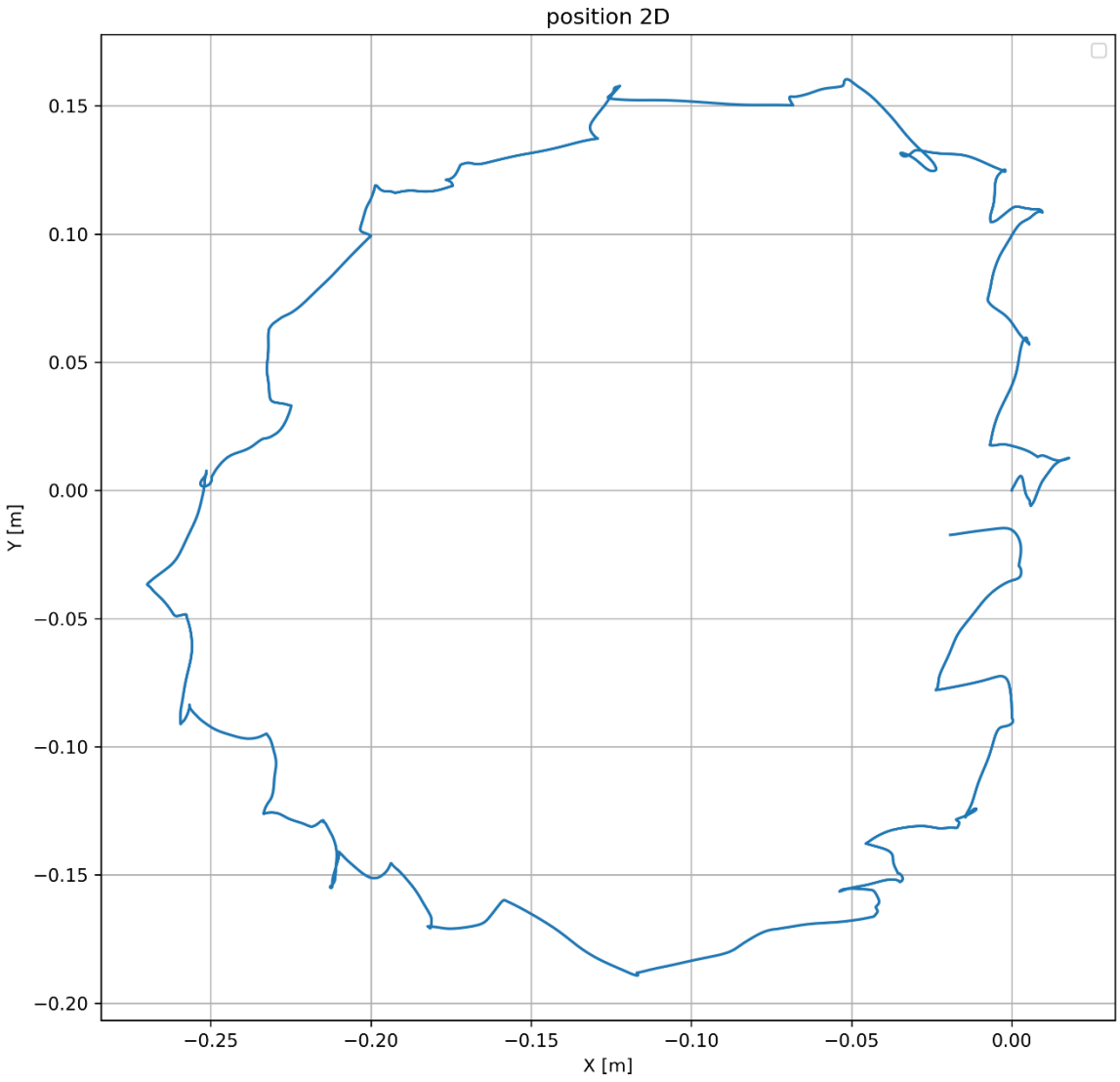
در ادامه یک مثال از خروجی کامل الگوریتم حاوی داده‌های الگوریتم AHRS که در این مورد Madgwick بوده برای یک مسیر بیضوی در حالت handheld و توسط فرد بلند قد (مرد) طی شده.

شکل 3داده‌های خام سنسورها، جهت سنسور در سه محور yaw، pitch و roll ، میزان خطای شتاب خطی نسبت به شتاب دورانی ، زمان استفاده از ژیروسکوپ به تنهایی برای تخمین جهت و زمان تا شروع استفاده ترکیبی دوباره از تمامی سنسورها

شکل 4شتاب بدست آمده در جهات تصحیح شده و در قالب NED ، نمودار نشانگر درحال حرکت (is moving) ، سرعت محاسبه شده نهایی، میزان رانش (drift) سرعت که از انتها و ابتدای هر بازه حرکتی درونیابی خطی شده و از سرعت اولیه خام کاسته میشود و در نهایت مقادیر مکان در سه بردار دکارتی

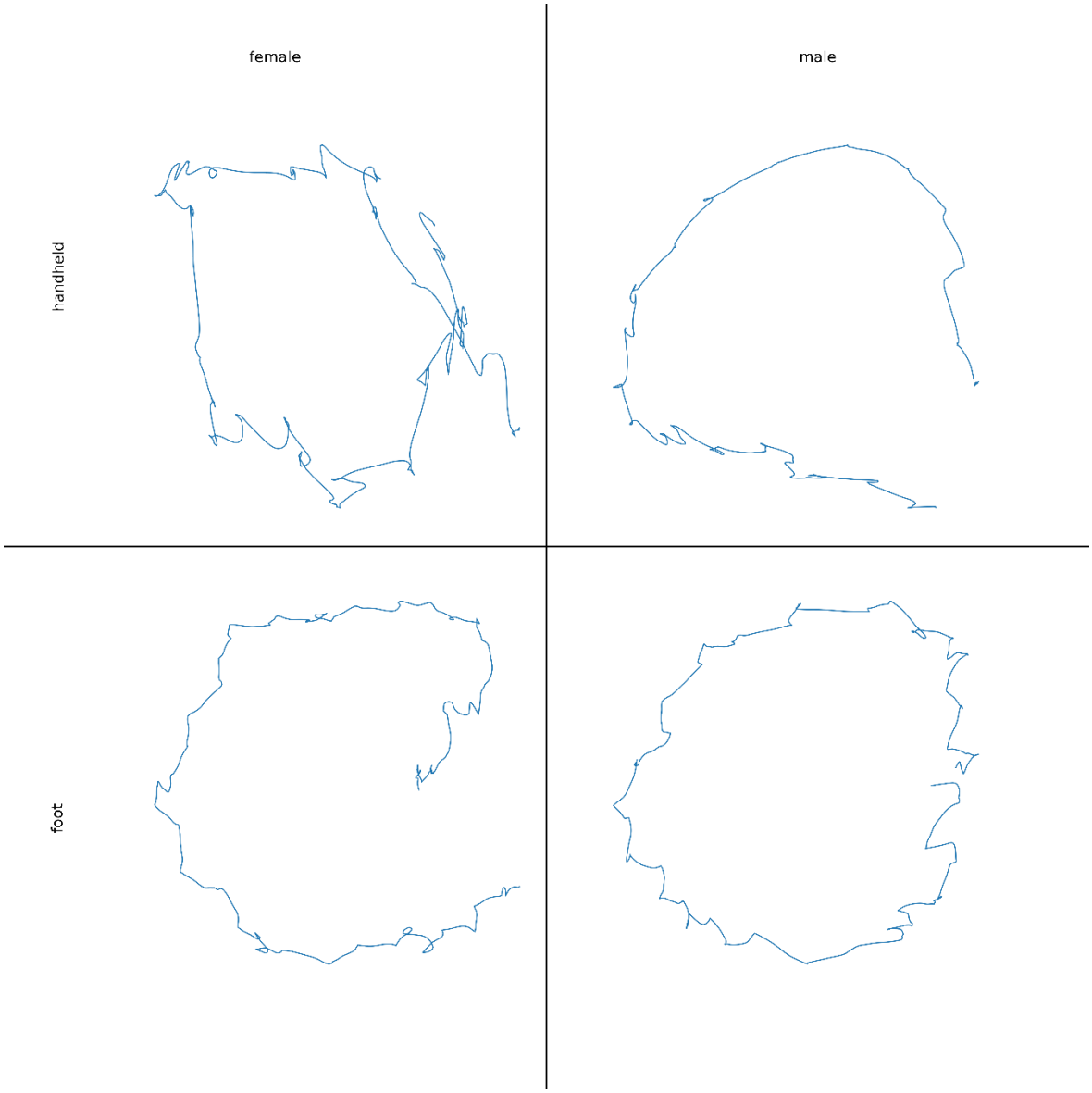


شکل 5 درنهایت مسیر دو بعدی رسم شده از خروجی نیز برای مقایسه نهایی ذخیره می‌شود



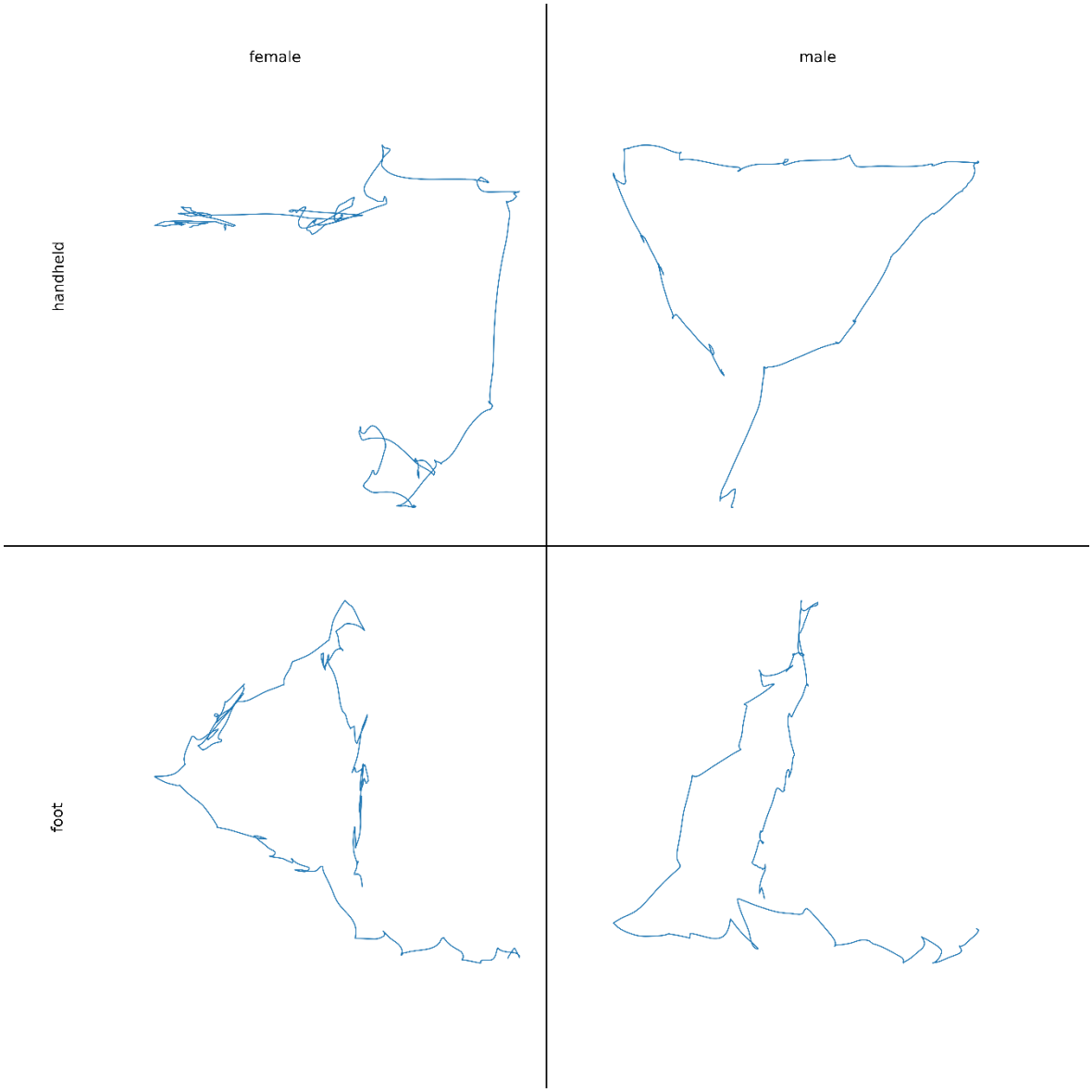
در ادامه به مقایسه سه مسیر ارایه شده با دو حرکت کننده متفاوت و محل نصب در دست و پاشنه پا می‌پردازیم. در این بخش به علت مشابه بودن دو نمودار ابتدایی تنها مسیر نهایی کنار یکدیگر ارایه شده و در انتها تعداد قدم‌ها با مقدار واقعی مقایسه شده است.

شکل 6 مقایسه قد، جنسیت و محل نصب در مسیر بیضوی



شکل 7 مقایسه قد، جنسیت و محل نصب در مسیر مستطیلی

شکل 8 مقایسه قد، جنسیت و محل نصب در مسیر مثلثی

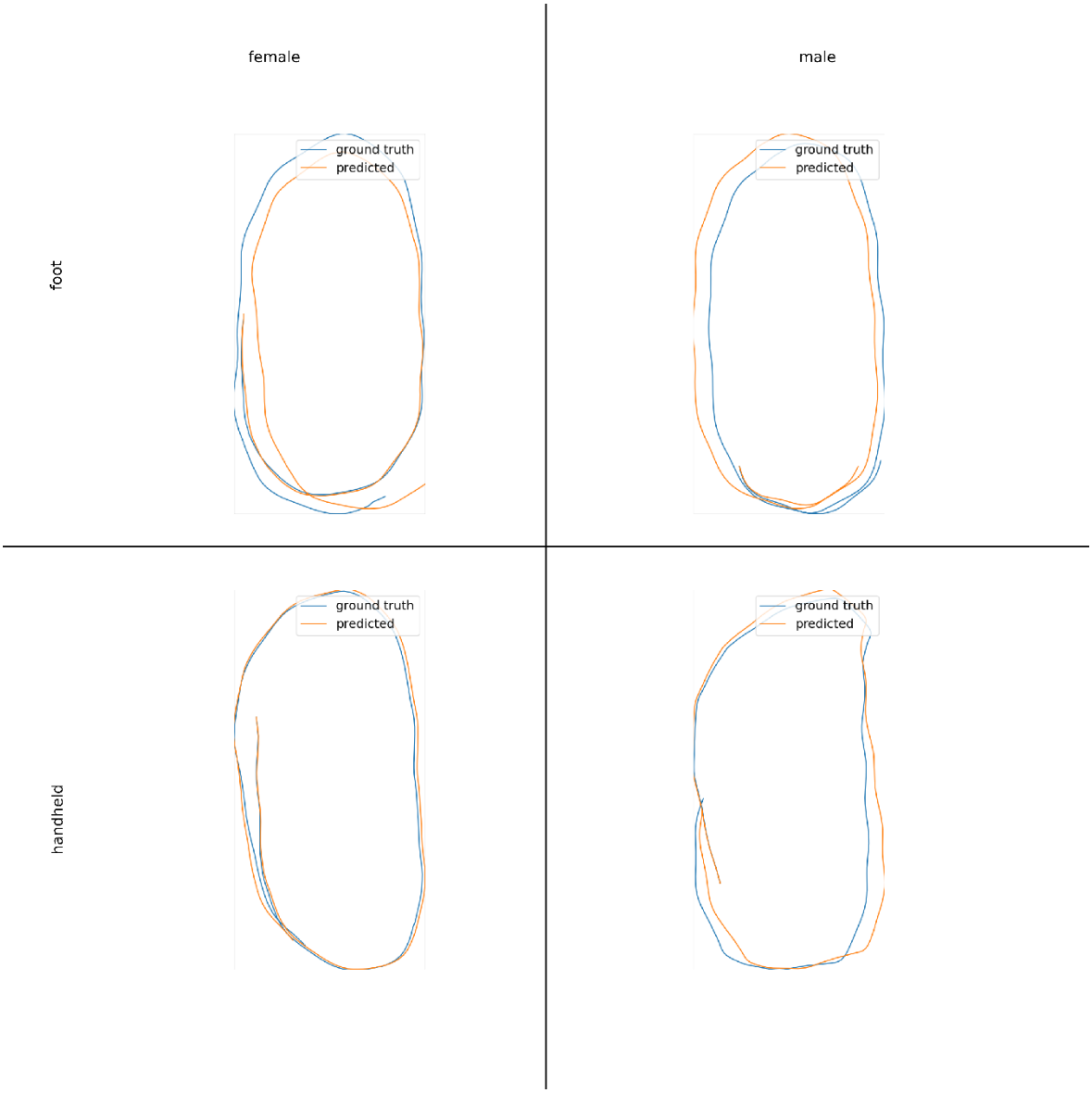


مقایسه تعداد قدم‌های ثبت شده، قدم‌های یافت شده و خطای نسبی درصدی فرایند3 جدول

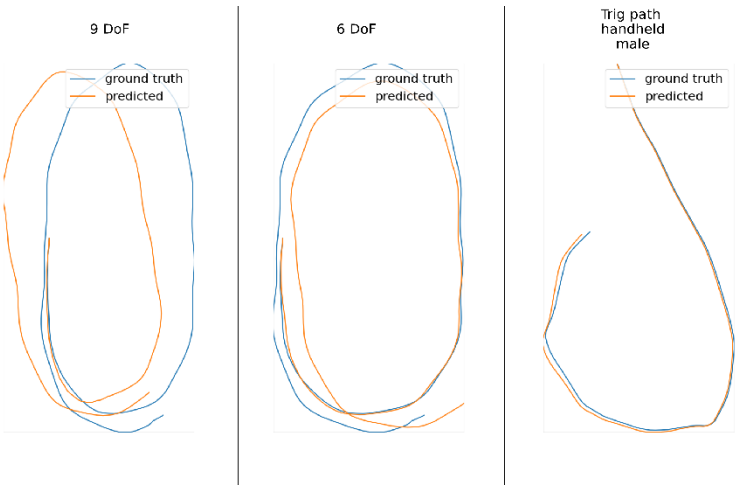
Height/Gender	Handheld			Foot mounted			type
	Rectangular	Circular	Triangular	Rectangular	Circular	Triangular	
Male	65	48	46	68	46	38	**real**
Female	68	48	44	70	53	47	
Male	61	42	34	35	24	21	**processed**
Female	68	41	43	34	28	28	
Male	6.2	12.5	26.1	-2.9	-4.3	-10.5	**error [%]**
Female	0	14.6	2.3	2.9	-5.7	-19.1	

4-3-3- خروجی شبکه عصبی

شکل 9 مقایسه قد، جنسیت و محل نصب در مسیر بیضوی



شکل 10 مقایسه شبکه با داده‌های مغناطیسی، بدون داده‌های مغناطیسی و مسیر مثلثی



فصل پنجم

نتیجه گیری

5-1- مقدمه

نتایج این تحقیق به علت نبود سیستم‌های ثبت دقیق موقعیت و جهت (ViCon) در دسترس نویسنده تنها که در مقالات دیگر توسط آنها یا سیستم‌های مشابه نتایج نهایی بررسی شده است، نتایج به صورت ظاهری و کیفی بررسی شده است.

5-2- خلاصه نتایج حاصله

با توجه به کیفی بودن نتیاج شبکه عصبی عملکرد بهتری و نزدیک‌تر به شکل مسیر ارایه داده است که در هر دو مسیر و نسبت به افراد با قد متفاوت این نتیجه قابل مشاهده می‌باشد.

لازم به ذکر است در الگوریتم ZUPT، پارامترهای اولیه همچون آستانه شتاب برای حرکت، طول قدم و... برای هرکدام از نمونه‌ها برای نزدیک‌ترین نتیجه به شکل نهایی تغییر و تنظیم شده است که این الگوریتم را که تعمیم‌دهی به شرایط، قد، جنسیت و مسیرهای متفاوت را مشکل می‌کند.

5-3- پیشنهاد برای پژوهش های آتی

- اخذ داده‌های foot mounted توسط سیستم‌هایی مشابه ViCon یا مکان‌یابی مثلثی توسط روش UWB triangulation برای مقایسه دقیق‌تر و یک به یک دو مدل پیشنهادی شبکه عصبی و ZUPT. (مدل شبکه عصبی با داده‌های handheld آموزش دیده است که شباهت کافی به داده‌های foot mounted ندارد)
- تلفیق یک مدل شبکه عصبی یا الگوریتمی برای تخمین طول قدم و پارامترهای قابل تنظیم در قسمت ZUPT برای دقت بهتر و تعمیم پذیری بالاتر برای انواع شرایط محیطی و کاربری
- داده‌های آزمایش نهایی برای مسیرهایی با شیب افقی بالا همخوانی با شکل مسیر نداشتند و علت این را می‌توان به فرکانس دقت و فاکتورهای مقیاس حداکثری سنسور مرتبط دانست که با ورود سنسورهای متنوع و دقیق‌تر و با قیمت مناسب به بازار کشور این مشکل مرتفع پذیر است.

شایان به ذکر است که به علت متناظر نبودن توابع این دو محیط Torch و Keras نویسنده اطمینان کافی از شباهت دقیق این مدل‌ها در محیط Torch ندارد و این موضوع نیاز به بررسی بیشتر دارد.

فهرست منابع

Alex Kendall ،Yarin Gal و ،Roberto Cipolla .(2018) .Multi-Task Learning Using Uncertainty to Weigh Losses for Scene Geometry and Semantics .*CVPR*.

Carl Fischer ،Poorna Talkad Sukumar و ،Mike Hazas .(2013) .Tutorial: Implementing a Pedestrian Tracker Using Inertial Sensors .*IEEE Pervasive Computing* .27-17 ، doi:10.1109/MPRV.2012.16

Daniel Weber ،Clemens Ghmann و ،Thomas Seel .(2020) .Neural Networks Versus Conventional Filters for Inertial-Sensor-based Attitude Estimation.

Eric Foxlin .(2005) .Pedestrian Tracking with Shoe-Mounted Inertial Sensors .*IEEE Computer Graphics and Applications* .46-38 ،doi:10.1109/MCG.2005.140

João Paulo Silva do Monte Lima ،Hideaki Uchiyama و ،Rin-ichiro Taniguchi .(2019) .End-to-End Learning Framework for IMU-Based 6-DOF Odometry .*Mobile Robot Navigation* . doi:https://doi.org/10.3390/s19173777

Rahul P Suresh ،Vinay Sridhar ،Pramod J و ،Viswanath Talasila .(2018) .Zero Velocity Potential Update (ZUPT) as a Correction Technique*3 .rd International Conference On Internet of Things: Smart Innovation and Usages (IoT-SIU)* .doi:10.1109/IoT-SIU.2018.8519902

Sebastian O.H. Madgwick .(2010) .*An efficient orientation filter for inertial and inertial/magnetic sensor arrays* .

Sebastian O. H. Madgwick ،Samuel Wilson ،Ruth Turk ،Jane Burridge ،Christos Kapatos و ،Ravi Vaidyanathan .(2020) .An Extended Complementary Filter for Full-Body MARG Orientation Estimation .*IEEE/ASME Transactions on Mechatronics (Volume: 25, Issue: 4, August 2020)* .2064 - 2054 ،doi:10.1109/TMECH.2020.2992296

Simone Ludwig و ،Kaleb Burnham .(2018) .Comparison of Euler Estimate using Extended Kalman Filter, Madgwick and Mahony on Quadcopter Flight Data .*ICUAS*.

YIQIONG MIAO .(2021) .*MEMS-MARG-BASED DEAD RECKONING FOR* .McMaster University.

Yi-Shan Li و ،Fang-Shii Ning .(2018) .Low-Cost Indoor Positioning Application Based on Map Assistance and Mobile Phone Sensors .*Sensors* .(12)18 ،
doi:https://doi.org/10.3390/s18124285

Zengshan Tian ،Yuan Zhang ،Mu Zhou و ،Yu Liu .(2014) .Pedestrian dead reckoning for MARG navigation using a smartphone .*EURASIP J. Adv. Signal Process* .65 ،
doi:https://doi.org/10.1186/1687-6180-2014-65

پیوست

در ادامه 4 برنامه که نقش اصلی پردازش را ایفا کرده‌اند آورده شده است. که شامل کد دو الگوریتم ZUPT و AHRS و همچنین مدل و آموزش شبکه عصبی ذکر شده بهمراه پلتفرم تلفیق کننده این دو (PDR) و موتور بصری‌سازی برای درک و بینش عمیق‌تر نسبت به داده‌های ورودی زنده و ترتیب پردازش داده‌ها در پلتفرم می‌باشد.

بدیهی است که برای خلاصه نگه داشتن و کاهش حجم این دانشنامه تمامی برنامه‌های استفاده شده در این پروژه پیوست نشده و تنها برنامه‌های اصلی قابل دسترسی هستند. برای دسترسی به کد کل پروژه، پروژه در تمامیت در پلتفرم GitHub در دسترسی عموم قرار گرفته است.

(برای اطلاعات بیشتر راجع به برنامه و ساختار درونی آن و دیدن نمونه‌های تکمیلی به ریپو مراجعه شود)

GitHub repo: aradng/MARG-Dead-reckoning

Code1 ZUPT Algorithm

```python
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import imufusion
from scipy import signal


def zupt(
    df,
    fn="data",
    sample_rate=200,
    zupt_tresh=3,
    margin=0.1,
    debug=False,
    lp_filter=False,
):
    plt.style.use("default")

    df = df.copy().reset_index()
    df.index /= sample_rate
    df["gyro"] *= 180 / np.pi
    # df['accel'] /= 9.80665
```

```
    dt = 1 / sample_rate
    margin = int(margin * sample_rate)  # 100 ms
    # debug = False

    df.index /= sample_rate

    # filter results
    if lp_filter:
        b, a = signal.butter(10, 20, fs=200, btype="lowpass", analog=False)
        df = df.apply(lambda x: signal.filtfilt(b, a, x))

    offset = imufusion.Offset(sample_rate)
    ahrs = imufusion.Ahrs()
    ahrs.settings = imufusion.Settings(
        imufusion.CONVENTION_NED,
        0.5,  # gain
        2000,  # gyroscope range
        10,  # acceleration rejection
        30,  # magnetic rejection
        5 * sample_rate,
    )  # rejection timeout = 5 seconds

    def update(x):
        # ahrs.update(x['gyro'].to_numpy(), x['accel'].to_numpy(),
x['mag'].to_numpy(), 0.005)
        ahrs.update_no_magnetometer(
            x["gyro"].to_numpy(), x["accel"].to_numpy(), 0.005
        )

        euler = ahrs.quaternion.to_euler()
        Q = ahrs.quaternion.wxyz
        # acceleration = ahrs.earth_acceleration * 9.80665  # convert g to m/s/
        acceleration = ahrs.earth_acceleration  # convert g to m/s/

        ans = {}
        ans.update(
            {"x": acceleration[0], "y": acceleration[1], "z": acceleration[2]}
        )
        ans.update({"roll": euler[0], "pitch": euler[1], "yaw": euler[2]})
        ans.update({"Q_T": Q})
        ans.update({"accel_err": ahrs.internal_states.acceleration_error})
        ans.update({"accel_igr": ahrs.internal_states.accelerometer_ignored})
        ans.update(
            {"accel_rec": ahrs.internal_states.acceleration_recovery_trigger}
        )

        ans.update({"ang_rrec": ahrs.flags.angular_rate_recovery})
        ans.update({"accel_rrec": ahrs.flags.acceleration_recovery})
        return ans

    sf = df.apply(update, axis=1)
    sf = pd.DataFrame(list(sf), index=df.index)

    fig, ax = plt.subplots(
        nrows=6,
        sharex=True,
        figsize=(20, 15),
        tight_layout=True,
```

```
        gridspec_kw={"height_ratios": [6, 6, 6, 2, 1, 1]},
    )

    ax[0].plot(df.index, df["gyro", "x"], "tab:red", label="gyro x")
    ax[0].plot(df.index, df["gyro", "y"], "tab:green", label="gyro y")
    ax[0].plot(df.index, df["gyro", "z"], "tab:blue", label="gyro z")
    ax[0].set_ylabel("Degrees/s")
    ax[0].set_title("Gyroscope")
    ax[0].legend()
    ax[0].grid()

    ax[1].plot(df.index, df["accel", "x"], "tab:red", label="Accelerometer x")
    ax[1].plot(
        df.index, df["accel", "y"], "tab:green", label="Accelerometer y"
    )
    ax[1].plot(
        df.index, df["accel", "z"], "tab:blue", label="Accelerometer z"
    )
    ax[1].set_ylabel("Acceleration [g]")
    ax[1].set_title("Accelerometer")
    ax[1].legend()
    ax[1].grid()

    ax[2].plot(sf["yaw"], "tab:red", label="Yaw")
    ax[2].plot(sf["pitch"], "tab:green", label="Pitch")
    ax[2].plot(sf["roll"], "tab:blue", label="Roll")
    ax[2].set_ylabel("Degrees")
    ax[2].grid()
    ax[2].legend()

    ax[3].plot(sf["accel_err"], "tab:olive", label="Acceleration error")
    ax[3].set_ylabel("Error")
    ax[3].set_xlabel("Time [s]")
    ax[3].legend()

    ax[4].plot(sf["accel_igr"], "tab:cyan", label="Acceleration ignored")
    ax[4].legend()

    ax[5].plot(
        sf["accel_rec"], "tab:orange", label="Acceleration recovery trigger"
    )
    ax[5].legend()

    for axes in ax:
        axes.set_xlim(0, sf.index.max())

    fig.savefig(f"ypr.png", dpi=300)

    from scipy.signal import find_peaks

    hf = sf[["x", "y", "z"]].to_numpy()
    cols = pd.MultiIndex.from_product([["acceleration"], ["x", "y", "z"]])
    hf = pd.DataFrame(hf, columns=pd.MultiIndex.from_tuples(cols))

    # subtract earth gravity
    g_end = np.linalg.norm(hf["acceleration"], axis=1)[-100:].mean()
    g_start = abs(hf["acceleration", "z"][-100:].mean())
    g = min(g_start, g_end)
```

```
print(f"calculated g : {g}")

# ZUPT
fig, ax = plt.subplots(
    nrows=4, sharex=True, figsize=(20, 10), tight_layout=True
)

hf["is_moving"] = (
    hf["acceleration"].apply(np.linalg.norm, axis=1) > zupt_tresh + g
)

ax[0].plot(
    hf["acceleration"].apply(np.linalg.norm, axis=1) - g, label="norm"
)
ax[1].plot(hf["is_moving"], label="is_moving")

for index in range(len(hf) - margin):
    hf.loc[index, "is_moving"] = any(
        hf.loc[index : (index + margin), "is_moving"]
    )  # add leading margin

ax[2].plot(hf["is_moving"], label="is_moving trailing")

for index in range(len(hf) - 1, margin, -1):
    hf.loc[index, "is_moving"] = any(
        hf.loc[(index - margin) : index, "is_moving"]
    )  # add trailing margin

ax[3].plot(hf["is_moving"], label="is_moving leading")

for axes in ax:
    axes.legend()
ax[0].set_ylim(0, 10)

if debug:
    fig.savefig(f"zupt.png", dpi=300)
    for axes in ax:
        l = len(hf)
        axes.set_xlim(l / 2 - l / 10, l / 2 + l / 10)
    fig.savefig(f"zupt_zoom.png", dpi=300)

peaks, _ = find_peaks(hf["is_moving"].astype(int))
steps = len(peaks)
print(f"steps : {steps}")

# velocity caluclations
velocity = np.zeros((len(hf), 3))
cols = pd.MultiIndex.from_product([["velocity"], ["x", "y", "z"]])
hf[cols] = hf["acceleration"] * dt
for idx in range(1, len(hf)):
    if hf.loc[idx, "is_moving"][0]:
        velocity[idx] = velocity[idx - 1] + hf.loc[idx, "velocity"]

hf["velocity"] = velocity

# velocity drift
is_moving_diff = hf["is_moving"].astype(int).diff().fillna(0)
idx_shift_diff = is_moving_diff[is_moving_diff < 0].index
```

```
    is_moving_diff[idx_shift_diff] = 0
    is_moving_diff[idx_shift_diff - 1] = 1
    is_moving_diff = is_moving_diff.astype(bool)

    hf["step"] = False
    hf.loc[is_moving_diff, "step"] = True

    cols = pd.MultiIndex.from_product([["velocity_drift"], ["x", "y", "z"]])
    hf[cols] = hf["velocity"].apply(lambda x: x * is_moving_diff)
    idx_to_interp = hf[hf["is_moving"]][
        "is_moving"
    ].index.symmetric_difference(is_moving_diff[is_moving_diff].index)
    hf.loc[idx_to_interp, "velocity_drift"] = np.nan
    hf["velocity_drift"] = hf["velocity_drift"].interpolate()
    hf["velocity"] = hf["velocity"] - hf["velocity_drift"]

    # Calculate pos
    cols = pd.MultiIndex.from_product([["position"], ["x", "y", "z"]])
    hf[cols] = hf["velocity"] * dt
    pos = np.zeros((len(hf), 3))
    for idx in range(1, len(hf)):
        pos[idx] = pos[idx - 1] + hf.loc[idx, "position"]

    hf["position"] = pos

    fig, ax = plt.subplots(
        nrows=5,
        sharex=True,
        figsize=(20, 10),
        tight_layout=True,
        gridspec_kw={"height_ratios": [6, 1, 6, 6, 6]},
    )

    ax[0].plot(hf["acceleration", "x"], "tab:red", label="X")
    ax[0].plot(hf["acceleration", "y"], "tab:green", label="Y")
    ax[0].plot(hf["acceleration", "z"], "tab:blue", label="Z")
    ax[0].set_title("Acceleration")
    ax[0].set_ylabel("m/s/s")
    ax[0].grid()
    ax[0].legend()

    ax[1].plot(hf["is_moving"], "tab:cyan", label="Is moving")
    ax[1].grid()
    ax[1].legend()

    ax[2].plot(hf["velocity", "x"], "tab:red", label="X")
    ax[2].plot(hf["velocity", "y"], "tab:green", label="Y")
    ax[2].plot(hf["velocity", "z"], "tab:blue", label="Z")
    ax[2].set_title("Velocity")
    ax[2].set_ylabel("m/s")
    ax[2].grid()
    ax[2].legend()

    ax[3].plot(hf["velocity_drift", "x"], "tab:red", label="X")
    ax[3].plot(hf["velocity_drift", "y"], "tab:green", label="Y")
    ax[3].plot(hf["velocity_drift", "z"], "tab:blue", label="Z")
    ax[3].set_title("Velocity Drift")
    ax[3].set_ylabel("m/s")
```

```
ax[3].grid()
ax[3].legend()

ax[4].plot(hf["position", "x"], "tab:red", label="X")
ax[4].plot(hf["position", "y"], "tab:green", label="Y")
ax[4].plot(hf["position", "z"], "tab:blue", label="Z")
ax[4].set_title("Position")
ax[4].set_ylabel("m")
ax[4].grid()
ax[4].legend()

fig.savefig(f"path_{len(peaks)}.png", dpi=300)

# plot position 2D

fig, axes = plt.subplots(nrows=1, figsize=(10, 10))

axes.plot(hf["position", "x"], hf["position", "y"], label="path")
axes.scatter(
    hf.loc[idx_shift_diff, "position"]["x"],
    hf.loc[idx_shift_diff, "position"]["y"],
    color="red",
    label="steps",
)
axes.set_xlabel("X [m]")
axes.set_ylabel("Y [m]")
axes.set_title("position 2D")
axes.legend()
axes.grid()

fig.savefig(f"path2D_{fn}.png", dpi=300)
plt.show()
```

Code2 Neural Network Model (Architecture & Custom Loss Layer)

```
import tfquaternion as tfq
import tensorflow as tf

from keras.models import Sequential, Model
from keras.layers import (
    Bidirectional,
    LSTM,
    Dropout,
    Dense,
    Input,
    Layer,
    Conv1D,
    MaxPooling1D,
    concatenate,
)
from keras.initializers import Constant
from keras.optimizers import Adam
from keras.losses import mean_absolute_error
from keras import backend as K


def quat_mult_error(y_true, y_pred):
    q = tfq.Quaternion(y_pred).normalized()
    q_hat = tfq.quaternion_conjugate(y_true)
    q_prod = q * q_hat
    q_prod = tf.convert_to_tensor(q_prod)
    w, x, y, z = tf.split(q_prod, num_or_size_splits=4, axis=-1)
    return tf.abs(tf.multiply(2.0, tf.concat(values=[x, y, z], axis=-1)))


def quaternion_mean_multiplicative_error(y_true, y_pred):
    return tf.reduce_mean(quat_mult_error(y_true, y_pred))


# Custom loss layer
class CustomMultiLossLayer(Layer):
    def __init__(self, nb_outputs=2, **kwargs):
        # def __init__(self, nb_outputs=3, **kwargs):
        self.nb_outputs = nb_outputs
        self.is_placeholder = True
        super(CustomMultiLossLayer, self).__init__(**kwargs)

    def build(self, input_shape=None):
        # initialise log_vars
        self.log_vars = []
        for i in range(self.nb_outputs):
            self.log_vars += [
                self.add_weight(
                    name="log_var" + str(i),
                    shape=(1,),
                    initializer=Constant(0.0),
                    trainable=True,
                )
            ]
        super(CustomMultiLossLayer, self).build(input_shape)
```

```
    def multi_loss(self, ys_true, ys_pred):
        assert (
            len(ys_true) == self.nb_outputs
            and len(ys_pred) == self.nb_outputs
        )
        loss = 0

        # for y_true, y_pred, log_var in zip(ys_true, ys_pred, self.log_vars):
        #    precision = K.exp(-log_var[0])
        #    loss += K.sum(precision * (y_true - y_pred)**2., -1) + log_var[0]

        precision = K.exp(-self.log_vars[0][0])
        loss += (
            precision * mean_absolute_error(ys_true[0], ys_pred[0])
            + self.log_vars[0][0]
        )
        precision = K.exp(-self.log_vars[1][0])
        loss += (
            precision
            * quaternion_mean_multiplicative_error(ys_true[1], ys_pred[1])
            + self.log_vars[1][0]
        )
        # loss += precision * quaternion_phi_4_error(ys_true[1], ys_pred[1]) +
self.log_vars[1][0]
        return K.mean(loss)

    def call(self, inputs):
        ys_true = inputs[: self.nb_outputs]
        ys_pred = inputs[self.nb_outputs :]
        loss = self.multi_loss(ys_true, ys_pred)
        self.add_loss(loss, inputs=inputs)
        # We won't actually use the output.
        # return K.concatenate(inputs, -1)
        return self.multi_loss(ys_true, ys_pred)


def create_pred_model_6d_quat(window_size=200, mag=False):
    x1 = Input((window_size, 3), name="x1")
    x2 = Input((window_size, 3), name="x2")
    x3 = Input((window_size, 3), name="x3")

    convA1 = Conv1D(128, 11)(x1)
    convA2 = Conv1D(128, 11)(convA1)
    poolA = MaxPooling1D(3)(convA2)
    convB1 = Conv1D(128, 11)(x2)
    convB2 = Conv1D(128, 11)(convB1)
    poolB = MaxPooling1D(3)(convB2)
    convC1 = Conv1D(128, 11)(x3)
    convC2 = Conv1D(128, 11)(convC1)
    poolC = MaxPooling1D(3)(convC2)

    if mag:
        AB = concatenate([poolA, poolB, poolC])
    else:
        AB = concatenate([poolA, poolB])

    lstm1 = Bidirectional(LSTM(128, return_sequences=True))(AB)
    drop1 = Dropout(0.25)(lstm1)
```

```
    lstm2 = Bidirectional(LSTM(128))(drop1)
    drop2 = Dropout(0.25)(lstm2)
    y1_pred = Dense(3)(drop2)
    y2_pred = Dense(4)(drop2)

    model = Model([x1, x2, x3], [y1_pred, y2_pred])

    model.summary()

    return model


def create_train_model_6d_quat(pred_model, window_size=200):
    x1 = Input((window_size, 3), name="x1")
    x2 = Input((window_size, 3), name="x2")
    x3 = Input((window_size, 3), name="x3")
    y1_pred, y2_pred = pred_model([x1, x2, x3])
    y1_true = Input(shape=(3,), name="y1_true")
    y2_true = Input(shape=(4,), name="y2_true")
    out = CustomMultiLossLayer(nb_outputs=2)(
        [y1_true, y2_true, y1_pred, y2_pred]
    )
    train_model = Model([x1, x2, x3, y1_true, y2_true], out)
    train_model.summary()
    return train_model
```

PDR Platform (driver for the input/output and data processing)3 Code

```
import numpy as np
import pandas as pd
from ahrs.filters import Mahony, Madgwick, EKF
import ahrs
import logging
import itertools

import matplotlib.pyplot as plt
from pyparsing import col
import PDR.util as util
from PDR.udp import UDP
from PDR.calibrate import Calibrate, sampler
from scipy import signal


class PDR:
    def __init__(
        self,
        filter=Mahony,
        frequency=100,
        lp=False,
        cutoff=5,
        order=10,
        port=1234,
        lattitude=None,
        longitude=None,
    ):
        self.logger = logging.getLogger(__name__)
        self.logger.setLevel(logging.DEBUG)
        self.formatter = logging.Formatter(
            "%(asctime)s - %(name)s - %(levelname)s - %(message)s"
        )
        self.frequency = frequency
        self.filter = filter(frequency=frequency)
        self.filter_update = (
            self.filter.update
            if filter.__name__ == "EKF"
            else self.filter.updateMARG
        )
        self.dt = self.filter.Dt
        self.lp = lp
        self.port = port

        # magnetic declination and magnetic norm
        self.magnetic_declination = 0
        self.norm = 55
        if lattitude and longitude:
            wmm = ahrs.utils.WMM()
```

```
            wmm.magnetic_field(lattitude, longitude, 0)["I"]
            self.norm = wmm.magnetic_elements["I"]
            self.magnetic_declination = wmm.magnetic_elements["D"]
        self.udp_init()

        self.Q = np.array([1.0, 0.0, 0.0, 0.0])

        try:
            self.calib = pd.read_csv(
                "calib.csv", sep="\t", header=[0, 1], index_col=[0, 1]
            ).T
        except:
            self.logger.warning("No calibration file found")
            columns = tuple(
                itertools.product(("accel", "gyro", "mag"), ("x", "y", "z"))
            )
            index = tuple(itertools.product(("A"), ("x", "y", "z"))) + (
                ("b", "b"),
            )
            data = np.array([np.eye(3)] * 3).reshape(9, 3).T
            data = np.append(data, np.zeros(shape=(1, 9)), axis=0)
            self.calib = pd.DataFrame(data, columns=columns, index=index)

        self.calib_b = self.calib.loc["b", "b"]
        self.calib_A = self.calib.loc["A"]

        self.lp_init(cutoff=cutoff, order=order, fs=frequency)

        self.data = pd.DataFrame()
        self.capture = False

    def udp_init(self):
        self.udp = UDP()

    def lp_init(self, cutoff=5, order=10, fs=200):
        b, a = signal.butter(
            order, cutoff, fs=fs, btype="lowpass", analog=False
        )
        self.lfilter = {
            col: util.LiveLFilter(b, a) for col in self.calib.columns
        }

    def update(self):
        data = self.udp.read()
        q = []
        data -= self.calib_b
        data["mag"] = (self.calib_A["mag"] @ data["mag"].T).T
        # data['accel'] = (self.calib_A['accel'] @ data['accel'].T).T
        if self.lp:
            data = data.agg(self.lfilter)
```

```
        for idx, v in data.iterrows():
            self.Q = self.filter_update(
                self.Q, v["gyro"], v["accel"], v["mag"]
            )
            q.append(self.Q)
        data["Q"] = q
        if self.capture:
            self.data = pd.concat([self.data, data], axis=0)

        return self.Q

    @staticmethod
    def quat_to_rot_mat(q):
        r00 = 2 * (q[0] * q[0] + q[1] * q[1]) - 1
        r01 = 2 * (q[1] * q[2] - q[0] * q[3])
        r02 = 2 * (q[1] * q[3] + q[0] * q[2])
        r10 = 2 * (q[1] * q[2] + q[0] * q[3])
        r11 = 2 * (q[0] * q[0] + q[2] * q[2]) - 1
        r12 = 2 * (q[2] * q[3] - q[0] * q[1])
        r20 = 2 * (q[1] * q[3] - q[0] * q[2])
        r21 = 2 * (q[2] * q[3] + q[0] * q[1])
        r22 = 2 * (q[0] * q[0] + q[3] * q[3]) - 1

        return np.array([[r00, r01, r02], [r10, r11, r12], [r20, r21, r22]])

    def calibrate_gyro(self):
        self.udp.close()
        calib = Calibrate(
            sampler=sampler.linear, N_single=1000, sensor="gyro", lim=0.05
        )
        calib.run()
        self.calib_b["gyro"] = calib.calib_b
        self.save_calib()
        self.udp_init()
        return calib.calib_b

    def calibrate_accel(self):
        self.udp.close()
        calib = Calibrate(
            norm=9.8,
            sampler=sampler.single,
            N_single=100,
            sensor="accel",
            lim=15,
        )
        calib.run()
        self.calib_b["accel"] = calib.calib_b
        self.calib_A["accel"] = calib.calib_A
        self.save_calib()
        self.udp_init()
```

```
        return calib.calib_b, calib.calib_A

    def calibrate_mag(self):
        self.udp.close()
        calib = Calibrate(
            norm=self.norm, sampler=sampler.continuous, sensor="mag", lim=200
        )
        calib.run()
        self.calib_b["mag"] = calib.calib_b
        self.calib_A["mag"] = calib.calib_A
        self.save_calib()
        self.udp_init()
        return calib.calib_b, calib.calib_A

    def save_calib(self):
        self.calib.T.to_csv("calib.csv", sep="\t")
        self.calib_b = self.calib.loc["b", "b"]
        self.calib_A = self.calib.loc["A"]

    def plot_path(self):
        df["qtrm"] = df["Q"].apply(self.quat_to_rot_mat)
        df = df.apply(lambda x: x["qtrm"] @ x["accel"], axis=1).copy()
        # velocity
        df.columns = tuple(itertools.product(["accel"], ["x", "y", "z"]))
        sf = df.shift(1).apply(lambda x: x * self.dt)
        sf.columns = (("vel", "x"), ("vel", "y"), ("vel", "z"))
        sf = sf.cumsum()
        df = pd.concat([df, sf], axis=1)
        df.fillna(0, inplace=True)
        # position
        sf = pd.DataFrame(
            0,
            index=df.index,
            columns=(("pos", "x"), ("pos", "y"), ("pos", "z")),
        )
        df = pd.concat([df, sf], axis=1)
        df["pos"] = (
            df["vel"].shift(1) * self.dt
            + df["accel"].shift(1) * (self.dt**2) / 2
        )
        df.fillna(0, inplace=True)
        df["pos"] = df["pos"].cumsum()
        # plot
        fig, ax = plt.subplots(1, 3, figsize=(15, 10), tight_layout=True)
        ax.flatten()

        ax[0].plot(df["accel"]["x"], df["accel"]["y"], label="accel")
        ax[0].set_title("acceleration")
        ax[0].axis("equal")
        ax[0].grid()
```

```
ax[1].plot(df["vel"]["x"], df["vel"]["y"], label="vel")
ax[1].set_title("velocity")
ax[1].axis("equal")
ax[1].grid()

ax[2].plot(df["pos"]["x"], df["pos"]["y"], label="pos")
ax[2].set_title("position")
ax[2].axis("equal")
ax[2].grid()

plt.show()
```

```
import numpy as np
import pandas as pd
import pygame
import math
from OpenGL.GL import *
from OpenGL.GLU import *
from pygame.locals import *
from PDR.pdr import PDR
from PDR.zupt import zupt
import time


class Visulize:
    def __init__(self, width, height, pdr):
        self.width = width
        self.height = height

        self.frames = 0
        self.ticks = pygame.time.get_ticks()

        self.capture = False
        self.running = False

        self.pdr = pdr

        self.calib_menu = False

        self.timings = pd.DataFrame()

    def run(self):
        self.init()
        self.running = True
        while self.running:
            self.event = pygame.event.poll()
            self.event_handler()
            if pdr.filter.__class__.__name__ == "Ahrs":
                self.pdr.update_madgwick_fusion()
            else:
                self.pdr.update()
            self.draw()
            pygame.display.flip()
            self.frames += 1

    def init(self):
        pygame.init()
        video_flags = OPENGL | DOUBLEBUF
        self.screen = pygame.display.set_mode(
            (self.width, self.height), video_flags
        )
        pygame.display.set_caption("IMU orientation visualization")
        self.resizewin()
        glShadeModel(GL_SMOOTH)
        glClearColor(0.0, 0.0, 0.0, 0.0)
        glClearDepth(1.0)
        glEnable(GL_DEPTH_TEST)
        glDepthFunc(GL_LEQUAL)
        glHint(GL_PERSPECTIVE_CORRECTION_HINT, GL_NICEST)
```

```
    def event_handler(self):
        if self.event.type == QUIT or (
            self.event.type == KEYDOWN and self.event.key == K_ESCAPE
        ):
            self.running = False
            if len(self.pdr.data):
                self.pdr.data.to_csv("data.csv")
                # self.pdr.plot_path()
                zupt(
                    df=self.pdr.data,
                    fn="data",
                    sample_rate=self.pdr.frequency,
                    zupt_tresh=5,
                    margin=0.12,
                    debug=False,
                )
        if self.event.type == KEYDOWN and self.event.key == K_f:
            pygame.display.toggle_fullscreen()
        if self.event.type == KEYDOWN and self.event.key == K_c:
            self.calib_menu = ~self.calib_menu
        if self.event.type == KEYDOWN and self.event.key == K_d:
            self.capture = ~self.capture
            self.pdr.capture = self.capture
        if self.event.type == KEYDOWN and self.event.key == K_r:
            self.frames = 0
            self.ticks = pygame.time.get_ticks()
        if self.calib_menu:
            if self.event.type == KEYDOWN and self.event.key == K_1:
                self.pdr.calibrate_mag()
                self.calib_menu = False
            elif self.event.type == KEYDOWN and self.event.key == K_2:
                self.pdr.calibrate_gyro()
                self.calib_menu = False
            elif self.event.type == KEYDOWN and self.event.key == K_3:
                self.pdr.calibrate_accel()
                self.calib_menu = False
        if self.event.type == KEYDOWN and self.event.key == K_UP:
            self.pdr.filter.gain = 5
        if self.event.type == KEYDOWN and self.event.key == K_DOWN:
            self.pdr.filter.gain = 0.033
        if self.event.type == KEYDOWN and self.event.key == K_BACKSPACE:
            self.pdr.lp = not self.pdr.lp

    def resizewin(self):
        if self.height == 0:
            self.height = 1
        glViewport(0, 0, self.width, self.height)
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        gluPerspective(45, 1.0 * self.width / self.height, 0.1, 100.0)
        glMatrixMode(GL_MODELVIEW)
        glLoadIdentity()

    def draw(self):
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        glLoadIdentity()
        glTranslatef(0, 0.0, -7.0)
        self.drawText(
```

```
            (-2.6, 1.8, 2),
            f"filter Module {str(self.pdr.filter.__class__.__name__)}",
            18,
        )
        self.drawText(
            (-2.6, 1.7, 2),
            f"PDR config gain {self.pdr.filter.gain if
(self.pdr.filter.__class__.__name__ == 'Madgwick') else None} lp {self.pdr.lp}",
            16,
        )
        if not self.calib_menu:
            self.drawText(
                (-2.6, -2, 2),
                "Press Escape to Exit , C to Calibrattion Settings, D to
capture",
                16,
            )
        else:
            self.drawText(
                (-2.6, -2, 2),
                "Press 1 for magnetometer calibration, 2 for gyro calibration, 3
for accel calibration",
                16,
            )

        self.drawText(
            (-2.6, -1.8, 2),
            f"Q: {self.pdr.Q[0]:.3f}, x: {self.pdr.Q[1]:.3f}, y:
{self.pdr.Q[2]:.3f}, z: {self.pdr.Q[3]:.3f}",
            16,
        )
        self.drawText(
            (2, 1.8, 2),
            f"Recording {'X' if self.capture else ' '}",
            16,
            color=(0, 255, 0, 255) if self.capture else (255, 0, 0, 255),
        )
        self.drawText(
            (2, 1.9, 2),
            f"fps: {self.frames / ((pygame.time.get_ticks() - self.ticks) /
1000.0):.2f}",
            16,
        )
        if self.capture:
            accel = (
                self.pdr.quat_to_rot_mat(self.pdr.Q)
                @ self.pdr.data["accel"].iloc[-1].to_numpy()
            )
            self.drawText(
                (-2.6, -1.9, 2),
                f"Q accel : x: {accel[0]:.1f}, y: {accel[1]:.1f}, z:
{accel[2]:.1f}",
                16,
            )
            accel_s = np.array(["0", "0", "0"])
            for i in range(len(accel)):
                if accel[i] > 1:
                    accel_s[i] = "+"
```

```
                elif accel[i] < -1:
                    accel_s[i] = "-"
            self.drawText(
                (1, -1.9, 2),
                f"x: {accel_s[0]}, y: {accel_s[1]}, z: {accel_s[2]}",
                16,
            )

        # print(self.pdr.Q)
        if self.pdr.filter.__class__.__name__ in ["Mahony", "EKF"]:
            glRotatef(
                2 * math.acos(self.pdr.Q[0]) * 180.00 / math.pi,
                self.pdr.Q[1],
                1 * self.pdr.Q[3],
                -1 * self.pdr.Q[2],
            )
        elif self.pdr.filter.__class__.__name__ in ["Madgwick", "Ahrs"]:
            glRotatef(
                2 * math.acos(self.pdr.Q[0]) * 180.00 / math.pi,
                -1 * self.pdr.Q[1],
                -1 * self.pdr.Q[3],
                1 * self.pdr.Q[2],
            )
        self.draw_box()

    @staticmethod
    def drawText(position, textString, size, color=(255, 255, 255, 255)):
        font = pygame.font.SysFont("Courier", size, True)
        textSurface = font.render(textString, True, color, (0, 0, 0, 255))
        textData = pygame.image.tostring(textSurface, "RGBA", True)
        glRasterPos3d(*position)
        glDrawPixels(
            textSurface.get_width(),
            textSurface.get_height(),
            GL_RGBA,
            GL_UNSIGNED_BYTE,
            textData,
        )

    @staticmethod
    def quat_to_ypr(q):
        yaw = math.atan2(
            2.0 * (q[1] * q[2] + q[0] * q[3]),
            q[0] * q[0] + q[1] * q[1] - q[2] * q[2] - q[3] * q[3],
        )
        pitch = -math.asin(2.0 * (q[1] * q[3] - q[0] * q[2]))
        roll = math.atan2(
            2.0 * (q[0] * q[1] + q[2] * q[3]),
            q[0] * q[0] - q[1] * q[1] - q[2] * q[2] + q[3] * q[3],
        )
        pitch *= 180.0 / math.pi
        yaw *= 180.0 / math.pi
        # yaw   -= self.pdr.magnetic_declination
        roll *= 180.0 / math.pi
        return [yaw, pitch, roll]

    @staticmethod
    def quat_to_rot_mat(q):
```

```
        r00 = 2 * (q[0] * q[0] + q[1] * q[1]) - 1
        r01 = 2 * (q[1] * q[2] - q[0] * q[3])
        r02 = 2 * (q[1] * q[3] + q[0] * q[2])
        r10 = 2 * (q[1] * q[2] + q[0] * q[3])
        r11 = 2 * (q[0] * q[0] + q[2] * q[2]) - 1
        r12 = 2 * (q[2] * q[3] - q[0] * q[1])
        r20 = 2 * (q[1] * q[3] - q[0] * q[2])
        r21 = 2 * (q[2] * q[3] + q[0] * q[1])
        r22 = 2 * (q[0] * q[0] + q[3] * q[3]) - 1

        return np.array([[r00, r01, r02], [r10, r11, r12], [r20, r21, r22]])

    @staticmethod
    def draw_box():
        glBegin(GL_QUADS)

        glColor3f(0.0, 1.0, 0.0)
        glVertex3f(2.0, 0.2, -1.0)
        glVertex3f(-2.0, 0.2, -1.0)
        glVertex3f(-2.0, 0.2, 1.0)
        glVertex3f(2.0, 0.2, 1.0)

        glColor3f(1.0, 0.5, 0.0)
        glVertex3f(2.0, -0.2, 1.0)
        glVertex3f(-2.0, -0.2, 1.0)
        glVertex3f(-2.0, -0.2, -1.0)
        glVertex3f(2.0, -0.2, -1.0)

        glColor3f(1.0, 0.0, 0.0)
        glVertex3f(2.0, 0.2, 1.0)
        glVertex3f(-2.0, 0.2, 1.0)
        glVertex3f(-2.0, -0.2, 1.0)
        glVertex3f(2.0, -0.2, 1.0)

        glColor3f(1.0, 1.0, 0.0)
        glVertex3f(2.0, -0.2, -1.0)
        glVertex3f(-2.0, -0.2, -1.0)
        glVertex3f(-2.0, 0.2, -1.0)
        glVertex3f(2.0, 0.2, -1.0)

        glColor3f(0.0, 0.0, 1.0)
        glVertex3f(-2.0, 0.2, 1.0)
        glVertex3f(-2.0, 0.2, -1.0)
        glVertex3f(-2.0, -0.2, -1.0)
        glVertex3f(-2.0, -0.2, 1.0)

        glColor3f(1.0, 0.0, 1.0)
        glVertex3f(2.0, 0.2, -1.0)
        glVertex3f(2.0, 0.2, 1.0)
        glVertex3f(2.0, -0.2, 1.0)
        glVertex3f(2.0, -0.2, -1.0)
        glEnd()


from ahrs.filters import Mahony, Madgwick, EKF
from ahrs.utils import WMM

if __name__ == "__main__":
```

```
    pdr = PDR(lp=False, filter=Madgwick, frequency=200)

    # EKF
    # wmm = WMM()
    # wmm.magnetic_field(35.74276511014527, 51.49721575854581, 1.1)
    # ml = wmm.magnetic_elements
    # pdr.filter = EKF(frequency=200, magnetic_ref=[ml['X'], ml['Y'], ml['Z']])
#, noises=[0.00000035, 0.0003, 0.5])
    # pdr.filter.gain = 0.5
    # print(pdr.filter.noises)

    vis = Visulize(1280, 720, pdr)
    vis.run()
```

Abstract

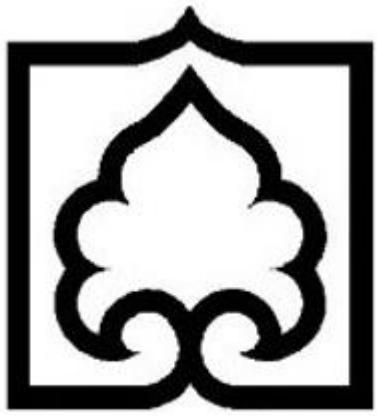
Pedestrian Dead Reckoning (PDR) is a crucial aspect of indoor localization systems, relying on the integration of inertial sensor data to estimate the user's position and orientation. In this paper, we present a comparative study of two distinct approaches to PDR: a neural network-based solution and a Zero Velocity Potential Update (ZUPT) method.

The neural network approach involves the utilization of deep learning techniques to process raw sensor data from Inertial Measurement Units (IMU) and Magnetic, Angular Rate, and Gravity (MARG) sensors. The network transforms this information into position and quaternion representations, subsequently computing the final relative position.

In contrast, the ZUPT method leverages gaiting techniques to isolate individual steps, integrating accelerometer data during these zero-velocity intervals to determine relative position changes. The method further incorporates Altitude and Heading Reference Systems (AHRS) to capture heading information, culminating in the calculation of the relative trajectory and position.

Our comparative analysis evaluates the performance of these two approaches in terms of accuracy, robustness, and computational efficiency. The findings contribute valuable insights into the strengths and limitations of each methodology, providing a foundation for optimizing PDR solutions in diverse indoor environments. Ultimately, this research aims to enhance the understanding of PDR techniques and guide the development of effective indoor localization systems.

Keywords: Pedestrian Dead Reckoning (PDR), Attitude and Heading Reference System (AHRS), Zero Velocity Potential Update (ZUPT), Inertial Measurement Unit (IMU), Deep Neural Networks (DNN), Quaternions, Indoor Positioning System (IPS)



University of Zanjan

Faculty of Engineering
Department of Industrial Engineering

A Thesis Submitted in partial fulfillment of the requirements for the degree of Bachelor of Science in Industrial Engineering

Pedestrian Dead Reckoning

By:

Arad Arang

Supervisor:

Dr. Mehdrad Babazadeh

[Feb 2024]