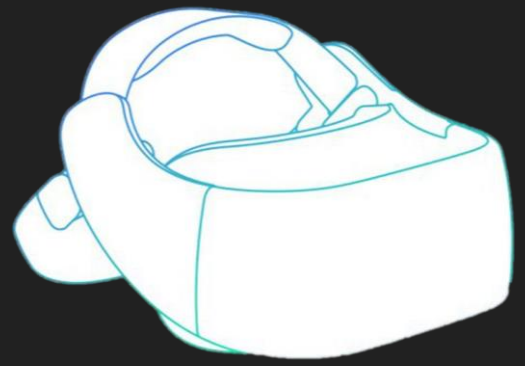
Reconstructing locomotion in Virtual Reality (VR) from Walking-In-Place (WIP) motion : an IMU-based approach

JiGang Kim

Motivation

- *Develop a low-cost (IMU), natural interface (WIP) for navigating Virtual Environments (VE)*
- *Synthesize a personalized natural motion from WIP with gait tracking results of user's normal walking.*

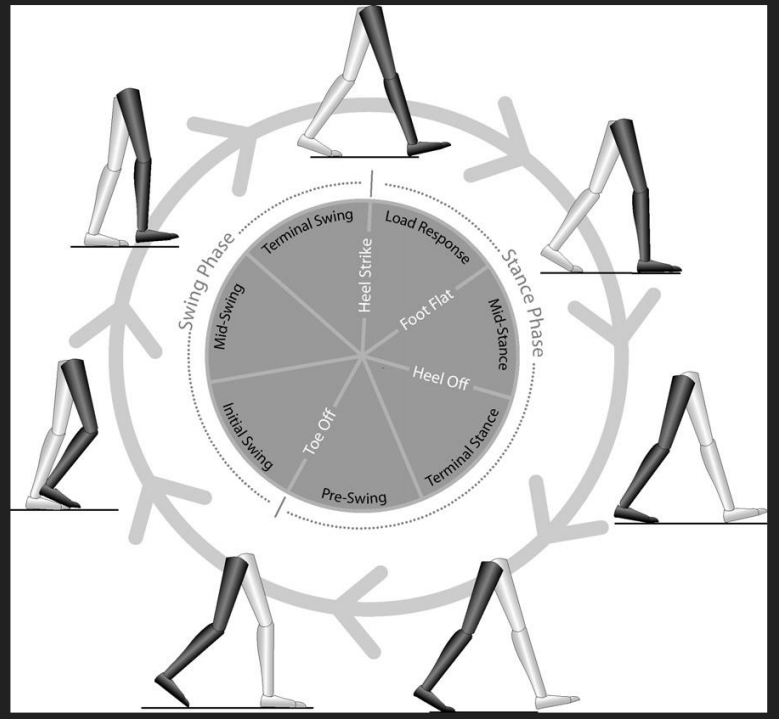
- *Performance Criterion :*
  - *Latency : rapid transition between stationary and moving state*
  - *Smoothness : no sudden jerks in the frame*
  - *Precision : sensitivity/accuracy of motion*
  - *Speed & Efficiency : little effort needed to complete a task*



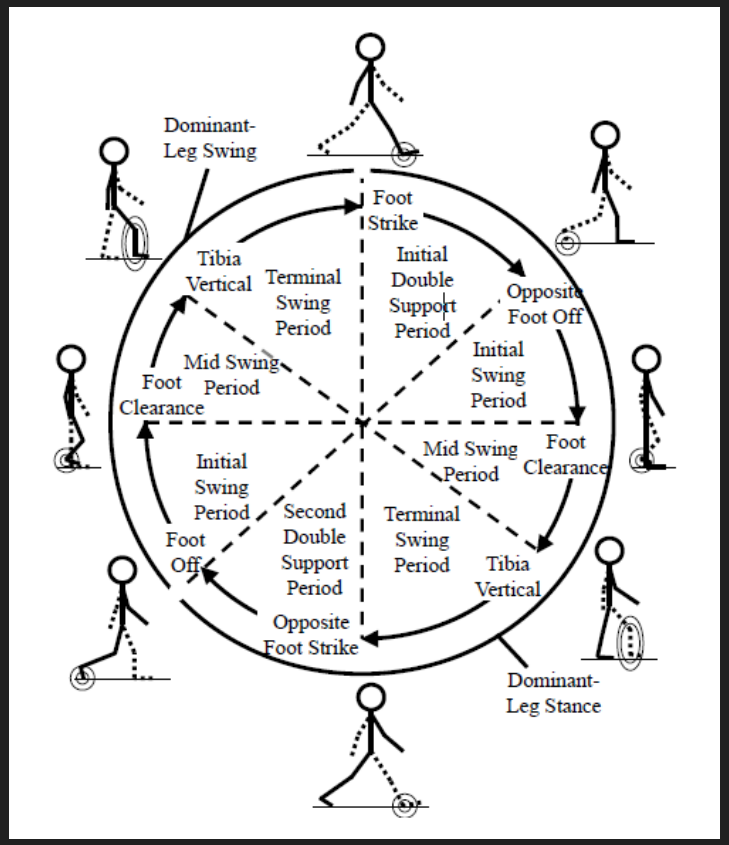
Introduction & Past Literatures

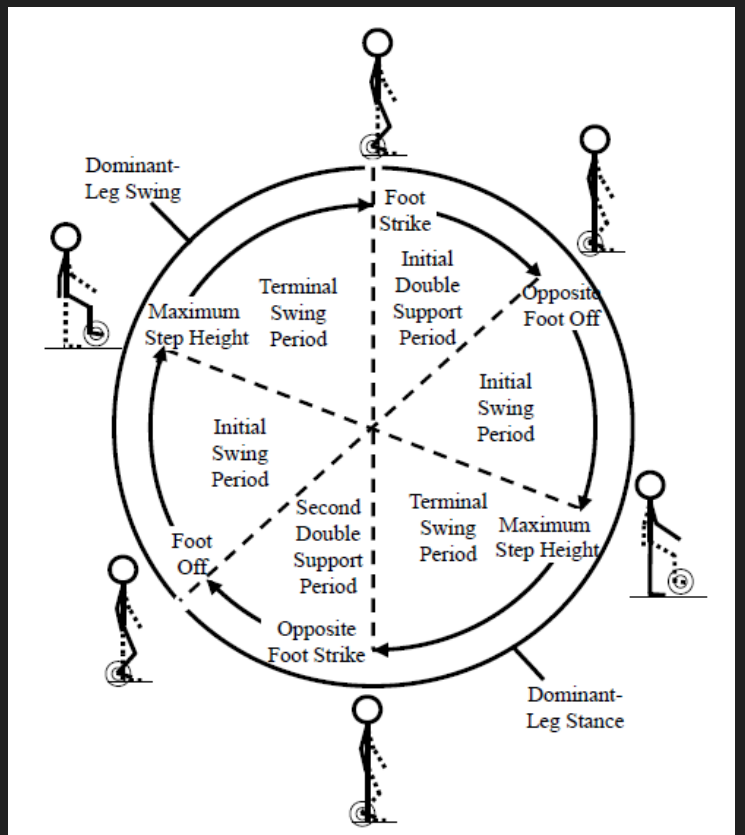
Gait Cycle Diagram

Typical gait cycle

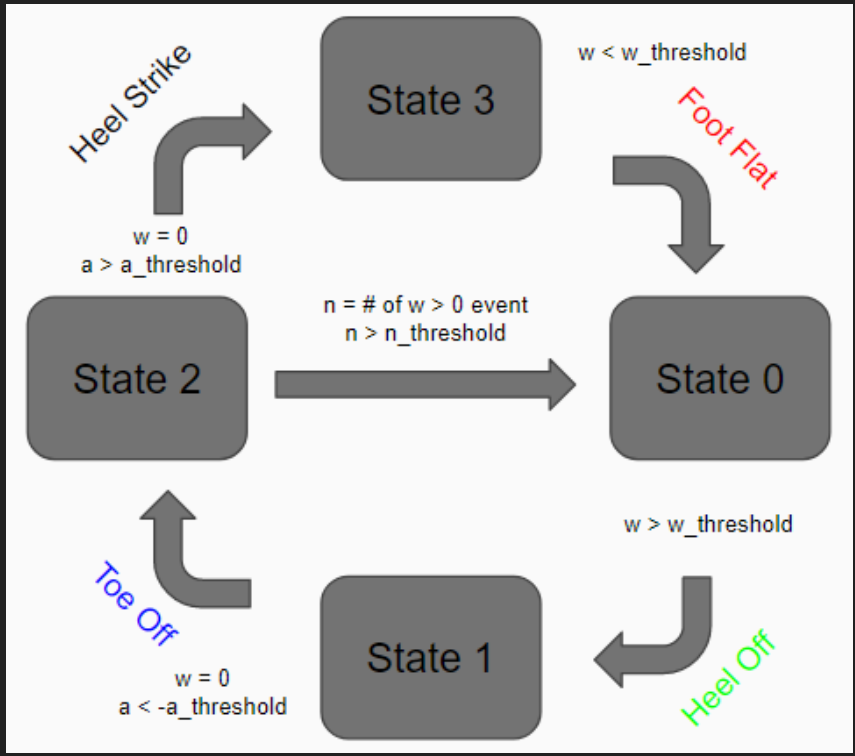


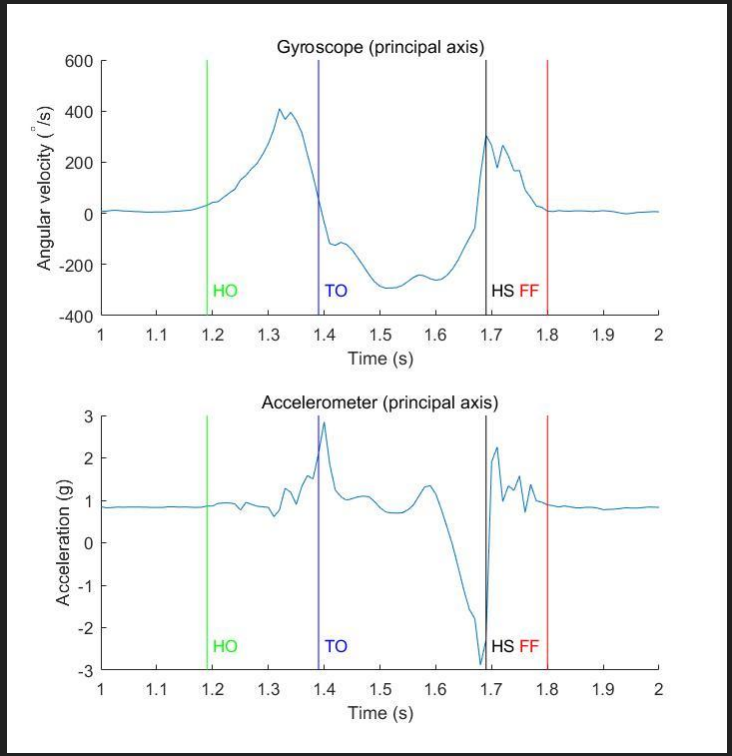
Real walking vs. WIP





Gait Event Detection

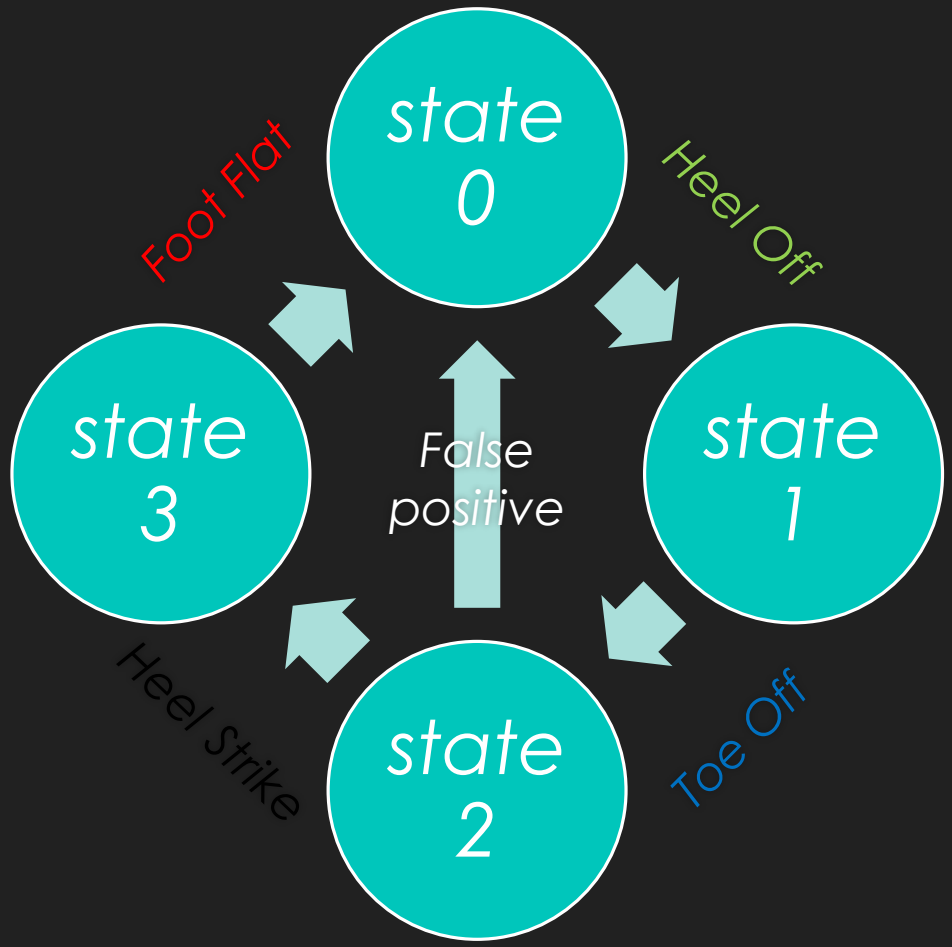




Normal walking
- *Finite State Machine (FSM) coupled with threshold ftn.*
- *4 states and 5 transitions*
- *gyro-based approach*
- *offline analysis*

WIP
- *Binary State Machine (BSM)*
- *acc & gyro-based threshold ftn.*
- *real-time detection*

Gait Event Detection (revised)



Gait Tracking

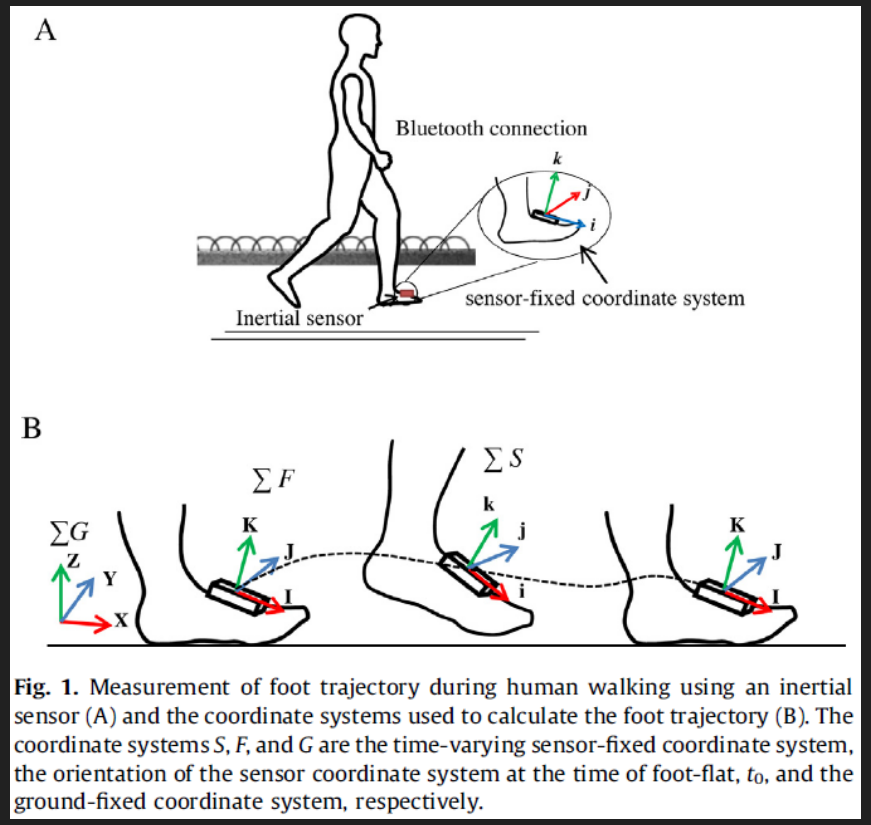
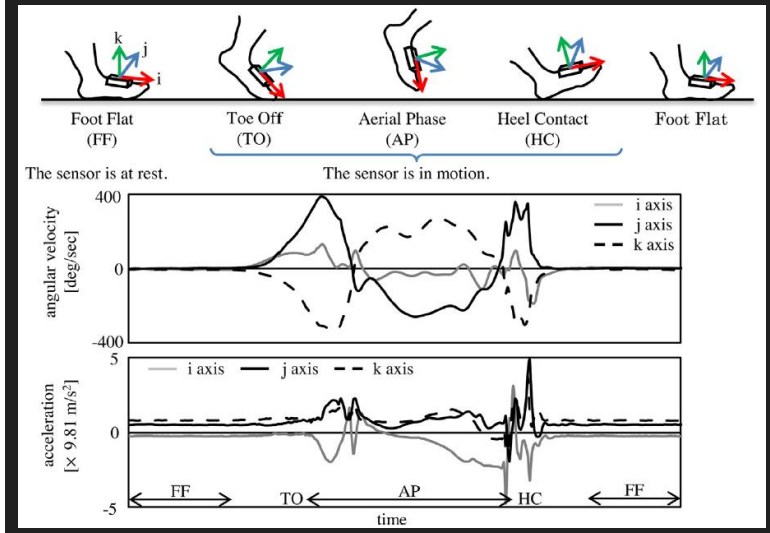


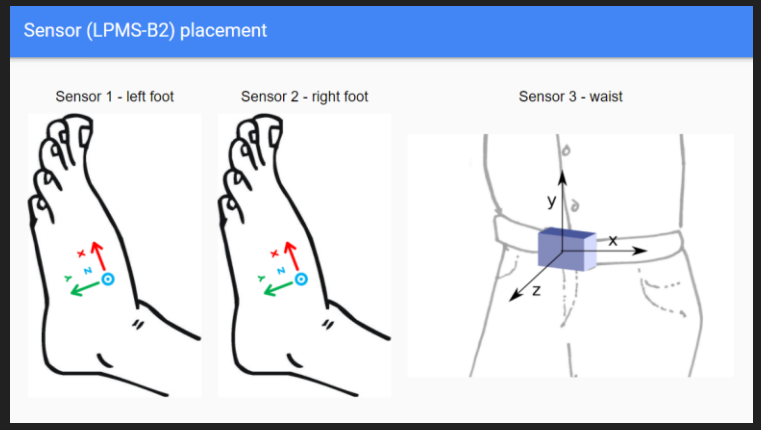
Fig. 1. Measurement of foot trajectory during human walking using an inertial sensor (A) and the coordinate systems used to calculate the foot trajectory (B). The coordinate systems *S*, *F*, and *G* are the time-varying sensor-fixed coordinate system, the orientation of the sensor coordinate system at the time of foot-flat, t_0, and the ground-fixed coordinate system, respectively.

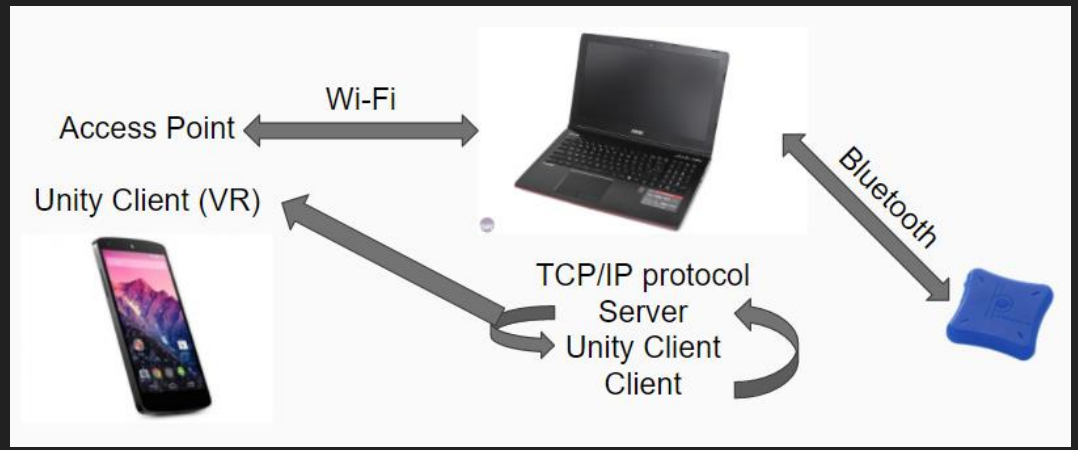


Normal walking
- Kitagawa 2016, Gait & Posture
- extract gait parameters and the relation between vertical & transverse velocity

WIP
- ZUPT (zero velocity update)
- velocity drift rate compensation
- position drift compensation (flat floor assumption)

Locomotion Synthesis in VR





Past Literature
- *Discrete : step detection → position update*
- *Continuous : velocity update*
  - *LLCM : left/right heel speed superposition*
  - *GUD, VR-STEP : step frequency*
  - *SAS : heel speed + maximum heel height*

System Configuration
- *LPMS-B2 x 2*
- *VE interface using Unity*
- *VR headset using Google Cardboard*
- *TCP/IP communication protocol*

LLCM-WIP (Low-Latency, Continuous-Motion), 2008

- *Emphasis on low latency and smoothness*
- *Direct mapping from heel speed to virtual locomotion speed*
- *Implemented with magnetic foot trackers and chest-orientation tracker*
- *Implementation prone to magnetic disturbances*

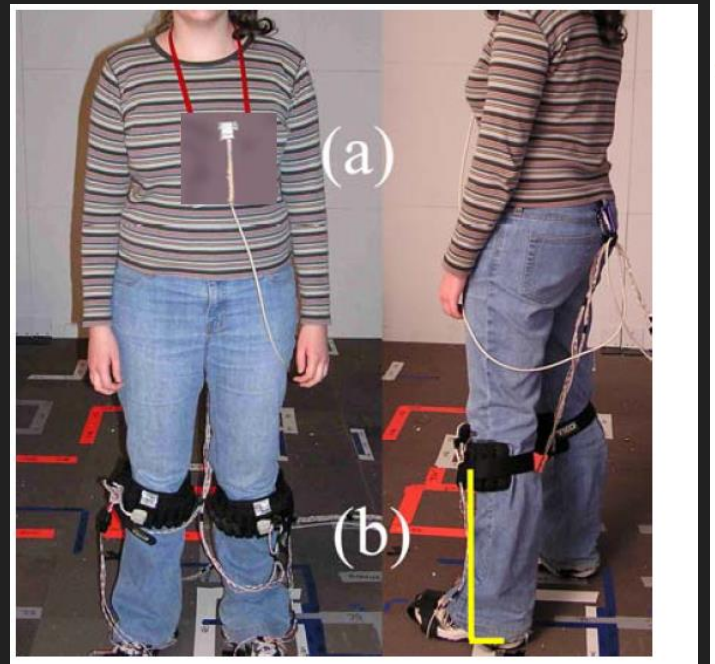
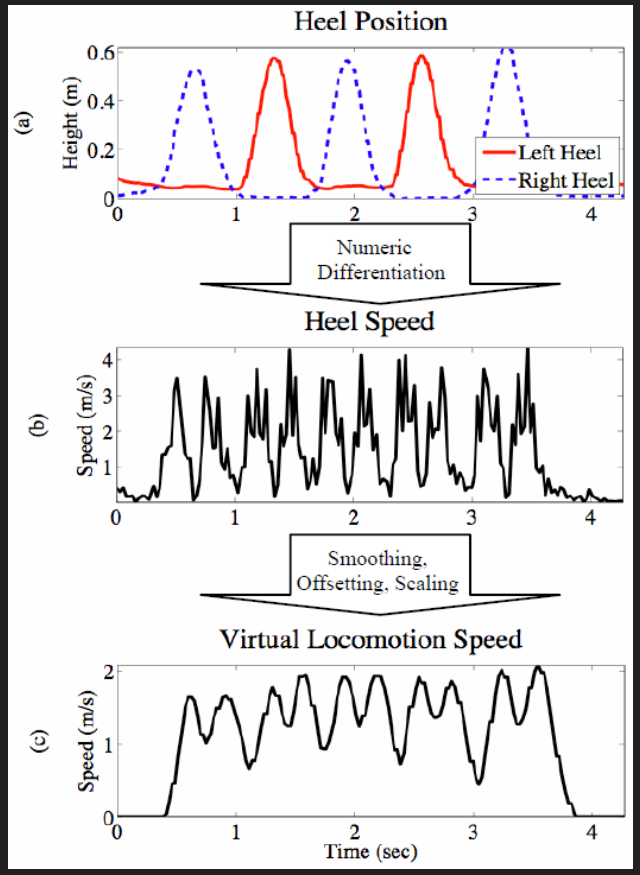


Figure 1: The LLCM-WIP hardware. (a) Chest-orientation tracker (b) Foot trackers (rigid offset from mount-point to heel shown),



GUD-WIP (Gait-Understanding-Driven), 2010

- *Emphasis on natural locomotion synthesis with understandings of gait characteristics*
- *Gait event detection using state machines*
- *Implemented with optical tracker*

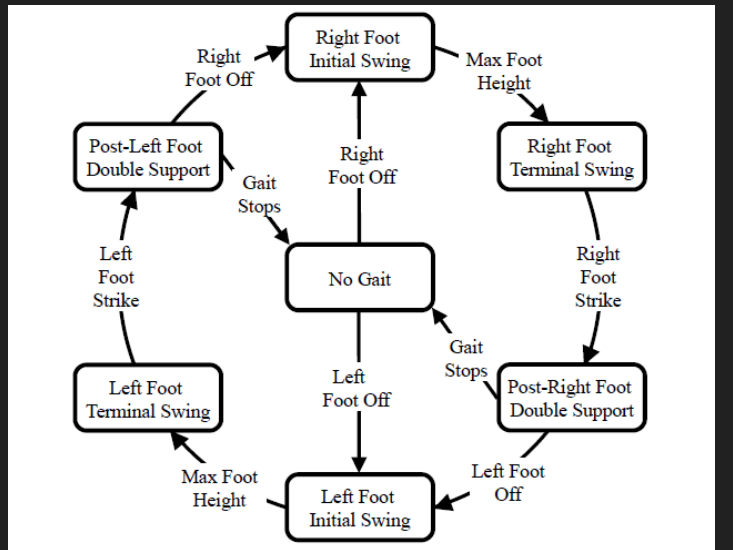


Figure 6: GUD WIP state machine. The current state is maintained until a state-exit criterion (shown on transitions) is fulfilled.

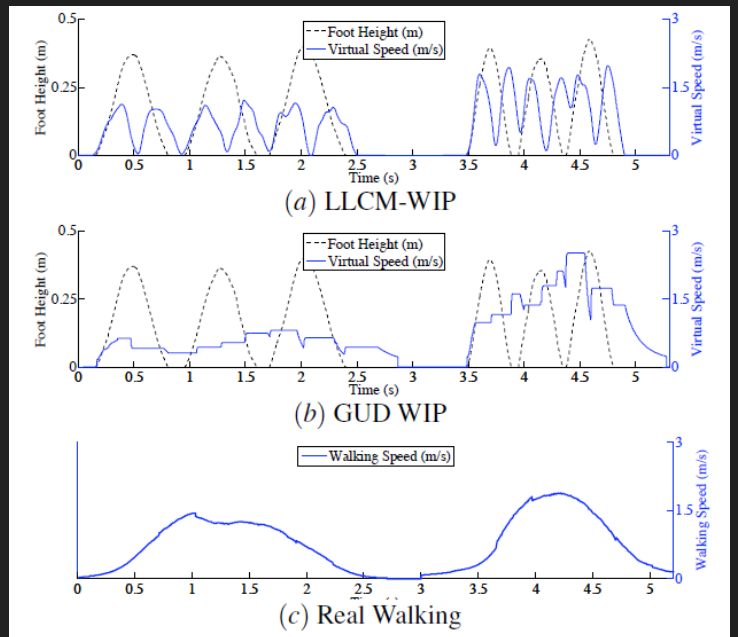


Figure 1: Resulting motion (thicker solid line, right y-axis) from (a) LLCM-WIP [2], (b) GUD WIP, and (c) Real Walking. In this example, the two WIP systems are driven by the same in-place steps (dashed line, left y-axis). The Real-Walking motion is caused by steps taken at approximately the same frequencies as the in-place steps for the WIP systems. Note LLCM-WIP's resulting speed varies considerably. GUD WIP's resulting speed is far steady – similar to Real Walking.

SAS-WIP (Speed-Amplitude-Supported), 2013

- *Emphasis on reducing user fatigue and better control of walking speed*
- *Improved performance (faster long distance travel and more precise short distance travel) compared to GUD-WIP*
- *Implemented with optical tracker*

4.4 Apparatus

This experiment was developed in the same laboratory where the exploratory study occurred. The output of the virtual environment was displayed on a large-scale screen (4x2.25 square meters) as shown in Fig. 4 (bottom picture). The participants are positioned approximately 2.0 m from the screen, within a 3x3 meter square area. The foot motion capture was supported by an optical tracker system [11], which determines the heel position for each foot.

Fig. 4. The 3D virtual environment and the experiment apparatus (from left to right)

VR-STEP (Unity Asset Store), 2016

- *Virtual locomotion synthesis from resulting head bobbing motion during WIP*
- *Gaze-directed locomotion*
- *Implemented with onboard (smartphone) IMU only*

Evaluation & Result

- *Evaluation*
  - *Qualitative evaluation (user survey)*
  - *Quantitative evaluation (time to completion for a given task)*

- *Result*
  - *Pedestrian Dead Reckoning (PDR) + IMU-based WIP VR interface*

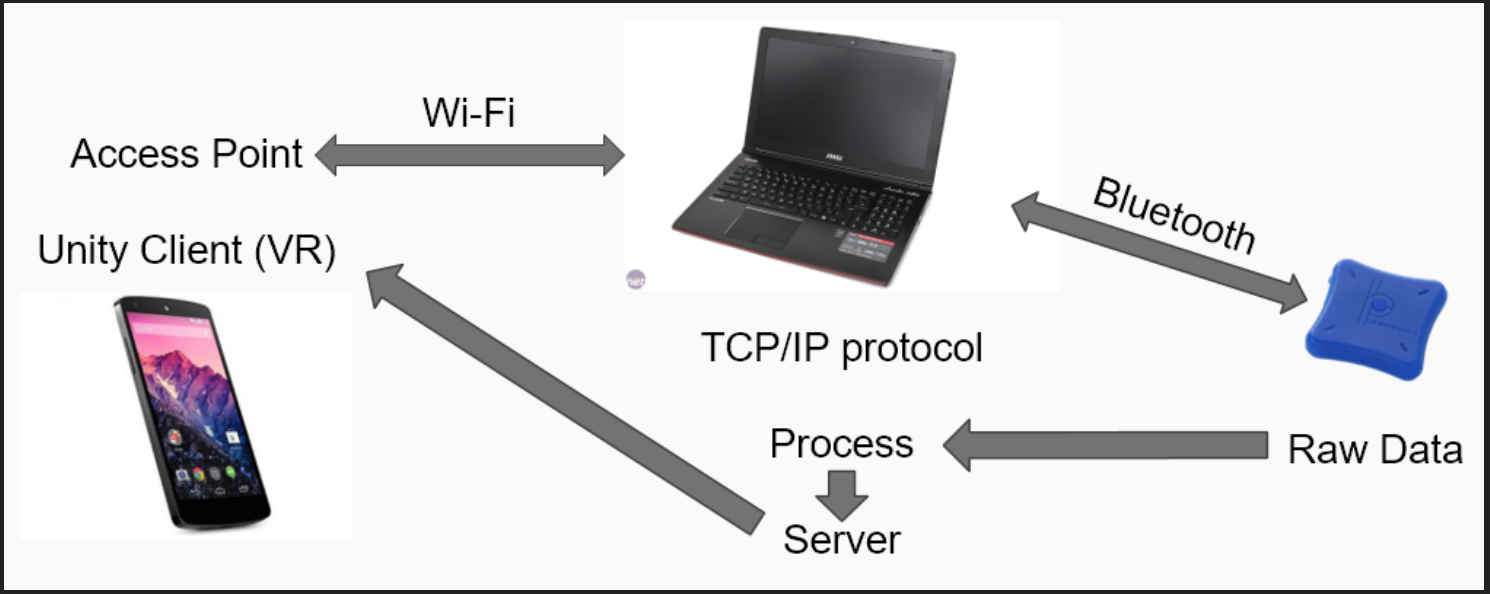
    → personalized & natural interface for navigating VE
  - *Comparison with other similar solutions (e.g. VR-STEP, pressure mat based WIP VR interface)*

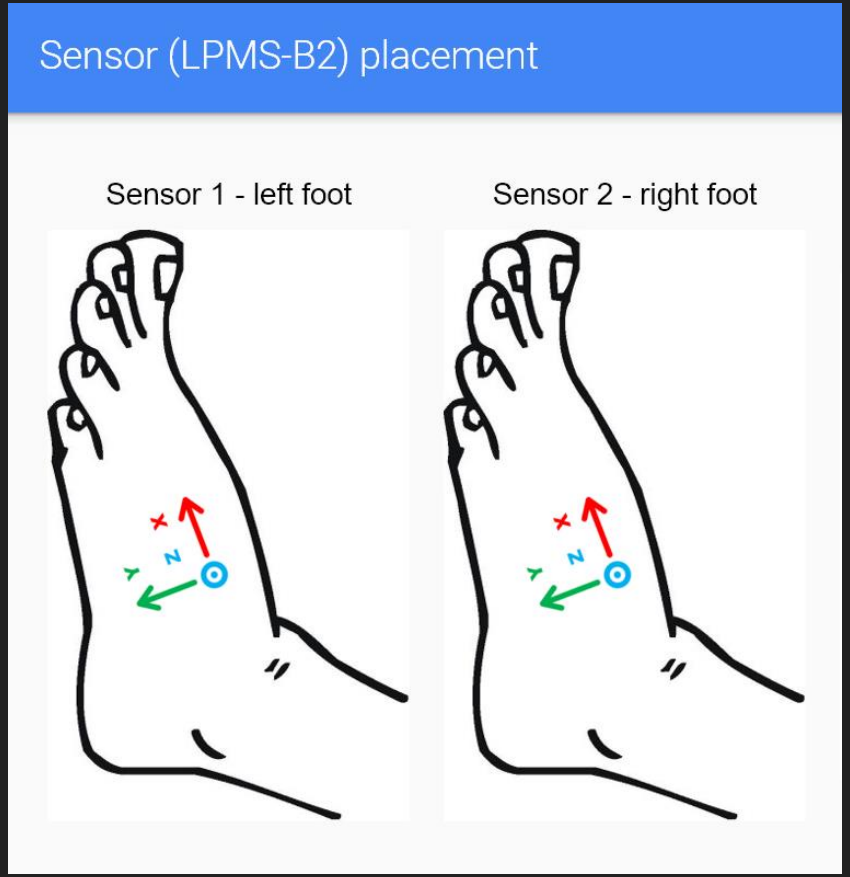
Methodology

System Overview

- *Process:*
  - *calibrate, kita (offline version of 'calibrate') : obtain gait parameters (Kitagawa, 2016)*
  - *run, playback (offline version of 'run') : track WIP motion and generate locomotion*
- *Language:*
  - *C# (server & unity client), C++ (processing client)*
- *Platform & IDE:*
  - *Windows 10 & Visual Studio 2015, Unity 5.6.1f1*
  - *Android 4.4 KitKat or above (requires Google Cardboard)*
- *Communication:*
  - *TCP/IP socket communication (server)*
  - *Bluetooth (LPMS-B2 IMU sensor)*

System Overview (cont.)





Processing Client (class overview)

- *LpmsSensorI, LpmsSensorManagerI: interface for LPMS sensors*
- *ImuData: struct for handling LPMS sensor data*
- *csv: handles csv files, supports reading/writing of csv files in (header, fields) format*
- *LowPassFilter: implementation of IIR filter*
- *Kitagawa2016: implementation of Kitagawa's gait feature extraction algorithm*
- *MadgwickAHRS: supports IMU-based AHRS, ported from C and repackaged into C++ class*
- *WIP: tracks state, velocity, position, attitude of WIP motion*
- *WIPmanager: generates locomotion from WIP motion of feet*

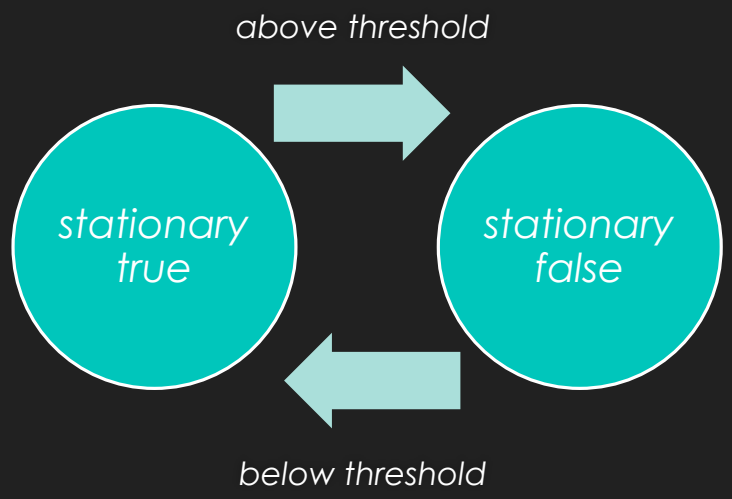
WIP states (stationary, state_vv)

- *stationary: simple thresholding (acceleration, angular velocity) to determine whether stationary or not (binary state, true/false)*
- *state_vv (vertical velocity): 5-state FSM (state 0-4) from functional analysis of WIP vertical velocity*
  - *state 0: stationary true*
  - *state 1: between foot-off (FO) and maximum upwards vertical velocity (maxUp)*
  - *state 2: between maxUp and zero-passing of vertical velocity (zeroPass)*
  - *state 3: between zeroPass and maximum downwards vertical velocity (maxDown)*
  - *state 4: between maxDown until stationary true*

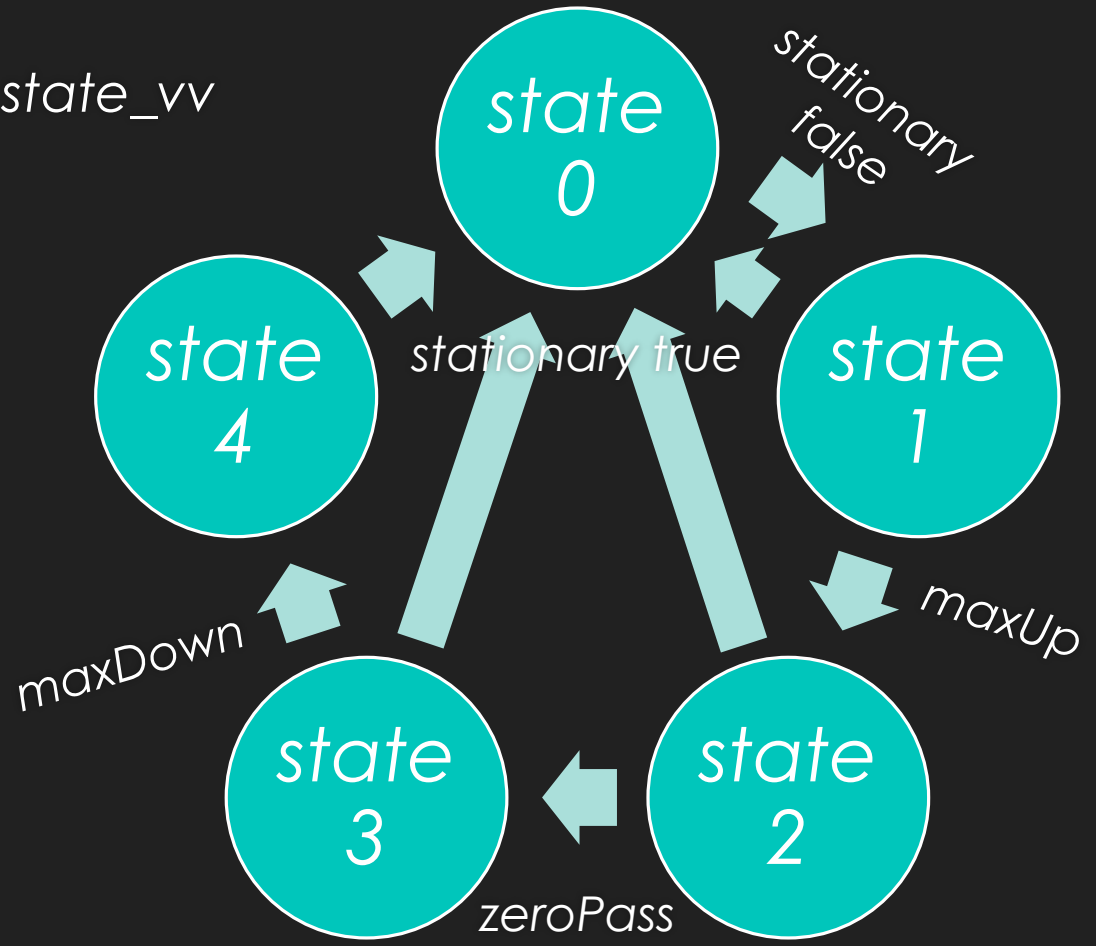
  ** state transition to state 0 possible from any state, otherwise state must increase by 1*

WIP states (cont.)

- *stationary*



- *state_vv*



WIP position tracking

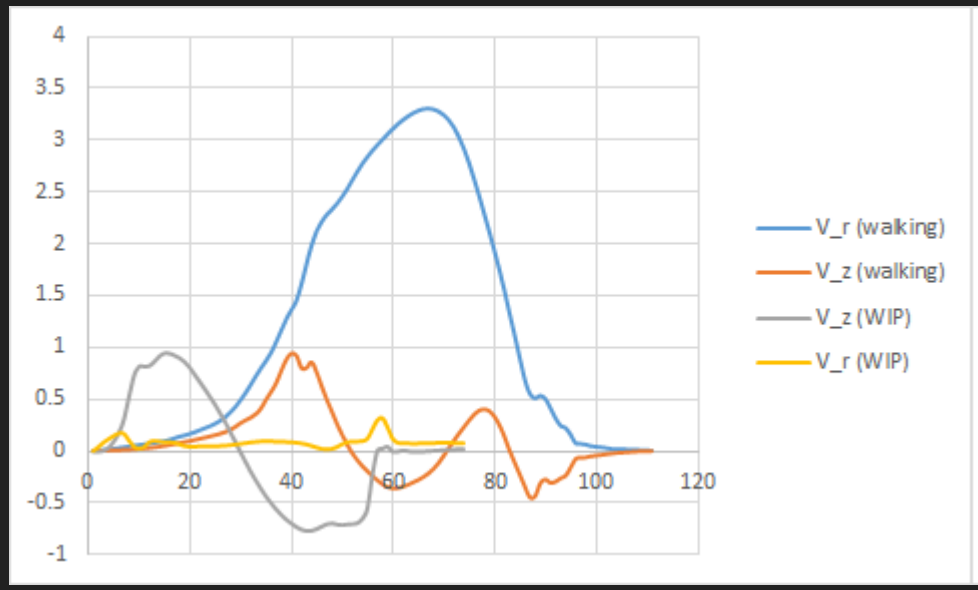
- *Basic idea: double integration with velocity drift correction (ZuPT)*

1. *Obtain quaternion (directly from LPMS-B2 or from Madgwick)*
2. *Convert acceleration from sensor frame to global frame and compensate for gravity*
3. *Integrate once to obtain velocity and twice to obtain position*
4. *When foot becomes stationary update velocity drift rate with residual velocity and WIP duration*
5. *When foot becomes stationary or vertical position is negative set position to origin*

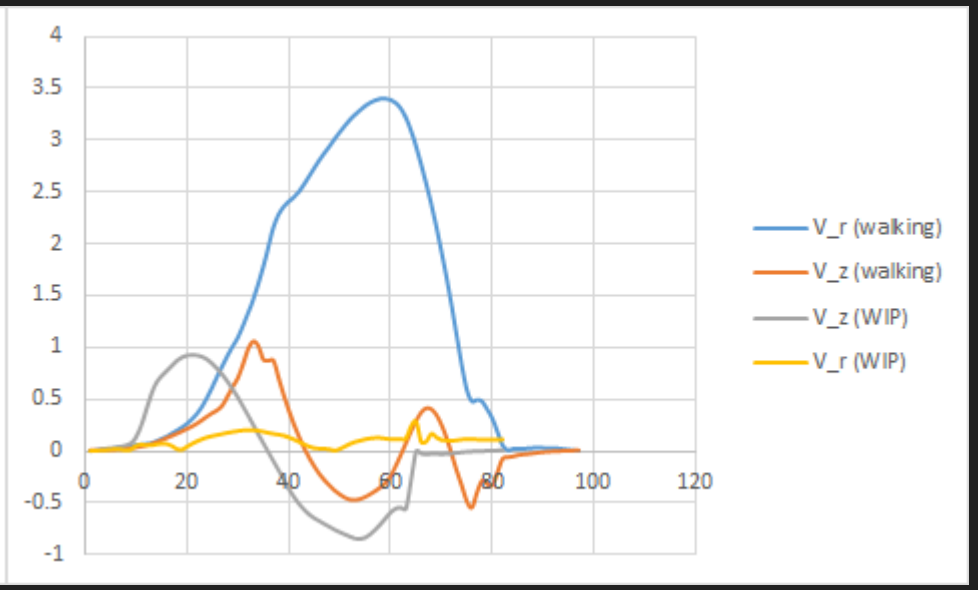
** 1-3 double integration, 4-5 drift compensation*

getVirtualSpeed() – motivation

walking #2, WIP #1



walking #3, WIP #2



$$V_z = \sin(t),\ V_r = 1 - \cos(t) \rightarrow V_r = k \int V_z = kP_z$$

Sample velocity curves (walking - 0728/w_normal, left & WIP - 1006/dorsal_100Hz_1, left) @ 100Hz, Madgwick filter

getVirtualSpeed() – summary

foot velocities can be modeled as such:

$$v_{z,walking} = V_z \sin\left(\frac{2\pi}{T} t\right)$$

$$v_{r,walking} = V_r \left[1 - \cos\left(\frac{2\pi}{T} t\right)\right]$$

$$p_{z,walking} = \int v_{x,walking} = \frac{TV_z}{2\pi}\left[1 - \cos\left(\frac{2\pi}{T} t\right)\right]$$

$$v_{r,walking} = k * p_{z,walking}$$

coefficient k can be determined from Kitagawa2016 measurements:

$$\mathrm{k} = \frac{2\pi V_r}{TV_z}$$

recovering horizontal velocity from WIP motion:

$$v_{r,recovered} = f(k, p_{z,WIP})$$

resulting locomotion in virtual environment:

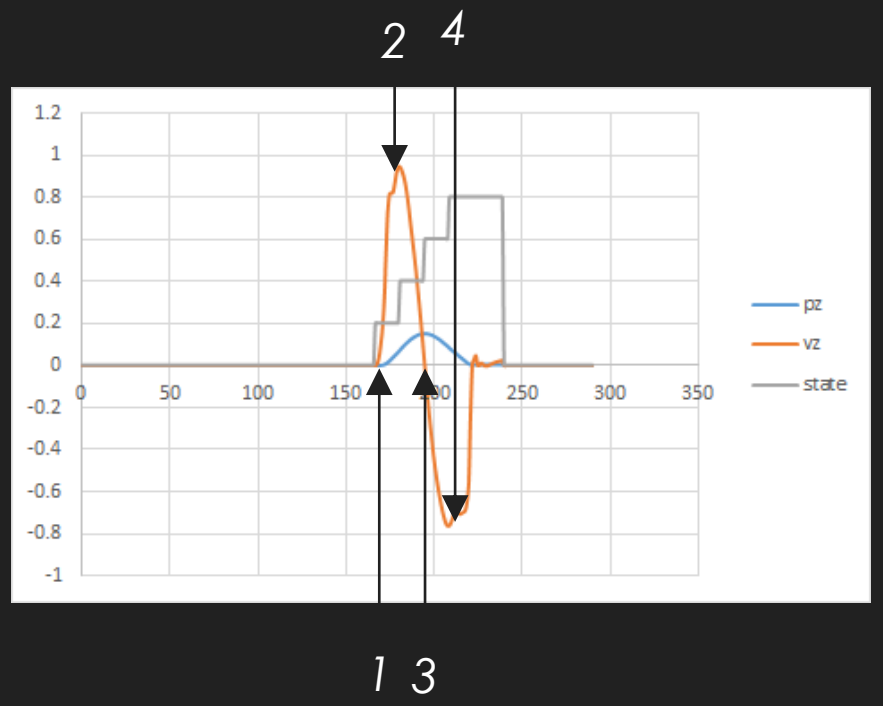
$$v_{r,virtual} = f(v_{r,recovered(left)}, v_{r,recovered(right)})$$

getVirtualSpeed() – implementation

- *Practical issues*
  - *Simple addition of foot speeds results in jerkiness (just as in LLCM-WIP)*
  - *Starting/stopping latency should be minimal*
- *Strategy to mitigate jerkiness without increasing latency*
  - *Add padding at the end (state_vv 4)*

$$v_{r,recovered}(t) = k * p_{z,WIP}(t) \quad (t_1 \le t \le t_4)$$

$$= 0.5k * (p_{z,WIP}(t) + p_{z,WIP}(t_3)) \quad (t_4 < t)$$



getVirtualSpeed2()

- *Unlike getVirtualSpeed(), getVirtualSpeed2() does not utilize the entire time series of foot vertical position but only requires maximum foot height*
- *getVirtualSpeed2() locomotion scheme also takes WIP period into account*
- *Smoothness of motion has been improved over getVirtualSpeed()*

- *As of now, locomotion velocity is given as below (subject to change)*

$$v = f(maxHeight, WIPPeriod) = scaleCoefficeint * maxHeight * 2/(WIPPeriod + 1)$$

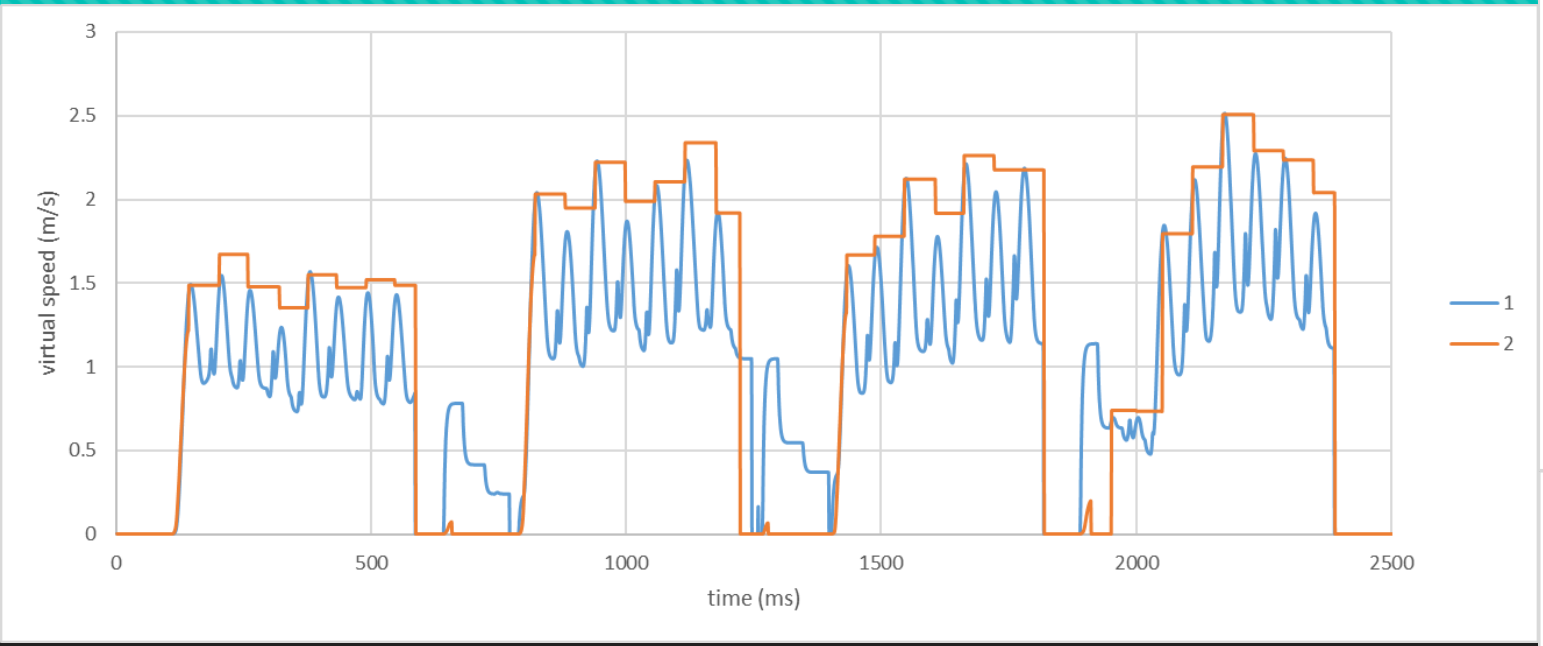
getVirtualHeading()

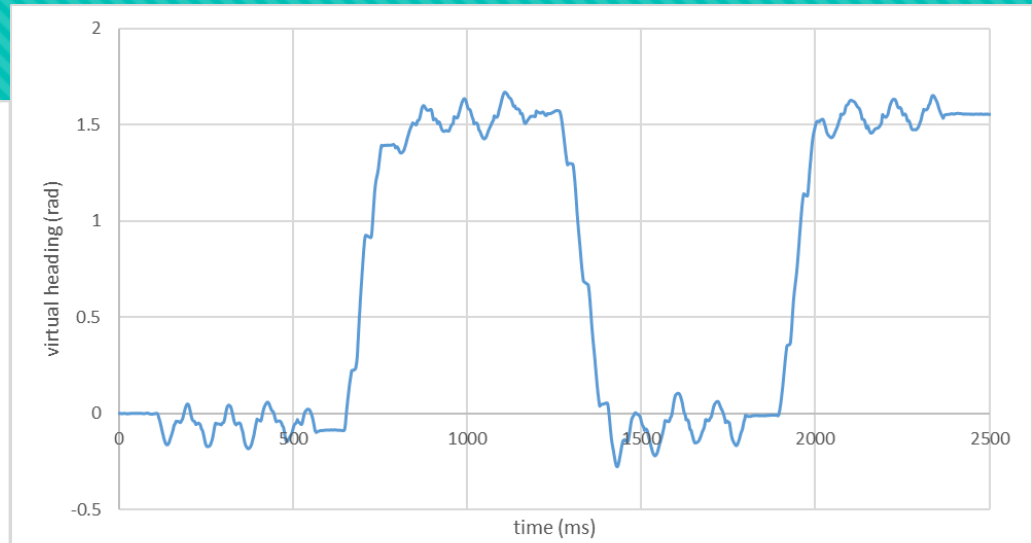
- *Basic idea: compute the midway direction of the feet (syncing/offset compensation between VR headset and LPMS-B2 may be required)*

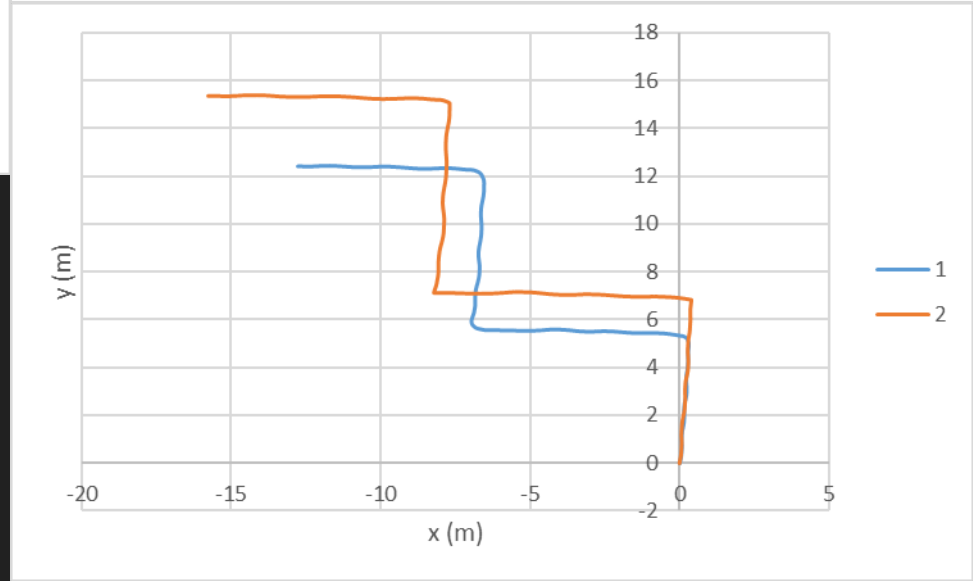
1. *Compute relative quaternion to initial quaternion (optional)*
2. *Find yaw angle of both feet*
3. *Add the two-dimensional direction vector of both feet to obtain the midway direction vector (optional – exception handling may be required)*

Results

1006dorsal_100Hz_4 normal WIP with rotation







Demo