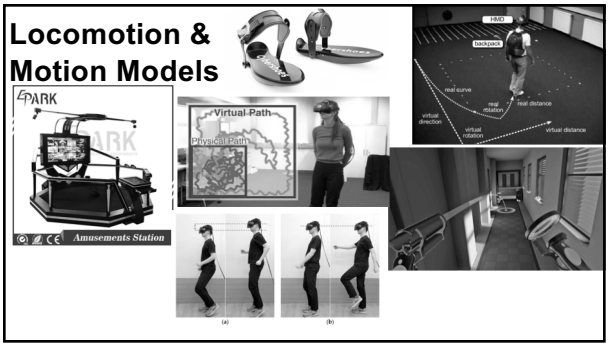
Locomotion & Motion Models

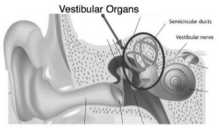


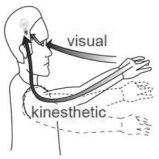
Locomotion

Senses:

- **Vestibular**: sense of balance/movement
- **Kinesthetic**: sense of how a body part is oriented

General idea: try to make motion in game closest to what user physically does to avoid sickness





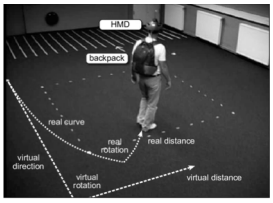
Movement Perception

Locomotion

Goal: find a mapping between **physical** motions/actions and **virtual** motions

Physical Actions (Button Presses, etc.)

Physical Motions (Walking)



Important ideas to keep in mind

- VE & PE sizes don't usually align (distance perception and/or actual sizes)
  - Audio, Locomotion, Depth, Etc.
  - Interrante 06: "Distance Perception in Immersive Virtual Environments, Revisited"
- There's a lot of noise in motion
  - Saccades, micro-motions of head, decisional randomness (game theory), etc.
- We want to avoid distorting mental map too much
- **General rule**: the closer the physical/virtual motions match, the better

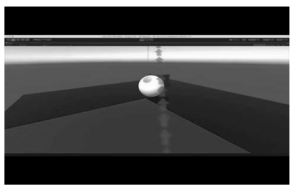
https://www-users.cs.umn.edu/~interran/papers/ieeevr06.pdf

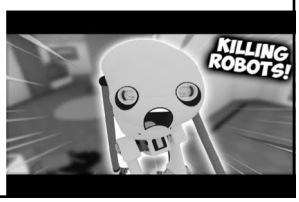
Controller Movement

- Joysticks cause motion sickness
  - Mismatch between what vestibular sense tells us about movement and what we see
- Implementing tunnel vision (restrict FOV) can help
  - Columbia/Microsoft: https://engineering.columbia.edu/news/fighting-virtual-reality-sickness

Teleportation

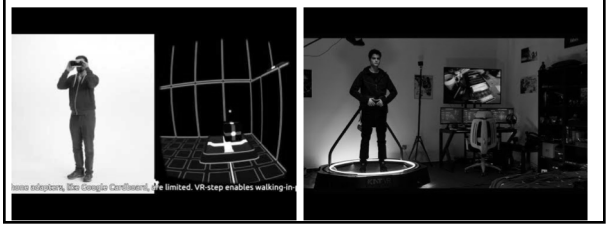
- One of the least sickening
- One of the least immersive
- Can still restrict FOV to avoid sickness. Should also black out display when moving for light-sensitive people
- Great paper about this: Boletsis '2019 http://downloads.hindawi.com/journals/ahci/2019/7420781.pdf
- Cool implementation: Budget Cuts





Walking-In-Place (WIP)

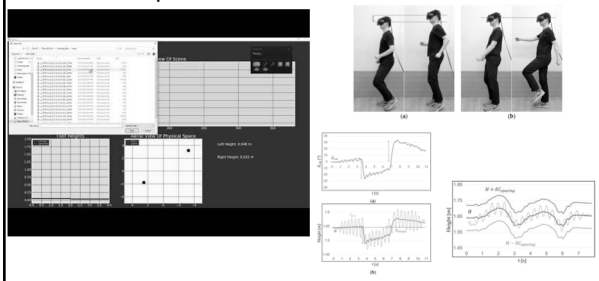
- Map head motions to translational motion in restricted PE
- Usually requires calibration of head bob, arm motion, gait, etc.
- Also have WIP treadmills & rugs



Walk Cycle for WIP

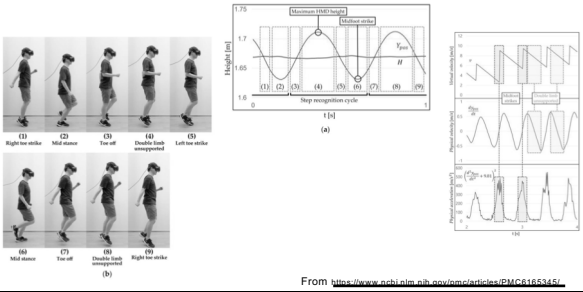
Want to replicate how people actually walk to avoid sickness

Quick Example of Head/Controller Bob in WIP



From https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6165345/

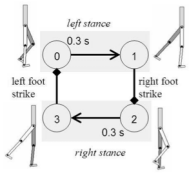
Some more: Leg position state machine



From https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6165345/

Why is the state machine important?

- Ideas?
  - Mitigate drift & noise
  - Not every "head bob" maps to same 2D translation input
  - Different heights & body shapes = different stride (how far you move each step)
  - Feet are not usually tracked… so we need to infer based on state machines and head motion
- Issue in practice: people in WIP don't move naturally



Arm Movement

Usually used in unison with WIP
E.g. if you want user to fly, should have them flap their arms to avoid sickness
Great free demo of WIP+arm movement: Freedom Locomotion VR on Steam

Great comparison by Boletsis '19

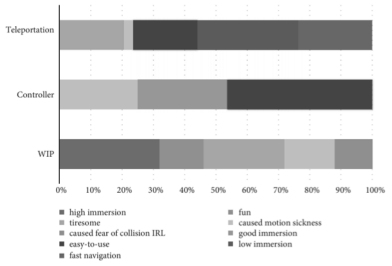
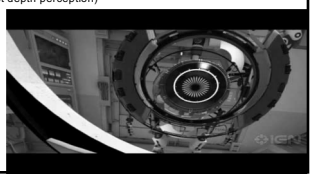


FIGURE 8: A summary of the VR locomotion techniques' qualities based on the participants' interviews.

1:1 Walking

- Physical & virtual spaces exactly aligned (or VE fits within PE)
- Doesn't necessarily work well… any ideas why?
  - **Distance compression**: users feel like they are walking slower in VR than in real life
    - Walking 1 meter in the VE doesn't "feel" the same as walking 1 meter in real life
  - VR "camera" doesn't match user's eyes… HMD/lens design matters (weight, IPD, etc.)
  - Inaccurate audio (affects distance compression)
  - Inaccurate rendering (effects like AO affect depth perception)



Real Walking & Distortion Methods

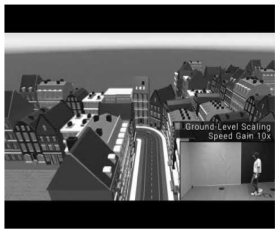
- **General problem**: we only have so much PE tracking space….but want to navigate big VEs
- Also want to use real walking b/c it's most natural while mitigating distance compression

VE (game world)

PE (tracking space)

Translational Gain

Addresses distance compression. Makes you walk virtually much faster than in real life. People generally feel like they walk much slower in VR (mostly b/c they do! VR space is very small)



Practice exercise: How to implement translation gain?

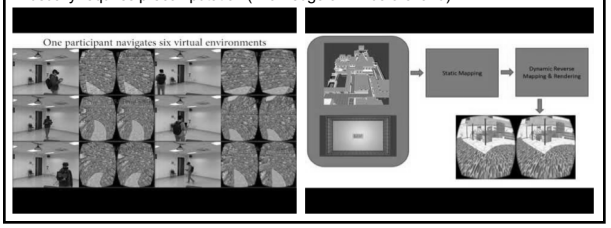
- Things to remember:
  - Cannot mutate Camera or controller positions….but we can always get them
  - Can move entire VR space around (e.g. VRRoot)
- How do we know how much user WANTS to move?
  - Save prev frame head position. On current frame, **deltaPos**=Camera.transform.postion-prevPos
- How do we move them MORE?
  - Add that vector to their Pawn/PlayerController location….shift entire space by their deltaHeadLoc
  - VRRoot.transform.position=OVRPlayerController.transform.position+deltaHeadLoc
- How much gain is this?
  - 100%.... It doubles their movement
- How to reduce gain?
  - Multiple the deltaHeadPos by a threshold (e.g. 0.5f)
- Final equation
  - VRRoot.transform.position=VRRoot.transform.position+thres*deltaHeadLoc
  - VRRoot.transform.position=VRRoot.transform.position+thres*(Camera.transform.position-prevPos)

Translational Gain Perception

- Partially depends on size of environment, display, etc. 50-100% gain usually good enough
- Too slow: Feels like walking in VR is really slow
- Too fast: Feels like walking on ice
- In terms of physics/calc, what IS deltaHeadLoc?
  - Velocity!
  - Much like fixing the momentum problem in Unity physics, we can average velocity over a few frames as well to get acceleration, which is even better

Motion Compression

Use mapping from VE to PE to fit VE inside PE
Limitations: can distort environment & might not work in open scenes…. Also usually requires precomputation (knowledge of VE beforehand)



Mechanical methods

- Have machine automatically move you
- Treadmills
- Moving stairs

Reorientation (Explicit)

- User hits/holds button to rotate world themselves
- E.g. Fine China demo

Redirected Walking (Implicit)

- Idea: rotate the world more virtually than physically to distort trajectory (usually as a function of eye rotation/saccades, head rotation, etc.)
- Why? People are not as good at knowing their exact orientation as they think
- Developed at UNC in 2000-2001 by Sharif Razzaque
- Why is it implicit? User shouldn't know it's happening!

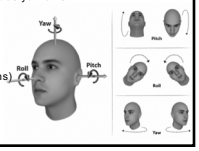
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.13_2.4818&rep=rep1&type=pdf

Can also do this with eye rotation (saccadic redirection)

Qi Sun SIGGRAPH 2018. Different rotational threshold

Redirected Walking (RDW) (Implicit)

- Does this idea sound similar to another (that we just learned!)?
  - Similar to translational gain…. But rotational gain!
  - Also in the same vein: function of how much user actually rotates (effectively function of rotational velocity)
- Which rotational measurement is good for head velocity?
  - Yaw!
  - Why are others not appropriate?
    - Should always let user natural head motion handle pitch/roll
    - PE/Tracking Space effectively 2D square with rotational pivot as yaw axis
  - Global or local yaw (wrt head or world axes)?
    - World axes: everything gets projected to ground plane
  - Wrt PE or VE?
    - PE: how much user physically rotated
    - VE: how much they rotated in-game (including prev distortions)



Redirected Walking (RDW) (Implicit)

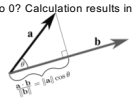
- Design technicalities:
  - Rotations towards/away from goal are different (gain vs. loss)
  - No rotational gain in center of PE
    - So RDW algorithm needs to be sensitive to distance from PE center
  - Free roam?
    - Dynamic reference for what the "goal" is

RDW: Rotational Velocity

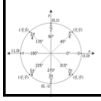
- Can we not just add 360 degrees to yaw to get deltaYaw?
  - We can for just getting amount of head rotation & general left/right head direction…. But we'll see why we need to do cross-product way in a bit anyway
- How can we change the algorithm to be sensitive to which way the center of the PE is?
  - Use cross product to figure out if user is rotating towards or away from PE center
- Basic idea: decelerate head rotation away from PE center (loss), accelerate rotations towards it (gain)

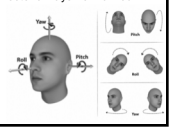
Redirected Walking (Implicit)

- How to get deltaYaw (rotational velocity)?
  - CurrentYaw-PrevYaw? Problems?
  - What happens when trajectory switches from 360 to 0? Calculation results in very high or low velocity
  - Any other way of getting rotation between vectors?
    - Dot product!



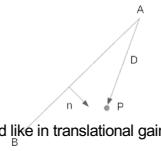
- Which vectors to use?
  - Forward vector of camera!
- Problems with dot product?
  - Functions strangely on non-normalized vectors…. Fortunately, forward vector always normalized
  - Rotation always positive…. No direction





How to know **direction** of rotational velocity?

- Figure out where previous projected position was relative to new line
- How to get projected position?
  - Camera.transform.position+Camera.transform.forward
- To get direction, basically compute cross-product
- d=(x−x1)(y2−y1)−(y−y1)(x2−x1)
  - d<0 is one side (usually left), d>0 other side
- So left or right indicates deltaYaw * -1 or +1
- Resulting value can now be multiplied by threshold like in translational gain!
- Final process:
  - Use dot product on prev & current forward vector to get absolute value of rotational velocity
  - Use cross product to figure out left or right (multiply above value by + or -1)
  - Multiply resultant value by threshold
  - Add this to yaw of OVRPlayerController or VRPawn



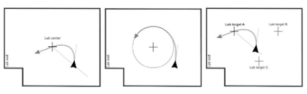
Good explanation here: https://math.stackexchange.com/questions/274712/calculate-on-which-side-of-a-straight-line-is-a-given-point-located

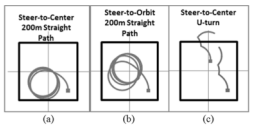
Quick vector subtraction tip

Destination-Source gives vector starting at source and ending at destination (easier to remember IMO than A-B)

Steering Methods

- Point of RDW is to keep user in space…. Usually trying to keep them facing the PE instead of looking out of bounds (e.g. at a wall)
- Thus, need to also include some vector keeping track of "target" PE location (usually the center….where they have most walking space) to steer them to center and keep them in bounds…. Called S2C





S2C basic algorithm (Z-up)

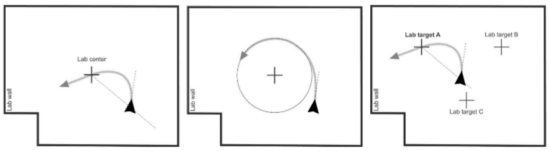
- **d(Vector3/Vector2 A,Vector3/Vector2 B,Vector3/Vector2 C)**=(A.x−B.x)(C.y−B.y)−(A.y−B.y)(C.x−B.x)
- **angleBetweenVectors(Vector2 A, Vector2 B)**=arccos(dot(normalize(A),normalize(B)))
- Beginning of game (Start/BeginPlay)
  - **prevForwardVector**=Camera.forward
  - **prevYawRelativeToCenter**=angleBetweenVectors(Camera.forward,VRTrackingOrigin.position-Camera.position)
- Each frame (Tick/Update):
  - **howMuchUserRotated**=angleBetweenVectors(prevForwardVector,Camera.forward)
  - **directionUserRotated**=(d(Camera.position+prevForwardVector, Camera.position, Camera.position + Camera.forward)<0)?1:-1
  - **deltaYawRelativeToCenter**=prevYawRelativeToCenter-angleBetweenVectors(Camera.forward,VRTrackingOrigin.position-Camera.position)
  - **distanceFromCenter**= Camera.localPosition.magnitude OR (Camera.position-VRTrackingOrigin.position).magnitude
  - **longestDimensionOfPE**=[some value you define in m or cm]
  - **howMuchToAccelerate**=((deltaYawRelativeToCenter<0)? -decelerateThreshold [-13%]: accelerateThreshold[30%]) * howMuchUserRotated * directionUserRotated * clamp(distanceFromCenter/longestDimensionOfPE/2,0,1)
  - VRTrackingOrigin.RotateAround(Camera.position,(0,1,0),howMuchToAccel)
  - **prevForwardVector**=Camera.forward
  - **prevYawRelativeToCenter**=angleBetweenVectors(Camera.forward,VRTrackingOrigin.position-Camera.position)

RDW threshold

- For systems with visuals, <30% max of head velocity
  - S2C: 13% max deceleration/loss, 30% max gain/acceleration
- For systems relying only on audio, <20% max of head velocity gain
- Again, we can average velocities over a few frames to get more accurate measurements
- Anything higher will make the rotation noticeable or sickening
- Seems really small…. Means that you need at least 30mx30m space to rotate user walking straight with only natural rotational drift
  - https://illusioneering.cs.umn.edu/papers/azmandian-ieat2015.pdf
- So….what do we do? Remember RDW is function of rotational velocity
  - Give user a reason to rotate their heads!
  - Distractors!

Can also do steer-to-physical-object...

- Generally requires object to be cylindrical (why?)
  - Because RDW allows user to be redirected into any direction…. Physical object must be shaped such that distortion rotation doesn't matter



Good article: https://www.makinggames.biz/news/getting-forwards-locomotion-in-virtual-environments10143.html

"Combining passive haptics with redirected walking" Luv Kohli 2005

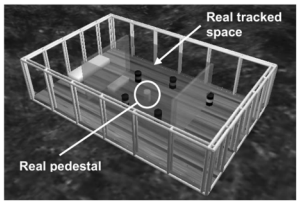


Fig. 1. A view of the virtual environment with the five striped virtual pedestals. A box indicating the size of the real tracked space is superimposed, along with the position of the real pedestal in the center.

Fig. 2. A user touches the one cylindrical object intended to provide haptic feedback. The Styrofoam walls mark the limits of the tracked space.

The "Stuck" problem

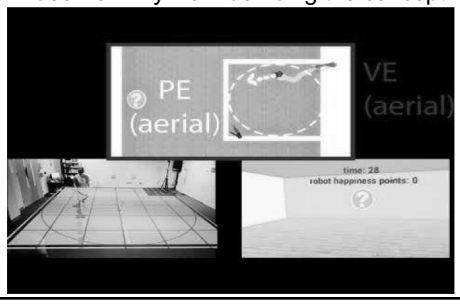
- If direction is ambiguous, S2C will fail and oscillate but never really shift back to center
- Can sort of address with behavior graphs…. But still hard to deal with
  - My distractor work addresses it though….with some constraints!
- Demo through my example app

Original Distractors

First major study by Tabitha Peck 2008, "Evaluation of Reorientation Techniques and Distractors for Walking in Large Virtual Environments"

- Distractors appeared and rotated around head until user looked back into center
- Distractors way better than forcing rotation when they're too close to bounds
- Found that better-designed distractors lead to less notice of rotation. Distractor moving too quickly causes sickness. Using audio helps a lot

Video from my work demoing the concept

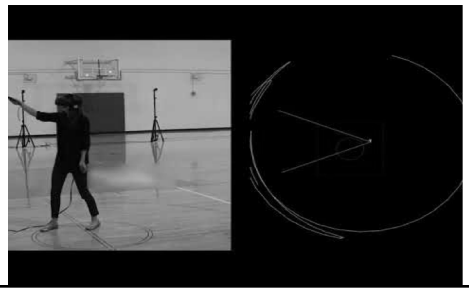


Original Distractors

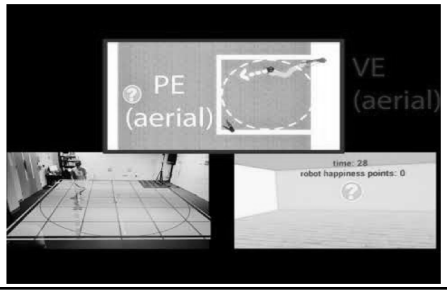
First major study by Tabitha Peck 2008, "Evaluation of Reorientation Techniques and Distractors for Walking in Large Virtual Environments"

- Distractors appeared and rotated around head until user looked back into center
- Distractors way better than forcing rotation when they're too close to bounds
- Found that better-designed distractors lead to less notice of rotation. Distractor moving too quickly causes sickness. Using audio helps a lot

Gamified example from UNC



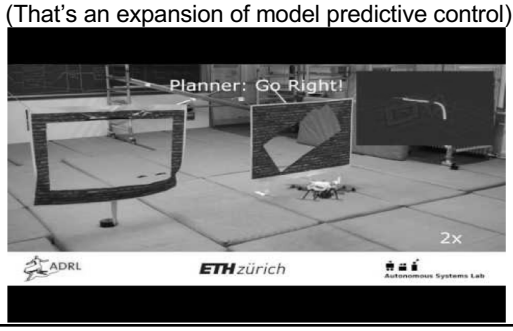
Video from my work demoing the concept



Great talk comparing methods (SIGCHI 2019)

Game-theory-esque prediction methods

(That's an expansion of model predictive control)



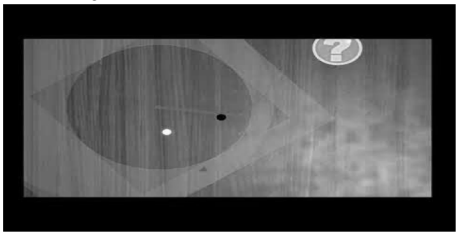
Some DL extension from VR 2019

Constraints on user motion

- (Our work) Haptic constraints
- Objects in environment constraints (e.g. deterrents… a fence around the VE…. decreases immersion)
- Others….? Not very densely-explored area (yet)

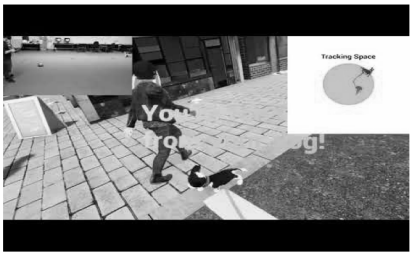
(Our Work)

- Use distractor's trajectory to choose a direction to redirect since that's consistent throughout the distraction



(Our application)

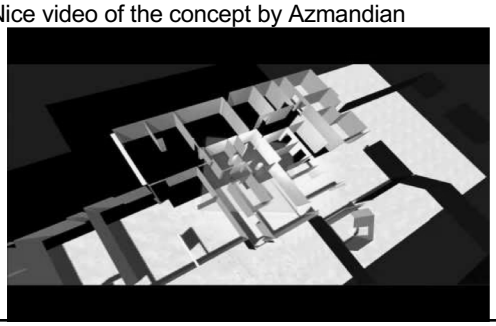
- Guide dogs in VR



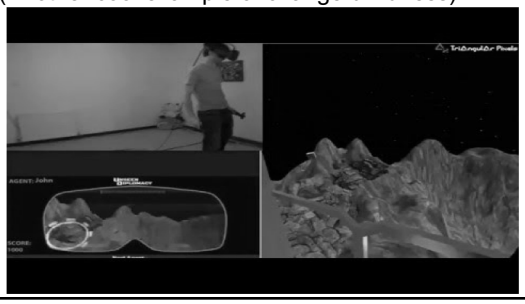
Other cool applications

Taking advantage of maze-like VEs

Nice video of the concept by Azmandian



(Another cool example of change blindness)



Redirected Walking Toolkit by USC

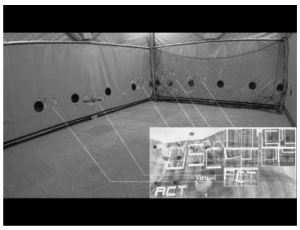
- Nice Unity project to play with params
- Not set up for VR by default and has way too many params to make A7 any easier….but worth trying out!

http://projects.ict.usc.edu/mxr/rdw/

HRI In Locomotion

- **Haptic proxy**: physical object/robot representing virtual haptic actor
- **Co-location**: physical and virtual agent are **exactly** aligned

HRI In XR



Redirected Touching/Haptics

Luv Kohli 2010

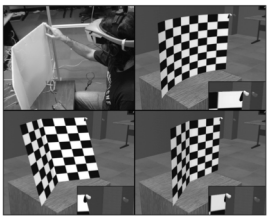
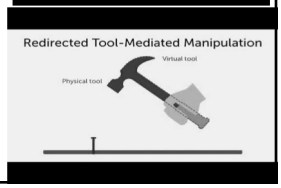


Figure 2: A user touches the corner of the physical foam board. The virtual finger position is warped to touch the corners of three different virtual objects. The insets show the distances between the virtual finger and the superimposed real finger.

Redirected Touching/Haptics

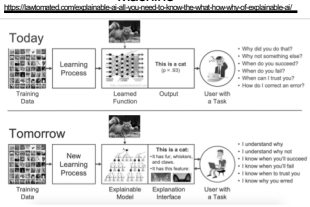


Neuroscience & Explainable AI

Same problem… hard to explain in detail workings of human or machine decision-making. Requires a model to break down components

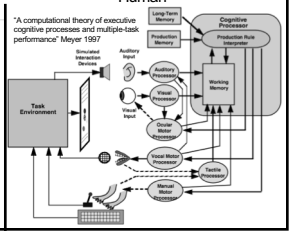
Machine

https://towardsdatascience.com/explainable-ai-xai-what-you-need-to-know-the-what-how-why-of-explainable-ai/



Human

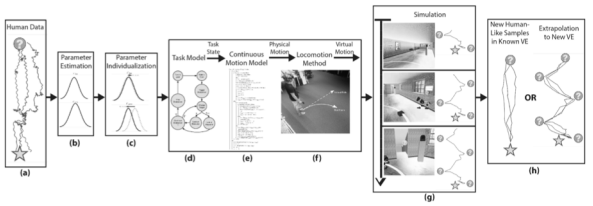
"A computational theory of executive cognitive processes and multiple-task performance" Meyer 1997



Reinforcement Learning Approach

My attempt to treat the robot dog as an RL problem: link

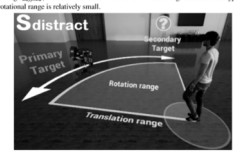
Simulating VR Users Project: Pipeline



Simulating VR Users Project: States

Motion parameters are different in different states… so we need (1) a state machine for a simulated user and (2) extracted motion params for each state

(a) During $s_{waypoint}$, the user's translation is a narrow forward range towards the waypoint and the rotational range is relatively small.



(b) During $s_{distract}$, the user's translation is a small range of noise (as they generally stand still) and the rotational range is very high, not only due to distractor motion, but due to the tendency to glance towards the original waypoint (which is now a secondary target).

Simulating VR Users Project: Motion Params

Estimated Simulation Parameters

Description		Symbol	Source	Distribution ($\forall p \in P$)	Units
Stimulus response time, or length of a decision		R	[16]	$\mathcal{U}(0.03, 0.5)$	s
Probability of glancing to secondary target		Π	[5]	$\mathcal{U}(22, 30)$	%
Angle of "looking at" cone		Θ	[24]	$\mathcal{U}(10, 50)$	°
Motion	**Type**	**Task State ($Q(s\|t)$)**		**p=Audio (A)**	**p=Vision (V)**	
Translation	speed ($x^{\cdot}$)	$s_{distract}$	Extracted from real users in task L3 in population p [48]	$\mathcal{N}(0.15, 0.88)$	$\mathcal{N}(0.12, 0.032)$	m/s
		$s_{waypoint}$		$\mathcal{N}(0.24, 0.082)$	$\mathcal{N}(0.36, 0.14)$	
	acc (x^{+})	$s_{distract}$		$\mathcal{N}(0.42, 0.26)$	$\mathcal{N}(0.39, 0.63)$	m/s^2
		$s_{waypoint}$		$\mathcal{N}(1.0, 1.1)$	$\mathcal{N}(0.66, 1.0)$	
	dec (x^{-})	$s_{distract}$		$\mathcal{N}(0.38, 0.24)$	$\mathcal{N}(0.37, 0.53)$	
		$s_{waypoint}$		$\mathcal{N}(1.1, 1.2)$	$\mathcal{N}(0.66, 1.2)$	
Rotation	speed ($r^{\cdot}$)	$s_{distract}$		$\mathcal{N}(39, 14)$	$\mathcal{N}(26, 3.5)$	$°/s$
		$s_{waypoint}$		$\mathcal{N}(21, 6.5)$	$\mathcal{N}(17, 8.7)$	
	acc (r^{+})	$s_{distract}$		$\mathcal{N}(138, 75)$	$\mathcal{N}(65, 22)$	$°/s^2$
		$s_{waypoint}$		$\mathcal{N}(85, 47)$	$\mathcal{N}(90, 49)$	
	dec (r^{-})	$s_{distract}$		$\mathcal{N}(125, 66)$	$\mathcal{N}(64, 20)$	
		$s_{waypoint}$		$\mathcal{N}(85, 43)$	$\mathcal{N}(83, 44)$	

Table 1: This summarizes the sample distributions for basic cognitive parameters as well as the estimated real human motion parameters E as derived from [48] for the two task states of that paper, $s_{waypoint}$ and $s_{distract}$. Note that $\mathcal{U}$ indicates the uniform distribution $\mathcal{U}(a, b)$ and $\mathcal{N}$ is the normal distribution $\mathcal{N}(\mu, \sigma)$.

Simulating VR Users Project: Motion Param Individualization

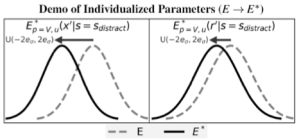


Fig. 5: An example of how a particular simulated user u's parameters may shift to individualize them for the simulation. For each estimated parameter $m \in M$, its mean m_μ is shifted by a uniformly random amount of its deviation m_σ, but m_σ is unmodified, resulting in the set of individualized parameters $E^*_{u,p}(m|s)$.

Simulating VR Users Project: Param Updating

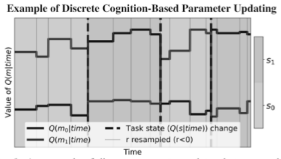


Fig. 6: An example of discrete parameter updates that occur when the stimulus response time r has passed–that is, the user is cognitively prepared to make a new decision. The grey lines show when this occurs. These example motion parameters m_0 and m_1 are sampled for a user of population p in state x at time t–that is, $Q(m_0|t)$ and $Q(m_1|t)$ are sampled from $E^*_p(m_0|s)$ and $E^*_p(m_1|s)$ with every new decision. The black dotted lines indicate changes in the task state s (i.e. changing from $s_{waypoint}$ to $s_{distract}$), which noticeably affects sampled parameters.

Simulating VR Users Project: Motion Models

- **Velocity-based:**
  - We only sample velocities each time user makes decision
- **Acceleration-based**
  - We sample acceleration each time user makes decision, and clamp velocity values
- **Behavior-based**
  - Acceleration-based model with 2 behaviors accounted for:
    - Users tend to drift back to PE center during distraction
    - Users' heads oscillate around a view target

Simulating VR Users Project: Results

Velocity model too accurate & unnatural

Acceleration model pretty good in general

Behavioral model works well for vision users but not for audio users (b/c the 2 behaviors applied mainly to users with vision)... basically, it overfits on the user population that actually does those behaviors (might be desirable). Drift behavior causes jagged paths

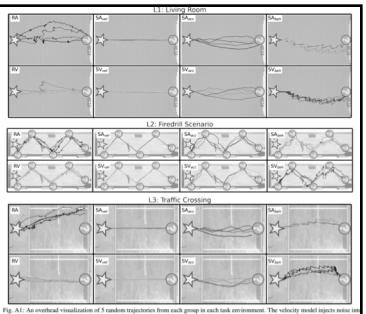


Fig. A1: An overhead visualization of 5 random trajectories from each group in each task environment. The velocity model injects noise int
decisions, but due to sampling suitable velocities directly, it does not significantly deviate from the ideal path. The acceleration model generates
good variety of paths, but there is not a significant amount of translational deviation in its decisions. The behavioral model better captures the
noisiness of the human paths, but the "drift back to PE center" behavior causes unrealistically strong zig-zag patterns. Its deceleration behavio
also causes it to walk in wide arcs since it is less likely to significantly overshoot its target.