

Supplementary Table S1

Marker location of the Oxford Foot Model used in this study

Segment	Marker Name	Description - Position	Notes
	LASI / RASI	Anterior Superior Iliac Spine	Conventional Plug-in-Gait model
	LPSIS / RPSIS	Posterior Superior Iliac Spine	Conventional Plug-in-Gait model
	LTHI / RTHI	Thigh marker	Conventional Plug-in-Gait model
	LKNE / RKNE	Standard lateral knee	Conventional Plug-in-Gait model
	RTIB	Tibial marker	Conventional Plug-in-Gait model
Shank	RHFB	Lateral head of fibula	-
	RTUB	Tibial tuberosity	-
	RSHN	Anterior aspect of the shin	-
	LANK / RANK	Ankle marker	Conventional Plug-in-Gait model
	RMMA	Medial Malleolus	Static trial only – remove for dynamic trials
Hindfoot	RCPG	Peg marker – posterior end of the calcaneus	-
	LHEE / RHEE	Heel marker	Conventional Plug-in-Gait model
	RPCA	Posterior calcaneus proximal	Static trial only – remove for dynamic trials
	RLCA	Lateral calcaneus	-
	RSTL	Sustentaculum tali	-
Forefoot	RP1M	1st metatarsal, proximal dorsal	-
	LD1M / RD1M	1st metatarsal, distal medial	Static trial only – remove for dynamic trials
	RP5M	5th metatarsal, proximal lateral	-
	RD5M	5th metatarsal, distal lateral	-
	RTOE	Toe marker	Conventional Plug-in-Gait model
Hallux	RHLX	Hallux – proximal end of 1st distal phalanx	-

Supplementary Table S2

The coupling angle (γ_i : 0–360°) was divided into eight categories in 45° increments, and the coordination pattern at each time point was determined accordingly. Intervals were treated as half-open, and 360° was considered equivalent to 0°. The “+ / –” symbols indicate the direction of angular change for the proximal ($\Delta\theta_P$) and distal ($\Delta\theta_D$) segments at each time point. For example, in the analysis of shank–hindfoot coordination, the proximal segment refers to the shank and the distal segment to the hindfoot. In this case, “+” represents shank external rotation and hindfoot inversion, whereas “–” represents shank internal rotation and hindfoot eversion.

γ_i (deg)	Phase	Dominance	Proximal direction	Distal direction	Label used in manuscript/code
[0, 45)	In-phase	Proximal	+	+	In-Phase Proximal Dominancy + Proximal + Distal
[45, 90)	In-phase	Distal	+	+	In-Phase Distal Dominancy + Proximal / + Distal
[90, 135)	Anti-phase	Distal	–	+	Anti-Phase Distal Dominancy - Proximal + Distal
[135, 180)	Anti-phase	Proximal	–	+	Anti-Phase Proximal Dominancy - Proximal / + Distal
[180, 225)	In-phase	Proximal	–	–	In-Phase Proximal Dominancy - Proximal / - Distal
[225, 270)	In-phase	Distal	–	–	In-Phase Distal Dominancy - Proximal / - Distal
[270, 315)	Anti-phase	Distal	+	–	Anti-Phase Distal Dominancy + Proximal / - Distal
[315, 360)	Anti-phase	Proximal	+	–	Anti-Phase Proximal Dominancy + Proximal / - Distal

Notes:

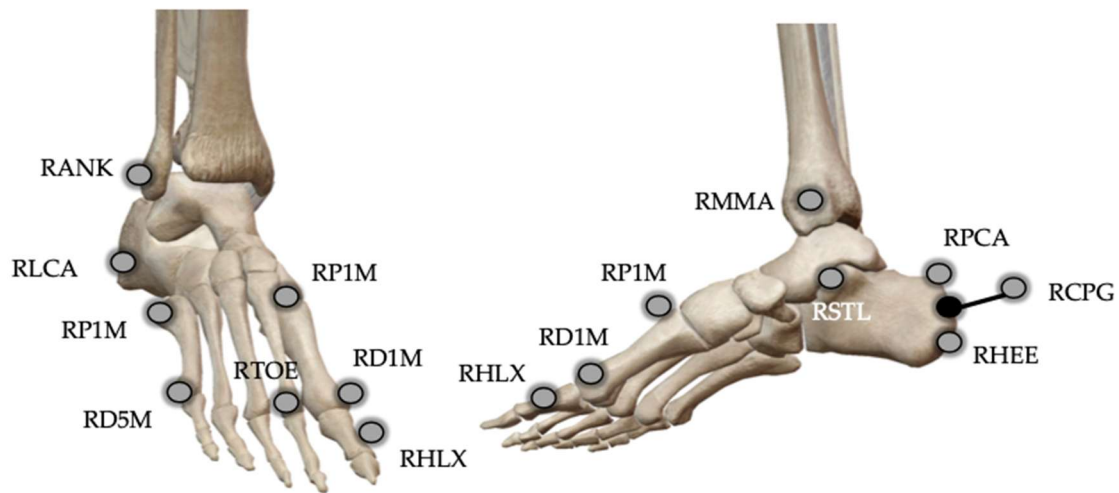
In-phase: Indicates that the proximal and distal segments are moving in the same direction.

Anti-phase: Indicates that the proximal and distal segments are moving in opposite directions.

Dominance: Indicates which segment (proximal or distal) exhibits the greater magnitude of movement.

Supplementary Figure S1

Visualization of the marker locations of the Oxford Foot Model



Supplementary Algorithm S1

Supplementary Algorithm S1: Modified Vector Coding (MVC) with circular-mean gamma

Author: <Your Name / Affiliation>

Python >= 3.10, numpy >= 1.24, pandas >= 2.0

Inputs (DataFrame)

- 'normalized_time': float in [0..100] for a single movement phase (ascending or descending)
- Proximal angle columns (θ_P): list of column names you provide
- Distal angle columns (θ_D): list of column names you provide
(The two lists must have the same length; each (θ_P , θ_D) pair defines one trial.)

Outputs (DataFrames)

- gamma_df: per-trial gamma_i time series (long format)
- mean_df: normalized_time, mean_gamma_i, Phase (Early/Mid/Late), Coordination (8 labels)

```
from __future__ import annotations
import numpy as np
import pandas as pd
from typing import Iterable, List, Tuple

# ----- 8-class labeling (labels kept exactly as requested) -----

def classify_coordination_direct(gamma: float) -> str | None:
    """Classify mean_gamma_i (deg) into the 8 coordination labels."""
    if not np.isfinite(gamma):
        return None
    g = float(gamma) % 360.0
    if 0 <= g < 45: return "In-Phase Proximal Dominancy + Proximal + Distal"
    elif 45 <= g < 90: return "In-Phase Distal Dominancy + Proximal / + Distal"
    elif 90 <= g < 135: return "Anti-Phase Distal Dominancy - Proximal + Distal"
    elif 135 <= g < 180: return "Anti-Phase Proximal Dominancy - Proximal / + Distal"
    elif 180 <= g < 225: return "In-Phase Proximal Dominancy - Proximal / - Distal"
    elif 225 <= g < 270: return "In-Phase Distal Dominancy - Proximal / - Distal"
    elif 270 <= g < 315: return "Anti-Phase Distal Dominancy + Proximal / - Distal"
    else: return "Anti-Phase Proximal Dominancy + Proximal / - Distal" #
[315,360)

# ----- per-trial coupling angle series -----

def coupling_angle_series(theta_P: np.ndarray, theta_D: np.ndarray) -> np.ndarray:
    """
    Compute gamma_i time series from proximal ( $\theta_P$ ) and distal ( $\theta_D$ ) angles (deg).
    Returns an array of length len(theta_*) - 1.
    """
    theta_P = np.asarray(theta_P, dtype=float)
    theta_D = np.asarray(theta_D, dtype=float)

    dP = np.diff(theta_P)
    dD = np.diff(theta_D)

    gamma = np.degrees(np.arctan2(dD, dP))

    # Special cases (consistent with prior work)
    gamma = np.where((dP == 0) & (dD > 0), 90.0, gamma)
    gamma = np.where((dP == 0) & (dD < 0), -90.0, gamma)
    gamma = np.where((dP < 0) & (dD == 0), -180.0, gamma)
    gamma = np.where((dP == 0) & (dD == 0), np.nan, gamma)

    # Map to [0, 360)
    gamma = np.where(gamma < 0, gamma + 360.0, gamma)
    return gamma

# ----- circular mean across trials (timewise) -----
```

```

def circular_mean_deg(angle_matrix_deg: np.ndarray) -> np.ndarray:
    """
    Circular mean across trials at each time index.
    angle_matrix_deg: shape (T, Ntrials), columns = trials, values in degrees.
    Returns mean_gamma (deg in [0,360)) of length T.
    """
    rad = np.deg2rad(angle_matrix_deg)
    cosv = np.nanmean(np.cos(rad), axis=1)
    sinv = np.nanmean(np.sin(rad), axis=1)
    mean_rad = np.arctan2(sinv, cosv)
    mean_deg = np.degrees(mean_rad)
    return np.where(mean_deg < 0, mean_deg + 360.0, mean_deg)

# ----- Early / Middle / Late labels -----

def add_period_labels(normalized_time: np.ndarray) -> pd.Categorical:
    """
    Early [0,33), Middle [33,66), Late [66,100].
    (Use the time vector aligned to gamma length.)
    """
    nt = np.asarray(normalized_time, dtype=float)
    return pd.cut(nt, bins=[-1, 33, 66, 100],
                  labels=['Early Phase', 'Mid Phase', 'Late Phase'])

# ----- main API -----

def mvc_mean_then_classify(
    df: pd.DataFrame,
    proximal_cols: Iterable[str],
    distal_cols: Iterable[str],
    time_col: str = "normalized_time",
) -> Tuple[pd.DataFrame, pd.DataFrame]:
    """
    Pipeline:
    1) For each (θP, θD) pair, compute per-trial gamma_i series.
    2) Stack per-trial gamma_i -> (T, Ntrials) and compute circular mean -> mean_gamma_i.
    3) Classify each time point from mean_gamma_i into 8 labels.
    Returns:
    - gamma_df (long): trial, prox_col, dist_col, normalized_time, gamma_deg
    - mean_df (wide): normalized_time, mean_gamma_i, Phase, Coordination
    """
    assert time_col in df.columns, f"Missing time column: {time_col}"
    nt_full = df[time_col].to_numpy()
    prox_cols = list(proximal_cols)
    dist_cols = list(distal_cols)
    assert len(prox_cols) == len(dist_cols), "proximal_cols and distal_cols must have the same length"

    gamma_list: List[np.ndarray] = []
    gamma_rows: List[pd.DataFrame] = []

    for trial_idx, (pcol, dcol) in enumerate(zip(prox_cols, dist_cols), start=1):
        if pcol not in df.columns or dcol not in df.columns:
            missing = pcol if pcol not in df.columns else dcol
            raise KeyError(f"Missing angle column: {missing}")

        theta_P = df[pcol].to_numpy()
        theta_D = df[dcol].to_numpy()
        gamma = coupling_angle_series(theta_P, theta_D)

        gamma_list.append(gamma)
        gamma_rows.append(pd.DataFrame({
            "trial": trial_idx,
            "prox_col": pcol,
            "dist_col": dcol,

```

```

        "t_index": np.arange(len(gamma)),
        "normalized_time": nt_full[:-1], # aligned to diff length
        "gamma_deg": gamma
    )))

gamma_df = pd.concat(gamma_rows, ignore_index=True) if gamma_rows else pd.DataFrame()

if len(gamma_list) == 0:
    mean_df =
pd.DataFrame(columns=["normalized_time", "mean_gamma_i", "Phase", "Coordination"])
    return gamma_df, mean_df

gamma_mat = np.column_stack(gamma_list) # shape: (T, Ntrials)
mean_gamma_i = circular_mean_deg(gamma_mat) # length T (= len(nt_full)-1)
nt_aligned = nt_full[:-1]

phase_labels = add_period_labels(nt_aligned)
coord_labels = [classify_coordination_direct(g) for g in mean_gamma_i]

mean_df = pd.DataFrame({
    "normalized_time": nt_aligned,
    "mean_gamma_i": mean_gamma_i,
    "Phase": phase_labels,
    "Coordination": coord_labels
})

return gamma_df, mean_df

# ----- example usage (commented) -----
# df = pd.read_excel("your_data.xlsx")
# proximal_cols = ["Prox_Trial1", "Prox_Trial2", "Prox_Trial3"] # 0P columns
# distal_cols = ["Dist_Trial1", "Dist_Trial2", "Dist_Trial3"] # 0D columns
# gamma_df, mean_df = mvc_mean_then_classify(df, proximal_cols, distal_cols,
# time_col="normalized_time")
# gamma_df.to_excel("gamma_timeseries_per_trial.xlsx", index=False)
# mean_df.to_excel("mean_gamma_and_classification.xlsx", index=False)

```
