Inverse Optimal Robust Adaptive Controller for Upper Limb Rehabilitation Exoskeletons with Inertia and Load Uncertainties

Supplemental Document for MATLAB Simulation Codes

Jiamin Wang[1], and Oumar R. Barry[1,2]

January 11, 2021

Nomenclature

The mathematical notations used are listed as following:

$\mathbf{F}(\underline{\mathbf{Z}})$	Single-input function $\mathbf{F}$ with argument $\mathbf{Z}$ (to differentiate from multiplications of parenthesized terms)
$\| \mathbf{Z} \|_n$	n-norm of a matrix $\mathbf{Z}$ ($n = 2$ if not specified)
$\bar{\mathbf{z}}$	Conjugation of quaternion $\mathbf{z}$ (4×1)
$\mathbf{c}_{m\times n}$	$m \times n$ matrix whose elements equal to $\mathbf{c} \in \mathbb{R}$ ($m,\ n$ fit with neighboring blocks if not specified)
$\mathbf{c}_m$	$m \times 1$ vector whose elements equal to $\mathbf{c} \in \mathbb{R}$
$\mathbf{I}_n$	Identity matrix of dimension n ($m,\ n$ fit with neighboring blocks if not specified)
$\mathrm{diag}(\underline{\mathbf{z}})$	Convert a m-dimensional vector $\mathbf{z}$ into a $m \times m$ diagonal matrix with elements from $\mathbf{z}$
$\mathbf{Z} > 0$	A square matrix $\mathbf{Z}$ is positive definite
$\mathbf{Z}^+$	The Moore-Penrose pseudo inverse of $\mathbf{Z}$
$\mathbf{Z}^{-\mathrm{T}}$	Transposed inverse of $\mathbf{Z}$ (since $(\mathbf{Z}^{-1})^{\mathrm{T}} = (\mathbf{Z}^{\mathrm{T}})^{-1}$)
$\mathbf{z}_1 \times \mathbf{z}_2$	Product of quaternions $\mathbf{z}_1$ (4×1) and $\mathbf{z}_2$ (4×1)

Contents

[1]Department of Mechanical Engineering, Virginia Tech, Blacksburg, VA, 24061, USA
[2]Corresponding author: obarry@vt.edu

1 Instructions and Files

This document contains the instructions and explanations for the MATLAB simulation codes of the main paper "Inverse Optimal Robust Adaptive Controller for Upper Limb Rehabilitation Exoskeletons with Inertia and Load Uncertainties". The codes are tested with MATLAB 2018b, and should be compatible with any versions above MATLAB 2017b.

The dynamical models for both simulation examples (i.e., the stationary upper limb exoskeleton (Arm4D) and the wearable wrist exoskeleton (TAWE)) are generated and pre-compiled for 64-bit Windows (`.mexw64`) and Linux (`.mexa64`) systems [1]. For any issues encountered with the use of pre-compiled functions (e.g., `XXX_mex.mexw64` or `XXX_mex.mexa64`), please try switch to the corresponding function scripts (e.g., `XXX.m`), which are slower but more compatible with different MATLAB versions.

The open source simulation codes can be be found at `https://github.com/VibRoLab-Group/IORAC_Exo` (after the publication of the main paper). We will update this document for possible future contents and corrections. To report error and technical problems, please feel free to contact the authors (**jmechw@vt.edu** or **obarry@vt.edu**). Thank you!

1.1 Instructions

The project root folder is `IORAC_RA_L_20_1319/`, or equivalently `./` hereinafter. The main simulation scripts use `PathSetup.m` to recursively include all subfolders within `./` to the project directory search path. Hence, please do not remove `PathSetup.m` from the root folder.

The simulations require tracking references that can be randomly generated by `./Arm4D_JacFuncGen.m` and `./TAWE_TrajGen.m`. Once the simulation results are generated, the user can use `Arm4D_Figures.m` and `TAWE_Figures.m` to plot the results in a way similar to those presented in the paper. The user can also download sample tracking reference data from `https://github.com/VibRoLab-Group/IORAC_Exo` to observe simulation results identical to those shown in the paper. The simulations also support 3D visualizations. This utility requires `.stl` 3D mesh models available in the online repository as well.

Since the simulation code adopts the fixed step size Runge Kutta 4th method for numerical integration, it should be noted that the simulation and control sampling rates are crucial to the stability of simulations. The control sampling rate also limits the controller application in practice. When extremely large gains, disturbances, and noises are implemented, please adjust the sampling rates accordingly. When the sampling rates are changed, please also remember to generate new tracking references.

1.2 Main Simulation Scripts

The following files located in `./` are the main simulation scripts with detailed comments:

(1) `Arm4D_Figures.m` generates the plots for the stationary upper limb exoskeleton (Arm4D) simulations. The plots generated are for Fig. 2, Fig. 3, and Fig. 4 in the main paper.

(2) `Arm4D_InertiaMatrixObservation.m` demonstrates how Eq.(22) is obtained from the stationary upper limb exoskeleton (Arm4D) model, and shows that it is not uncertain inertia parameters, but their linear combinations that uniquely affect the inertia matrix.

(3) `Arm4D_JacFuncGen.m` generates the uncertain parameter Jacobian function introduced by a unknown body. The uncertain parameters include mass, moment of inertia, and load parameters at load tetrahedron (for center of mass estimation). The generated function is used in stationary upper limb exoskeleton (Arm4D) simulations.

(4) `Arm4D_Parameters.m` contains the simulation, model, and control parameters for all stationary upper limb exoskeleton (Arm4D) simulations.

(5) `Arm4D_Simulation_Disturbance.m` contains the code for stationary upper limb exoskeleton (Arm4D) simulations under disturbance. It contains both Link 3 and Link 4 uncertainties. The performance of PD, SMC, and IO-RAC feedback controllers are generated for comparison.

(6) `Arm4D_Simulation_NoDisturbance.m` contains the code for stationary upper limb exoskeleton (Arm4D) simulations under no disturbance. It contains the simulations with two uncertainty conditions (i.e., with Link 4 uncertainty only, and with both Link 3 and Link 4 uncertainties).

(7) `Arm4D_TrajGen.m` randomly generates the tracking references and disturbances for all stationary upper limb exoskeleton (Arm4D) simulations.

(8) `TAWE_EKFParam.m` contains the default EKF parameters for the wrist kinematic identifications in the control simulation of wearable wrist exoskeleton (TAWE).

(9) `TAWE_Figures.m` generates the plots for TAWE simulations. The plots generated are for Fig. 6 in the main paper.

(10) `TAWE_JacFuncGen.m` generates the uncertain parameter Jacobian function introduced by a unknown body. The uncertain parameters includes mass, moment of inertia, and load parameters at load tetrahedron (for center of mass estimation). The generated function is used in wearable wrist exoskeleton (TAWE) simulations.

(11) `TAWE_Parameters.m` contains the simulation, model, and control parameters for all wearable wrist exoskeleton (TAWE) simulations.

(12) `TAWE_Simulation_NoWristID.m` contains the simulation code for the wearable wrist exoskeleton (TAWE) without wrist kinematic identification (WKI), tremor, or disturbance.

(13) `TAWE_Simulation_WristID.m` contains the simulation code for the wearable wrist exoskeleton with wrist kinematic identification and disturbance. The simulations also feature tremor and its suppression via the proposed controller.

(14) `TAWE_TrajGen.m` randomly generates the tracking references and disturbances for all wearable wrist exoskeleton (TAWE) simulations.

1.3 Common Files

The following folders contain codes and data for 3D visualizations, which will not be discussed in detail:

(1) `./STL/` contains the 3D .stl mesh models for simulations obtained from CAD designs: (`arm4DBasePart.STL`, `arm4DLink1.STL`, `arm4DLink2.STL`, `arm4DLink3.STL`, `arm4DLink4.STL`, `taweArmBase.STL`, `taweArmHand.STL`, `taweExoBaseLink.STL`, `taweExoLink1.STL`, `taweExoLink2.STL`, `taweExoLink3.STL`, `taweExoLink4.STL`, `taweExoLinkJoint.STL`, `taweExoPan.STL`).

(2) `./Visualization/` contains a simple toolbox for 3D visualization during simulations, and codes for plot generations based on collected simulation data: (`XYZFramePatch.mat`, `drawingParam.m`, `jDic.m`, `jObj.m`, `sAxes.m`, `sPatch.m`, `sPlot.m`, `stlread.m`, `subplotPosSet.m`).

The following mathematical functions included in `./CommonFunction/` are used by both simulations (details of the codes are commented in the scripts):

(1) `ekfStep.m` iterates one step of the extended Kalman filter process.

(2) `genFreqTraj.m` generates periodic or quasiperiodic trajectories (and their 1st and 2nd order time-derivatives) composed of harmonic waves of different frequencies, amplitudes, and phases within a provided time span. Note that the amplitudes are randomly generated.

(3) `quat2Mat.m` converts a unit quaternion to its corresponding rotation Matrix.

(4) `quat2YPR.m` converts a unit quaternion to its corresponding yaw-pitch-roll Euler angles.

(5) `quatConj.m` calculates the conjugate of a quaternion.

(6) `quatMultiply.m` cross-multiplies two quaternions.

(7) `skew3.m` convert 3×1 vector to a skew symmetric matrix.

(8) `rk4.m` numerically integrates an ODE with Runge-Kutta 4th method.

(9) `ypr2Mat.m` converts a set of yaw-pitch-roll Euler angles to its corresponding rotation Matrix.

(10) `ypr2Quat.m` converts a set of yaw-pitch-roll Euler angles to its corresponding unit quaternion.

The `./Data/` folder contains the simulation data for both simulations. These files can be generated by the main simulation scripts:

(1) `simLink34Data.mat` contains the simulation results for Arm4D simulation (Fig. 3), which assumes no disturbance, and both Link 3 and Link 4 have model uncertainties. This data can be generated by `./Arm4D_Simulation_NoDisturbance.m`.

(2) `simLink34DataDisturbance.mat` contains the simulation results for Arm4D simulation (Fig. 4), which assumes model disturbance, and both Link 3 and Link 4 have model uncertainties. The data contains the performances of PD, SMC, and proposed IO-RAC controllers. This data can be generated by `./Arm4D_Simulation_Disturbance.m`.

(3) `simLink34Ref.mat` contains the tracking reference for Arm4D simulation (Fig. 3), which assumes no disturbance, and both Link 3 and Link 4 have model uncertainties. This data can be generated by `./Arm4D_TrajGen.m`.

(4) `simLink34RefDisturbance.mat` contains the tracking reference for Arm4D simulation (Fig. 4), which assumes model disturbance, and both Link 3 and Link 4 have model uncertainties. This data can be generated by `./Arm4D_TrajGen.m`.

(5) `simLink4Data.mat` contains the simulation results for Arm4D simulation (Fig. 2), which assumes no disturbance, and only Link 4 has model uncertainties. This data can be generated by `./Arm4D_Simulation_NoDisturbance.m`.

(6) `simLink4Ref.mat` contains the tracking reference for Arm4D simulation (Fig. 2), which assumes no disturbance, and only Link 4 has model uncertainties. This data can be generated by `./Arm4D_TrajGen.m`.

(7) `simTAWEDataNoWKI.mat` contains the simulation results for TAWE simulation (Fig. 6) that does not involves wrist kinematics identification (WKI). This data can be generated by `./TAWE_Simulation_NoWristID.m`

(8) `simTAWEDataWKI.mat` contains the simulation results for TAWE simulation (Fig. 6) that involves wrist kinematics identification (WKI). The data contains the performances of the controllers in the presence of (1) disturbance only; (2) tremor and disturbance (passive suppression); and (3) tremor, disturbance, and implementation of BMFLC with adaptive control (active suppression). This data can be generated by `./TAWE_Simulation_WristID.m`.

(9) `simTAWERef.mat` contains the tracking reference, disturbance, sensor noises, and tremor for TAWE simulation (Fig. 6). This data can be generated by `./TAWE_TrajGen.m`.

Note: The sample data in `./STL/` and `./Data/` are large and have to be downloaded separately from: `https://github.com/VibRoLab-Group/IORAC_Exo`. The sample tracking references in `./Data/` can lead to simulation results identical to those in the main paper.

1.4 Model-Specific Files

The remaining files are model-specific to either Arm4D or TAWE. The model-specific functions are pre-compiled for 64-bit Windows (`.mexw64`) and Linux (`.mexa64`) systems. Hence, the following will introduce the original `.m` scripts only.

1.4.1 Model Functions for Stationary Upper Limb Exoskeleton (Arm4D)

The following functions in `./ModelFunction_Arm4D/` include the model information for Arm4D:

(1) `Flow_Arm4D.m` is the ODE function of the Arm4D system.

(2) `Frames_Arm4D.mat` lists the coordinate frames of the Arm4D system used in 3D visualization only.

(3) `ModelInfo_Arm4D.mat` contains the symbolic information (e.g., inertia matrix, generalized forces) of the Arm4D system. This file is used in `./Arm4D_InertiaMatrixObservation.m` only.

(4) `System_Arm4D.m` numerically calculates the dynamical properties of the Arm4D system based on states, parameters and inputs.

(5) `numTF_Arm4D.m` numerically calculates the transformations between frames in the Arm4D system, which is used in 3D visualization only.

(6) `uncertainBodyParamJacobianFunc_Arm4D.m` numerically calculates adaptive parameter Jacobian matrix for uncertainties introduced by an unknown body in Arm4D (used for both Link 3 and Link 4 uncertainties).

1.4.2 Model Functions for Wearable Wrist Exoskeleton (TAWE)

The following functions in `./ModelFunction_TAWE/` include the model information for TAWE:

(1) `Flow_TAWE.m` is the ODE function of the TAWE system.

(2) `Frames_TAWE.mat` lists the coordinate frames of the TAWE system used in 3D visualization only.

(3) `System_TAWE.m` numerically calculates the dynamical properties of the TAWE system based on states, parameters and inputs.

(4) `numTF_TAWE.m` numerically calculates the transformations between frames in the TAWE system, which is used in 3D visualization only.

(5) `uncertainBodyParamJacobianFunc_TAWE.m` numerically calculates adaptive parameter Jacobian matrix for uncertainties introduced by an unknown body (on the hand) in TAWE.

(6) `taweWKI.mat` contains the extended Kalman filter (EKF) states, parameters, and functions used in the wrist kinematics identification (WKI) process of TAWE simulation (see `./TAWE_WKIParam.m` for details, also see `./CommmonFunction/ekfStep.m` for usage).

2 Additional Information for Simulations

Here we provide additional information for better insights into the simulation codes, which covers: (a) the multibody formulation and model functions, (b) additional information for Arm4D simulation, and (c) additional information for TAWE simulation.

2.1 Multibody Formulation and Dynamical Model Functions

The multibody models of Arm4D and TAWE are generated with a toolbox in MATLAB [1]. The toolbox adopts a combination of symbolic and numerical multibody modeling based on the Kanes Method [2]. By defining the variables and coordinate frames of the system, the toolbox first recursively constructs the kinematic tree, then formulates the system dynamics based on dynamical objects (body, force, torque, etc.) established on the coordinate frames and kinematic constraints. The following explanation only covers the generated model functions used in the control simulations.

2.1.1 Multibody Dynamics and Formulations

Similar to the definitions in the main paper, the variables in a multibody model are categorized as follows: $t \geq 0$ is the time variable; $\mathbf{q}$ is the generalized coordinate vector; $\mathbf{u}$ is the control input vector. The Arm4D and TAWE models also feature model parameters $\boldsymbol{\kappa}$, which include both known parameters and uncertain parameters ($\mathbf{p}$ in the main paper). Some models may also feature nonholonomic state $\boldsymbol{\rho}$ such as unit quaternions. However, $\boldsymbol{\rho}$ is excluded from further discussions since it does not exist in Arm4D and TAWE models.

Both Arm4D and TAWE models have a number of coordinate frames. After recursive formulation, the kinematic relationship between the Frame i and the global frame can be represented by a homogeneous transformation matrix:

$$\mathbf{T}_i = \begin{bmatrix} \boldsymbol{\Omega}_i(\mathbf{q}, \boldsymbol{\kappa}) & \mathbf{d}_i(\mathbf{q}, \boldsymbol{\kappa}) \\ \mathbf{0}_{1\times 3} & 1 \end{bmatrix} \tag{1}$$

where $\boldsymbol{\Omega}_i$ is the 3D rotation matrix, and $\mathbf{d}_i$ is the 3D translational displacement with respect to the global frame. Note that the rotation $\boldsymbol{\Omega}_i$ can also be numerically converted to a unit quaternion $\boldsymbol{\mu}$. Furthermore, we can define the angular velocity of Frame i with respect to the global frame as $\boldsymbol{\omega}_i$, which leads to

$$\dot{\mathbf{d}}_i = \mathbf{J}_{d,i}(\mathbf{q}, \boldsymbol{\kappa})\dot{\mathbf{q}}; \quad \boldsymbol{\omega}_i = \mathbf{J}_{\omega,i}(\mathbf{q}, \boldsymbol{\kappa})\dot{\mathbf{q}} \tag{2}$$

where $\mathbf{J}_{d,i}$ and $\mathbf{J}_{\omega,i}$ are the translational and rotational Jacobian matrices of Frame i, respectively. These properties will be used to form the dynamical properties of the system. For example, the inertia properties of Body j with mass $m_j > 0$ ($m_j \in \boldsymbol{\kappa}$) and moment of inertia matrix $\boldsymbol{\Phi}_j(\boldsymbol{\kappa})$ (note that $\boldsymbol{\Phi}_j = \boldsymbol{\Phi}_j^{\mathrm{T}} > 0$) established at Frame i can be written as

$$\mathbf{J}_{r,j} = \begin{bmatrix} \mathbf{J}_{d,i} \\ \mathbf{J}_{\omega,i} \end{bmatrix}; \quad \mathbf{J}_{l,j} = \begin{bmatrix} m_j\mathbf{J}_{d,i} \\ \boldsymbol{\Omega}_i\boldsymbol{\Phi}_j\boldsymbol{\Omega}_i^{\mathrm{T}}\mathbf{J}_{\omega,i} \end{bmatrix}; \quad \mathbf{M}_j = \mathbf{J}_{l,j}^{\mathrm{T}}\mathbf{J}_{r,j}; \quad \mathbf{C}_j = \mathbf{J}_{l,j}^{\mathrm{T}}\dot{\mathbf{J}}_{r,j} + \mathbf{J}_{\omega,i}^{\mathrm{T}}\underline{\mathrm{skew}}(\boldsymbol{\omega}_i)\boldsymbol{\Omega}_i\boldsymbol{\Phi}_j\boldsymbol{\Omega}_i^{\mathrm{T}}\mathbf{J}_{\omega,i} \tag{3}$$

with the skew matrix conversion function

$$\underline{\mathrm{skew}}\left(\begin{bmatrix} c_x & c_y & c_z \end{bmatrix}^{\mathrm{T}}\right) = \begin{bmatrix} 0 & -c_z & c_y \\ c_z & 0 & -c_x \\ -c_y & c_x & 0 \end{bmatrix}; \tag{4}$$

where $\mathbf{M}_j$ is the inertia matrix of Body j that satisfies $\mathbf{M}_j = \mathbf{M}_j^{\mathrm{T}} > 0$; and $\mathbf{C}_j$ is the Coriolis and centripetal matrix (C-matrix) of Body j. We also refer to $\mathbf{J}_{l,j}$ and $\mathbf{J}_{r,j}$ as the left and right inertia component matrices, respectively. As another example, the input Jacobian matrix $\mathbf{J}_{u,k}$ and generalized force $\mathbf{h}_{u,k}$ of a scalar input u_k established at the $-z$ direction of Frame i can be calculated as

$$\mathbf{J}_{u,k} = \begin{bmatrix} 0 & 0 & -1 \end{bmatrix}\mathbf{J}_{d,i}; \quad \mathbf{h}_{u,k} = \mathbf{J}_{u,k}^{\mathrm{T}}u_k \tag{5}$$

After the summation of inertia matrices, C-matrices, and generalized forces, the generic structure of the models similar to Eq.(1) in the main paper can be written as

$$\mathbf{M}(\mathbf{q}, \boldsymbol{\kappa})\ddot{\mathbf{q}} = -\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\kappa})\dot{\mathbf{q}} - \mathbf{h}(t, \mathbf{q}, \dot{\mathbf{q}}, \boldsymbol{\kappa}) + \mathbf{J}_u^{\mathrm{T}}(\mathbf{q}, \boldsymbol{\kappa})\mathbf{u} + \mathbf{J}_\lambda^{\mathrm{T}}(\mathbf{q}, \boldsymbol{\kappa})\boldsymbol{\lambda} \tag{6}$$

Here, $\mathbf{h}$ excludes the generalized forces of control input $\mathbf{u}$ and perturbation/disturbance $\mathbf{w}$ (currently considered as a part of $\mathbf{u}$). A difference between this model and Eq.(1) in the main paper is the constraint force enforced by the Lagrangian multiplier $\boldsymbol{\lambda}$. The constraint Jacobian matrix $\mathbf{J}_\lambda$ is acquired from kinematic constraint $\mathbf{r}_\lambda$, which, along with its 1st and 2nd order time-derivatives, can be written as

$$\mathbf{r}_\lambda(\mathbf{q}) = 0; \quad \dot{\mathbf{r}}_\lambda = \mathbf{J}_\lambda\dot{\mathbf{q}} = 0; \quad \ddot{\mathbf{r}}_\lambda = \mathbf{J}_\lambda\ddot{\mathbf{q}} + \dot{\mathbf{J}}_\lambda\dot{\mathbf{q}} = 0 \tag{7}$$

Note that kinematic constraints do not exist in the Arm4D model, but exist in the TAWE model due to the closed kinematic chain formed by the user bodies and exoskeleton mechanisms [3]. A general approach to calculate the constraint force is by substituting $\ddot{\mathbf{q}}$ from Eq.(6) into $\ddot{\mathbf{r}}_\lambda$ from Eq.(7)

$$\boldsymbol{\lambda} = \boldsymbol{\Lambda}_\lambda^{-1}(\mathbf{J}_\lambda\mathbf{M}^{-1}(\mathbf{C} + \mathbf{h} - \mathbf{J}_u^{\mathrm{T}}\mathbf{u}) - \dot{\mathbf{J}}_\lambda\dot{\mathbf{q}}) \tag{8}$$

where $\boldsymbol{\Lambda}_\lambda = \mathbf{J}_\lambda\mathbf{M}^{-1}\mathbf{J}_\lambda^{\mathrm{T}}$ is defined as the constraint decoupling matrix.

2.1.2 Dynamical Model Functions in Control Simulations

Based on Eq.(3) and Eq.(4), the scripts `./Arm4D_JacFuncGen.m` and `./TAWE_JacFuncGen.m` respectively generates `./ModelFunction_Arm4D/uncertainBodyParamJacobianFunc_Arm4D.m` and `./ModelFunction_TAWE/uncertainBodyParamJacobianFunc_TAWE.m` to calculate the uncertain parameter Jacobian $\mathbf{J}_p$ introduced by an uncertain body. The calculated $\mathbf{J}_p$ satisfies the structure in Eq.(3) and Eq.(15b) from the main paper.

The ODE functions of the multibody models (i.e., `./ModelFunction_Arm4D/Flow_Arm4D.m` and `./ModelFunction_TAWE/Flow_TAWE.m`) are formulated by substituting $\boldsymbol{\lambda}$ from Eq.(8) back into Eq.(6). The ODE function also adopts an auxiliary "soft" generalized constraint force $\mathbf{h}_\lambda$ that eliminates the numerical

drift error accumulated during numerical integration [1, 4].

Finally, the "System" functions for both models (i.e., `./ModelFunction_Arm4D/System_Arm4D.m` and `./ModelFunction_TAWE/System_TAWE.m`) numerically calculate and output a collection of the above mentioned kinematic and dynamical properties. The input and output arguments of the functions are

$$\left(\mathbf{T},\ \begin{bmatrix}\dot{\mathbf{d}}\\ \boldsymbol{\omega}\end{bmatrix},\begin{bmatrix}\dot{\mathbf{J}}_d\\ \dot{\mathbf{J}}_\omega\end{bmatrix}\dot{\mathbf{q}},\begin{bmatrix}\mathbf{J}_d\\ \mathbf{J}_\omega\end{bmatrix},\mathbf{d},\boldsymbol{\mu},\mathbf{M},\mathbf{C}\dot{\mathbf{q}},\mathbf{h},\mathbf{h}_u,\mathbf{J}_u,\mathbf{J}_\lambda,\dot{\mathbf{J}}_\lambda\dot{\mathbf{q}},\mathbf{h}_\lambda,(\mathbf{C}-\mathbf{J}_l^{\mathrm{T}}\dot{\mathbf{J}}_r),\mathbf{J}_l^{\mathrm{T}},\mathbf{J}_r\right)=\mathrm{System}(t,\mathbf{q},\boldsymbol{\kappa},\mathbf{u},\boldsymbol{\rho}) \quad (9)$$

where each output argument contains the individual properties of corresponding coordinate frames, bodies, force effects, inputs, and constraints.

2.2 Additional Information for Arm4D Simulation

As mentioned in the paper, the feedback controller comparison in `./Arm4D_Simulation_NoDisturbance.m` involves a PD controller (with higher gain), and a sliding mode controller (SMC) [5, 6]. Note that the adaptive control term $\mathbf{J}_p^{\mathrm{T}}\hat{\mathbf{p}}$ and feedforward control term $\mathbf{u}_f$ from Eq.(14) in the main paper remain identical for all cases. Based on Eq.(14) and Eq.(15) in the main paper, recall that $\boldsymbol{\xi}=\dot{\boldsymbol{\epsilon}}+\mathbf{K}_1\boldsymbol{\epsilon}$ where $\boldsymbol{\epsilon}=\mathbf{q}-\mathbf{r}$ is the position tracking error, the feedback controller $\mathbf{u}_2$ for the PD and SMC controllers are designed as

$$\mathbf{u}_{b,\mathrm{PD}}=-4c_1\mathbf{K}_2\boldsymbol{\xi};\quad \mathbf{u}_{b,\mathrm{SMC}}=-\mathbf{K}_{\mathrm{S}}\underline{\boldsymbol{\delta}(\boldsymbol{\xi})}-c_1\mathbf{K}_2\boldsymbol{\xi}; \quad (10)$$

where the gain switching control term is realized with the stabilizing sliding surface for $\mathbf{x}$ selected as $\boldsymbol{\xi}=0$, a positive definite diagonal gain matrix $\mathbf{K}_{\mathrm{S}}=\mathrm{diag}(\underline{[1;0.5;1;0.5]})$, and a real-domain function $\boldsymbol{\delta}$ that approximates (or is identical to) the sign function, which satisfies the following conditions:

$$(1):\ \underline{\boldsymbol{\delta}(z)}>0\ \text{ for }\ z>0;\quad (2):\ \underline{\boldsymbol{\delta}(\boldsymbol{\xi})}=\mathrm{diag}(\underline{\mathrm{sign}(\boldsymbol{\xi})})\underline{\boldsymbol{\delta}(|\boldsymbol{\xi}|)};\quad (3):\ \lim_{|z|\to\infty}\underline{\boldsymbol{\delta}(z)}=\underline{\mathrm{sign}(z)} \quad (11)$$

In the simulation, we have chosen $\boldsymbol{\delta}$ to be

$$\underline{\boldsymbol{\delta}(z)}=2(\underline{\mathrm{sigmoid}(c_{\mathrm{S}}z)}-0.5)=\frac{1-e^{-c_{\mathrm{S}}z}}{1+e^{-c_{\mathrm{S}}z}} \quad (12)$$

where $c_S=100$. This continuous function mimics the sign function to slightly reduce the chattering problem.

Stability: The stability proofs for both PD and SMC controllers adopt the same Lyapunov function as shown in Eq.(17) and Eq.(A-1) from the main paper:

$$\mathcal{V}(\mathbf{x},\hat{\mathbf{p}})=\frac{1}{2}\mathbf{x}^{\mathrm{T}}\begin{bmatrix}\mathbf{K}_3+\mathbf{K}_1\mathbf{M}\mathbf{K}_1 & \mathbf{K}_1\mathbf{M}\\ \mathbf{M}\mathbf{K}_1 & \mathbf{M}\end{bmatrix}\mathbf{x}+\frac{1}{2}\tilde{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\Gamma}\tilde{\mathbf{p}}=(\boldsymbol{\xi}^{\mathrm{T}}\mathbf{M}\boldsymbol{\xi}+\boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_3\boldsymbol{\epsilon}+\tilde{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\Gamma}\tilde{\mathbf{p}})/2 \quad (13)$$

However, at the step of Eq.(A-3) from the main paper, we set $\mathbf{J}_w=0$ in the proofs of PD and SMC controllers. Therefore, similar to Eq.(A-4) from the main paper, we have

$$\dot{\mathcal{V}}=\boldsymbol{\xi}^{\mathrm{T}}\mathbf{u}_2+\big(\boldsymbol{\xi}^{\mathrm{T}}(\dot{\mathbf{M}}-2\mathbf{C})\boldsymbol{\xi}\big)/2+\boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_3\dot{\boldsymbol{\epsilon}}+\tilde{\mathbf{p}}^{\mathrm{T}}\boldsymbol{\Gamma}\big(\dot{\hat{\mathbf{p}}}+\boldsymbol{\Gamma}^{-1}\mathbf{J}_p\boldsymbol{\xi}\big) \quad (14)$$

Again, for a multibody system, $\dot{\mathbf{M}}-2\mathbf{C}$ is a skew matrix so that $\boldsymbol{\xi}^{\mathrm{T}}(\dot{\mathbf{M}}-2\mathbf{C})\boldsymbol{\xi}=0$. By inserting the updater of parameter estimates $\dot{\hat{\mathbf{p}}}$ in Eq.(14b) from the main paper, we have

$$\dot{\mathcal{V}}=\boldsymbol{\xi}^{\mathrm{T}}\mathbf{u}_2+\boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_3\dot{\boldsymbol{\epsilon}} \quad (15)$$

From here, the stability proof will respectively branch into the cases of PD and SMC controllers:

(1) In the case of PD feedback controller $\mathbf{u}_2=\mathbf{u}_{b,\mathrm{PD}}$, we have

$$\dot{\mathcal{V}}=-4c_1\boldsymbol{\xi}^{\mathrm{T}}\mathbf{K}_2\boldsymbol{\xi}+\boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_3\dot{\boldsymbol{\epsilon}}=-c_1\boldsymbol{\xi}^{\mathrm{T}}(4\mathbf{K}_2-\mathbf{K}_5/c_1)\boldsymbol{\xi}-\dot{\boldsymbol{\epsilon}}^{\mathrm{T}}\mathbf{K}_5\dot{\boldsymbol{\epsilon}}-\boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_1\mathbf{K}_5\mathbf{K}_1\boldsymbol{\epsilon} \quad (16)$$

where $\mathbf{K}_5$ is a positive definite symmetric matrix. It is required that $(4\mathbf{K}_2-\mathbf{K}_5/c_1)>0$, and: (1) $\mathbf{K}_5=\mathbf{I}_{n_q}/2$ for $\mathbf{K}_1=\mathbf{K}_3$; or (2) $\mathbf{K}_5=\mathbf{K}_1^{-1}\mathbf{K}_3/2=\mathbf{K}_3\mathbf{K}_1^{-1}/2$ for diagonal $\mathbf{K}_1$ and $\mathbf{K}_3$ matrices. Thus, with any $c_1\geq 2$, there exists a smooth function $\underline{Q(\mathbf{x})}\geq 0$ ($Q=0$ iff. $\mathbf{x}=\mathbf{0}$) so that

$$\dot{\mathcal{V}}\leq -Q \quad (17)$$

and $\dot{\mathcal{V}}=-Q$ iff. $\mathbf{x}=\mathbf{0}$.

(2) In the case of SMC feedback controller $\mathbf{u}_2 = \mathbf{u}_{b,\text{SMC}}$, we have

$$\begin{aligned}\dot{\mathcal{V}} =& -\boldsymbol{\xi}^{\mathrm{T}}\mathbf{K}_{\mathrm{S}}\boldsymbol{\delta}(\underline{\boldsymbol{\xi}}) - c_1\boldsymbol{\xi}^{\mathrm{T}}\mathbf{K}_2\boldsymbol{\xi} + \boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_3\dot{\boldsymbol{\epsilon}} \\ =& -(|\boldsymbol{\xi}|)^{\mathrm{T}}\mathbf{K}_{\mathrm{S}}\boldsymbol{\delta}(\underline{|\boldsymbol{\xi}|}) - c_1\boldsymbol{\xi}^{\mathrm{T}}(4\mathbf{K}_2 - \mathbf{K}_6/c_1)\boldsymbol{\xi} - \dot{\boldsymbol{\epsilon}}^{\mathrm{T}}\mathbf{K}_6\dot{\boldsymbol{\epsilon}} - \boldsymbol{\epsilon}^{\mathrm{T}}\mathbf{K}_1\mathbf{K}_6\mathbf{K}_1\boldsymbol{\epsilon}\end{aligned} \tag{18}$$

where $\mathbf{K}_6$ is a positive definite symmetric matrix. Similar to previous case, it is required that $(\mathbf{K}_2 - \mathbf{K}_6/c_1) > 0$, and: (1) $\mathbf{K}_6 = \mathbf{I}_{n_q}/2$ for $\mathbf{K}_1 = \mathbf{K}_3$; or (2) $\mathbf{K}_6 = \mathbf{K}_1^{-1}\mathbf{K}_3/2 = \mathbf{K}_3\mathbf{K}_1^{-1}/2$ for diagonal $\mathbf{K}_1$ and $\mathbf{K}_3$ matrices. Finally, for the gain switching term, since $\mathbf{K}_{\mathrm{S}}$ is positive definite and diagonal, based on the conditions in Eq.(11), we obtain that $-(|\boldsymbol{\xi}|)^{\mathrm{T}}\mathbf{K}_{\mathrm{S}}\boldsymbol{\delta}(\underline{|\boldsymbol{\xi}|}) < 0$ for any $\boldsymbol{\xi} \neq \mathbf{0}$. Thus, with any $c_1 \geq 2$, there exists a smooth function $Q(\underline{\mathbf{x}}) \geq 0$ ($Q = 0$ iff. $\mathbf{x} = \mathbf{0}$) so that

$$\dot{\mathcal{V}} \leq -Q \tag{19}$$

and $\dot{\mathcal{V}} = -Q$ iff. $\mathbf{x} = \mathbf{0}$.

Hence, both $\mathbf{u}_{b,\text{PD}}$ and $\mathbf{u}_{b,\text{SMC}}$ in the simulations are globally asymptotically stabilizing feedback controllers.

2.3 Additional Information for TAWE Simulation

The modeling detail of TAWE can be found in our previous work [3]. The dynamical model of TAWE generated with the toolbox [1] has also been numerically validated by the V-Rep [3, 7].

The following discusses a few details about the control simulations of TAWE, which covers the coupling of human and exoskeleton system, the wrist kinematic identification, and tremor suppression via the proposed controller in the main paper.

2.3.1 Coupling of Subsystems

In Section 2.1.1, the coupling between multibody systems through kinematic constraints is realized with the Lagrange multiplier $\boldsymbol{\lambda}$ in Eq.(8). However, Eq.(8) does not reduce the model dimensionality to the degrees of freedom (DOF) of the constrained system. To be specific, the 6-DOF TAWE mechanism will be fully constrained to the forearm. Following the definitions in the main paper, the human and exoskeleton subsystems can be written as

$$\mathbf{M}_1\ddot{\mathbf{q}}_1 = -\mathbf{C}_1\dot{\mathbf{q}}_1 - \mathbf{h}_1 + \mathbf{J}_{u,1}^{\mathrm{T}}\mathbf{u}_1 + \mathbf{J}_{w,1}^{\mathrm{T}}\mathbf{w}_1 + \mathbf{J}_{\lambda,1}^{\mathrm{T}}\boldsymbol{\lambda} \tag{20a}$$

$$\mathbf{M}_2\ddot{\mathbf{q}}_2 = -\mathbf{C}_2\dot{\mathbf{q}}_2 - \mathbf{h}_2 + \mathbf{J}_{u,2}^{\mathrm{T}}\mathbf{u}_2 + \mathbf{J}_{w,2}^{\mathrm{T}}\mathbf{w}_2 + \mathbf{J}_{\lambda,2}^{\mathrm{T}}\boldsymbol{\lambda} \tag{20b}$$

where "1" and "2" are labels of the human and exoskeleton subsystems, respectively. Similar to Section 2.1.1, the two subsystems are coupled by the kinematic constraints $\mathbf{r}_\lambda$. As explained in the main paper, $\mathbf{r}_\lambda$ not only constrains the exoskeleton motion $\mathbf{q}_2$ to the wrist motion $\mathbf{q}_1 = [\theta_{\text{RUD}},\ \theta_{\text{FE}},\ \theta_{\text{SP}}]^{\mathrm{T}}$ (i.e., radial-ulnar deviation (RUD), flexion-extension (FE), and supination-pronation (SP)), but also internally constrains θ_{SP} to $\mathbf{q} = [\theta_{\text{RUD}},\ \theta_{\text{FE}}]^{\mathrm{T}}$, since the wrist is commonly considered a 2-DOF condyloid/ellipsoidal joint. Hence, we have

$$\dot{\mathbf{r}}_\lambda = \mathbf{0} = \mathbf{J}_{c,q}\dot{\mathbf{q}} + \mathbf{J}_{c,\rho}\dot{\boldsymbol{\rho}}; \quad \boldsymbol{\rho} = \begin{bmatrix}\theta_{\text{SP}} & \mathbf{q}_2^{\mathrm{T}}\end{bmatrix}^{\mathrm{T}} \tag{21}$$

Note that $\boldsymbol{\rho}$ here is the constrained internal state calculated through a nonholonomic manner. The dynamics of $\boldsymbol{\rho}$ can be obtained as

$$\dot{\boldsymbol{\rho}} = -\mathbf{J}_{c,\rho}^{-1}\mathbf{J}_{c,q}\dot{\mathbf{q}}; \quad \ddot{\boldsymbol{\rho}} = -\mathbf{J}_{\lambda,\rho}^{-1}(\mathbf{J}_{\lambda,q}\ddot{\mathbf{q}} + \dot{\mathbf{J}}_{\lambda,q}\dot{\mathbf{q}} - \dot{\mathbf{J}}_{\lambda,\rho}\mathbf{J}_{\lambda,\rho}^{-1}\mathbf{J}_{\lambda,q}\dot{\mathbf{q}}) \tag{22}$$

Unlike the previous case where the constrained dynamics is formulated through a general approach in Eq.(8), for TAWE, we can directly obtain the constrained dynamical model based on Eq.(20) and Eq.(22):

$$\mathbf{M}\ddot{\mathbf{q}} = -\mathbf{C}\dot{\mathbf{q}} - \mathbf{h} + \mathbf{J}_u\mathbf{u} + \mathbf{J}_w^{\mathrm{T}}\mathbf{w} \tag{23a}$$

$$\dot{\boldsymbol{\rho}} = -\mathbf{J}_{c,\rho}^{-1}\mathbf{J}_{c,q}\dot{\mathbf{q}} \tag{23b}$$

where

$$\mathbf{J}_c = \begin{bmatrix} \mathbf{I}_2 \\ -\mathbf{J}_{c,\rho}^{-1}\mathbf{J}_{c,q} \end{bmatrix}; \quad \mathbf{M} = \mathbf{J}_c^{\mathrm{T}} \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 \end{bmatrix} \mathbf{J}_c; \quad \mathbf{C} = \mathbf{J}_c^{\mathrm{T}} \begin{bmatrix} \mathbf{C}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{C}_2 \end{bmatrix} \mathbf{J}_c + \mathbf{J}_c^{\mathrm{T}} \begin{bmatrix} \mathbf{M}_1 & \mathbf{0} \\ \mathbf{0} & \mathbf{M}_2 \end{bmatrix} \dot{\mathbf{J}}_c; \tag{24a}$$

$$\mathbf{h} = \mathbf{J}_c^{\mathrm{T}} \begin{bmatrix} \mathbf{h}_1 \\ \mathbf{h}_2 \end{bmatrix}; \quad \mathbf{J}_u = \mathbf{J}_c^{\mathrm{T}} \begin{bmatrix} \mathbf{J}_{u,1}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_{u,2}^{\mathrm{T}} \end{bmatrix}; \quad \mathbf{u} = \begin{bmatrix} \mathbf{u}_1 \\ \mathbf{u}_2 \end{bmatrix}; \quad \mathbf{J}_w = \mathbf{J}_c^{\mathrm{T}} \begin{bmatrix} \mathbf{J}_{w,1}^{\mathrm{T}} & \mathbf{0} \\ \mathbf{0} & \mathbf{J}_{w,2}^{\mathrm{T}} \end{bmatrix}; \quad \mathbf{w} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \end{bmatrix} \tag{24b}$$

The above formulation is adopted in `./TAWE_Simulation_NoWristID.m` and `./TAWE_Simulation_WristID.m`.

2.3.2 Wrist Kinematics Identification

In `./TAWE_Simulation_WristID.m`, a simple real-time WKI through extended Kalman filter (EKF) [8] is carried out as proposed in [9]. The goal is to identify the kinematic model between two coordinate frames at TAWE attachment locations on the forearm (Frame A) and hand dorsum (Frame B), respectively.

The WKI includes two parts: (1) the identification of the constraint between θ_{SP} and $[\theta_{\mathrm{RUD}},\ \theta_{\mathrm{FE}}]$; and (2) the translational displacement between Frame A and Frame B. In the above TAWE model, $\mathbf{q}_1$ is also a set of Euler angle that represents the rotation matrix $\mathbf{\Omega}_{\mathrm{w},1}$. The regression of rotational constraint adopts a quaternion-based model structure [3, 9]:

$$r_{\mathrm{w}}(\mathbf{q}_1, c_{\mathrm{w}}) = \begin{bmatrix} 1 & \mathbf{0}_{1\times 3} \end{bmatrix} \left(\begin{bmatrix} \mathbf{0}_{1\times 2} & 1 & 0 \end{bmatrix}^{\mathrm{T}} \times \boldsymbol{\mu}_1(\mathbf{q}_1) \right) + c_{\mathrm{w}} \sin(\theta_{\mathrm{RUD}}/2) \sin(\theta_{\mathrm{FE}}/2) = 0 \tag{25}$$

where c_μ is an unknown parameter, $\boldsymbol{\mu}_1$ is the unit quaternion that represents the same 3D rotation as $\mathbf{\Omega}_{q,1}$. Note that based on the value of c_{w}, r_{w} can represent different rotation models [3, 9]. The 3D translational displacement model between Frame A and Frame B can be written as:

$$\mathbf{d}_{\mathrm{w}}(\mathbf{q}_1, \mathbf{d}_{\mathrm{w},0}, \mathbf{d}_{\mathrm{w},1}, \boldsymbol{\mu}_0) = \mathbf{d}_{\mathrm{w},0} + \mathbf{\Omega}_{\mathrm{w},0}(\boldsymbol{\mu}_0)\mathbf{\Omega}_{\mathrm{w},1}(\mathbf{q}_1)\mathbf{d}_{\mathrm{w},1} \tag{26}$$

where $\mathbf{d}_{\mathrm{w},0}$ and $\mathbf{d}_{\mathrm{w},1}$ are unknown translational displacement parameters, and $\boldsymbol{\mu}_0$ is an unknown unit quaternion that represents the initial rotation $\mathbf{\Omega}_{\mathrm{w},0}$.

Note that in the current simulation, the WKI model structure in Eq.(25) and Eq.(26) is identical to the wrist kinematic structure in the TAWE multibody model. We also add sensor noises to the WKI process as a more realistic simulation condition. The EKF model functions of WKI are contained in `./ModelFunction_TAWE/taweWKI.mat`.

It is important to note that during the simulation, we assume that the sensors in TAWE can only provide $\mathbf{\Omega}_{\mathrm{A,B}}$, which is the rotation from Frame A to Frame B is calculated by

$$\mathbf{\Omega}_{\mathrm{A,B}} = \mathbf{\Omega}_{\mathrm{w},0}\mathbf{\Omega}_{\mathrm{w},1} \tag{27}$$

Hence, $\mathbf{q}_1$ is also assumed to be unknown, and its estimation $\hat{\mathbf{q}}_1$ is obtained by $\mathbf{\Omega}_{\mathrm{A,B}}$ and the estimated $\hat{\mathbf{\Omega}}_{\mathrm{w},0}$. The controller is designed with $\hat{\mathbf{q}}_1$ and the estimated parameters, while the tracking performance shown in Fig. 6 of the main paper is evaluated based on the true value of $\mathbf{q}_1$.

2.3.3 Tremor Suppression with Adaptive Control

The band-limited multi-frequency Fourier linear combiner (BMFLC) is a model used for real-time modeling of tremors [10, 11]. This model assumes that tremor signals are combinations of harmonic waves with different frequencies within a certain bandwidth. The structure of a BMFLC model with a total of η frequency components can be written as

$$u_{\mathrm{BMFLC}}(t) = \sum_{i=1}^{\eta} \left(p_{u,i} \sin(c_{\omega,i}t) + p_{u,i+\eta} \cos(c_{\omega,i}t) \right) \tag{28}$$

where $c_{\omega,i}$ is the ith constant frequency, and $p_{u,i}$ is the ith uncertain amplitude parameter. Notice that all the harmonic terms in BMFLC are combined linearly. Hence, we can obtain the parameter Jacobian matrix for the 2η uncertain amplitudes as

$$\mathbf{J}_{p,\mathrm{BMFLC}} = \begin{bmatrix} \sin(c_{\omega,1}t) & \sin(c_{\omega,2}t) & \cdots & \sin(c_{\omega,\eta}t) & \cos(c_{\omega,1}t) & \cos(c_{\omega,2}t) & \cdots & \cos(c_{\omega,\eta}t) \end{bmatrix}^{\mathrm{T}} \tag{29}$$

so that

$$u_{\mathrm{BMFLC}} = \mathbf{J}_{p,\mathrm{BMFLC}}^{\mathrm{T}} \begin{bmatrix} p_{u,1} & p_{u,2} & \cdots & p_{u,2\eta} \end{bmatrix}^{\mathrm{T}} \tag{30}$$

Hence, BMFLC can be directly implemented as a reference model for the proposed robust adaptive controller in the main paper. The application of BMFLC for tremor suppression (where tremors are considered as model uncertainties) is shown in `./TAWE_Simulation_WristID.m`.

References

[1] Jiamin Wang, Vinay R Kamidi, and Pinhas Ben-Tzvi. A multibody toolbox for hybrid dynamic system modeling based on nonholonomic symbolic formalism. In *ASME 2018 Dynamic Systems and Control Conference*, pages V003T29A003–V003T29A003. American Society of Mechanical Engineers, 2018.

[2] Thomas R Kane and David A Levinson. *Dynamics, theory and applications*. McGraw Hill, 1985.

[3] Jiamin Wang and Oumar Barry. Multibody analysis and control of a full-wrist exoskeleton for tremor alleviation. *Journal of Biomechanical Engineering*, 2020.

[4] Andrew Witkin. An introduction to physically based modeling: Constrained dynamics. *Robotics Institute*, 1997.

[5] Brahim Brahmi, Abdelkrim Brahmi, Maarouf Saad, Guy Gauthier, and Mohammad Habibur Rahman. Robust adaptive tracking control of uncertain rehabilitation exoskeleton robot. *Journal of Dynamic Systems, Measurement, and Control*, 141(12), 2019.

[6] Fucheng Cao, Chunfeng Li, and Yuanchun Li. Robust sliding mode adaptive control for lower extremity exoskeleton. In *2015 Chinese Automation Congress (CAC)*, pages 400–405. IEEE, 2015.

[7] Eric Rohmer, Surya PN Singh, and Marc Freese. V-rep: A versatile and scalable robot simulation framework. In *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1321–1326. IEEE, 2013.

[8] Simon S Haykin. *Adaptive filter theory*. Pearson Education India, 2005.

[9] Jiamin Wang and Oumar Barry. Real-time identification of wrist kinematics via sparsity-promoting extended kalman filter based on ellipsoidal joint formulation. *(Submitted to) IEEE Transactions on Biomedical Engineering*, 2020.

[10] Kalyana C Veluvolu and Wei Tech Ang. Estimation of physiological tremor from accelerometers for real-time applications. *Sensors*, 11(3):3020–3036, 2011.

[11] Shengxin Wang, Yongsheng Gao, Jie Zhao, and Hegao Cai. Adaptive sliding bandlimited multiple fourier linear combiner for estimation of pathological tremor. *Biomedical Signal Processing and Control*, 10: 260–274, 2014.