ExoKit: A Toolkit for Rapid Prototyping of Interactions for Arm-based Exoskeletons

Marie Muehlhaus, Alexander Liggesmeyer, Jürgen Steimle

Supplemental Material: Algorithms

This supplemental material contains the pseudocode for the algorithms presented in section 4 of the paper, organized by the higher-level motion augmentation strategies. The algorithms build on the following basic functions:

- MOVETO$(j_i, \theta_i, \upsilon)$ moves the joint j_i towards a target angle θ_i with a user-specified velocity υ. All angles are absolute unless explicitly stated otherwise. The function internally checks if the goal angle is beyond a joint's calibrated range of motion (RoM). If so, the joint would be moved just towards the bound but never beyond. Unless the torque was updated before (see UPDATE_TORQUE(j_i, τ)), the motor will be moved with the largest possible torque.
- LOCK(j_i) locks the joint j_i in place. UNLOCK(j_i) unlocks the joint j_i.
- UPDATE_TORQUE(j_i, τ) ensures the motor of joint j_i is moving with the specified torque τ; if $\tau = 0$, then the the torque is disabled. TORQUE_IS_ENABLED(j_i) and TORQUE_IS_DISABLED(j_i) indicate whether $\tau > 0$ and $\tau = 0$, respectively.
- GET_ANGLE(j_i), GET_VELOCITY(j_i), GET_DIRECTION(j_i) return the current joint angle, velocity, and movement direction (extension/adduction, flexion/abduction, or both) of j_i.
- GET_MAX_VELOCITY(j_i) returns the maximum velocity that the motor attached to j_i can provide.
- GET_UPPER_RoM(j_i) and GET_LOWER_RoM(j_i) return the maximum and minimum angles of j_i's calibrated range of motion
- IS_MOVING(j_i) indicates whether j_i is currently moving.
- PARALLEL$(s_1, \ldots, s_k)$ executes k statements $s_1, \ldots, s_k$ in parallel (e.g., two MOVETO$(\cdot)$ at the same time).

In the following, we refer to the joint modules attached to the elbow (flexion-extension), side (flexion-extension) and back (abduction-adduction) of the shoulder as j_{elbow}, j_{sside}, and j_{sback}. Without loss of generality, in the following, we assume that (1) $D \in \{$"extension", "flexion", "both"$\}$ for any direction D and (2) $j_i \in \{j_{elbow}, j_{sside}\}$. We leave out code for $D \in \{$"*adduction*", "*abduction*"$\}$ and $j_i = j_{sback}$ which works analogously to extension and flexion and the other joints.

Further, we decided on the values of constants in the code after doing preliminary tests in the research team. This refers to, for instance, $dirChangeSuspensionTime$ (amplify) and $linearGrowth$ (resist).

All algorithms except the one for the waving gesture can also have an optional condition as input. In these algorithms, while-loops will run until they are aborted through this user-defined condition, such as a timer, or an emergency shutdown. To keep the pseudocode abstract, we do not include this input in the *require* parameters and only state the conditions explicitly if needed.

Please note that the algorithms represent the logic of the implemented functions in the form of readable pseudocode, which abstracts away the details of specific programming languages. Our implementation was realised in C++ with an event-based architecture and dedicated Condition and Action classes. The pseudocode below therefore looks different from the actual program structure.

1 Scripted Motions

Wave This function executes a sequence of movements to perform a waving gesture with the arm.

Algorithm 1 Wave

Require: exoskeleton arm with active joints at the elbow, shoulder side and back: j_{elbow},j_{sback}, j_{sside}

PARALLEL(MOVETO($j_{elbow}, 90, 70$), MOVETO($j_{sside}, 0, 70$), MOVETO($j_{sback}, 90, 70$))
PARALLEL(LOCK(j_{sside}), LOCK(j_{sback}), MOVETO($j_{elbow}, 45, 50$))
MOVETO($j_{elbow}, 90, 50$)
MOVETO($j_{elbow}, 45, 50$)
MOVETO($j_{elbow}, 90, 50$)
MOVETO($j_{elbow}, 45, 50$)
PARALLEL(UNLOCK(j_{sside}), UNLOCK(j_{sback}))

Vibrate This function generates a vibrating sensation at a joint j_i through rapid back-and-forth movements of adjustable amplitude and frequency. The function will loop until aborted. An abort can be triggered by a user defined condition or an emergency shutdown.

Algorithm 2 Vibrate

Require: joint j_i, amplitude amp, frequency f

$\upsilon_{required} \leftarrow \min(4f \cdot amp, \text{GET\_MAX\_VELOCITY}(j_i))$ ▷ *ensure that $\upsilon_{required}$ does not exceed the max. available velocity of j_i*
while not aborted **do** ▷ *the loop will run until it is aborted through a user-defined condition, such as a timer, or an emergency shutdown*
    MOVETO($j_i, 2 \cdot amp, \upsilon_{required}, relative$)
    MOVETO($j_i, -2 \cdot amp, \upsilon_{required}, relative$)

---

2 Motion Transfer

Mirror The mirror function facilitates motion transfer between two exoskeletons or joint instances, where in every time interval, the state of the source joints j_s is read and the target joints j_t are moved accordingly. Users can optionally scale the motion up or down by a user-defined factor, while the $\textsc{moveTo}(\cdot)$ function internally ensures that the scaled motion does not exceed the user's calibrated limits of their range of motion. The function runs in a loop until it is aborted.

Algorithm 3 Mirror

Require: target joint j_t, source joint j_s, scale factor f

```
1: while not aborted do
2:     moveTo(j_t, f · get_angle(j_s), get_max_velocity(j_i))
```

3 Effort Modulation

Amplify Amplify applies a torque in a user's inherent motion, while preserving the user's ability to freely navigate in space. The function includes parameters that allow designers to adjust the assistive torque τ_{amp} and to specify whether the strategy should be continuously active or only triggered when the user moves in a certain direction $D \in \{\text{“flexion”}, \text{“extension”}, \text{“both”}\}$. To avoid false activations triggered by slight jerks of the user or amplifying the user beyond a safe speed, the function additionally includes parameters that define the minimum and maximum speed where the amplification should start or pause respectively. The function runs in a loop until it is aborted.

Algorithm 4 Amplify

Require: joint j_i, amplifying torque τ_{amp}, supported movement direction D, minimum velocity to start amplification υ_{start}, max. velocity to which arm should be amplified υ_{stop}

```
dirChangeSuspensionTime ← 200  ▷ a constant that tunes the sensitivity to direction changes (in ms)
lastDirChangeTime, lastTorqueEnable ← 0
firstStep ← true
currentDirIsFlex ← false
while not aborted do
    if (|GET_VELOCITY(j_i)| > |υ_stop|) then
        /* stop amplification because j_i moves too fast */
        UPDATE_TORQUE(0)
    else if (|GET_VELOCITY(j_i)| > |υ_start| ∧ D ∈ {GET_DIRECTION(j_i)} ∪ {"both"} then
        /* j_i moves sufficiently fast in desired direction */
        dirIsFlexUpdated ← GET_DIRECTION(j_i) = flexion
        if currentDirIsFlex ≠ dirIsFlexUpdated then
            /* because direction has changed, we pause the amplification */
            currentDirIsFlex ← dirIsFlexUpdated
            if firstStep then
                firstStep ← false
            else
                lastDirChangeTime ← MILLIS()
            UPDATE_TORQUE(0)
        if lastDirChangeTime + dirChangeSuspensionTime < MILLIS() then
            /* because enough time has passed between direction change and now, we resume amplification */
            if TORQUE_IS_DISABLED(j_i) then
                /* amplify by letting j_i pull the user with τ_amp towards the end of user's RoM. */
                UPDATE_TORQUE(j_i, τ_amp)
                lastTorqueEnable ← MILLIS()
            if currentDirIsFlex then
                MOVETO(j_i, GET_UPPER_ROM(j_i), υ_stop)
            else
                MOVETO(j_i, GET_LOWER_ROM(j_i), υ_stop)
    else if lastTorqueEnable + dirChangeSuspensionTime < MILLIS() then
        /* eventually pause amplification if j_i moves too slow or in opposite direction */
        UPDATE_TORQUE(0)
```

Resist Resist applies a torque opposite to a user's inherent motion, while preserving the user's ability to freely navigate in space. The function includes parameters that allow designers to adjust the resistive torque τ_{res} and to specify whether the strategy should be continuously active or only triggered when the user moves in a certain direction $D \in \{$"flexion", "extension", "both"$\}$. To avoid that the user is slowed down below a specific speed, the function additionally includes a parameter that defined the minimum speed. The function runs in a loop until it is aborted.

Algorithm 5 Resist

Require: joint j_i, resisting torque τ_{res}, supported movement direction D, min. velocity to which arm should be slowed down υ_{stop}

$linearGrowth \leftarrow 100$ ▷ *a constant that tunes the adaptation to velocity changes (in deg/sec)*
$firstStep \leftarrow true$
$currentDirIsFlex \leftarrow false$
while not aborted **do**
  /* *adapt the resistance to j_i's velocity by $\tau_{fraction} \in [0,1]$ percent; the slower the motion, the weaker the resistance, so that the user is not abruptly slowed down when the resistance is triggered.* */
  $\tau_{fraction} \leftarrow \min(1, \max(0, \frac{|\text{GET\_VELOCITY}(j_i)| - |\upsilon_{stop}|}{|\upsilon_{stop}| + linearGrowth}))$
  $\tau_{adapt} \leftarrow \tau_{res} \cdot \tau_{fraction}$
  if $|\text{GET\_VELOCITY}(j_i)| < |\upsilon_{stop}|$ **then** ▷ *pause resistance if j_i is too slow*
    $\text{UPDATE\_TORQUE}(0)$
  else if $D \in \{\text{GET\_DIRECTION}(j_i)\} \cup \{\text{"}both''\}$ **then**
    /* *if j_i moves sufficiently fast in desired direction, provide resistance; if user is not moving anymore in the same direction as in the previous loop iteration, update behavior* */
    $dirIsFlexUpdated \leftarrow \text{GET\_DIRECTION}(j_i) =$ "flexion"

    /* *resist by letting j_i continuously try to push the user towards j_i, making it hard to move away from this fix point* */
    $\text{UPDATE\_TORQUE}(j_i, \tau_{adapt})$ ▷ *update applied torque now*
    if $\text{TORQUE\_IS\_DISABLED}(j_i)$ **then**
      $\text{MOVETO}(j_i, \text{GET\_ANGLE}(j_i), \upsilon_{stop})$ ▷ *torque was disabled, so update fix point*
    else if $currentDirIsFlex \neq dirIsFlexUpdated$ **then**
      $currentDirIsFlex \leftarrow dirIsFlexUpdated$
      if firstStep **then**
        $firstStep \leftarrow false$
      else
        $\text{MOVETO}(j_i, \text{GET\_ANGLE}(j_i), \upsilon_{stop})$ ▷ *direction changed, so update fix point*
  else
    $\text{UPDATE\_TORQUE}(0)$

4 Motion Style

Jerks The jerk function introduces short exoskeleton movements at a user-specified velocity, with varying magnitude and at varying time intervals, creating brief perturbations in the user's inherent motion. The loop runs until aborted or the user-specified number of jerks is performed.

Algorithm 6 Jerk

Require: joint j_i, min./max. jerk size $\theta_{minJerk}/\theta_{maxJerk}$, min./max. time interval between jerks t_{min}/t_{max}, jerk velocity υ, number of jerks $nrJerks$

$lastLoopRun, lastJerk \leftarrow$ MILLIS()
$currentWaitDuration \leftarrow 0$
$jerksPerformed \leftarrow 0$
while not aborted $\wedge$ $jerksPerformed < nrJerks$ **do**
    $currentTime \leftarrow$ MILLIS()
    $deltaLastLoopRun \leftarrow currentTime - lastLoopRun$
    $lastLoopRun \leftarrow currentTime$
    if $\neg$IS_MOVING(j_i) **then** ▷ *no jerks if user does not move*
        $lastJerk \leftarrow lastJerk + deltaLastLoopRun$
        continue
    if $\neg(currentTime > lastJerk + currentWaitDuration)$ **then** ▷ *not yet time for the next jerk*
        continue
    $lastJerk \leftarrow$ MILLIS()
    $currentWaitDuration \leftarrow$ RANDOM($[t_{min}, t_{max}]$)
    $valid \leftarrow true$
    repeat
    /* *randomly search for a valid choice for θ_{jerk}, i.e., within the user's specifications* */
        $jerkDirection \leftarrow$ RANDOM({"flexion", "extension"})
        $\theta_{jerk} \leftarrow$ RANDOM($[\theta_{minJerk}, \theta_{maxJerk}]$)
        if $jerkDirection$ = "extension" **then**
            $\theta_{jerk} \leftarrow -\theta_{jerk}$ ▷ *a jerk towards extension decreases current angle*
        if (GET_ANGLE(j_i) $+ \theta_{jerk} >$ GET_UPPER_RoM(j_i)) $\vee$ (GET_ANGLE(j_i) $+ \theta_{jerk} <$ GET_LOWER_RoM(j_i)) **then**
        /* *if jerk moves out of calibrated RoM, jerk is invalid* */
            $valid \leftarrow$ **false**
    until $valid$
    MOVETO(j_i, GET_ANGLE(j_i) $+ \theta_{jerk}, \upsilon$)
    $jerksPerformed \leftarrow jerksPerformed + 1$

Filter Speed Filter speed tries to keep a user's motion speed within a pre-defined range, by applying assistive torques if they are too slow or resisting ones if they are moving too fast. Designers can fine-tune these effects by setting the velocity range, the forces applied to maintain it and by specifying whether the strategy should be continuously active or only triggered when the user moves in a certain direction $D \in \{$"flexion", "extension", "both"$\}$. The function runs in a loop until it is aborted.

Algorithm 7 Filter Speed

Require: joint j_i, min. and max. speed υ_{min} and υ_{max}, supported movement direction D, amplifying/resisting torque during flexion $\tau_{ampFlex}/\tau_{resFlex}$, amplifying/resisting torque during extension $\tau_{ampExt}/\tau_{resExt}$

while not aborted **do**
    if $D \in \{$"both", "extension"$\}$ **then** ▷ *set filter only for the right direction(s)*
        if $|$GET_VELOCITY$(j_i)| > |\upsilon_{max}|$ **then** ▷ *as long as user is too fast, resist*
            RESIST$(j_i, \tau_{resExt}, D, |\upsilon_{max}|)$ with abort condition $|$GET_VELOCITY$(j_i)| \leq |\upsilon_{max}| \vee$ GET_DIRECTION$(j_i) \neq$ "extension"
        else if $|$GET_VELOCITY$(j_i)| < |\upsilon_{min}|$ **then** ▷ *as long as user is too slow, amplify*
            AMPLIFY$(j_i, \tau_{ampExt}, D, 10, |\upsilon_{min}|)$ with abort condition $|$GET_VELOCITY$(j_i)| \geq |\upsilon_{min}| \vee$ GET_DIRECTION$(j_i) \neq$ "extension"
    if $D \in \{$"both", "flexion"$\}$ **then**
        if $|$GET_VELOCITY$(j_i)| > |\upsilon_{max}|$ **then**
            RESIST$(j_i, \tau_{resFlex}, D, |\upsilon_{max}|)$ with abort condition $|$GET_VELOCITY$(j_i)| \leq |\upsilon_{max}| \vee$ GET_DIRECTION$(j_i) \neq$ "flexion"
        else if $|$GET_VELOCITY$(j_i)| < |\upsilon_{min}|$ **then**
            AMPLIFY$(j_i, \tau_{ampFlex}, D, 10, |\upsilon_{min}|)$ with abort condition $|$GET_VELOCITY$(j_i)| \geq |\upsilon_{min}| \vee$ GET_DIRECTION$(j_i) \neq$ "flexion"

5 Motion Guidance

Constrain to This function constrains the range of motion of a joint j_i to an area around an absolute angle θ_i with range ϵ. As soon as the user attempts to move beyond this range, the exoskeleton tries to move the user back to the boundary, thereby keeping the joint within the specified limits. The function runs in a loop until it is aborted.

Algorithm 8 Constrain to

```
Require: joint j_i, central angle θ_i, range ϵ
while not aborted do
    if GET_ANGLE(j_i) ≤ θ_i − ϵ then
        goalAngle ← θ_i − ϵ + 0.55          ▷ move the user back into the allowed area
        MOVETO(j_i, goalAngle, GET_MAX_VELOCITY(j_i))
    else if GET_ANGLE(j_i) ≥ θ_i + ϵ then
        goalAngle ← θ_i + ϵ − 0.55
        MOVETO(j_i, goalAngle, GET_MAX_VELOCITY(j_i))
```

Guide Towards This function guides the user towards an area centered around θ_i with range ϵ. The exoskeleton applies assistive torques τ_{amp} when moving towards the desired area and resists with τ_{res} otherwise. The function runs in a loop until it is aborted.

Algorithm 9 Guide towards

Require: joint j_i, central angle θ_i, range ϵ, amplifying torque τ_{amp}, resisting torque τ_{res}

while not aborted **do**
  if GET_ANGLE(j_i) $\geq \theta_i + \epsilon$ **then** ▷ *if above upper bounds*
    while GET_ANGLE(j_i) $\geq \theta_i + \epsilon$ **do** ▷ *not back in motion range*
      / to get back in the area, resist during flexion and amplify during extension */*
      if GET_DIRECTION(j_i) = “flexion” **then** ▷ *If user moves further away, resist*
        RESIST(j_i, τ_{res}, “flexion”, 0) with abort condition GET_DIRECTION(j_i) $\neq$ “flexion”
      else ▷ *If user approaches the area, amplify*
        AMPLIFY(j_i, τ_{amp}, “extension”, 0, GET_MAX_VELOCITY(j_i)) with abort condition GET_ANGLE(j_i) $< \theta_i + \epsilon \vee$ GET_DIRECTION(j_i) $\neq$ “extension”
  else if GET_ANGLE(j_i) $\leq \theta_i - \epsilon$ **then** ▷ *if below lower bounds*
    while GET_ANGLE(j_i) $\leq \theta_i + \epsilon$ **do** ▷ *not back in motion range*
      / to get back in area, resist extension and amplify flexion */*
      if GET_DIRECTION(j_i) = “flexion” **then**
        AMPLIFY(j_i, τ_{amp}, “flexion”, 0, GET_MAX_VELOCITY(j_i)) with abort condition GET_ANGLE(j_i) $> \theta_i + \epsilon \vee$ GET_DIRECTION(j_i) $\neq$ “flexion”
      else
        RESIST(j_i, τ_{res}, “extension”, 0) with abort condition GET_DIRECTION(j_i) $\neq$ “extension”

Guide Away This function keeps j_i away from the area around θ_i, applying resistance with torque τ_{res} if the user is approaching the area and assistive torques τ_{amp} otherwise. The function runs in a loop until it is aborted.

Algorithm 10 Guide away

Require: joint j_i, central angle θ_i, range ϵ, amplifying torque τ_{amp}, resisting torque τ_{res}

while not aborted **do**
  if GET_ANGLE(j_i) $\geq (\theta_i - \epsilon) \wedge$ GET_ANGLE(j_i) $\leq (\theta_i + \epsilon)$ **then**
  / if j_i is in dead area, move it out */*
    if GET_ANGLE(j_i) $\geq \theta_i$ **then** ▷ *if closer to upper side, move out there*
      MOVETO($j_i, \theta_i + \epsilon + 0.55, 0$)
    else
      MOVETO($j_i, \theta_i - \epsilon - 0.55, 0$)
  else
  / if j_i not in dead area, use amplify an resist to keep j_i away from it */*
    if GET_ANGLE(j_i) $> \theta_i + \epsilon$ **then**
      while GET_ANGLE(j_i) $> \theta_i + \epsilon$ **do** ▷ *above dead area, prevent user from moving toward it*
        if GET_DIRECTION(j_i) = "flexion" **then**
        / support user when moving away from the dead area */*
          AMPLIFY(j_i, τ_{amp}, "flexion", 0, GET_MAX_VELOCITY(j_i)) with abort condition GET_DIRECTION(j_i) $\neq$ "flexion"
        else
        / prevent user when moving towards the dead area */*
          RESIST(j_i, τ_{res}, "extension", 0) with abort condition GET_ANGLE(j_i) $\leq \theta_i + \epsilon \vee$ GET_DIRECTION(j_i) $\neq$ "extension"
    else if GET_ANGLE(j_i) $< \theta_i - \epsilon$ **then**
      while GET_ANGLE(j_i) $< \theta_i - \epsilon$ **do** ▷ *below dead area, prevent user from moving toward it*
        if GET_DIRECTION(j_i) = "flexion" **then**
          RESIST(j_i, τ_{res}, "flexion", 0) with abort condition GET_ANGLE(j_i) $\geq \theta_i - \epsilon \vee$ GET_DIRECTION(j_i) $\neq$ "flexion"
        else
          AMPLIFY(j_i, τ_{amp}, "extension", 0, GET_MAX_VELOCITY(j_i)) with abort condition GET_DIRECTION(j_i) $\neq$ "extension"