A Modelica Library for Manual Tracking

James J. Potter

VTT Technical Research Centre of Finland
Vuorimiehentie 3, Espoo, Finland

Abstract

Many systems require a human to perform real-time control. To simulate these systems, a dynamic model of the human's control behavior is needed. The field of manual control has developed and validated such models, and their implementation in Modelica could support researchers of human-machine systems. This paper presents a Modelica library with models from the manual control literature. Python-based tools allow users to perform, in real time, the manual tracking tasks they design in Modelica. Parameter values in the manual controller models can be automatically tuned to either maximize tracking performance, or to match recorded control input from a user experiment.
Keywords: manual control, parameter estimation, FMI, Python, OpenModelica, JModelica.org

1 Introduction

There are many situations where a human operator attempts to make the output of a system follow a desired trajectory. For example, the top of Figure 1 shows the task of recording an athlete with a tripod-mounted video camera. The goal of the camera operator is to keep the athlete centered in the camera frame. The actual camera direction is compared to its desired direction (pointed directly at the athlete), and corrective actions are made by applying force to the tripod handle. This activity is similar to eye tracking, where a human keeps a moving object in the center of his or her vision (Jagacinski, 1977). In these activities, the human is an active part of a feedback control system. Other examples of manual tracking tasks include aiming a tank turret (Tustin, 1947; Kleinman and Perkins, 1974), driving an automobile (Bekey et al., 1977; Hess and Modjtahedzadeh, 1990), and piloting an aircraft (McRuer and Jex, 1967).

The bottom of Figure 1 shows a simplified diagram of the task. Blocks represent the camera operator's control behavior, and the camera and tripod's rotational dynamics. The athlete's direction relative to the tripod is the reference signal, $r(t)$, the camera's actual direction is the camera state, $y(t)$, and the angle between the actual and desired directions is the error, $e(t)$. The operator's force on the handle is the command input, $u(t)$.

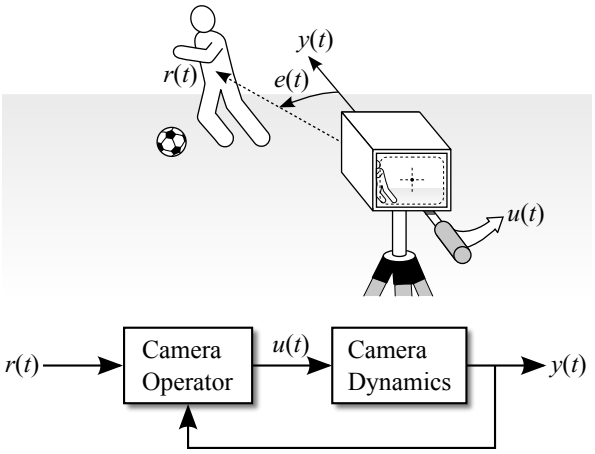


Figure 1. One-dimensional video camera tracking task.

Note that to simulate this system, a model of the human's control behavior must be specified. Such models can be found in the field of *manual control*, which uses the tools and techniques of control theory to study the control behavior of humans. A Modelica library that captures knowledge from this field would be useful to modelers of human-machine systems.

This paper presents a library with models of human control behavior from the manual control literature. In addition, tools allow users to perform manual tracking tasks designed in Modelica, and to tune parameter values in the manual controller models to either maximize tracking performance, or to match recorded control input from user experiments. The next section gives background information about manual control and manual tracking tasks. Then, Sections 3 and 4 present the ManualTracking Modelica library and the supporting functions, respectively. Example tracking tasks are described in Section 5, and conclusions are drawn in Section 6.

2 Manual Tracking

Previous studies have made extensive use of single-axis manual tracking tasks to investigate the control behavior of humans performing continuous control. In a typical experimental tracking task, a human operator views

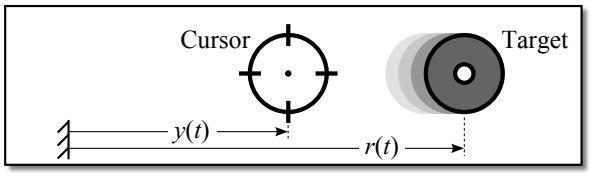


Figure 2. Display for manual tracking task.

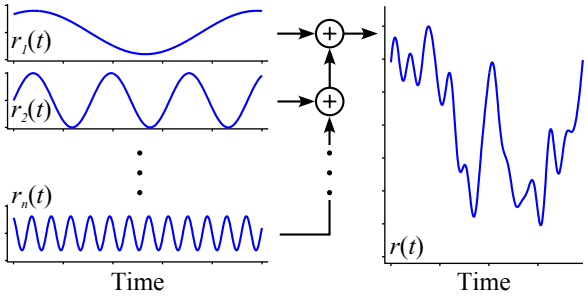


Figure 3. Sum of sines forcing function.

a display on a computer screen and uses an input device, such as a joystick or force stick, to generate control input. An example display is shown in Figure 2. There are two objects on the screen: one is a *target* that represents the reference (desired) state, and the other is a *cursor* that represents the actual state of the controlled system. The human's goal is to make the cursor follow the target as closely as possible.

Many situations require humans to perform multi-axis, multi-loop control tasks, so it might seem that studying one-dimensional control would be an unreasonable oversimplification. However, it has been found that multi-axis tracking performance is highly related to one-axis tracking (Todosiev et al., 1967), and that information about the human controller derived from single-axis tracking tasks can be applied to multi-loop tasks (McRuer et al., 1975).

2.1 Forcing Functions

In the tracking display of Figure 2, the target's motion is prescribed by a *forcing function*. This function should appear random to prevent the operator from predicting future behavior of the target, unless the real-world control task consists of highly predictable signals. This library, and much of manual control theory, focuses on the tracking of unpredictable signals.

From past studies, it has been shown that the sum of 5 or more sine waves is unpredictable to human operators (McRuer et al., 1965). An example summed-sine forcing function is shown in Figure 3. The individual sine waves on the left of Figure 3 are combined to yield the more complicated function on the right. In equation form,

$$r(t) = \sum_{i=1}^{n} A_i sin(\omega_i t + \phi_i), \qquad (1)$$

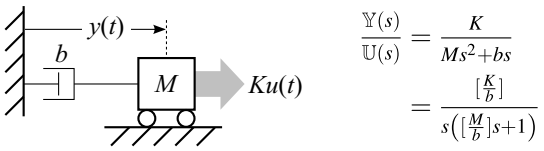


Figure 4. Mechanical example of a controlled element.

where A_i, ω_i, and ϕ_i are the sine wave amplitudes, frequencies, and phase angles, respectively. In general, low-frequency sine waves are given large amplitudes, and waves with increasing frequency are given increasingly small amplitudes (Jagacinski and Flach, 2003). The difficulty of tracking a given forcing function depends heavily on the velocity and acceleration of the target motion (Damveld et al., 2010).

2.2 Controlled Elements

The *controlled element* is the dynamic response of the cursor to control input, and it represents the real-world system under human control. A simple mechanical example is shown in the left side of Figure 4. A rolling cart with mass M is attached to ground by a damper with damping coefficient b, and the control input pushes the cart with a force of magnitude $Ku(t)$. The equivalent controlled-element transfer function is shown in the right side of Figure 4. The cart exhibits a lagged velocity response with time constant M/b and steady-state velocity K/b. The units of these parameters depend on the units chosen for M, b, and K.

Simple models have been used to capture the primary behavior of certain degrees of freedom in aircraft (McRuer and Jex, 1967), automobiles, and other complicated systems. Many experiments have used the simplest controlled elements with position, velocity, and acceleration responses.

2.3 Manual Controllers

Human control behavior while tracking an unpredictable signal can be modeled using tools and techniques from control theory. A specific model will be called a *manual controller* model. These models are generally either structural or algorithmic in nature (McRuer, 1980). Structural models use explicit equations and parameters to model human control pathways and the human's resulting input-output response. Algorithmic models use a more implicit optimal control formulation, where only the human's total response is computed. This library includes only structural models. For a review of both kinds of models, see Hess (2006).

Structural manual controller models have taken many forms, but most include one or more of the control pathways shown in Figure 5. Nearly all controllers include the *compensatory* pathway, which acts on the error $e(t)$ between the reference and measured state. Manual track-

DOI 10.3384/ecp15118329

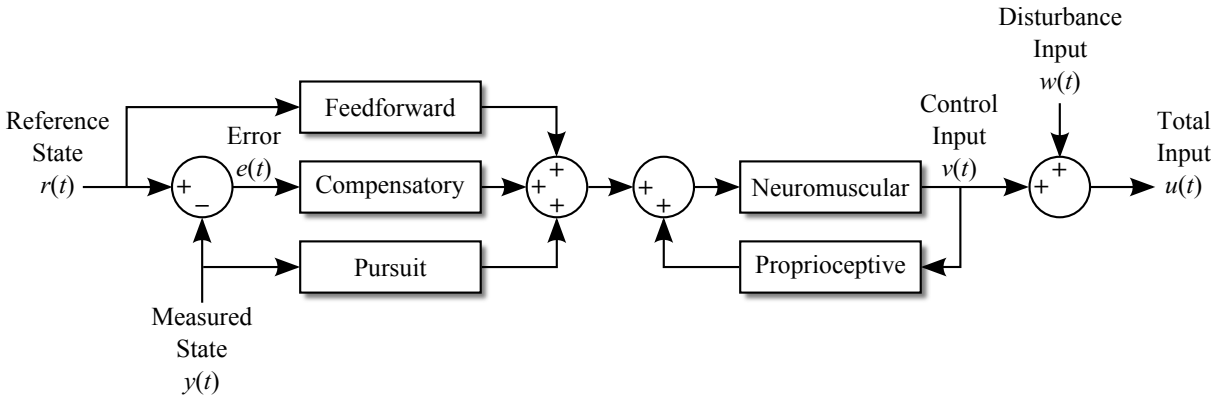


Figure 5. Manual controller signals and control pathways.

ing experiments that display only this error, and not the reference and measured states independently, are called compensatory tracking tasks.

If both the reference state and the measured state are displayed to the human, then they can be used for the *feedforward* and *pursuit* control actions. The presence of pursuit information does not guarantee pursuit control will be used, and the absence of pursuit information does not guarantee pursuit control will not be used.

The *neuromuscular* filter accounts for the lag imposed by limb dynamics and neuromuscular delays. The human senses the filtered input using the *proprioceptive* pathway, and compares it to the desired input.

Once the human's control input is determined, a *disturbance input* is added. This can be used for a disturbance rejection task (Van Paassen and Mulder, 2006), or to add *remnant* to the controller model. Remnant accounts for the human's control input that is not predicted by the model. Most of the remnant appears to come from fluctuations in the effective time delay (McRuer, 1980), nonsteady control behavior, and nonlinear anticipation or relay-like operations (McRuer et al., 1967). These effects are larger when tracking conditions are difficult (Hess, 1979). The remnant has been found to have fairly constant power with no major peaks, and it tends to be relatively small when tracking conditions are favorable (Wade and Jex, 1972).

Perhaps the most influential model has been the *Crossover model* proposed by McRuer and Jex (1967). The Crossover model states that in a compensatory task (when only $e(t)$ is displayed) for a variety of controlled-element dynamics, the operator acts to make the overall human-machine system assume the form:

$$\frac{\mathbb{Y}(j\omega)}{\mathbb{E}(j\omega)} = \frac{Ke^{-j\omega\tau}}{j\omega} \quad \text{near } \omega = K, \tag{2}$$

where K is the open-loop system gain, and τ is the effective time delay. Note that the transfer function is written with the frequency operator $j\omega$ instead of the Laplace variable s. This is to emphasize that the model is only intended to apply in the frequency domain, and may not be accurate for non-sinusoidal inputs such as steps or ramps. Furthermore, the Crossover model is only meant to characterize the system near the crossover frequency – hence its name. The control system's closed-loop response is generally dominated by its behavior near the crossover frequency (McRuer and Jex, 1967).

Note that control input does not appear explicitly in the Crossover model. It is an implicit model that depends on the controlled-element dynamics. Therefore it is not included in the ManualTracking library, which requires explicit models to generate the control input. However, many of these explicit models were originally formed using the Crossover model as a basis.

3 Modelica Library

The previous section introduced the elements of typical manual tracking tasks, and this section presents their implementation in the ManualTracking Modelica library. An overview of the library structure is shown in Figure 6. Packages in the library will be described in order from least to most complex, ending with the TrackingTasks package. An instance of a tracking task requires instances from the ManualControllers, ControlledElements, and ForcingFunctions packages. The Blocks and Icons packages are straightforward, and will not be discussed.

ForcingFunctions

This package only includes the summed sine wave signal, which is by far the most common signal used in manual tracking tasks. Frequency values can be either in units of Hz (with SumOfSinesHz), or radians per second (with SumOfSinesRadPerSec). If the user wishes to make a custom forcing function for either the reference signal or disturbance input, the signal must be contained in one block with a single output, and it must be stored in the appropriate package. If these rules are violated, the Python functions will not be able to parse the text of the tracking task.

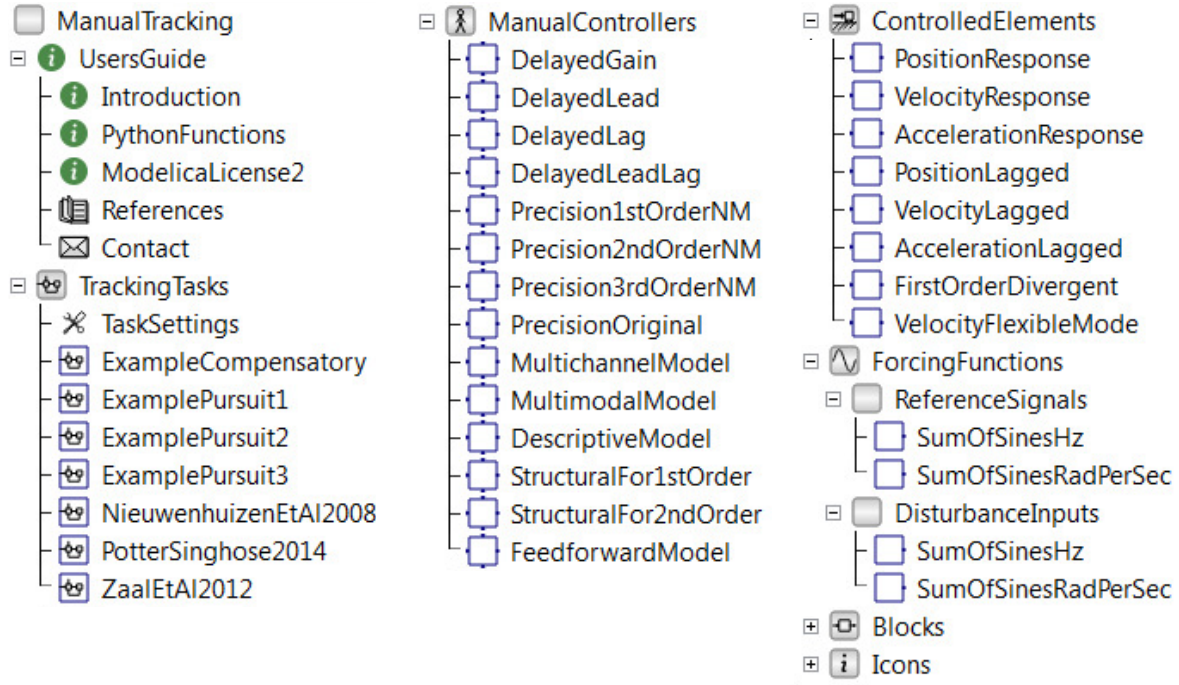


Figure 6. ManualTracking library overview.

ControlledElements

All controlled elements included in the library are shown in Table 1. There are the basic position, velocity, and acceleration responses. There are also versions of these basic responses with an added first-order lag, making them less responsive at first, but eventually reaching the same steady-state position/velocity/acceleration.

The unstable FirstOrderDivergent controlled element was used to investigate limitations of a human's effective time delay in Jex et al. (1966). The VelocityFlexibleMode includes a second order mode that can be oscillatory. This controlled element was studied with relatively high damping in Shirley and Young (1968), and very low damping in Potter and Singhose (2014).

Table 1. Controlled elements.

Class Name	*Transfer Function*
PositionResponse	K
VelocityResponse	K/s
AccelerationResponse	K/s^2
PositionLagged	$K/(Ts+1)$
VelocityLagged	$K/(s[Ts+1])$
AccelerationLagged	$K/(s^2[Ts+1])$
FirstOrderDivergent	$K/(Ts-1)$
VelocityFlexibleMode	$K\omega^2/(s[s^2+2\zeta\omega s+\omega^2])$

ManualControllers

Table 2 shows all manual controller models in equation form. The ManualTracking library documentation contains block diagrams of each controller model, and these block diagrams are sometimes more intuitively useful than the equations.

McRuer and Jex (1967) describe how the first four manual controllers are combined with specific controlled elements to yield the form of the Crossover model. The model PrecisionOriginal was proposed by McRuer and his colleagues, and various simplified versions with one of the lead-lag terms removed have been used since then. These versions mainly differ in how they represent the human's neuromuscular filter. The MultichannelModel, MultimodalModel, and DescriptiveModel each include pursuit control, but they have different ways of organizing the human's control pathways.

The following two models, StructuralFor1stOrder and StructuralFor2ndOrder, include proprioceptive feedback, and are designed to control first-order and second-order controlled elements, respectively. The FeedforwardModel includes feedfoward control. For this control pathway, an inverted model of the plant dynamics is needed, and the controller automatically uses the block ManualTracking.Blocks.FeedForward for this purpose. To change the inverse dynamics block, or to make the controller use a different block, the Modelica file text must be modified manually.

Once the controller is selected, the next task is to choose values for controller parameters. Tools described in Section 4 can help select these values – they are selected either to yield optimal performance, or the closest fit to experimental control behavior.

DOI 10.3384/ecp15118329

Table 2. Manual controller models.

Class Name	*Control Input,* $\mathbb{V}(s)$ *as a function of Laplace-transformed* $r(t)$, $y(t)$, *and* $e(t)$, *defined in Figure 5*
DelayedGain	$\mathbb{E}(K)e^{-\tau s}$
DelayedLead	$\mathbb{E}\left(K(Ts+1)\right)e^{-\tau s}$
DelayedLag	$\mathbb{E}\left(K\frac{1}{T_2 s+1}\right)e^{-\tau s}$
DelayedLeadLag	$\mathbb{E}\left(K\frac{T_1 s+1}{T_2 s+1}\right)e^{-\tau s}$
Precision1stOrderNM	$\mathbb{E}\left(K\frac{T_1 s+1}{T_2 s+1}\right)\left(\frac{1}{T_3 s+1}\right)e^{-\tau s}$
Precision2ndOrderNM	$\mathbb{E}\left(K\frac{T_1 s+1}{T_2 s+1}\right)\left(\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}\right)e^{-\tau s}$
Precision3rdOrderNM	$\mathbb{E}\left(K\frac{T_1 s+1}{T_2 s+1}\right)\left(\frac{\omega^2}{(T_3 s+1)(s^2+2\zeta\omega s+\omega^2)}\right)e^{-\tau s}$
PrecisionOriginal (McRuer et al., 1965)	$\mathbb{E}\left(K\frac{T_1 s+1}{T_2 s+1}\right)\left(\frac{T_3 s+1}{T_4 s+1}\right)\left(\frac{\omega^2}{(T_5 s+1)(s^2+2\zeta\omega s+\omega^2)}\right)e^{-\tau s}$
MultichannelModel (Nieuwenhuizen et al., 2008)	$\left[\mathbb{E}\left(K_1(T_1 s+1)\right)e^{-\tau_1 s}+\mathbb{Y}\left(K_2\frac{s^2(T_2 s+1)}{T_3 s+1}\right)e^{-\tau_2 s}\right]\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}$
MultimodalModel (Zaal et al., 2012)	$\left[\mathbb{E}\left(K_1\frac{(T_1 s+1)^2}{T_2 s+1}\right)e^{-\tau_1 s}+\mathbb{Y}\left(K_2 s\right)e^{-\tau_2 s}\right]\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}$
DescriptiveModel (Hosman and Stassen, 1999)	$\left[\mathbb{E}\left(K_1 e^{-\tau_1 s}+K_2 s e^{-\tau_2 s}\right)+\mathbb{Y}\left(K_3 s e^{-\tau_3 s}+K_4\frac{s^2(T_1 s+1)}{(T_2 s+1)(T_3 s+1)}\right)\right]e^{-\tau_4 s}$
StructuralFor1stOrder (Hess, 1980)	$\mathbb{E}\left(K_1+K_2 s e^{-\tau_1 s}\right)\left(\frac{\omega^2(Ts+1)}{\omega^2 K_3 s+(s^2+2\zeta\omega s+\omega^2)(Ts+1)}\right)e^{-\tau_2 s}$
StructuralFor2ndOrder (Hess, 1980)	$\mathbb{E}\left(K_1+K_2 s e^{-\tau_1 s}\right)\left(\frac{\omega^2(T_1 s+1)(T_2 s+1)}{\omega^2 K_3 s+(s^2+2\zeta\omega s+\omega^2)(Ts+1)(T_2 s+1)}\right)e^{-\tau_2 s}$
FeedforwardModel (Drop et al., 2013)	$\left[\mathbb{E}(K_1)e^{-\tau_1 s}+\mathbb{R}\left(K_2\frac{1}{Ts+1}\right)[\mathsf{FeedForward}]\,e^{-\tau_2 s}\right]\frac{\omega^2}{s^2+2\zeta\omega s+\omega^2}$

TrackingTasks

Blocks from the ForcingFunctions, ControlledElements, and ManualControllers packages can be useful by themselves, in any configuration that supports the user's model. However, to use the parameter tuning and user experiment features of this library, a specific configuration of the components is required.

A tracking task model should have the standard form shown in Figure 7, and it should be stored inside the TrackingTasks package. It includes one block from the ForcingFunctions.ReferenceSignals, ForcingFunctions.DisturbanceInputs, and ControlledElements packages, each connected to the appropriate port of a ManualController block. Several manual tracking tasks from the literature are provided.

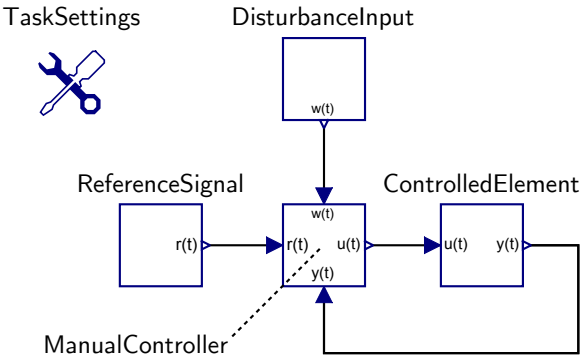


Figure 7. Example of tracking task.

An additional component, the TaskSettings block, must be included. This block contains important details of the tracking task: **taskDuration** is the total length of the task; **previewTime** is the amount of time in advance to show the target motion; and **backgroundVisible** determines whether or not pursuit (background) information is shown with hatch marks. The last three parameters are only used in the user experiment, and not in the parameter tuning or simulation functions.

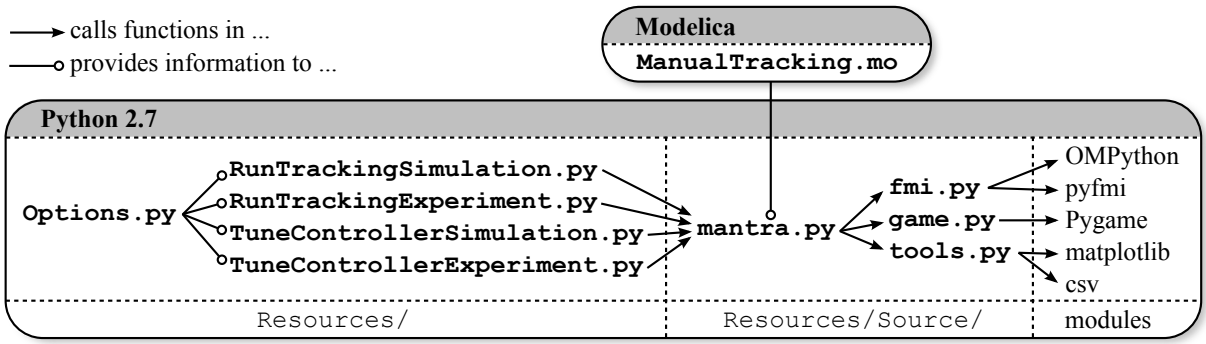


Figure 8. Software overview.

4 Python Functions

The previous sections have described purely Modelica-based components that can be run from within a Modelica simulation environment. Two additional capabilities are provided in the ManualTracking library: automatically tuning manual controller parameters, and performing real-time tracking experiments. These features are implemented in the Python programming language.

An overview of the software is shown in Figure 8. In the `Resources/` directory, there are 5 .py files. Four of these are function scripts that can be run from a terminal or Python IDE, and the fifth file is `Options.py`, where options are set by the user. The functions do not use input arguments, and instead get them from `Options.py`. Variables in this file include: **taskModel**, the tracking task model to run; **saveFormat**, the format with which to save backup data files; **printVerbose**, whether or not to print all runtime messages to the console. The user may also experiment with different framerates and optimization/simulation methods.

`Options.py` also contains Boolean input arguments for each of the four functions: **useSaved** makes the functions use most recent saved data (in `Resources/Temp/` directory) instead of re-running an experiment; **plotResults** generates a figure with the resulting trajectories; and **saveResults** saves a backup results file in the `Resources/Data/` directory. The results file contains values for the reference state, measured state, disturbance input, and control input at each sample time.

Each of the four main functions call `mantra.py`, which reads the `ManualTracking.mo` file for details of the tracking task, and then calls functions defined in `fmi.py`, `game.py`, and `tools.py`. These functions use standard Python modules, as well as OMPython, pyfmi, Pygame, and matplotlib. Note that OMPython requires an installation of OpenModelica (Fritzson et al., 2005). Additionally, pyfmi has many dependencies that must be installed first, and it may be more convenient to install JModelica.org (Åkesson et al., 2010) instead of installing them individually. After all required Python modules have been installed, the following functions should run successfully.

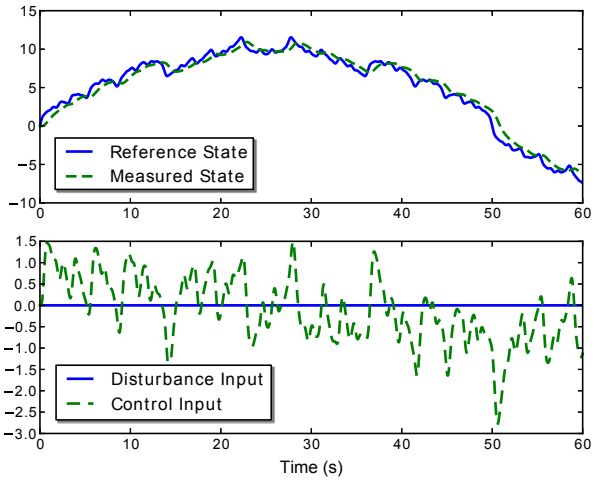


Figure 9. Plot of simulation results.

4.1 `RunTrackingSimulation`

This function simulates the tracking task model specified in `Options.py`. The simulation stop time is specified by **taskDuration** in the TaskSettings block. All files are saved into the `Resources/Temp/` directory. Generated files include log files, FMU build files, and a comma-separated value (CSV) file of simulation results. If **saveResults** is true, this CSV file is also saved in the `Resources/Data/` directory. If **plotResults** is true, then time-curves of $r(t)$, $y(t)$, $w(t)$, and $u(t)$ are plotted, as shown in Figure 9. The Python module *matplotlib* is required for this feature.

4.2 `RunTrackingExperiment`

This function allows the user to perform a tracking task in real time. The tracking task model specified in `Options.py` is parsed, and details of the experiment are extracted from the TaskSettings block. Next, the reference signal and disturbance input are generated by building them into separate FMUs, and simulating them for the duration of the experiment. These signals are not affected by the user input or controlled element state, and therefore they can be simulated open loop.

DOI 10.3384/ecp15118329

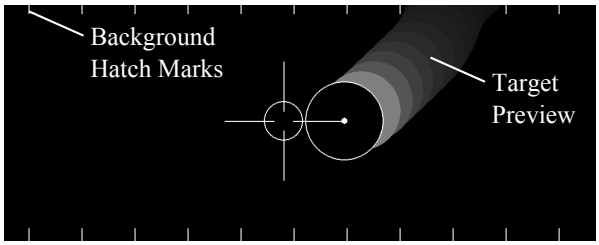


Figure 10. Display of manual tracking experiment.

Next, Pygame looks for any joysticks connected to the computer. If no joystick is found, then the keyboard arrow keys may be used for control input. When a joystick is used, the experiment runs more smoothly and the parameter-fitting functions work much more effectively. Therefore, using a joystick for the experiment is highly recommended.

Then, two scaling factors are automatically calculated. One is the *display gain*, which determines how far to move display objects (in pixels) based on the reference and measured state magnitudes, which are in unknown units of length. The other is the *input gain*, which determines the magnitude of input to the controlled element based on the joystick or keyboard input between -1 (full left) and 1 (full right).

Finally, the user is prompted to start the experiment. The tracking display is shown in Figure 10. Lines on the top and bottom of the screen mark the global coordinates, so that the target motion can be seen independently of the cursor motion. These lines can be hidden to create a compensatory task by setting **backgroundVisible** to false in the TaskSettings block.

Figure 10 also shows a preview of the target motion. Future motion is indicated by circles falling from the top of the screen. The topmost circle shows where the target will be **previewTime** seconds in the future. This feature can be disabled by setting **PreviewTime** to 0.

If desired, the user may adjust fundamental settings of the game in the `Resources/Source/game.py` file. Modifying these settings may cause errors, so it is a good idea to save a backup of the `game.py` file before making modifications.

4.3 `TuneControllerSimulation`

This function repeatedly simulates the tracking task with different parameter values in the manual controller, and finds values which yield the best tracking performance. This reflects an important finding in the literature: an experienced human operator has inherent human limitations[1] but behaves in a nearly optimal fashion given these limitations.

Mathematically, the function tries to minimize the integrated squared difference between $y(t)$ and $r(t)$. This is shown conceptually in Figure 11(a), where $c(t)$ is the continuous cost to minimize. Because tracking performance is not a differentiable function of the controller parameters, a derivative-free optimization method such as the Nelder-Mead simplex method (Gedda et al., 2012) must be used.

Some manual controller models contain many parameters, and attempting to tune all of them at once would be time-consuming and would likely yield poor results. Therefore, the user is allowed to select a subset of the parameters using a console prompt like this:

```
Tunable controller parameters:
  1. K -- Proportional gain
  2. T2 -- Time constant of phase
     lag compensation (s)
  3. T1 -- Time constant of phase
     lead compensation (s)

Please enter a comma-separated number
list specifying parameters to tune: _
```

To tune `K` and `T1`, for example, the user should type `1,3` and press `Enter`. The rest of the parameters are fixed so that they remain the same as in the manual controller component definition, unless they are re-assigned in the tracking task model.

4.4 `TuneControllerExperiment`

This function tunes the automatic manual controller to behave as much as possible like the human controller. The concept is shown in Figure 11(b). The goal is to minimize the difference between the experimentally recorded control input, $u(t)$, and the simulated control input, $\hat{u}(t)$.

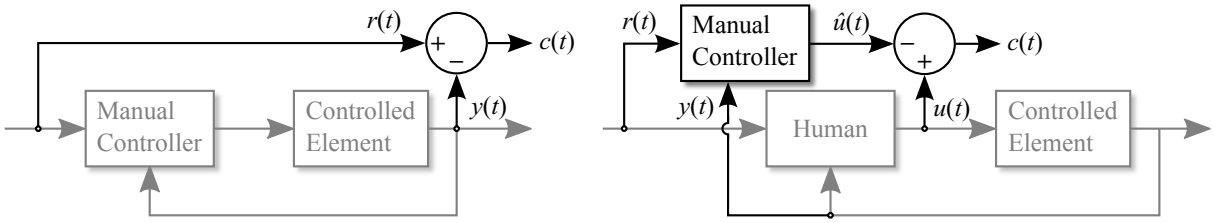
The input to the manual controller is the reference signal (and disturbance input, not shown in Figure 11), and the experimentally recorded controlled element state. Note that the tracking performance of the tuned controller might be very poor, because it does not attempt to optimize tracking performance. It simply tries to match control behavior of the user.

4.5 Notes

- When selecting a controller model, one should consider details of the real-world task and controlled-element dynamics. The Python functions do not check the appropriateness of a manual controller for the given tracking conditions.

- User-created ReferenceSignal, DisturbanceInput, ControlledElement, and ManualController classes must be stored inside the appropriate packages. The Python functions only search in these locations when parsing the tracking task.

[1] For example, reaction time delay, neuromuscular lag, and ability to generate derivatives and higher-order leads.



a) Optimize tracking performance

b) Match experimental control behavior

Figure 11. Tuning manual controller parameters by minimizing $\int [c(t)]^2 dt$.

- While the keyboard can be used for control input in the tracking experiment, a joystick is highly recommended. The parameter tuning functions work much better, and the display is smoother.
- Automatic syncing programs (Dropbox in particular, but possibly others) seem to cause a problem with the real-time user experiment. Try exiting, or at least pausing, these programs if the experiment crashes repeatedly.

5 Example

Basic use of the Modelica library and Python functions will now be demonstrated. Load the ManualTracking library, go to the TrackingTasks package, and simulate one of the example tasks. Plots of controlledelement1.y and referencesignal1.y should resemble the top plot of Figure 9, with the controlled element following the reference signal closely, but with a small time delay.

Next, navigate to `ManualTracking/Resources/`, and open `Options.py`. The **taskModel** variable should be assigned to one of the example tracking tasks, and **printResults** should be set to true. Then run the script `RunTrackingSimulation.py`, either from a Python IDE or from a console window. A few diagnostic messages should print to the console, and then a figure similar to Figure 9 should appear. To reduce the amount of console output, set **printVerbose** to false.

If the function executed successfully, then try the real-time experiment. Run `RunTrackingExperiment.py` and wait for the reference signal and disturbance inputs to be generated. After a short time, the console should show this prompt:

```
Press 'Enter' to bring up the display,
then press any key except 'q' to start
the experiment: _
```

After following these instructions, a window similar to Figure 10 appears. Use either the arrow keys or a joystick to make the crosshairs follow the target. Once the experiment is complete, a plot of the state and input trajectories is shown if **printResults** is set to true. To examine the data file used for this plot, go to the `Resources/Temp/` directory, and look for the comma-separated-value (CSV) file `ExampleTaskName_exp.csv`. The data file for the tracking simulation should also be in the same directory, saved as `ExampleTaskName_sim.csv`.

Next, try the manual controller tuning functions. Run the `TuneControllerSimulation.py` script. When prompted, type `1,3` and press enter. The optimization function prints information about the current parameter guesses and cost function value to the console. Within a few minutes, the solver should converge, and parameter values for the parameters should be displayed. A plot shows the simulated tracking performance using these parameter values.

Instead of tuning parameters to yield the best tracking performance, they can be tuned to fit experimental tracking performance. First, make sure **useSaved** is set to true in `Options.py`, otherwise the user experiment will be run again. Then run the `TuneControllerExperiment.py` script. Just like in the previous example, select the parameters you would like to tune, and the function should find their optimal values and display the simulated tracking results with the chosen controller values.

6 Conclusions

This paper presents a Modelica library and supporting functions for studying human control behavior in continuous tracking tasks. These tools can increase Modelica's usefulness for modeling human-machine systems. For now, further development, debugging, and testing on different platforms is needed. The library is open source and available for download at http://jjpotterkowski.github.io/.

Acknowledgements

This work was fully supported by an ERCIM "Alain Bensoussan" post-doctoral research fellowship, hosted by VTT Technical Research Centre of Finland. The author wishes to thank A. Ashgar, A. Pop, and M. Sjölund for technical advice related to OMPython and FMUs.

References

G. A. Bekey, G. O. Burnham, and J. Seo. Control theoretic models of human drivers in car following. *Human Factors*, 19(4):399–413, Aug. 1977.

H. J. Damveld, G. C. Beerens, M. M. van Paassen, and M. Mulder. Design of forcing functions for the identification of human control behavior. *AIAA Journal of Guidance, Control, and Dynamics*, 33(4):1064–1081, Jul.-Aug. 2010.

F. M. Drop, D. M. Pool, H. J. Damveld, M. M. van Paassen, and M. Mulder. Identification of the feedforward component in manual control with predictable target signals. *IEEE Transactions on Cybernetics*, 43(6):1936–1949, Dec. 2013.

P. Fritzson, P. Aronsson, H. Lundvall, K. Nyström, A. Pop, L. Saldamli, and D. Broman. The OpenModelica Modeling, Simulation, and Software Development Environment. *Simulation News Europe*, 44(45), Dec. 2005.

S. Gedda, C. Andersson, J. Åkesson, and S. Diehl. Derivative-free parameter optimization of functional mock-up units. In *Proc. 9th Int. Modelica Conf.*, Munich, Germany, Sep. 2012.

R. A. Hess. A rationale for human operator pulsive control behavior. *Journal of Guidance and Control*, 2(3):221–227, May-Jun. 1979.

R. A. Hess. A structural model of the adaptive human pilot. *AIAA Journal of Guidance, Control, and Dynamics*, 3(5): 416–423, Sep.-Oct. 1980.

R. A. Hess. *Feedback Control Models – Manual Control and Tracking*, chapter 38, pages 1249–1294. John Wiley & Sons, Inc., Hoboken, NJ, 3 edition, 2006.

R. A. Hess and A. Modjtahedzadeh. A control theoretic model of driver steering behavior. *IEEE Control Systems Magazine*, 10(5):3–8, Aug. 1990. doi:10.1109/37.60415.

R. Hosman and H. Stassen. Pilot's perception in the control of aircraft motions. *Control Engineering Practice*, 7:1421–1428, 1999.

R. J. Jagacinski. A qualitative look at feedback control theory as a style of describing behavior. *Human Factors*, 19:331–347, Aug. 1977.

R. J. Jagacinski and J. M. Flach. *Control Theory for Humans: Quantitative Approaches to Modeling Performance*. CRC Press, New York, NY, 2003.

H. R. Jex, J. D. McDonnell, and A. V. Phatak. A "critical" tracking task for manual control research. *IEEE Transactions on Human Factors in Electronics*, HFE-7(4):138–145, Dec. 1966. doi:10.1109/THFE.1966.232660.

J. Åkesson, K.-E. Årzén, M. Gäfvert, T. Bergdahl, and H. Tummescheit. Modeling and optimization with optimica and jmodelica.org—languages and tools for solving large-scale dynamic optimization problems. *Computers and Chemical Engineering*, 34(11):1737–1749, Nov. 2010.

D. L. Kleinman and T. R. Perkins. Modeling human performance in a time-varying anti-aircraft tracking loop. *IEEE Transactions on Automatic Control*, AC-19(4):297–306, Aug. 1974.

D. T. McRuer. Human dynamics in man-machine systems. *Automatica*, 16(3):237–253, May 1980.

D. T. McRuer and H. R. Jex. A review of quasi-linear pilot models. *IEEE Transactions on Human Factors in Electronics*, HFE-8(3):231–249, Sep. 1967. doi:10.1109/THFE.1967.234304.

D. T. McRuer, D. Graham, E. S. Krendel, and W. Reisner. Human pilot dynamics in compensatory systems. Technical Report AFFDL-TR-65-15, Air Force Flight Dynamics Laboratory, Wright-Patterson AFB, OH, 1965.

D. T. McRuer, D. Graham, and E. S. Krendel. Manual control of single-loop systems: Part I. *Journal of the Franklin Institute*, 283(1):1–29, Jan. 1967.

D. T. McRuer, D. H. Weir, H. R. Jex, R. E. Magdaleno, and R. W. Allen. Measurement of driver-vehicle multiloop response properties with a single disturbance input. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-5(5): 490–497, 1975. doi:10.1109/TSMC.1975.5408371.

F. M. Nieuwenhuizen, P. M. T. Zaal, M. Mulder, M. M. van Paassen, and J. A. Mulder. Modeling human multichannel perception and control using linear time-invariant models. *Journal of Guidance, Control, and Dynamics*, 31(4):999–1013, Jul.-Aug. 2008.

J. J. Potter and W. E. Singhose. Effects of input shaping on manual control of flexible and time-delayed systems. *Human Factors*, 56(7):1284–1295, Nov. 2014.

R.S. Shirley and L.R. Young. Motion cues in man-vehicle control: effects of roll-motion cues on human operator's behavior in compensatory systems with disturbance inputs. *IEEE Transactions on Man-Machine Systems*, 9(4):121–128, Dec. 1968.

E. P. Todosiev, R. E. Rose, and L. G. Summers. Human performance in single and two-axis tracking systems. *IEEE Transactions on Human Factors in Electronics*, HFE-8(2): 125–129, Jun. 1967.

A. Tustin. The nature of the operator's response in manual control, and its implications for controller design. *Journal of the Institution of Electrical Engineers*, 94(2):190–206, May 1947.

M. M. Van Paassen and M. Mulder. *International Encyclopedia of Ergonomics and Human Factors*, volume 1, chapter Identification of Human Control Behavior, pages 400–407. Taylor and Francis, London, 2 edition, 2006.

A. R. Wade and H. R. Jex. A simple Fourier analysis technique for measuring the dynamic response of manual control systems. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-2(5):638–643, Nov. 1972. doi:10.1109/TSMC.1972.4309192.

P. M. T. Zaal, D. M. Pool, M. M. van Paassen, and M. Mulder. Comparing multimodal pilot pitch control behavior between simulated and real flight. *Journal of Guidance, Control, and Dynamics*, 35(5):1456–1471, Sep.-Oct. 2012.