ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE069

GENERAL FUNCTIONALITY FOR TURN-DEPENDENT ELEMENT PROPERTIES IN SIXTRACK*

K. Sjobak†, H. Burkhardt, R. De Maria, A. Mereghetti, A. Santamaría García
CERN, Geneva, Switzerland

Abstract

In order to facilitate studies of how dynamically changing element attributes affect the dynamics of the beam and beam losses, the functionality for dynamic kicks (DYNK) of SixTrack has been significantly extended. This functionality can be used for the simulation of dynamic scenarios, such as when crab cavities are switched on, orbit bumps are applied, optics are changed, or failures occur. The functionality has been extended with a more general and flexible implementation, such that arbitrary time-dependent functions can be defined and applied to different attributes of families or individual elements, directly from the user input files. This removes the need for source code manipulation, and it is compatible with LHC@Home which offers substantial computing resources from volunteers. In this paper, the functionality and implementation of DYNK is discussed, along with examples of application to the HL-LHC crab cavities.

INTRODUCTION

SixTrack is a 6D single particle tracking code [1,2], which is routinely used at CERN to study the dynamic aperture and collimation system in high-energy circular machines like the LHC. There are also a large number of tools built around SixTrack, both for analyzing the results and for handling very large numbers of initial conditions. For this reason, SixTrack is the natural tool to use for studying fast failure scenarios at the HL-LHC, and for other transient phenomena at other similar machines.

Functionality for applying dynamic kicks (DYNK) – i.e. time-dependent machine element parameters – was therefore added to the code [3,4] and significantly extended. It makes it possible for the user to specify the functional form of these parameters directly in the input file, or to load them from a file. This eliminated the need for multiple private code forks, freeing up and focusing developer resources. The specification of the functions, to which elements they should be applied, and when they should be applied, is done using a simple mini-language. Some examples of this language are given below, and a full description is given on the TWIKI page [5].

DYNK currently supports setting the strength of all the standard thin elements, and also setting the voltage, phase, and frequency of crab cavities. It also supports a wide variety of functions, which may use the turn number or the output of other functions as input. It is also possible for the functions to store and retrieve data from memory, as is used for pseudo random number generating functions that stores the random seed between turns, and when loading the functions as tables from files. There is also an option to output the setting of the affected elements at every turn to a file. In order to work with LHC@Home [6], which is used for large tracking campaigns, DYNK supports checkpointing and restarting from a checkpoint. It interacts correctly with the collimation routines, including resetting the elements and generating exactly the same values for each pass of 64 particles. The ripple module is made redundant by DYNK, which can exactly reproduce its results, and is therefore removed.

IMPLEMENTATION

In order to make DYNK work, there are two main components: a function parser and evaluator, and a setter and getter for the element properties. Additionally, there are hooks for calling DYNK in the tracking loops, and for initialization before the start of tracking. The data storage for the functions are provided by one master table (one row per function) and a "free memory" for each of the major data types (integers, floats and strings).

This architecture is very easily extendable, making adding support for new types of functions a matter of adding a few lines to the parser and evaluator. Similarly, new elements or element attributes can be supported by adding them to the setter and getter functions – the difficulty of this is determined by the complexity of the memory structures and initialization scheme used for that element.

EXAMPLE USE CASES

Some examples for the use of DYNK are provided below. All of these are ran using the HL-LHC v1.1 lattice [7] with vertical crab cavities around the first interaction point (IP1, ATLAS). The beam was sampled at IP1 as a Gaussian distribution using the nominal HL-LHC parameters, cropped so to only include particles inside of the RF bucket.

The crab cavities closing the bump (at s=153.6, 154.6, 160.2 and 161.2 m, relative to IP1) are in this simulation called CRAB_IP1_R1$\cdots$4, while the cavities creating the bump (at s=26494.3, 26495.3, 26500.9 and 26501.9 m) are called CRAB_IP1_L1$\cdots$4. Their frequencies are set to 400.8 MHz, i.e. the same as for the accelerating cavities. The standard voltages of the closing- and opening cavities opening cavities are calculated using Eq. 4 from [8]. For the opening cavities, it is assumed that the transverse betatron β-function and phase advance is the same for all the cavities, while the voltage of each of the closing cavities are chosen such that they cancel the symmetrically positioned opening

* Research supported by EU FP7 HiLumi LHC – Grant Agreement 284404
† kyrre.ness.sjoebaek@cern.ch

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE069

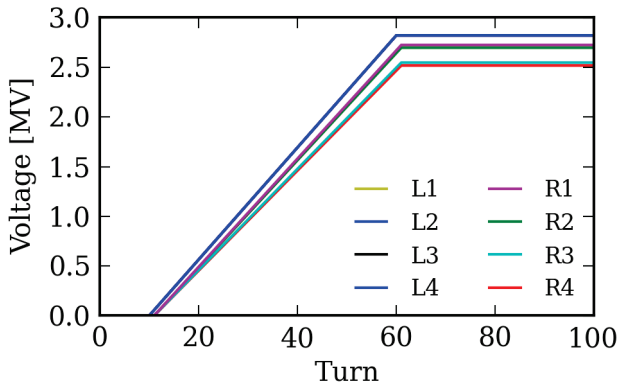

Figure 1: Ramping of crab cavity voltages.

cavity. For these examples, the crab cavities around IP5 (CMS) are switched off.

Starting a Tracking Run with Crabs Switched On

If the tracking is started with a uncrabbed beam inside a bump, such as produced by a crab cavity, the bump must be temporarily disabled during injection. To achieve this, the following DYNK block can be used:

```
DYNK
FUN zero CONST 0.0
FUN CV_1R1 GET CRAB_IP1_R1 voltage
FUN CV_1R2 GET CRAB_IP1_R2 voltage
FUN CV_1R3 GET CRAB_IP1_R3 voltage
FUN CV_1R4 GET CRAB_IP1_R4 voltage
SET CRAB_IP1_R1 voltage zero 1 1 0
SET CRAB_IP1_R2 voltage zero 1 1 0
SET CRAB_IP1_R3 voltage zero 1 1 0
SET CRAB_IP1_R4 voltage zero 1 1 0
SET CRAB_IP1_R1 voltage CV_1R1 2 2 0
SET CRAB_IP1_R2 voltage CV_1R2 2 2 0
SET CRAB_IP1_R3 voltage CV_1R3 2 2 0
SET CRAB_IP1_R4 voltage CV_1R4 2 2 0
NEXT
```

This creates a constant function `zero` which always returns 0.0, and four constant functions `CV_1R1`$\cdots$`4` which always return the original voltage of the crab cavity (as defined in the lattice description file `fort.2`). The voltage of these cavities are then set equal to zero at turn 1 (ending at 1), and then changed to the original value for turn 2 (ending at 2). Since no further settings are applied, the current settings remain after turn 2.

Linear Ramping of Crab Cavities

Another case is a simulation where the voltage of the crab cavities are linearly ramped up to the maximum voltage in 50 turns, starting at turn 10. The voltage is then flat for 40 turns. This profile is illustrated in Fig. 1, and the input block looks like (shortened by removing duplicate instructions to control multiple elements):

```
DYNK
FUN zero CONST 0.0
FUN CV_R1 GET CRAB_IP1_R1 voltage
(etc. for R2,R3,R4)
```

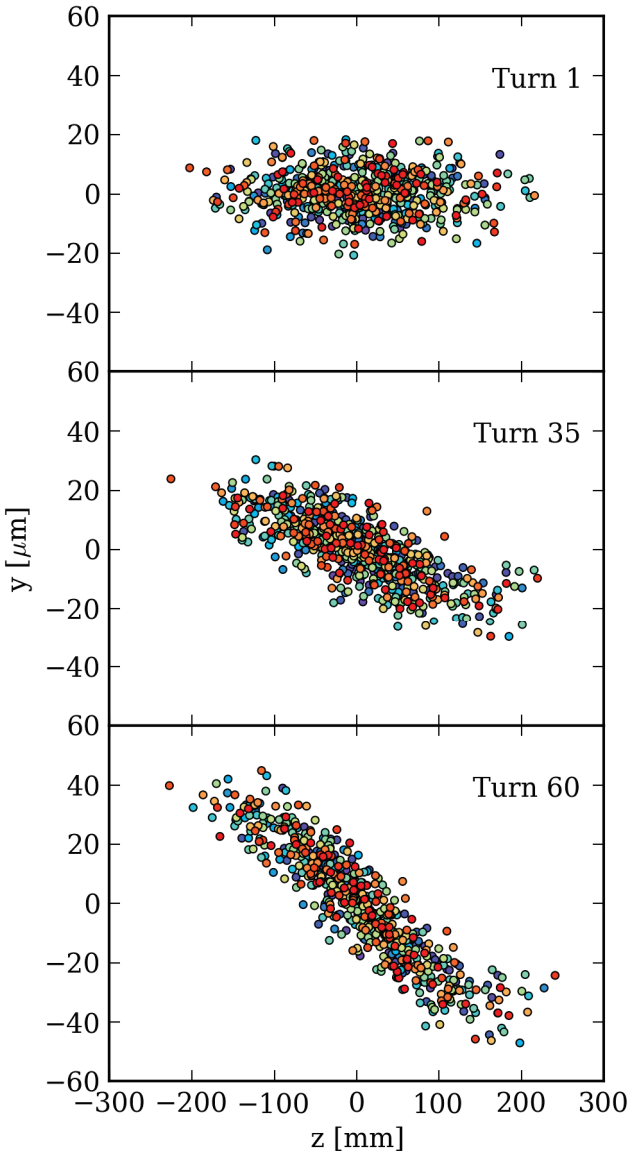

Figure 2: Simulated Z-Y distribution (640 particles) of the beam during ramping of the crab cavity voltages.

```
FUN CV_L  GET CRAB_IP1_L1 voltage
FUN ramp LIN 0.02 0
FUN ramp_R1 MUL CV_R1 ramp
(etc. for R2,R3,R4)
FUN ramp_L  MUL CV_L  ramp
SET CRAB_IP1_R1 voltage zero 1 10 0
(etc. for R2,R3,R4)
SET CRAB_IP1_L1 voltage zero 1 9 0
(etc. for L2,L3,L4)
SET CRAB_IP1_R1 voltage ramp_R1 11 61 -11
(etc. for R2,R3,R4)
SET CRAB_IP1_L1 voltage ramp_L 10 60 -10
(etc. for L2,L3,L4)
NEXT
```

Here the function `ramp` is given as $y(x) = 0.02x + 0.0$, and this normalized ramp is then multiplied by the wanted final voltages. When setting the voltages during the ramp, the point where the functions are evaluated is then shifted

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE069

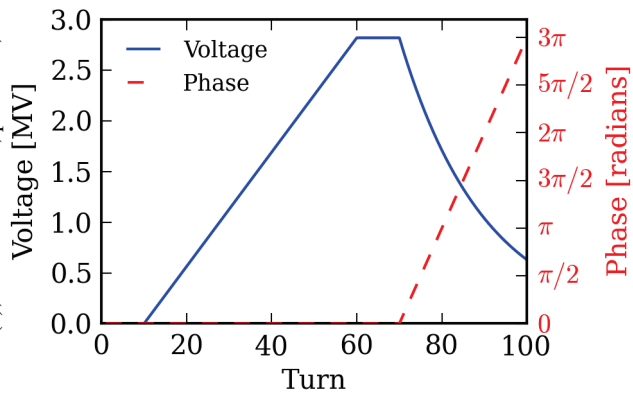

Figure 3: Voltage and phase of crab cavities CRAB_IP1_L1···L4 as function of time, for the exponential decay/linear drift example.

by -11 and -10 turns, as specified in the last column in the SET statement. The resulting beam distributions are shown in Fig. 2, which show the bunch is increasingly tilted as the voltage ramps up.

Exponential Decay of Crab Voltage, Combined with a Linear Drift of Crab Phase

A slightly more complicated case is if the crab cavities on the side upstream of the IP sees an exponential decay in the kick voltage, combined with a linear drift in phase. This failure case corresponds to the cavities being detuned, while also no longer being driven, similarly to what is described in [9]. In this example, the failure starts after 70 turns, following a ramping like in the above example. The extra lines in the DYNK block then looks like:

```
FUN expCore LIN -0.05 0.0
FUN decay EXP expCore
FUN decayScaled MUL decay CV_L
SET CRAB_IP1_L1 voltage decayScaled 70 100 -70
(etc. for L2,L3,L4)
FUN phasedrift LIN 0.3141592654 0.0
SET CRAB_IP1_L1 phase phasedrift 70 100 -70
(etc. for L2,L3,L4)
```

The exponential decay is here given as $V(t') = V_0 \exp(-t'/20)$, where V_0 is the standard voltage and t' is the number of turns after the start of the failure, i.e. $t' = \text{turn} - 70$. The phase drift is given as $\phi(t') = \pi t'/10$. The resulting voltage and phase program is shown in Fig. 3.

REPLACEMENT OF RIPP BLOCK

As the functionality of the ripple module in SixTrack is a subset of what is possible with DYNK, it has been deprecated. The RIPP input block is therefore no longer accepted. A special function COSF_RIPP has therefore been provided in DYNK, exactly mirroring the old RIPP input format. This function is calculated as

$$f(t; A, T, \phi_0) = A \cos\left(2\pi \frac{(t-1)}{T} + \phi_0\right), \qquad (1)$$

where t is the current turn number, A is the specified amplitude, T the period, and ϕ_0 the initial phase.

This was tested against the prob1 and prob3 example cases from sixtest [10], as well as against the results from a recent BOINC campaign. It was confirmed that replacing the RIPP block with a DYNK block exactly reproduced the tracking results, even after 10^6 turns.

As an example, a part of the RIPP block from prob1 is shown below. It specifies a ripple of amplitude $A = \pm 3.2315 \cdot 10^{-10}$ radians/meter and period $T = 244.9$ turns (equivalent to 50 Hz) for the quadrupole elements dmqx1f5015+2 and dmqx2af5015+2, with no start phase or continuation turn number specified:

```
RIPPLE OF POWER SUPPLIES
        dmqx1f5015+2      3.2315D-10    224.9
        dmqx2af5015+2    -3.2315D-10    224.9
(etc. for more elements)
```

An equivalent DYNK block is:

```
DYNK (autogenerated from RIPP block by rippconverter.py)
NOFILE
FUN RIPP-dmqx1f5015+2  COSF_RIPP  3.2315D-10 224.9 0.0
SET dmqx1f5015+2 average_ms  RIPP-dmqx1f5015+2 1  -1 0
FUN RIPP-dmqx2af5015+2 COSF_RIPP -3.2315D-10 224.9 0.0
SET dmqx2af5015+2 average_ms RIPP-dmqx2af5015+2 1 -1 0
(etc. for more elements)
```

Here the final turn number for the SET statement is given as -1, indicating it should be active until the end of the simulation. Also note the NOFILE statement, instructing DYNK to not write a file containing the settings of all elements affected by DYNK in all turns of the simulation, as such a file would in this case be very large. An automatic tool is available for doing the conversion of RIPP blocks to DYNK.

CONCLUSION

The new DYNK functionality allows for extremely flexible definition of time-dependent element settings in SixTrack. It allows the user to choose from and combine more than 25 different function types for computing the wanted setting, or load it from a file. The architecture is extensible, and allows for future additions of element types and functions. DYNK therefore opens many possibilities for future studies of the effect of time-dependent changes of element properties.

ACKNOWLEDGMENT

Thanks to Miriam Fitterer for providing one of the test cases for conversion from the ripple module to DYNK.

REFERENCES

[1] F. Schmidt, "SixTrack Version 4.2.16 Single Particle Tracking Code Treating Transverse Motion with Synchrotron Oscillations in a Symplectic Manner", CERN/SL/9456, 2012.

[2] G. Ripken, F. Schmidt, "A symplectic six-dimensional thin-lens formalism for tracking", DESY 95–63 and CERN/SL/95–12(AP), 1995.

ISBN: 978-3-95450-168-7 doi:10.18429/JACoW-IPAC2015-MOPJE069

[3] R.D. Maria et al., "Recent developments and future plans for SixTrack", IPAC'13, Shanghai, May 2013, MOPWO028.

[4] A. Mereghetti et al., "SixTrack-FLUKA active coupling for the upgrade of the SPS scrapers", IPAC'13, WEPEA064.

[5] SixTrack DYNK TWIKI: https://twiki.cern.ch/twiki/bin/viewauth/LHCAtHome/SixTrackDoc#Dynamic_Kicks_DYNK_input_block

[6] M. Giovannozzi et al., "LHC@HOME: A volunteer computing system for massive numerical simulations of beam dynamics and high energy physics events", IPAC'12, New Orleans, May 2012, MOPPD061.

[7] R.D. Maria et al., "HLLHCV1.1 Optics Version for the HL-LHC Upgrade", TUPTY037, *These Proceedings*, IPAC'15, Richmond, VA, USA (2015).

[8] Y. Sun, et al., Phys. Rev. ST Accel. Beams 12, 101002 (2009).

[9] A. Santamaria et al., "Limits on failure scenarios for crab cavities in the HL-LHC", THPF095, *These Proceedings*.

[10] E. McIntosh, private communication.