Commented parameter analysis

Helios++ allows customization via very many parameters. In the design of the new Python API, it is of great importance to analyse exactly what parameters there are and how they should be handled in the future. This document is meant for circulation across the dev team to come up with a plan that allows us to design a good API that avoids parameter bloat.

Explanation

The meaning of the columns in below table is the following:

- `Name`: Name of the parameter in Helios v2
- `Where in v3?`: Where should this be defined in the Python API of v3. Potential options are
  - `global`: Global state manipulated via dedicated functions in the API
  - `property:X`: Should be manipulated as a property of a class `X`. These classes are e.g. `Survey`, `Scanner`, `Platform`, `Leg`, `Scene`, `ScenePart`, `PlatformSettings`, `ScannerSettings`
  - `method:X`: Should be a parameter of a given method like e.g. `Survey.run` or `Survey.add_leg`.
  - `cli`: CLI-only parameter in the (still existing) CLI
  - `none`: Should be omitted in the Python API
- `Validation`: What type annotation/pydantic constraint should be used on this? If you do not know it exactly how to write one, just put as much information as possible in words
- `Group?`: If we would be looking at grouping parameters into suitable clusters, what cluster would that be
- `Remark`: What ever is worth mentioning

Command Line Parameters

Name	Where in v3?	Validation	Default	Group?	Remark
help	cli				
version	global				
test	none				Test should be separate executable
unzip	none				Could be added as a utility, if needed
assets	global	`list[Path]`	Work dir, Helios installation directory, `data` subdir of Helios installation directory		
output	method:Survey.run	`Optional[Path]`	`./output`	Output	
splitByChannel	global and/or method:Survey.run	`bool`	`False`	Output	
writeWaveform	global and/or method:Survey.run	`bool`	`False`	Output?	
writePulse	global and/or method:Survey.run	`bool`	`False`	Output	
calcEchowidth	global and/or method:Survey.run	`bool`	`False`	Execution and Output?	
fullwaveNoise	global and/or method:Survey.run	`bool`	`False`	?	
fixedIncidenceAngle	global and/or method:Survey.run	`bool`	`False`	Execution?	
seed	global	`conint(ge=0)`	random	Execution? Randomness?	
gpsStartTime	property:Survey or method:Survey.run?	`conint(ge=0)`	now		Apparently, `pydantic` coerces timestamp strings to integers
lasOutput	method:Survey.run	`bool`	`True`	Output	

Name	Where in v3?	Validation	Default	Group?	Remark
las10	method:Survey.run	`bool`	`False`	Output	Is this still relevant? *hw*: imo no
zipOutput	method:Survey.run	`bool`	`False`	Output	
lasScale	method:Survey.run	`confloat(gt=0)`	`0.0001`	Output	
parallelization	method:Survey.run	`Literal(X.chunk, X.warehouse)`	`X.chunk`	Execution	`X` would a new enum
num_threads	global and/or method:Survey.run	`conint(ge=1, le=n)`	`n`	Execution	`n` is the max number of physical threads. Or is oversubscription intended?
chunkSize	global and/or method:Survey.run	`conint(ge=1)`	`32`	Execution	
warehouseFactor	global and/or method:Survey.run	`conint(ge=1)`	`4`	Execution	
rebuildScene	method:Survey.from_xml or Scene.__init__	`bool`	`False`	Execution / Scene building?	
noSceneWriting	method:Survey.from_xml or Scene.__init__	`bool`	`False`	Execution / Scene building?	
kdt	method:Scene.from_xml etc.	`Literal(X.simple, X.median, X.sah, X.sah_approximation)`	`X.sah_approximation`	KDTree	`X` is a custom enum
kdtJobs	method:Scene.from_xml etc.	`conint(ge=1, le=n)`	`n`	KDTree	`n` is the number of physically available threads
kdtGeomJobs	method:Scene.from_xml etc.	`conint(ge=1, le=n)`	`1`	KDTree	`n` is the number of physically available threads
sahNodes	method:Scene.from_xml etc.	`conint(ge=1)`	`32`	KDTree	
disablePlatformNoise	property:Platform	`bool`	`False`		
disableLegNoise	property:Leg or property:Survey (for all legs)	`bool`	`False`		
logFile	global and/or Survey.run	`bool`	`False`	Execution or Logging?	
logFileOnly	global and/or Survey.run	`bool`	`False`	Execution or Logging?	
verbosity	global and/or Survey.run	`Literal(LogLevel.ERROR, ...)`	?	Execution or Logging?	`LogLevel` would be a new enum. Merges existing CLI flags (`-q`, `-vt`, `-v`, `-vv`)

ScannerSettings

`ScannerSettings` will for sure be grouped together like they were before. Ideally, we would build subclasses of that which only contain those parameters relevant to to the optics types `Rotating`, `Oscillating`, `Line`, `Conic` and `Risley`. The `Groups` column is used for that. A parameter can belong to multiple groups if it is relevant in more than one subset. From the technical side, the default for the same parameter could differ per group if you find that a useful thing.

`ScannerSettings` can be set hierarchically with the most specific set of given parameters taking precedence:

- Globally (Is this desirable?) *hw*: I'd say no
- property:Survey
- property:Leg
- Individual parameters given as keyword arguments to `Survey.add_leg`

Previously, these settings used a templating mechanism. In a Python API, I would argue that the power of a programming language to configure your simulations makes such concepts obsolete.

Name	Validation	Default	Groups	Remark
is_active	`bool`	`True`	?	
head_rotation	`confloat(ge=0, lt=360)`	`0`	Rotating head / TLS	*hw*: In deg/s. Can be positive for counterclockwise, negative for clockwise rotation. Limited by maximum head rotation speed of scanner device. - check validation
rotation_start_angle	`confloat(ge=0, lt=360)`	`0`	Rotating head / TLS	*hw:*: check validation
rotation_stop_angle	`confloat(ge=0, lt=360)`	`0`	Rotating head / TLS	*hw:* Expected to be > rotation_start_angle for counterclockwise rotation (and < for clockwise); Can be > 360 deg to have scanner rotate multiple full rotations, see `surveys/demo/box_survey_static_puck.xml` - check validation
pulse_frequency	`conint(gt=0)`	`300000`	?	
scan_angle	`float`	`0.349066`	?	What is the exact constraint?
min_vertical_angle	`float`	`nan`	TLS	What is the exact constraint? Is `nan` really a good default here? *hw:* Is set currently set internally by `scan_angle` (which defines the +- angle), would overwrite the min angle
max_vertical_angle	`float`	`nan`	TLS	What is the exact constraint? Is `nan` really a good default here? *hw*: See above, would overwrite the max angle
scan_frequency	`confloat(gt=0)`	`200`	?	*hw*: Constrained my max possible scan angle of device
beam_divergence_angle	`float`	`0.0003`	?	What is the exact constraint? *hw*: Should this be here, i.e., do we want to allow overriding the angle set for the scanner device
trajectory_time_interval	`confloat(gt=0)`	`0.01`	?	
vertical_resolution	`float`	`0`	TLS	Shouldnt this be > 0? *hw*: Remove this from C++ and move to Python? - overwrites `scan_frequency`
horizontal_resolution	`float`	`0`	Rotating head/TLS	Shouldnt this be > 0? *hw*: Remove this from C++ and move to Python? - overwrites `head_rotation`

PlatformSettings

"Normal" Platform Settings

By "normal" here we mean that they are part of the C++ class `PlatformSettings`. These `PlatformSettings` are dealt with similar to `ScannerSettings`, only that the groups refer to `TLS`, `ALS` etc.

Name	Validation	Default	Groups	Remark
x	`float`	`0`	universal	valid for any platform
y	`float`	`0`	universal	
z	`float`	`0`	universal	
is_yaw_angle_specified	`bool`	`False`	`multicopter`	*hw*: hw: decide whether to keep the platform
yaw_angle	`confloat(ge=0 , lt=360)`	`0`	`multicopter`	
is_on_ground	`bool`	`False`		
is_stop_and_turn	`bool`	`True`	`multicopter`	
is_smooth_turn	`bool`	`False`	`multicopter`	
is_slowdown_enabled	`bool`	`True`	`multicopter`	
speed_m_s	`confloat(gt=0)`	`70`	`linearpath` & `multicopter` (dynamic platforms)	*hw*: Not used for platform type `static`

Trajectory Settings

Trajectory settings are organized as a separate class and could be kept as such:

Name	Validation	Default	Remark
tStart	`float`	`std::numeric_limits<double>::lowest()`	I am not super happy with these defaults, as they are used as signals
tEnd	`float`	`std::numeric_limits<double>::max()`	I am not super happy with these defaults, as they are used as signals

Name	Validation	Default	Remark
teleportToStart	`bool`	`False`	

Additional XML Platform Settings

To my confusion, I can find a lot of platform settings in the XML file, that are not part of the `PlatformSettings` class, but instead are parsed from the XML and then directly set respective properties. I was wondering whether these should be implemented differently in the new API. Also, they do not seem to require fine grained hierarchical control (global/Survey/Leg etc.). This is the list, again with the `Group` column indicating opportunities to aggregate options into groups to keep interfaces slim:

Name	Where in v3?	Validation	Default	Group?	Remark
trajectory	Leg construction	`Path`	-	Trajectory Input	Would it also make sense to have this setting hierarchical and then cut a segment of it with [`tStart`, `tEnd`]
tIndex	Leg construction	`conint(ge=0, le=6)`	`0`	TrajectoryInput	
xIndex	Leg construction	`conint(ge=0, le=6)`	`1`	TrajectoryInput	
yIndex	Leg construction	`conint(ge=0, le=6)`	`2`	TrajectoryInput	
zIndex	Leg construction	`conint(ge=0, le=6)`	`3`	TrajectoryInput	
rollIndex	Leg construction	`conint(ge=0, le=6)`	`4`	TrajectoryInput	
pitchIndex	Leg construction	`conint(ge=0, le=6)`	`5`	TrajectoryInput	
yawIndex	Leg construction	`conint(ge=0, le=6)`	`6`	TrajectoryInput	
trajectory_separator	Leg construction	`constr(min_length=1, max_length=1)`	`,`	TrajectoryInput	
slopeFilterThreshold	?	`float`	`0.0`	TrajectoryInput	?
toRadians	none?	`bool`	`False`	TrajectoryInput	Seems to be only controlling the XML parsing
syncGPSTime	?	`bool`	`False`	TrajectoryInput	?
interpolationDomain	?	`Literal("", "position", "position_and_attitude"`	`""`	TrajectoryInput	?

FullWaveformSettings

You already described full wave form settings as a property of `Survey` in your API sketches. This is fine for me.

Name	Validation	Default	Remark
binSize_ns	`confloat(gt=0)`	`0.25`	
minEchoWidth	`confloat(gt=0)`	`2.5`	
peakEnergy	`confloat(gt=0)`	`500.0`	
apertureDiameter	`confloat(gt=0)`	`0.15`	
scannerEfficiency	`float`	`0.9`	What is the constraint? [0,1]? > yes
atmosphericVisibility	`float`	`0.9`	What is the constraint? [0,1]? > yes
scannerWaveLength	`confloat(gt=0)`	`1550.0`	
beamDivergence_rad	`confloat(ge=0, lt=2*math.PI)`	`0.0003`	*hw*: Is this inherited by the scanner or shall it be possible to overwrite this in the survey?
pulseLength_ns	`confloat(gt=0)`	`4.0`	
beamSampleQuality	`int`	`3`	What is the constraint? *hw*: > (>= 1) - not sure about how to determine an upper limit (-> computational constraints)
winSize_ns	`confloat(gt=0)`	`pulseLength_ns / 4.0`	Not sure yet how we can express defaults of this kind, but it is good to collect the information
maxFullwaveRange_ns	`confloat(ge=0)`	`0.0`	

XML Settings not currently matched to any of the above settings groups

I have found these settings when looking through all provided XMLs, but they came across as isolated occurences:

Name	Where in v2?	Where in v3?	Validation	Default	Remark
rotationSpec	XML Survey attribute	property:Survey	`Literal('CANONICAL', 'ARINC 705')`	`"ARINC 705"`	
scannerMount	XML Survey				I am confused by this one. Why is it on the survey and not on the platform? *hw*: Because we can (not saying we should/still need to)
detectorSettings	XML Survey				I only found one occurence and would need to learn more about this. What other similar options are there? *hw* Lets us override some detector settings from the scanner, see wiki/Survey#detector-settings

Omitted parameters

I omitted all XML parameters used to describe `Scene`, `Scanner` and `Platform`. For `Scene`, the translation of the parameters to a Python API seems fairly obvious (although the exact design of that API is not). For `Scanner` and `Platform`, their programmatic definition is something to be tackled in the future and I very much hope that the exact parameters required to do that would not interfere with any of the other parameters here.