2022 IEEE Workshop on Design Automation for CPS and IoT (DESTION) | 978-1-6654-7040-7/22/$31.00 ©2022 IEEE | DOI: 10.1109/DESTION56136.2022.00011

Rapid Design Space Exploration with Constraint Programming

Miklos Maroti
Bolyai Institute
University of Szeged
Szeged, Hungary
mmaroti@math.u-szeged.hu

Will Hedgecock
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
ronald.w.hedgecock@vanderbilt.edu

Peter Volgyesi
Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN, USA
peter.volgyesi@vanderbilt.edu

***Abstract*—Sample-efficient design space exploration (DSE) of complex CPS architectures remains a key challenge for identifying optimal configurations of components, design parameters and architectural choices. Detailed executable models require significant investment to build and are typically slow to evaluate. On the other hand, high-level conceptual models may lack the exactness or accuracy to evaluate and compare. In this paper we propose a constraint-based approach for capturing the design space and a vectorized, iterative solver for rapidly discovering Pareto-optimal design points. The paper describes the constraint-based modeling approach and developed tools through a concrete design optimization problem of unmanned underwater vehicles.**

***Index Terms*—design optimization, constraints, Pareto-optimal, UUV, CPS**

I. Introduction

Cyber-Physical Systems (CPS) are systems where the functionality emerges from the networked interaction of computational and physical processes. The tight interaction of physical and computational processes turns the design of these systems into a multi-domain co-design problem that integrates traditionally separated design domains into a coupled Design Space Construction (DSC), Design Composition (DC), and Design Space Exploration (DSE) process. Traditionally, these processes and the key design domains are targeted by separate groups of subject matter experts using highly different engineering approaches, design representations and tools. These domain-focused models and fragile, semi-manual integration interfaces are one of the primary reasons of conservative choices in the overall system architecture and a very slow turnaround time in design iterations. Thus, design optimization is restricted to a small fraction of the feasible design space. DARPA's AVM program was aimed at this problem by building horizontal models, tools, and execution integration platforms and end-to-end design automation tool suites [1]. Synthesizing, exploring and optimizing such multi-domain executable models with AI-based approaches is the goal of the DARPA Symbiotic Design program [2]–[4].

This work is supported by DARPA's Symbiotic Design for CPS program and by the Air Force Research Laboratory (FA8750-20-C-0537). Any opinions, findings, and conclusions, or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of DARPA or AFRL.

However, DSE automation and the synthesis of correct-by-construction designs require a large upfront engineering investment to produce a consistent library of multi-domain component models. One practical answer is to start this exploration with back-of-the-envelope calculations, and incomplete high-level conceptual models. One of the prevailing approaches in practice is to capture the key design parameters, performance metrics and fundamental constraints in spreadsheets. The value of such "design representation" should not be underestimated. It enables system-level human reasoning and very quick "what-if" analysis. Unfortunately, the spreadsheet approach breaks down fast as more and more design variables and constraints—rooted in physics and in the requirements—are added. Furthermore, design space exploration—by changing cell values and observing the ripple effects—is highly manual, error prone and follows a single linear exploration and evaluation path. In this paper we introduce a more systematic approach and tools supporting this design phase. The constraint programming approach captures the same conceptual level parameters and relationships of the model but automates and massively parallelize the exploration task. It also creates a bridge towards refining the conceptual design and integrating more detailed and accurate analysis models.

The constraint solver framework [5] is built on SymPy [6] for capturing the formal expressions and constraints of the design. These expressions are automatically translated to PyTorch [7] computational graphs for implementing a vectorized solver and design filters operating on thousands of design points concurrently. The solver and filter steps are executed iteratively (5–10 steps) and concurrently (1000–5000 design points) for finding representative designs near the Pareto-optimal surfaces. The overall process is shown in Figure 2.

We successfully applied the constraint-based approach for capturing the design space of underwater glider vehicles [8] and used the iterative optimization process for finding a large set of Pareto-efficient [9] design points for various mission-level requirements. The benefits of the constraint programming approach are: (1) well-defined formal models even at the conceptual phase, (2) the incremental refinement of multi-domain constraints, (3) rapidly generating large sets of feasible designs with a vectorized solver, (4) iterative pruning of the design space for sparse, Pareto-optimal designs.

978-1-6654-7040-7/22/$31.00 ©2022 IEEE
DOI 10.1109/DESTION56136.2022.00011

II. Design Problem

As a practical use-case for developing a constraint programming toolbox, we focus our initial problem on the design of Unmanned Underwater Vehicles (UUVs), sometimes referred to as Underwater Autonomous Vehicles (UAVs). This domain represents a good starting point as it is self-contained with a concrete and finite set of design components which nonetheless touch on a number of sub-domains common to many design regimes, including electrical, mechanical, physical, navigation, and control subsystems, all of which must be designed with an awareness of one another to achieve a working final product. Additionally, we possess a number of UUV-specific design spreadsheets used by actual domain experts that 1) provide us with a rudimentary ground truth to which to compare our own work, 2) provide many of the underlying mathematics- and physics-based models and equations governing UUV design, and 3) highlight the painstakingly manual design process currently used in industry, providing a direct barometer of how much our constraint-based tools could both speed up and improve the accuracy of the design process.

At the core of the UUV design problem is understanding the intended application of the system under design. It is not enough to simply design a functional vehicle if that vehicle cannot carry out the goals of its intended mission. As such, it is important to distill a number of concrete mission-level requirements that are used in every step of the design process and that dictate regions of viability in the overall design space. For UUV design, this list is remarkably tractable and includes:

- Target deployment environment characteristics (latitude, water salinity, temperature, and maximum depth)
- Mission minimum and/or maximum durations
- Mission minimum and/or maximum transit distances
- Required and nominal transit speeds
- Minimum and maximum neutral buoyancy requirements
- Allowable orientation errors (pitch and roll)
- Minimum and maximum course change speeds

This list represents the bare minimum set of requirements a design must achieve to be successful in its mission. It is important to note that these requirements are not subsystem- or domain-specific. For example, achieving a minimum target mission duration requires considerations on the overall shape and size of the vehicle (physical domain) to minimize drag and maximize lift (fluid dynamics domain), ensuring that the stored power (electrical domain) is sufficient for all electronics as well as for providing the required propulsion (dynamics domain) to achieve the specified transit distance, while ensuring that batteries are housed in a pressurized container of sufficient thickness and shape to withstand pressures at the target dive depth (physics domain), not taking into account any neutral buoyancy requirements (statics domain).

Changing a parameter in any one of these domains affects the design of all other subsystems, both directly and indirectly. Current design methodologies require manual and iterative approaches to designing these subsystems independently while converging on a final overall design; however, this requires that cross-domain optimizations be ignored completely, in addition to lengthening overall design time and requiring conservative choices to be made in the design of individual subsystems. Targeting these cross-domain requirements in a more intuitive and automated way is precisely the goal of the constraint programming tools introduced in this paper.

Let us consider the design of an expendable UUV whose purpose is to travel 1000 km from the edge of an Arctic ice sheet at 75°N to a target latitude of 85°N, release a capsule-shaped payload of size 1.22x0.46 m weighing 65.06 g, ascend to and penetrate the surface layer of ice using positive buoyancy, and remain there transmitting data for a duration of 1 week. In order to overcome adverse arctic currents of $\sim$1 m/s, a maximum speed of 1.5 m/s must be achievable, with a nominal speed of 0.5 m/s. Using this information along with oceanic profiles of the target region, we can determine that there are two sets of mission requirements that the vehicle must be able to satisfy, one corresponding to the time when the vehicle is in-transit with its payload (Transit Stage), and one in which the vehicle is stationary at the surface of the ice for a period of time (Stationary Stage). This produces the following set of mission requirements:

	Transit Stage	Stationary Stage
Average Latitude	80°N	85°N
Water Temperature	-2°C	0°C
Water Salinity	34 PSU	34 PSU
Maximum Depth	3000 m	0 m
Target Distance	1000 m	0 m
Target Duration	n/a	604800 s
Nominal Speed	0.5 m/s	0 m/s
Maximum Speed	1.5 m/s	0 m/s
Neutral Buoyancy	0% weight	3% weight
Maximum Pitch Error	1.5°	1.5°
Maximum Roll Error	0.5°	0.5°

To instantiate this design, we opt to use an underwater glider with a parametric floor plan. A glider is chosen because the transit path required by this design is primarily straight, the transit speeds are quite low, and the maximum depth is large, making it suitable for a design in which propulsion is achieved with very little energy expenditure by simply gliding along a path, using the weight and buoyancy of the vehicle itself to provide the necessary lift. In general, an underwater glider operates by using a "buoyancy engine" to change the volume of displaced water without changing the mass of the vehicle. This enables the vehicle to become heavier than the surrounding water when it wants to descend or lighter when it needs to resurface. By attaching wings to the glider, these vertical forces can be partially translated into horizontal forces. In such a way, forward motion is achieved by continually diving to a predefined depth, increasing buoyancy by moving an incompressible fluid from inside a pressurized container to a bladder outside the container, ascending to the surface, decreasing buoyancy by moving the bladder fluid back into the pressurized container, and repeating this process.

For illustrative purposes, let us also assume that all inter-

nal vehicle components (e.g., electronics, propulsion engines, batteries, pressurized containers, etc.) are well-defined and arranged in a specific, immutable order, and any external components (e.g., payloads, wings, etc.) are well-defined in terms of size, shape, and attachment points, leaving only the wing size alterable to achieve different magnitudes of lift. Furthermore, the glider is assumed to contain four spherical pressurized containers, three cylindrical syntactic foam modules for buoyancy, a set of wings comprising one solid structural piece that extends through the vehicle, a fillable bladder used to alter the buoyancy of the vehicle, a pitch control module, a roll control module, an external antenna, and an externally attached payload. Two of the pressurized containers are packed with battery cells to power the vehicle, one of the containers holds a reservoir of incompressible oil to be used for buoyancy, and the final container houses all navigation and control electronics.

The local coordinate system of the vehicle is designated such that x=0 is located at the nose of the glider, and the center-line at y=0 and z=0 runs along the geometric center of the vehicle. The order and shape of the components within the vehicle from nose to tail are (see cross section in Figure 1 and technical drawing in Figure 3):

1) Syntactic Foam #1: Cylinder
2) Pressure Vessel #1 (Contains Batteries): Sphere
3) Pressure Vessel #2 (Contains Reservoir): Sphere
4) Buoyancy Bladder: Cylinder
5) Syntactic Foam #2: Cylinder
6) Wings and Roll Control: Cuboid
7) Pressure Vessel #3 (Contains Electronics): Sphere
8) Syntactic Foam #3: Cylinder
9) Pressure Vessel #4 (Contains Batteries): Sphere

The roll control component lies directly on top of the interior portion of the wing and consists of a small lead trim weight that is able to move laterally to adjust the center of gravity of the vehicle in the y-direction. The pitch control component lies centered at the very bottom of the vehicle and runs underneath all other components. It also consists of a small lead trim weight that is able to move longitudinally to adjust the center of gravity of the vehicle in the x-direction. Due to the presence of the pitch control component, all components above it are not able to be centered on the center-line of the vehicle, but must be offset vertically by a small amount.

The external antenna component is located centrally on the top of the vehicle, where its attachment point on the x-axis is left as an optimizable parameter, while the external payload is located centrally on the bottom of the vehicle, again with its attachment point left as a free parameter. Additionally, the diameter of all internal components is set equal to the internal diameter of the vehicle when taking into account the added pitch control component. This assumption allows for reduction of the number of unknowns by 1 for every internal component, but it also means that the vehicle diameter will directly affect the number of battery cells and the amount of buoyancy fluid that can be stored in a pressure vessel. Under these assumptions, we have constrained the design problem to:

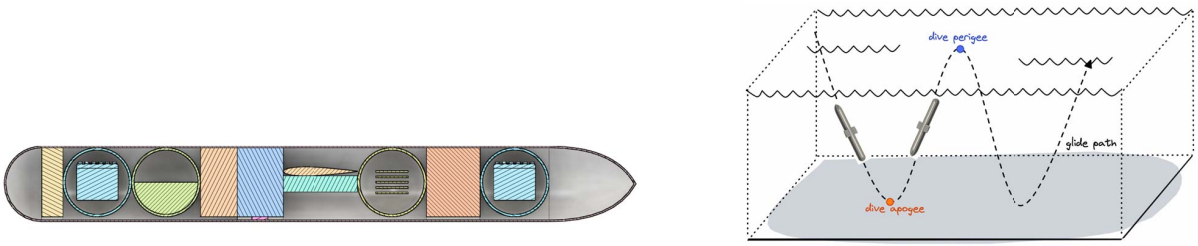
- Correctly size internal pressure containers to withstand ocean pressure,
- Determine the battery capacity required to power the vehicle through all mission stages,
- Pack the required number of battery cells into pressure containers to meet capacity demands,
- Size the wings to generate the lift needed to achieve the necessary transit speeds,
- Size internal buoyancy modules (syntactic foam) to produce the necessary buoyant forces through all mission stages,
- Size pitch and roll control components to allow for orientation changes while maintaining the minimally required pitch and roll error constraints, and
- Determine the optimal vehicle diameter to minimize overall mass.

Given the above design problem, the final set of design parameters that must be solved using our constraint programming tools including the battery capacities of the two battery packs, the uniform material thickness of all pressure vessels, the lengths of each syntactic foam module, the traversal width of the roll control component, the traversal length of the pitch control module, the x-position of the antenna attachment, the x-position of the external payload, and the overall vehicle diameter. Note that there are many additional parameters which may be required when solving the underlying equations for UUV design, but all of these parameters can be considered "derived parameters" as they are able to be expressed in terms of the above-listed free variables. One example of such a parameter is the net buoyancy attributable to a syntactic foam module. This value is determined by the module's absolute buoyancy and absolute gravitational forces, which themselves depend on the size, volume, and density of the foam, all of which are either known or are included in the list of free parameters.

The basis, then, of the design problem as it pertains to constraint programming is that we can generate a set of mathematically constraining expressions, taking into account a list of mission requirements in relation to the set of available free variables. These expressions can take the form of constraints on the length-to-diameter ratio of the vehicle, on the maximum achievable pitch and roll angles, on net buoyancy requirements, on geometrically feasible battery packing optimizations, or any other relationship that must be satisfied in order to generate a mission-compatible design. The next section explores the use of constraints to simultaneously optimize such a set of parameters, allowing the design space to be explored more fully, efficiently, and autonomously.

III. Constraint Programming

By a *design space* we simply mean a subset $S \subseteq \mathbb{R}^n$ where n is the number of *design variables* and each tuple $\bar{x} = (x_1, \ldots, x_n) \in S$ is a possible design. In practice we



(a) Cross section of glider assembly

(b) Typical glide-path

Fig. 1. Architecture and flight-path for the glider-based transit vehicle

often have natural bounds for each (or some) of the design variables $x_i \in [a_i, b_i]$. There are multiple ways to specify a design space, but for our purposes we will consider only algebraic equations and inequalities. By the equivalence of the formulas $f(\bar{x}) \geq 0$ and $\min(f(\bar{x}), 0) = 0$ we can assume that design space is given in the following form:

$$f_1(\bar{x}) = 0,$$
$$\vdots$$
$$f_m(\bar{x}) = 0,$$

where each function $f_j \colon \mathbb{R}^n \to \mathbb{R}$ is continuous and almost everywhere differentiable. Observe, that the typical activation function $\mathrm{ReLU}(x) := \max(x, 0)$ has this property, and thus any neural network based function approximation could be used in the specification of design spaces. At this moment we assume that all design variables are continuous, and handle discrete choices outside of this framework.

Once we have a fixed design space $S \subseteq \mathbb{R}^n$ we want to efficiently sample designs from S and calculate Pareto-optimal instances with respect to multiple variables. Typical Cyber-Physical Systems are severely under constrained, and optimality is not measured along a single variable. For the UUV design study the design space S has $n = 8$ primary design variables, all other parameters were explicitly derived from these, and the number of constraints was $m = 13$. However, the dimension of S was also 8, meaning that any valid design (not on the boundary of the design space) could by slightly modified in every design parameter by a tiny amount and it would still be a valid (but not optimal) design. We typically evaluate the high dimensional designs in a low dimensional evaluation space, thus formally we have maps $g_k : S \to \mathbb{R}$ for $k = 1 \ldots t$ and consider the evaluation space

$$T = \{\, (g_1(\bar{x}), \ldots, g_t(\bar{x})) \in \mathbb{R}^t \mid \bar{x} \in S \,\}.$$

Our goal is to find the Pareto-optimal designs in this space. For the UUV design study we had $t = 2$ or $t = 3$ depending on the choice of evaluation metrics.

Our solution to this problem was the development of a vectorized multi-variate Newton-Raphson solver to quickly find valid designs, that is quickly sample design points $\bar{x} \in S$. The single variate Newton method has been used to find the solution of an algebraic equation $f(x) = 0$ or to approximate transcendental functions in CPU implementations. The basic step of the process is to find an initial approximation x_0 and iteratively improve it as

$$x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}.$$

The same method works in multivariate setting, but one needs work with the Jacobian matrix at the given point and calculate its pseudo-inverse instead of dividing by it. Formally to solve $\bar{f}(\bar{x}) = 0$ we need to calculate

$$J = \left(\frac{\partial g_j}{\partial x_i}\right)_{n \times m}$$

and perform the approximation step

$$\bar{x}_{k+1} = \bar{x}_k - \bar{g}(x_k) \cdot J^+$$

where J^+ is the pseudo-inverse of J (as J is not even rectangular).

The novelty of our approach is that the multi-variate Newton-Raphson method was implemented in PyTorch where we solve multiple instances simultaneously [5]. We have implemented the simplest of the Newton-Raphson variants where we calculated the Jacobian directly using back propagation. Luckily, the machine learning community has done the heavy lifting already, and PyTorch is able to calculate the partial derivatives of the evaluation functions $f_1, \ldots, f_m$ in with respect to the input variables $x_1, \ldots, x_n$ in an m-step loop while retaining the forward computation and computational graph. It is interesting to observe, that compared to typical machine learning tasks the size of the input data, the current value of the variables, is minuscule (n=8) in our case. To effectively exploit the computational hardware we have used multiple data points (which would correspond to the minibatch-concept in machine learning), with the added twist that the errors and computations are not pulled together into a single metric (the loss function) but kept completely separate. In the UUV design study we typically solved 1000-10000 design points concurrently.

The hardest aspect of the implementation was the computation of the pseudo-inverse. Note, that for this step we have thousands of independent Jacobian matrices calculated in a single step and we want to calculate their pseudo inverse in a vectorized way as well. We have used the standard singlular value decomposition (SVD) method to decompose the Jacobian, and calculate the pseudo inverse. These matrices

are individually relatively small, so this is an efficient solution. Typically these matrices are of small rank, that is they have many zero or very small singular values. The small singular values can arise from calculation errors, but also pose a risk for the Newton-Raphson method because the tangent method will result in a big jump which we want to avoid. Thus we have used a relatively high cutoff singular value ($\epsilon = 10^{-3}$) below which we round them down to zero. We have experimented with various cutoff values, but they had very low impact to the overall performance of the Newton-Raphson method. One explanation could be that it is very unlikely that all singular values are small, so we are making progress in each step. After many experimentation we have concluded that it is better to calculate the SVD on the CPU and incur the transfer cost between the GPU and CPU than to use the SVD calculation on the GPU. This seems to be a known problem, and may be addressed in a future versions of PyTorch, but it also could be fundamental because of the sequential computational nature of the SVD calculation. The singular value decomposition treats all variables as equals, by decomposing the matrix into two unitary one (rotations) and a diagonal one. This means that proper scaling is very important in the specification of the design problem.

We have evaluated the vertorized multi-variate Newton-Raphson method on various mathematical datasets, such as elliptic curves on the plain and very flat but high-dimensional surfaces. Our implementation has performed significantly faster than solving the the problems sequentially on the CPU. We have also compared the solver to the typical stochastic gradient descent (SGD) algorithm which we have reimplemented again in a vectorized manner. Typical SGD works with a single problem with training data as constant and weight as input variables, and updates the weights iteratively based on the partial derivatives of the loss value with respect to the input variables. In out case we have multiple independent instances (design vectors $\bar{x} \in \mathbb{R}^n$), and evaluate each of them with a quadratic loss function $\text{loss}(\bar{x}) = \sum_{i=1}^{m} f_i^2(\bar{x})$. This is not the typical workload and significant effort was needed to reimplement the Adam optimizer. However, based on our informal evaluation the SGD algorithm with the Adam optimizer was at least 4-5 order of magnitude slower than vectorized multi-variate Newton-Raphson method, in many cases not converging fast enough.

With our implementation of the Newton-Raphson method we have solved the problem of rapid sampling from the design space S, however this does not solve the problem of finding Pareto-optimal designs in the evaluation space. To address this, we have developed a framework where we can work with sets of feasible designs using the following operations:

1) Given a finite set $X_0 \subset \mathbb{R}^n$ of possible designs, we find a set of feasible designs $X_1 \subseteq S$ using the vectorized multivariate Newton-Raphson method.
2) Given a finite set $X_1 \subseteq S$ of feasible designs we map them to the evaluation space $Y_1 \subseteq T$ using the evaluation map $\bar{x} \mapsto (g_1(\bar{x}), \ldots, g_t(\bar{x}))$.
3) Given a finite set of design and evaluation metrics $(X_1, Y_1) \in S \times T$ we can prune the set of feasible designs to the Pareto-front in the evaluation space and obtain (X_2, Y_2).
4) Given a set of designs X_2 we can prune it to X_3 by removing instances that are very close to each other (that form clusters in the design space).
5) We can enlarge a given a set of feasible designs X_3 with perturbed instance to get more possible designs X_4.

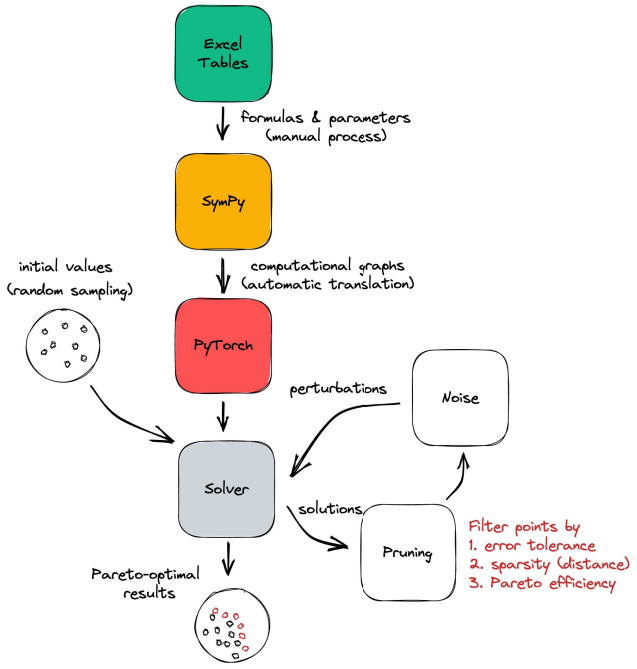
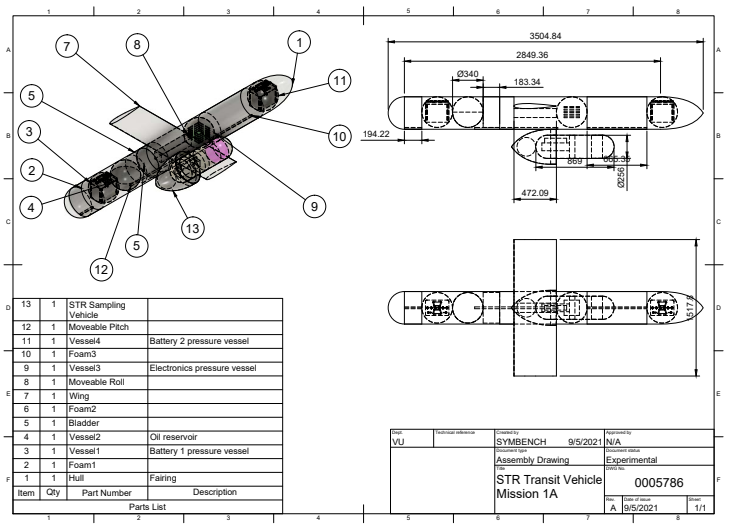


Fig. 2. Design optimization process using the constraint programming framework

The previous steps can be executed iteratively to obtain a larger and large number of feasible designs whose evaluation lies on the Pareto-front of all designs discovered so far. This algorithm is a genetic algorithm, but uses sets of designs at each step. In small mathematical examples and in the UUV study we found that even 5 iterations can give very good approximation of the Pareto-front. Note, that step 1 can return the empty set if the design space is empty or very sparse. We have found it beneficial to relax the design constrains in the beginning (enlarge the design space) and tighten them later when we have a good distribution of feasible designs within S with evaluation metrics in Y close to the Pareto-front.

The iterative genetic algorithm has worked very well in our use cases, but this method relies on the fact that a design is not going to be moved by the Newton-Raphson method if it already satisfies all design constraints. If the dimension of the feasible design manifold S is full, then this is not going to be a problem as the Jacobian is going to be the zero matrix. However, in smaller dimensions this could be challenging as perturbed designs could be moved back to the center of the design space by the Newton-Raphson

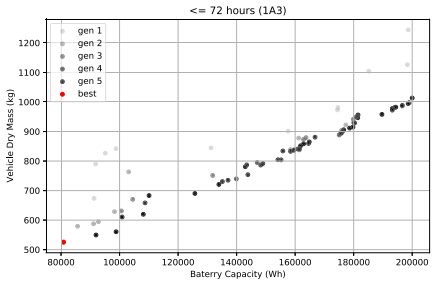


Item	Qty	Part Number	Description
13	1	STR Sampling Vehicle	
12	1	Moveable Pitch	
11	1	Vessel4	Battery 2 pressure vessel
10	1	Foam3	
9	1	Vessel3	Electronics pressure vessel
8	1	Moveable Roll	
7	1	Wing	
6	1	Foam2	
5	1	Bladder	
4	1	Vessel2	Oil reservoir
3	1	Vessel1	Battery 1 pressure vessel
2	1	Foam1	
1	1	Hull	Fairing

(a) Technical drawing of transit and sampling vehicle assembly

Weight:	289 kg	Volume:	0.21 m^3
Length:	2.85 m	Battery:	31,328 Wh
Battery weight:	50 kg (85% packing factor)		
Fairing:	4.8 mm (Fiberglass - PE)		
Pressure vessel	diameter:	0.37 mm (sphere)	
	thickness	11.8 mm (Aluminum-6061-T6)	
Hotel (transit):	1 W	Hotel (survey):	2 W

(b) Key design parameters



(c) Successive generations of Pareto-optimal design points

Fig. 3. Vehicle layout and sizing optimization results for Mission 1A1.

method away from the Pareto-front. To combat this, we plan to add a new constraint that expresses the fact that we are interested only those designs that are beyond the currently discovered Pareto-front. However, the vectorized calculation of the exact signed distance to the Pareto-front with PyTorch is computationnaly intensive and we are experimenting with good enough approximations.

IV. Results and Further Directions

The baseline glider model contains approximately 70 input design parameters, and 80 derived values. The constraint programming framework can generate a new generation (1000–5000) of design points within a few minutes. The actual size of the result set depends on the number of successful solutions (design points within a preset error tolerance), sparsity (close solutions are removed) and if the Pareto-optimal points are kept, only. Note, that for Pareto-efficiency we need additional decision/information from the SMEs. In this study we used the *total battery capacity* (higher is better) and the *vehicle dry mass* (lower value is desired) as the two primary dimensions for Pareto-pruning. Figure 3.c shows multiple generations of the iterative solver-pruning loop with a clearly noticeable trend towards the theoretical Pareto-front. At the end of the design optimization process we selected the lightest design—based on additional feedback from the domain experts. The most important design parameter values and vehicle geometry for this design point are shown in Figure 3.a-b.

Since the current model captures high-level conceptual design considerations, there are no detailed simulation and analysis tools available for gaining additional insights of the expected performance of the selected design(s). However, the constraint-based approach guarantees that all provided design points are consistent w.r.t the originally captured physics and requirements constraints. A more detailed analysis was carried out by the human domain experts. We supported these interactions with a parametric CAD model. Just like to actual set of expressions and constraints, the parametric CAD geometry was built manually. However—since the geometric parameters are a subset of the design parameters we solved for—the concrete CAD model was automatically generated for each solution.

We were able to adapt our glider model to changing mission-level requirements for (1) increased transit distance, (2) limited dive envelope, and (3) increased speed. The constraint-based approach enabled us to run these studies within a few hours. It also demonstrated the limitations of a fixed vehicle architecture. The limited dive envelope resulted in significantly *bulkier* designs. The required battery capacity increased more quickly with mass indicating that as the vehicle gets heavier, it takes more power to go the same distance using more shallow dive depths. Furthermore, we failed to find any valid design points when higher speed (shorter mission time) was required, cleary—and verifiably—showing the limitations of the glider-based approach.

Finally, the set of formal expressions and constraints can be easily extended with learned functions—e.g. integrating ML models trained on numerical simulation results instead of the current empirical algebraic approximations for hygroscopic stress (using FreeCAD/CalculiX), hydrodynamic lift and drag forces (using OpenFOAM). One important requirement for these surrogate models is to provide accurate estimates not only for the actual values of the design variables but on their true gradients—with respect to the design parameters. These extensions and a more composable, hiearchical representation and synthesis of the constraint model are the primary focus of our further efforts.

REFERENCES

[1] J. Sztipanovits, T. Bapty, X. Koutsoukos, Z. Lattmann, S. Neema, and E. Jackson, "Model and tool integration platforms for cyber–physical system design," *Proceedings of the IEEE*, vol. 106, no. 9, pp. 1501–1526, 2018.

[2] H. Vardhan, P. Volgyesi, and J. Sztipanovits, *Machine Learning Assisted Propeller Design*. New York, NY, USA: Association for Computing Machinery, 2021, p. 227–228. [Online]. Available: https://doi.org/10.1145/3450267.3452001

[3] A. Ozdagli and X. Koutsoukos, "Domain adaptation for structural health monitoring," *Annual Conference of the PHM Society*, vol. 12, p. 9, 11 2020.

[4] ——, "Model-based damage detection through physics guided learning," *Annual Conference of the PHM Society*, vol. 13, 11 2021.

[5] M. Maroti and Z. Vizi, "Constraint programming toolbox," https://github.com/symbench/constraint-prog, 2021.

[6] A. Meurer, C. P. Smith, M. Paprocki, O. Čertík, S. B. Kirpichev, M. Rocklin, A. Kumar, S. Ivanov, J. K. Moore, S. Singh, T. Rathnayake, S. Vig, B. E. Granger, R. P. Muller, F. Bonazzi, H. Gupta, S. Vats, F. Johansson, F. Pedregosa, M. J. Curry, A. R. Terrel, v. Roučka, A. Saboo, I. Fernando, S. Kulal, R. Cimrman, and A. Scopatz, "Sympy: symbolic computing in python," *PeerJ Computer Science*, vol. 3, p. e103, Jan. 2017. [Online]. Available: https://doi.org/10.7717/peerj-cs.103

[7] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala, "Pytorch: An imperative style, high-performance deep learning library," in *Advances in Neural Information Processing Systems 32*, H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, Eds. Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[8] W. P. Barker, "An analysis of undersea glider architectures and an assessment of undersea glider integration into undersea applications," Master's thesis, Naval Postgraduate School, 9 2012.

[9] M. Amir and T. Givargis, "Pareto optimal design space exploration of cyber-physical systems," *Internet of Things*, vol. 12, p. 100308, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2542660520301402