Automated Controller and Sensor Configuration Synthesis using Dimensional Analysis

Marcus Pirron, Damien Zufferey, and Phillip Stanley-Marbell

Abstract—**Automated controller synthesis methods for cyber-physical systems (CPS) often require precise knowledge of the system's state. Unfortunately, parts of the state may not be directly measurable, which limits the application of these methods. We present a design methodology for the co-design of software controllers and the required sensing capabilities. Our method leverages the knowledge of physical units in the model of a system to find ways of indirectly measuring parts of the system's state which cannot be measured directly. The method contains a search procedure which uses dimensional analysis to explore the space of physically well-typed expressions and it generates as an intermediate result possible sensor combinations.The integration between the physical and software design for CPS that we present make automated controller synthesis techniques more widely applicable. We have implemented our method and applied it to the design of robotic manipulators.**

Index Terms—**Control systems, Embedded software, Sensor systems, Measurement units, State estimation.**

I. Introduction

We present a design methodology for the co-design of software controllers along with the sensing infrastructure required by the controller. Automated controller synthesis methods, e.g., Model Predictive Control (MPC) [1] and Abstraction Based Controller Design (ABCD) [2], assume that the state of the system can be measured precisely. This is a strong assumption and it is up to users of these methods to make sure this is possible, or, if not, to devise a different system model. At the same time, progress in rapid manufacturing, e.g., 3D printing, has spurred creation of techniques and tools [3], [4], [5], [6], [7], [8], [9] to help non-expert users create and optimize robotic designs. Some of the tools can even generate basic software controllers for the robots [10], [11], [12], [13]. Unfortunately, the control strategies used by these tools are very simple. For future advanced robotics systems which integrate sensing, computation, and actuation directly into materials [14], [15], [16], [17], [18], the challenge of automated materials-specific controller synthesis becomes even greater. In this work, we show how to use dimensional analysis to automatically search over the space of physically well-typed expressions in order to derive a combination of sensors to measure and estimate the system's state. Our technique can be applied to bridge these robotic design tools with more advanced controller synthesis techniques.

Marcus Pirron and Damien Zufferey are with the Max Planck Institute for Software Systems, Germany. e-mail: {mpirron,zufferey}@mpi-sws.org

Phillip Stanley-Marbell is with the University of Cambridge, United Kingdom. e-mail: phillip.stanley-marbell@eng.cam.ac.uk

Manuscript received April 17, 2020; revised June 17, 2020; accepted July 6, 2020. This article was presented in the International Conference on Embedded Software 2020 and appears as part of the ESWEEK-TCAD special issue.

Our approach starts with (1) a high-level description of the *plant model* where each part of the system is annotated with its physical dimension, and (2) a specification of what the controller needs to achieve in the form of a *reach-avoid specification*. (Annotating the physical dimensions to the system covers both the system's state, i.e., what changes over time like the positions of actuators, and static elements used to build the system.)

In a first step, our method uses an off-the-shelf controller synthesis tool to obtain a controller. During that step, the method considers the plant model as a whitebox and allows the controller to read from the whole system's state. In a second step, our method inspects the controller to extract which elements within the overall system's state are used by the controller. Third, the method runs a search procedure to find the required sensors to measure the signals which the controller uses. This search procedure tries to match each dimension of the state with a sensor, or otherwise expands the search by generating candidate dimensionally-plausible invariants which can be used for indirect measurement.

Consider a robotic manipulator, whose range of motion is limited by its payload. With a small payload, the manipulator can extend further than when it carries a heavier payload. Our method would first propose a direct measurement, e.g., by adding a load cell in the effector. Expanding the search will first propose using the Euler-Bernoulli beam equation to estimate the payload by measuring deflection in the structure. Continuing the search will reach the actuator and propose to compute the load by looking at the force exerted by the actuator. These different possibilities are presented to the user who can select one. Once the configuration of sensors is fixed, we generate descriptions of the model in the Newton physical system specification language [19] and use Newton's estimation backend [20] to generate an extended Kalman filter which takes as input the sensor readings and controller outputs to estimate the overall system's state. This state is then sent back into the controller to close the feedback loop.

The candidate invariants used in our search are dimensionless groups, i.e., equations where all physical dimensions cancel out. We generate these dimensionless groups in an automated process using Newton. The Buckingham Π theorem [21] guarantees that any physically meaningful equation can be written in that form. However, formulas generated by this approach are only proportionality constraints. The first time the system is used, we run a calibration procedure to estimate the proportionality coefficients. The controller also monitors deviations between the predicted system state and sensor readings which can be used to detect failure.

We have implemented our method in a tool called MPERL-Π, which extends the Multipurpose Parallel End effector Robotics Language (MPERL) [12] with dimensional analysis and Π groups.

Motivating Example: Throughout the paper, we use the running example of a SCARA robot system to illustrate how MPERL-Π can be used to learn about the state of the robot and how this information can be used by the controller. Figure 1 shows the example SCARA manipulator. The arm structure consists of two revolute actuators connected by beams. The upper beam extends from the upper motor to the end effector and the lower motor is fixed to a base. The upper beam is 3D-printed in Nylon and includes a deflection sensor. The two actuators (brushed DC motors) are connected to current sensors.

We built the arm using MPERL and it uses a simple controller which moves the arm between positions by solving the inverse kinematic equations. The controller however only knows about the geometry of the system. We would like the robot to support dynamic effects and to take into account forces exerted on the motors and on the structure. Motors have a limited torque and we would like the controller to dynamically adjust the arm's working envelop depending on the load attached to the effector. With a bigger load attached to the effector, the controller should not extend the arm to its full reach to avoid overloading the motors.

We extended MPERL to compute the forces at different points of the structure and attach additional constraints to specify the maximal torque on each revolute actuator. MPERL originally supported only rotary sensors for position feedback; for the extensions to work properly, the synthesized controller needs to know about the weight of the load attached to the end effector. Using the method presented in this paper, our tool, MPERL-Π, explores possible instrumentations which allow the system to keep track of the weight.

Figure 1 shows the inputs, components, and stages of the method, showing how the components of the method are connected, and, for each component, a list of associated physical quantities. The first possibility would be to measure the weight on the effector but we do not have such a sensor. The next element is the beam attached to the effector. MPERL was designed with 3D printed structures in mind. These structures are not very stiff and MPERL supports flex sensors to compensate for structural deformation. the method also explores the properties of the beam which includes its length, height, width, flexural rigidity, deflection, etc. Using dimensional analysis, automated by Newton, the method can derive multiple candidate relations from these properties. The method filters these candidates in order to find a possible equation which can be used to derive the deflection.

If we continue the search for other sensor configurations, MPERL-Π will also explore other parts of the system. Next to the upper beam is the first actuator. A simple suggestion is to add a torque sensor to the actuators. As such sensors are quite expensive, we ask MPERL-Π to find other combinations of sensors. MPERL-Π considers properties of its actuators. Within the motor specification, there is a table which specifies the torque produced by the motor in relation to the rotational speed and current consumption. To use this equation, MPERL-Π looks at the available information by considering quantities with dimension T^{-1} (rotational speed) and I (current). Because the actuator contains a rotary encoder, MPERL-Π can determine its rotational speed. The current drawn, on the other hand, is unknown; MPERL-Π then suggests the addition of a current sensor to the motor. As current sensors are more cost-effective than torque sensors, this offers a better alternative. At that point we could build the arm with the current sensor. Figure 1 shows the SCARA arm including both current and deflection sensor. We include both sensor configurations to evaluate the system in both configurations.

Contributions:

- We present a physics-based search strategy to find configurations of sensors to directly or indirectly estimate the overall state of the system.
- We have implemented a prototype tool called MPERL-Π and explain the changes we had to make to off-the-shelf tools to support our method.
- We evaluate our prototype with both simulated and real experiments.

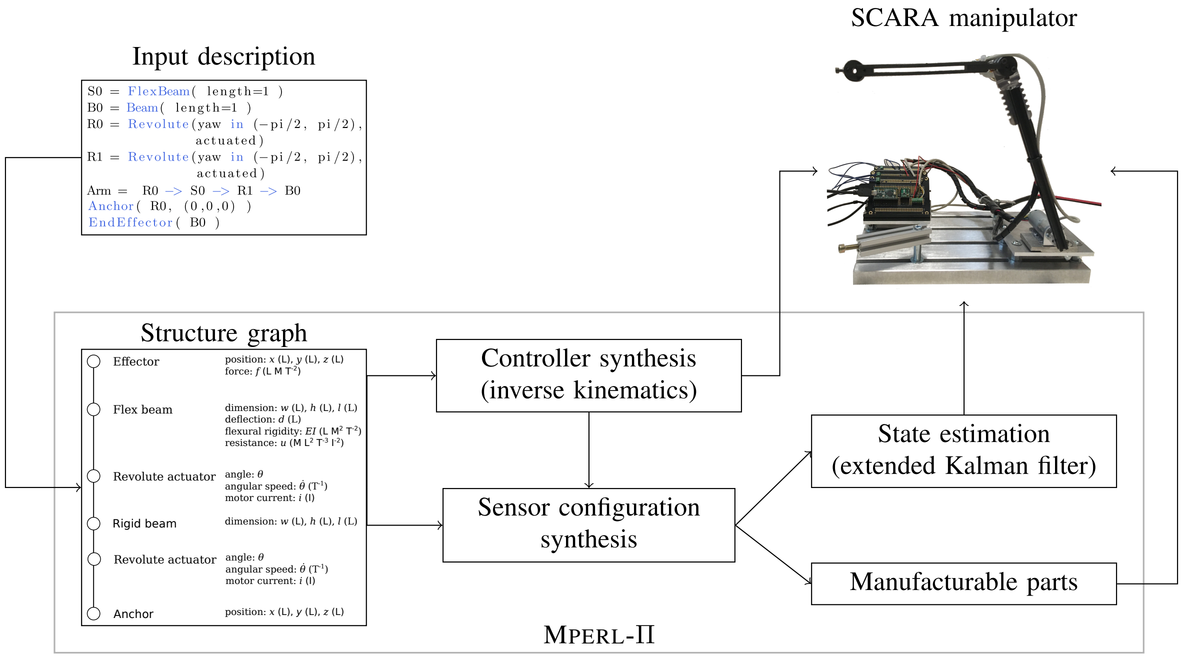
II. SYNTHESIS OF SENSOR CONFIGURATIONS

First, we describe the external tools needed for our method. We make use of tools for (a) automated controller synthesis, (b) dimensional analysis, and (c) state estimation.

a) Automated controller synthesis: A controller tries to get a physical system to behave according to a specification by influencing the system through actuators. A feedback controller also monitors the evolution of the system to correct for disturbances. With more computing power available, controllers can be automatically generated from a description of the physical system as a dynamical system and a specification. This can be done online, e.g., MPC [1], [22], [23], or offline, e.g., ABCD [2], [24], [25], [26]. Because controller synthesis is computationally difficult, strong assumptions are made on the knowledge of the system's state. For instance, ABCD tools [27], [28], [29] generate only state-feedback controllers and assume perfect measurements of the state. With only partial observation, controller synthesis becomes computationally even more difficult as the synthesis procedure needs to consider all the states compatible with the observations [30], [31], [32].

Recently in the field of robotics, we have seen the development of tools [10], [11], [12] which automatically generate software controllers, along with a matching description of manufacturable robotic parts, such as models for 3D printers. These tools require little to no expertise by the user but are limited to specific kinds of robots. Our goal is to take advantage of the design capabilities of these robotic design tools to make automated controller synthesis more applicable. We use dimensional analysis to generate an instrumentation which makes it possible to recover the state information needed by the controller.

b) Dimensional analysis: Dimensions and units are the type system of physics. Only quantities with the same dimensions can be compared and added. Multiplication combines



```
S0 = FlexBeam( length=1 )
B0 = Beam( length=1 )
R0 = Revolute(yaw in (-pi/2, pi/2),
              actuated)
R1 = Revolute(yaw in (-pi/2, pi/2),
              actuated)
Arm =  R0 -> S0 -> R1 -> B0
Anchor( R0, (0,0,0) )
EndEffector( B0 )
```

Input description
SCARA manipulator
Structure graph
Effector
position: x (L), y (L), z (L)
force: f (L M T-2)
Flex beam
dimension: w (L), h (L), l (L)
deflection: d (L)
flexural rigidity: EI (L M2 T-2)
resistance: u (M L2 T-3 I-2)
Revolute actuator
angle: θ
angular speed: θ̇ (T-1)
motor current: i (I)
Rigid beam
dimension: w (L), h (L), l (L)
Revolute actuator
angle: θ
angular speed: θ̇ (T-1)
motor current: i (I)
Anchor
position: x (L), y (L), z (L)
Controller synthesis (inverse kinematics)
Sensor configuration synthesis
State estimation (extended Kalman filter)
Manufacturable parts
MPERL-Π

Fig. 1: System overview

quantities and their dimensions. The SI system [33] identifies 7 base dimensions from which all other dimensions are derived. These dimensions and respective standard units are: time T (second), length L (meter), mass M (kilogram), electric current I (ampere), thermodynamic temperature Θ (kelvin), amount of substance N (mole), and luminous intensity J (candela). Dimensions should not be confused with units. The value of a quantity is expressed as a number along with a unit of the corresponding dimension. For instance, $0.1m$ and $10cm$ represent the same quantity of length.

Because there is a small number of base dimensions, Rayleigh's method of dimensional analysis has proven to be a powerful tool in a wide range of scientific and engineering fields. Dimensional analysis collects all variables in a system and tries to arrange them in dimensionally homogeneous equations. The theoretical justification of this method comes from the Buckingham Π theorem [21]. This theorem bounds the number of variables that have to be grouped in order to find dimensionless groups, i.e., expressions where all the dimensions cancel out. Recently, Wang et al. [34] showed that this theorem can be used to efficiently generate candidate invariants of physical systems. While invariants of physical systems are dimensionally correct, the converse is unfortunately not true; therefore, further analysis is required to identify actual invariants within the space of dimensionless groups.

c) State estimation: The goal of the state estimation, or filtering problem, is to find the most likely state of a system given a set of observables which varies as a function of the state. The state estimation can handle noise in the observations and make predictions about the system's evolution. We can also make use of the state estimation to compare observations against predictions to detect potential adequacy problems between the system and its model.

In this work we build on top of MPERL [12] and Newton [19].

MPERL is a tool to help non-experts build custom robotic manipulators. Using an abstract description of the robot's kinematic structure, MPERL generates both, a controller based on an inverse kinematic solver and 3D models to manufacture the corresponding manipulator. As MPERL contains detailed information about the robot's structure, e.g., joints, actuators, and sensors, we could easily associate dimensions to all the elements. Furthermore, as the mathematical model of the robot's motion and the physical object are generated from the same description, we know they match each other.

Newton is a specification language to describe constraints and invariants on values obtained from sensors embedded in physical objects. All the values in Newton are annotated with their dimensions. Moreover, Newton integrates the dimensional analysis [34] and can also generate extended Kalman filters [35] from the description of a system and the attached sensors.

A. System Model and Problem Definition

Let us recall some standard definitions to establish the setting and define the problem we are trying to solve.

Let $\{\mathsf{T}, \mathsf{L}, \mathsf{M}, \mathsf{I}, \Theta, \mathsf{N}, \mathsf{J}\}$ be the finite set of base dimensions. The set $\mathbb{D}$ of all dimension contains expressions of the form $\mathsf{T}^\alpha \mathsf{L}^\beta \mathsf{M}^\gamma \mathsf{I}^\delta \Theta^\varepsilon \mathsf{N}^\zeta \mathsf{J}^\eta$ with the dimensional exponent α, β, γ, δ, ε, ζ, η in $\mathbb{Z}$. A *quantity*, or variable, is a pair in $\mathbb{R} \times \mathbb{D}$. We assume that the value is expressed in the standard SI unit associated with the dimension. We write $dim(q)$ to obtain the dimension of q.

A dynamical *system* S is a tuple (X, X_0, U, F) where $X \subseteq (\mathbb{R} \times \mathbb{D})^n$ is the state space, X_0 is the set of initial states, $U \subseteq (\mathbb{R} \times \mathbb{D})^m$ is the control input space, and F defines the dynamics ($\dot{x} \in F(x, u)$).[1] We use the lower case letter corresponding to the name of the set to represent the elements, e.g. x is an element of X.

An execution, or trace, of duration T is a continuous right-differentiable function $\tau : [0, T] \to X$ such that $\tau(0) \in X_0$ and $\forall t \in [0, T]$. $\exists u$. $\dot{\tau}(t) \in F(\tau(t), u)$.

A state-feedback *controller* C is a function $(\mathbb{R} \times \mathsf{T} \to X) \to U$ which maps a history of the system's trajectory to a control input.

An execution τ is compatible with a controller C if it satisfies the additional condition $\forall t \in [0, T]$. $\dot{\tau}(t) \in F(\tau(t), C(\tau[0, t]))$.

A specification is a set of allowed traces $\mathcal{T}$ for a system S. The controller synthesis problem returns a controller C such that for all executions τ of S and C, we have $\tau \in \mathcal{T}$. Given a set of outputs Y for S and an observation function H that maps Y to X, the state estimation problem tries to recover the state of the system at time t by observing $H(\tau[0, t])$. The *sensor configuration synthesis* problem tries to find Y and H that have a possible physical realization and contain enough information to make the state estimation possible.

An invariant is a property of a robotic system which remains always true while the system is operational. For example, an invariant $R_i(\theta) \in [-\pi/4, \pi/4]$ would ensure that the rotation angle of revolute actuator R_i is always within the interval of $[-\pi/4, \pi/4]$. Invariants are also used to describe the interdependencies between different parts in the robotic system.

At the core of physics, conservation laws states the closed systems preserve some properties. These conservation laws are a specific kind of invariant and our method finds instances of these laws in concrete systems.

B. Sensor Configuration Synthesis

Let us explain how our method works and the details of the synthesis algorithm. In Figure 2, we show the different parts of the sensor synthesis workflow and how they interact. The user provides the initial input at the top of the figure and is also responsible for checking the sensor configuration proposed by the synthesis algorithm.

1) Goal state-space: Given a dynamical system S and a state-feedback controller C, the first step is to find which part of the state the controller actually needs. This step depends on the structure of C. For instance, if C is a linear function of the form $u(t) = -K(\tau(t))$, we select all columns of K with a non-zero entry. If the internal structure of the controller is not available, we can over-approximate necessary parts of the state by taking the full state. In the rest of this section, we denote by x the part of the state we need to reconstruct.

2) Dimensionless groups: When two physical quantities are equal, they must have the same dimensions. For instance, if $a = b$ then $dim(a) = dim(b)$. If a is non-zero, we can rewrite this as $1 = b/a$. The division cancels out the units

[1] Common system model often include an output space Y or and observation function H. We do not include them as our goal is to use the dimensional analysis to infer them.

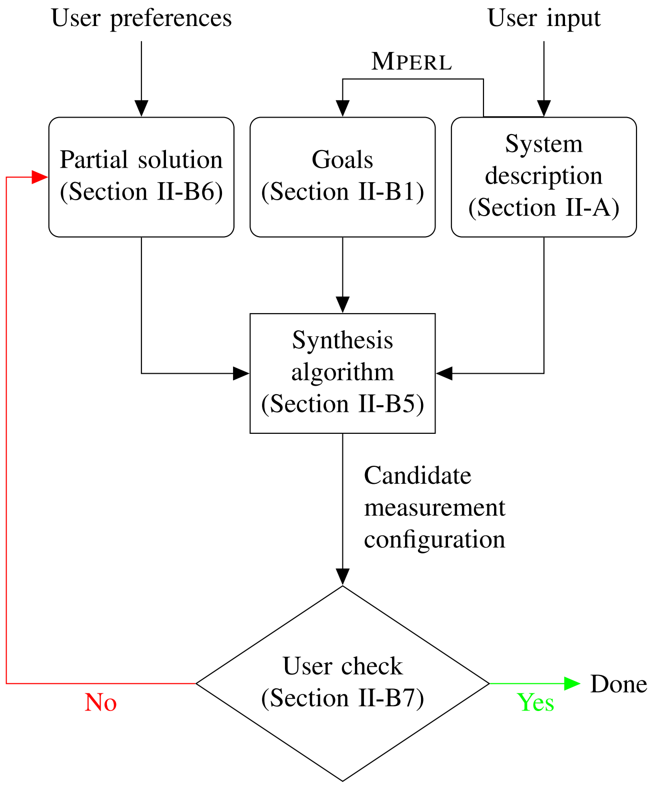

User preferences
User input
MPERL
Partial solution (Section II-B6)
Goals (Section II-B1)
System description (Section II-A)
Synthesis algorithm (Section II-B5)
Candidate measurement configuration
User check (Section II-B7)
No
Yes
Done

Fig. 2: Sensor synthesis workflow

and the result is dimensionless. The product ba^{-1} is called a dimensionless group or Π group, after Buckingham π theorem.

We can find Π groups by dimensional analysis. Consider Newton's second law as an example. We have force F, mass m, and acceleration a with $dim(F) = \mathsf{T}^{-2}\mathsf{LM}$, $dim(m) = \mathsf{M}$, and $dim(a) = \mathsf{T}^{-2}\mathsf{L}$. We are looking for exponents α, β, and γ such that $F^{\alpha} m^{\beta} a^{\gamma}$ is dimensionless. From the dimensions we extract the following constraints: $-2\alpha - 2\gamma = 0$ for T, $\alpha + \gamma = 0$ for L, and $\alpha + \beta = 0$ for M. Solving these linear constraints gives the solution $\alpha = -1$, $\beta = 1$, and $\gamma = 1$ which corresponds to ma/F. From this we get the equation $ma/F = C$ for some dimensionless constant C. Here, C is 1, but this is not always the case. For instance, if we consider load induced deflection of beams, which can be used to measure the force applied to the beam (see Section IV-B), it follows the law $3 = FL^3(EI)^{-1}d^{-1}$, where F is the force applied to the beam, L is the length of the beam, EI is the flexural modulus, and d is the deflection.

In the previous example, the constant C is 3, theoretically [36]. In general, we cannot know a priori the values of the dimensionless constants. We need to experimentally calibrate both, the system and the controller, to find theses values. In the case of the beam deflection, all quantities are constant except for F and d. Overall, the Π group tells us that the deflection is proportional to the force and the calibration finds this proportionality constant.

3) Sensors: While there can be multiple sensors with different ranges, precision, and accuracy, we assume at most one sensor per dimension for the sake of simplicity. The available sensors are represented by the function *sensor* which maps elements of $\mathbb{D}$ to sensors or $\bot$ when no sensor exists. Furthermore, the *cost* function maps the sensor to a cost. To simplify the presentation of the algorithm, we assume that $cost(\bot) = \infty$.

Our synthesis procedure minimizes the overall cost of the sensors, assuming that costs sum up. The cost function is a general function; for example, it can be applied in terms of monetary cost (to build a cost-efficient system), but also in terms of power consumption (to maximize the autonomy of a battery-powered system).

4) Extended system description and distance between variables: We envisage our method to be used when there is not only an abstract model of the system but also a more detailed description of the physical elements. Having a more detailed model makes it possible to find more sensing possibilities

For instance, the controller of the SCARA arm from Figure 1 needs to limit the range of the arm depending on the payload weight; otherwise, the maximal torque of the motors could be exceeded. A minimal model would only include the speed and acceleration of the payload and the torque exerted by the motor. In that setting, our algorithm would suggest to use a torque sensor which is an expensive sensor. With a more general model of the system, our algorithm can explore in more detail some elements. For instance, the motor is an electric motor, turning electric energy into torque. Therefore, the system can suggest measuring the current going into the motor along with its speed to derive the applied torque.

The downside of having access to all this information is the great increase in size of the search space and the number of Π groups. Therefore, we assume that we have a function $dist_S$ which returns the distance, for some notion of distance in S, between two quantities in the system. When generating the Π groups to measure a quantity q we only look for elements below some distance from q.

In our implementation, the structure of the system is represented by a graph where the nodes are physical elements and edges are the interactions between them. The shortest path between two nodes is the distance we use.

5) Search algorithm: Our search procedure, shown in Algorithm 1, is a backtracking search which progressively adds more sensors to the system while keeping the current best found solution to prune the search.

We start the search with the available sensors, their cost, the known constant quantities, and the variables $x_0 \dots x_n$ we need to measure. In our running example, known constant quantities are elements like the length of the beam, or motor characteristics like current consumption, which can be used without incurring any cost. For each variable, we try to find out if a sensor exists or expand the search by generating a candidate invariant involving that variable. The cost function drives the search into generating candidate invariants even if we have a sensor or cut the search if a branch becomes too expensive.

The expansion uses dimensional analysis to find candidate invariants involving the variable to measure. When we find a candidate invariant, we remove the variable to measure from the goal and add all other expressions occurring in the invariant to the goals. In the algorithm, this search is done by calling FINDPIGROUPS(q, Q). Q is the set of all the quantities that can be used to find the Π groups and q is a quantity which must be part of the Π groups, i.e., the exponent for q must be nonzero. FINDPIGROUPS is implemented using the algorithm by Wang et al. [34, Section 3.2] and filtering the groups where q is not present. As the number of candidate invariants grows exponentially with the available number of variables, and many can be spurious, we use the distance function to search only for candidate invariants involving quantities close to each other.

Algorithm 1 Sensor synthesis algorithm

```
Require: $x_0 \dots x_n$, the quantities to measure
Require: $initStatus$, all the known constant quantities of $S$
Require: $\Delta$ ($\geq 0$), the scope of the search
 1: function SEARCH($q, NA, status, c_{curr}, c_{max}$)
 2:     if $(q, \_) \in status$ then
 3:         return $(status, c_{curr})$
 4:     end if
 5:     $(status_{best}, c_{best}) \leftarrow (\bot, c_{max})$
 6:     if $cost(sensor(dim(q))) + c_{curr} \leq c_{max}$ then
 7:         $status_{best} \leftarrow status \cup \{(q, sensor(dim(q)))\}$
 8:         $c_{best} \leftarrow c_{curr} + cost(sensor(dim(q)))$
 9:     end if
10:     $Q \leftarrow \{q' \mid dist_S(q, q') \leq \Delta\} \setminus NA$
11:     $\mathcal{G} \leftarrow$ FINDPIGROUPS$(q, Q)$
12:     for $G \in \mathcal{G}$ do
13:         $(s, c) \leftarrow (status, c_{curr})$
14:         for $q' \in G \land c \neq \infty$ do
15:             $(s, c) \leftarrow$ SEARCH$(q', NA \cup \{q\}, s, c, c_{best})$
16:         end for
17:         if $c \leq c_{best}$ then
18:             $(status_{best}, c_{best}) \leftarrow (s \cup \{(q, G)\}, c)$
19:         end if
20:     end for
21:     if $status_{best} \neq \bot$ then
22:         return $(status_{best}, c_{best})$
23:     else
24:         return $(Failure, \infty)$
25:     end if
26: end function
27:                                   ▷ Apply the search to all the $x_i$
28: $s \leftarrow initStatus$
29: $c \leftarrow 0$
30: for $x \in x_0 \dots x_n \land s \neq Failure$ do
31:     $(s, c) \leftarrow$ SEARCH$(x_i, \emptyset, s, c, \infty)$
32: end for
33: return $s$
```

The algorithm search procedure (line 1–26) keeps track of (1) q the quantity that needs to be measured, (2) NA the quantities not available, (3) $status$ a partial solution, (4) c_{curr} the current cost, and (5) c_{max} the maximal cost. In the set NA, we keep track of the elements below the current branch of the search tree. These elements cannot be used as it would introduce circular dependencies in the result. $status$ is a set containing pairs of quantities and how they are measured: either directly with a sensor or derived indirectly through an invariant. The maximal cost c_{max} cuts the search when a better solution is already known.

In the search, we first try to check if a quantity is already known (line 2), in which case there is nothing to do. Then, we try to find a sensor which matches the quantity's dimension (line 6); if that fails, we consider indirect measurements. We gather the elements in the system in the neighborhood of q (line 10), find the dimensionless groups over these elements, and search recursively on them (line 11–20). In Section III, we discuss the scope of the search in more detail. The search is applied to all the elements we need to measure (line 28–33).

Restricting the search according to a distance has two goals: 1) improving the accuracy of the solution and 2) limiting the complexity of the search. Currently, we rely on the state estimation to deal with measurement errors. Measuring a quantity while being far means multiplying terms between the measurements and the goals. This compound errors and further measurements are less likely to yield good data. In Section II-D, we discuss methods to improve error handling. The second aspect is the scalability of the method. The number of Π groups is exponential in the size of the overall system. Therefore, a naive application of the search only works on small systems. On the other hand, the size of neighborhoods within a system should contain roughly the same number of elements independently of the overall system. This makes it possible to apply the search to larger systems.

6) User Constraints on the Search: Algorithm 1's presentation is minimal. It runs and returns one solution. If this solution is not satisfactory, we want the algorithm to explore other solutions, which can be achieved with minor changes to the algorithm's initial state. It is possible to enforce the usage of a specific sensor by adding it to $initStatus$. Conversely, we can prevent using a specific sensor by adding the corresponding quantity to NA at the beginning of the search. If the algorithm fails to find a solution the search can be tried again with a larger Δ.

Searching for different solutions can also be used to generate multiple ways of measuring the same quantity. Using multiple sensors can improve the quality of the state estimation if the elements used are independent across measurements.

7) Checking Candidate Solutions: The sensing configurations comes from dimensionally-correct equations, but these equations may not correspond to actual physical processes in the system. Therefore, we require that the user checks the output of the algorithm. We can distinguish between two sources of spurious solutions: (1) the Π group does not correspond to any physical law and (2) the Π group corresponds to a physical law, but associates the (right) dimension to the wrong element. The first case requires checking if there exists any law of physics corresponding to what the algorithm suggested. The sensing configuration can also be tested against a small number of known configurations to check if they hold. The second case happens because the search does not differentiate quantities with the same dimension. Such errors are harmless when they stem from constant terms, e.g. using the width instead of the length of a beam, as the Π groups are an equality up to a constant. When the error is not about a constant term, then the process to discard such solutions is similar to the first case.

This step still requires some knowledge from the user. As checking a given solution is easier than finding a solution, our method still lowers the expertise requirement to build CPS. A non-expert may be overwhelmed and not know how to solve the problem. With our method, the algorithm gives them a "place to start." If the user is already an expert, they may not need our system but using it can still help them accomplish the task faster.

C. Calibration and Runtime

After we have generated possible sensor configurations and the user has checked if they are meaningful for the system, we can generate the output space Y given by the sensors and the observation function H. The observation function corresponds to extracting the equations stored in $status$ and rewriting them such that their value is a function of the state. The dynamics model and the observation model are both used to generate an state estimator.

At this point, the observation function still contains proportionality constants coming from the Π groups. We need to calibrate the system and the sensors to find these values, which is system specific. In Section IV we explain how we did this in our evaluation.

Our approach requires this calibration phase. Currently, we only use the Π groups vetted by the user but we can also take advantage of this phase to reduce the burden on the user. As a mitigation technique, once the user has selected a sensor configuration, we gather all Π groups for that configuration and later, during the calibration phase, use regression to find the relevant ones. Furthermore, if the sensor and their placement get cheap enough [14], [15], [16], [17], [18], we could also embed more sensors for Π groups which have not been checked by the user. Then, during the calibration, we could use the data to learn which Π groups correspond to actual invariants of the system. In the future, we plan to add outlier detection [37] and consensus based method to detect faulty sensors [38] to the state estimation backend.

D. Limitations and Extensions of the Method

Buckingham Π theorem tells us how to compute dimensionless groups, but it does not tell us the physically meaningful ones. In our method, we rely on the user's judgment for checking that at least one Π group is physically meaningful as explained in Section II-B7 and II-C.

During the calibration, we need to get accurate measurements, which implies the possibility of measuring the entirety of the system's state. For some systems, this might not be possible.

Finally, while running the system we need to take measurement errors and the model adequacy into account. As we do not put any restriction on the model, we use an extended Kalman filter for the state estimation with non-linear dynamics. Unfortunately, this does not give any guarantees on the quality of the estimation which is fed to the controller. The filter also assumes Gaussian noise on the measurements, and therefore, we need to ensure that the sensors have this noise profile and no drift. Another limitation of the method is sensor selection and placement. Sensors can sense over different ranges, accuracies, and sampling frequencies. The range requirements for sensor could be established using interval analysis [39]. We currently use a sample-and-hold controller, and therefore, the controller-loop frequency determines the required sampling speed of the sensors. The method could be used with event-triggered control [40] for sensors which support programmable interrupts. Sensor placement can also affect the quality of their output. Some sensor are simple to place, such as a current sensor which can be placed anywhere on a wire, but sensors related to material properties are much more sensitive to location. For instance, optimizing the placement of deflection sensor requires finite element analysis [41]. The model adequacy is obviously outside of our control.

III. Implementation

We describe our prototype implementation MPERL-Π, built on top of MPERL and Newton, and discuss a concrete instantiation of the parts left abstract in the previous section.

MPERL *in a nutshell:* MPERL is a tool to help non-expert users build robotic manipulators and encompasses all aspects of system building, from synthesizing software to generating manufacturable hardware. The main data structure inside the tool is a graph whose vertices are physical elements and whose edges represent their interactions. Each vertex comes with a set of parameters that can be set or read by the controller and a rigid motion which constrains the relative position of the elements to which it is connected. With this information, MPERL generates a simple controller which moves the manipulator between different configurations. First, a kinematic model is extracted from the system's description. This model is used by an inverse kinematic solver and analyzed to compute the manipulator's range of motion. Within the workspace, MPERL determines which configurations are safe for the structure. By default, unsafe configurations are singularities, i.e., configuration where degrees of freedom are lost and the manipulator can either become uncontrollable or gets damaged.

Dimensions and element properties: The main modification of MPERL-Π compared to MPERL is the addition of dimensions. As with many tools, all quantities in MPERL are floating-point numbers and dimensions are implicit. We made dimensions explicit as a prerequisite of the dimensional analysis. Each element type of MPERL was revised and enhanced with additional properties, necessary for the dimensional analysis. As an example, revolute actuators have been refined with properties like power consumption, speed, torque, voltage or resistance, subject to their respective datasheets.

From kinematic to dynamic: With MPERL-Π having more detailed information about the nature of each physical element, we extended the model to generate the controller; while MPERL only knows about the geometry (kinematic) of the elements, and the synthesized controller supports only kinematics, for MPERL-Π, we extended this approach to take into account (some) dynamic effects: the system computes how forces propagate through the robots and the torque applied to

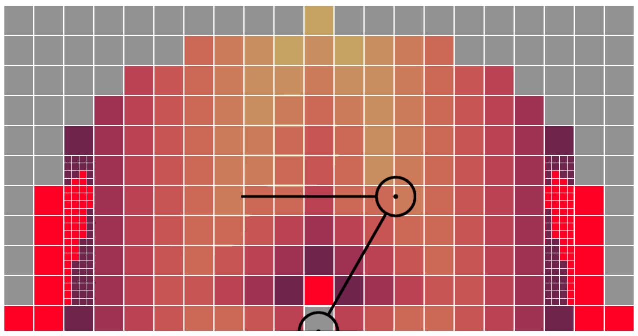


Fig. 3: Representation of the SCARA arm workspace while torque is applied to the actuators. Grey regions are outside of the reachable workspace. Regions with low torque are represented by lighter colors, darker regions represent a higher load. Red regions are unsafe regions of the workspace, where the motors' maximal torque is exceeded.

each element. For the safety properties of the controller we combine singularities of the robot with physical constraints of the model. We refine the workspace analysis to consider both kinematic and dynamic properties. We reuse the approach of [12] that pre-computes the safe region of the state space using adaptive sampling. We sample the configuration space with a varying density of samples depending on the distance to unsafe regions. In that model, the safety property must be modeled by a differentiable function from the system's state-space to $\mathbb{R}$ and the system is safe if the value returned by the function is larger than 0. The differentiablility requirement is needed by the adaptive sampling. When the function's value get close to 0, the sampling is refined toward the unsafe region to map it more accurately. While this is done offline and can take time, partitioning the state space into a grid is exponential in the number of variables in the system; the adaptive strategy reduces the computation. Figure 3 shows which part of the SCARA arm's workspace is safe or close to the limit of the actuators capabilities. The grid is used by the controller to find safe trajectories. For a start and end configuration, the controller tries to find a path within the safe region. Currently, MPERL-Π only supports state-based safety properties, but not temporal properties based on sequences of events.

Π *group generation:* For the Π group generation, MPERL-Π prepares the appropriate quantities and calls Newton to generate the Π groups. As the number of groups grows exponentially with the number of quantities, MPERL-Π uses a local search. The local search starts from a node in the graph and gathers all nodes within a certain range. It can also filter types. This allows a user to create local invariants, e.g. invariants spanning only parts / subsets of the robotic system, or type-specific invariants, e.g. invariants which are valid only for revolute joints. In the single SCARA example, instead of using a torque sensor attached to a revolute actuator, a local search deployed at range 0, would only find the revolute actuator (and thus, a torque sensor or a current sensor would be proposed); at range 1, the attached beam is found, and together with the beam's parameters like weight or acting force, the torque can be equationally derived. In Section IV-A, we evaluate the effectiveness of the local search and the number of Π group generated during the search.

While generating the Π groups, we observed that we get better results by having rather detailed models with more complex dimensions. The method from [34] tries to find the Π groups with the smallest exponents. If the model has multiple quantities with the same dimensions, they trivially cancel each other out.

This is another point against a large scope for the search; having several copies of the same element, e.g. two beams one on each side of the motor, adds spurious dimensionless groups.

Instead, we add more elements to the local search. For instance, our beam model includes the flexural rigidity property, which is described as $\mathsf{T}^{-2}\mathsf{L}^{3}\mathsf{M}$ increase the search space to include the Π groups we need.

Calibration: As explained in Section II-C, the model to convert sensor readings to observations of the state contains some unknown dimensionless constants. To find these values, we move the robot arm into configurations where we can measure the entire state space. With these measurements we estimate the constant terms using regression analysis. In our SCARA arm example, we move the system into an arbitrary position (here: lower beam vertical, upper beam horizontal) and attach a series of known weights. For each weight we record the beam deflection and how much current the motors consume. After the system is calibrated, the controller will be able to deduce the payload weight from sensor readings.

While the calibration adds some overhead to the method, it is likely to be needed anyway for two reasons. First, for modeling simplifications we ignore some terms, e.g. friction, due to the revolute actuators being equipped with gearboxes to increase the torque of the motors. This reduces their efficiency and by calibrating, the observation model recovers some of that information. Second, we may know that an element has a specific property, but we don't know its value. For instance, if we measure the load through deflection, it is necessary to know the flexural modulus. For pre-made parts, this figure is usually found in the accompanying data sheet, but for 3D printed parts it is not so easy to report an accurate figure, as printing conditions, technical parameters (printing temperature, layer adhesion and size, printer calibration, material, etc.) and printing parameters (infill, shell size) can significantly alter the characteristics of the part [42].

Runtime: After the calibration is done we have a complete observation model of the system. We use Newton which takes as input a description of the system's dynamics, the observation model and outputs an extended Kalman filter. The resulting filter is then connected to the sensors and to the controller.

IV. Evaluation

We show that our method works by building a complete system, including the physical realization and the sensor configuration. We show that the sensor configurations suggested by our method provides accurate measurements to the controller.

TABLE I: Parameters for the robot's components. Each parameter has the standard dimension associated with the parameter's name.

Element (# Parameters)	Parameters
Anchor (3)	position (x, y, z)
End effector (4)	position (x, y, z), force f
Revolute joint (1)	angle θ
Revolute actuator (2)	angle θ, current i
Revolute actuator with pulley (3)	angle θ, current i, radius r
Rigid beam (3)	width w, height h, length l
Flex beam (8)	width w, height h, length l, flexural rigidity EI, density ρ, deflection d, resistance u, voltage drop v
Pulley (4)	position (x, y, z), radius r

We also evaluate Algorithm 1 on three different robotic manipulator architectures. These experiments only evaluate the search, without the physical implementation of the robots. We show that sensor configurations can be efficiently generated and that our search heuristic effectively reduces the number of Π-groups to a manageable number. In this section, we use the term quantities and parameters interchangeably as the quantities are parameters of the design space for Mperl-Π. Constants like physical dimensions are only fixed at manufacturing time and they can "vary" during the exploration of the design space.

A. Search for Measurements

In this section, we show that our synthesis algorithm can effectively generate possible ways of measuring a goal quantity. The goal is given by its dimension and where we try to measure it. We evaluate the search on three manipulator architectures: The single SCARA manipulator as shown in Figure 1, a dual SCARA manipulator [43], and a CoreXY platform [44]. We give the number of quantities in the search space in Table I and II. Quantities include constant properties, e.g. the size of structural elements like beams, or variables like current consumption of an electric motor. Anchors are mounting points, used to attach the manipulator to the ground, the end effector is the element which effects the world; in this case, the part which carries an object. Beams are divided into rigid aluminum beams which do not deform under load, and 3D printed beams made of plastic which slightly deform under load. The flexible beams have a cavity where a flex sensor is inserted during printing. This flex sensor is fully enclosed and acts as a variable resistor, along with a Wheatstone bridge, which makes resistance and voltage drop parameters of the flexible beam.

For each manipulator, we show that our method can synthesize physically meaningful ways of measuring quantities of the system's state. We gather a set of 9 dimensions as goals to measure. The goals are acceleration, velocity, momentum, acceleration, their equivalents for rotation, torque, flexural modulus, and second area moment. For each goal, we try to take measurements at at various points in the system. For each manipulator and each goal, we search for physically valid Π groups which contains the goal, i.e., Π groups which can be used to derive the goal.

TABLE II: Components and the total number of quantities in the manipulators.

	Anchor	End effector	Revolute joint	Revolute actuator	Revolute actuator with pulley	Rigid beam	Flex beam	Pulley	$\sum$ Parameters
Single SCARA	1	1		2		1	1		22
Dual SCARA	2	1	3	2		2	2		41
CoreXY	2	1			2			8	48

TABLE III: Single SCARA arm, dimensional analysis summary. As starting node, we use the following numbering: (0) end effector, (1) wrist, (2) lower arm, (3) elbow actuator, (4) upper arm, (5) shoulder actuator.

Goal	Starting node	Range needed	Π groups searched	Time [s]
Velocity	0	1	93	0.049
	2	1	63	0.023
	4	1	54	0.024
Momentum	1	1	19	0.045
	3	3	278	0.398
	5	5	575	1.020
Acceleration	0	1	93	0.032
	2	1	63	0.063
	4	1	54	0.032
Angular momentum	1	1	75	0.482
	3	3	152	0.403
	5	5	385	1.202
Flexural modulus	2	2	148	0.243
Torque	3	2	261	0.560
	5	5	586	1.403

In Table III, IV, and V, we report some statistics about the search. The search uses iterative deepening and we report the value of the distance Δ in Algorithm 1, at which we find a valid Π group. Goals which can be found directly on the node where the search starts are omitted from the tables. As number of Π group searched we give the number of Π groups generated until we find a valid one. We only count unique Π groups by filtering out equivalent Π groups, e.g., groups where all the exponents are multiplied by -1. The time reported in the tables correspond to MPERL-Π running on a single core of an Intel i7-6920HQ (2.9Ghz) with Debian Linux.

We observe that the synthesis algorithm efficiently explores the search space of our manipulator designs even though the designs contain at least 20 and up to 48 parameters. The running time are low even when the search range span large portions of the systems. For larger distances, the number of Π groups considered can grow above 1000 but the local search usually keeps the number low.

B. End-to-end Case Study

We now present a more detailed end-to-end case study, including a physical prototype of the single SCARA system as shown in Figure 1. The arm's role is to perform pick-and-place tasks. While performing these tasks, it has to move different payloads, each with their own weight. The controller must ensure that the forces on the actuators stay within the operating ranges specified by the motors' respective datasheets. Exceeding these values would result in the arm collapsing in an uncontrolled way. Therefore, the controller needs a way of estimating the payload weight before deciding if it can reach the target location. In Figure 3, we give a visual representation of the arm's workspace and how it is affected by the load.

We describe using MPERL-Π from a user's perspective and we test two different sensor configurations found by our algorithm. First, we explain how the sensor configurations are derived and how the system is calibrated. Ensuing, we evaluate the measurement errors for the two configurations and show, that we can estimate the payload's mass with a reasonable accuracy. The root-mean-square error is around $5g$ for payloads ranging from $40g$ to $200g$.

TABLE IV: Dual SCARA arm, dimensional analysis summary. As the starting node, we use the following numbering: (0) end effector, (1) revolute joint middle, (2) beams (distal), (3) revolute joint, (4) beams (proximal), (5) revolute actuators to anchors. Due to the system's symmetry, the starting nodes are only in one half of the system.

Goal	Starting node	Range needed	Π groups searched	Time [s]
Velocity	0	1	93	0.083
	2	1	63	0.054
	4	1	54	0.053
Momentum	1	1	102	0.032
	3	3	531	0.036
	5	5	1667	1.356
Acceleration	0	1	93	0.045
	2	1	63	0.036
	4	1	54	0.060
Angular momentum	1	1	74	0.041
	3	3	361	0.683
	5	5	1054	1.210
Flexural modulus	2	2	242	0.398
Torque	3	2	504	0.705
	5	5	1382	1.225

TABLE V: CoreXY platform, dimensional analysis summary. As starting node, we use the following numbering: (0) revolute actuators, (1-4) pulleys. Due to the system's symmetry, the starting nodes are only in one half of the system.

Goal	Starting node	Range needed	Π groups searched	Time [s]
Velocity	0	4	1464	1.302
	1	3	662	0.892
	2	2	452	0.714
	3	1	272	0.352
Acceleration	0	4	1464	1.321
	1	3	662	0.912
	2	2	452	0.716
	3	1	272	0.420
Torque	0	4	1328	1.453
	1	3	568	1.032
	2	2	398	0.786
	3	1	244	0.596

The arm consists of two beams (shoulder and elbow), and two revolute actuators. The shoulder beam is solid, the elbow beam is allowed to flex and has a load cell embedded. The load cell is a resistor that changes with its deflection and is connected to a Wheatstone bridge for measuring the change in resistance (and therefore, the deflection). Both revolute actuators are brushed dc motors, equipped with 48 CPR quadrature encoders and gearboxes; in addition, their electric current and power are monitored by two INA219B sensors. Each actuator has a PID controller that follows the trajectory. The actuators and sensors are build as self contained units communicating with a central MPERL-Π controller via UART. The central controller computes trajectories for the overall system and dispatches commands to the actuator's controllers.

1) Measurement with the Euler-Bernoulli Beam Model: We describe how to use the flexural rigidity of the elbow beam by polling the system for its embedded sensors, creating the appropriate Π groups and calibrating the resulting equations by means of reference weights.

We start with a description of the SCARA system from [12] which already contains a flex sensor. In that work, the controller was simply adjusting the actuator's angle to compensate for the deflection, but this was not tied to any force. Now we use MPERL-Π to find how to use this sensor to compute the weight attached to the beam. As we do not know the equation, we use dimensional function analysis to synthesize suitable candidate equations. Starting with a local search of distance 1 from our beam, we get the following building parts: the beam itself, the effector, and the actuator. The beam has geometric properties (length, width, height), the sensor input (an angle) and sensor output (a resistance), and fabrication dependent parameters like the flexural rigidity and density. The effector applies a downward force which is proportional to the attached weight. Finally, the actuator's properties are its positions (angle), current, and torque.

From this generated description, we generate the Π groups; among them, we get the candidate $F^1(EI)^{-1}L^3d^{-1}$ where F is the force, L length of the beam, EI the flexural rigidity, and d the deflection. The Π group results in the tentative equation $C = \frac{FL^3}{EId}$.

It now remains to determine a value for C by calibration. We move the systems to a known pose where the two actuators have the angles $\theta_0 = \pi/4$ and $\theta_1 = -\pi/4$. These angles correspond to the pose where the shoulder beam is vertical and the elbow beam is horizontal. We attach known reference weights to the end effector and measure the deflection. To average out any noise or outliers, we repeat this step 10 times. Table VI summarizes the parameters and their values subject to different weights. Using these values we estimate that the constant C has the value 4.042.

2) Measurement with the Motor Current: Instead of relying on the flex sensor, we also have the possibility of deducing the weight by combining the current consumption of the revolute actuators, the length of the beams, and the known functional dependency between torque and power consumption from the motor's datasheet. Compared to the previous experiment, we add to the actuator specification a torque vs current diagram and perform the Π-group generation and calibration again. We now evaluate the accuracy of the sensor's predictions. Our model is idealized as it does not take the friction between the elements into account. Therefore, the system can stabilize at the same position but with different current consumption as long as the difference in torque output is smaller than the gearbox friction. For this experiment, we change the measurement method. The robot is put into a pose were it is possible for it to pull a precision spring scale. By pulling with varying intensity, the current consumption changes and can be used to draw conclusion about the force exerted (weight equivalent). In order to measure the error, the actuator pulls on the spring scale with similar intensity as when using reference weights and some intermediate values. Table VII and VIII reports the respective numbers. We observe that the model's simplification and change of setting results in small measurements errors. In the future, we plan to integrate a more realistic model of physics and try to derive confidence bounds on the measurements.

TABLE VI: Deflection measurements and calibration for reference weights. Each weight was lifted 10 times, the results are averaged and rounded.

Reference weight $[g]$	deflection $[mm]$	C
0	0.00	–
100	3.58	4.045
140	5.03	4.032
200	7.18	4.044
Average	–	4.042

TABLE VII: We measure current, derive force from torque/ampere curve, and compare it to actual force (measured via scale). The force columns has the weight equivalent of the force $[g]$ in parenthesis. The Root-Means-Square Error (RMSE) w.r.t. the weight in grams is 4.768.

Current $[A]$ (measured)	Force $[N]$ (calculated)	Force $[N]$ (measured)
0.20 A	0.39 (40)	0.33 (33.9)
0.31 A	0.59 (60)	0.52 (52.7)
0.46 A	0.78 (80)	0.77 (78.16)
0.60 A	0.98 (100)	0.99 (101.94)
0.69 A	1.18 (120)	1.15 (117.24)
0.85 A	1.47 (150)	1.42 (144.42)

TABLE VIII: We measure the deflection when lifting weights, derive force from the previously calibrated curve, and compare it to actual force (measured via scale). The force columns has the weight equivalent of the force $[g]$ in parenthesis. The RMSE is 6.770 w.r.t to weight in grams.

Deflection $[mm]$ (calculated)	Force $[N]$ (calculated)	Force $[N]$ (measured)
1.4223	0.39 (40)	0.49 (50)
2.1442	0.59 (60)	0.69 (70)
2.8262	0.78 (80)	0.83 (85)
3.5401	0.98 (100)	1.03 (105)
4.2498	1.18 (120)	1.18 (120)
5.2716	1.47 (150)	1.42 (145)

V. Related Work

Robotic hardware and software co-design: Mehta et al. work on robot creation from functional specification [10] is the closest to our work. The authors synthesize both a robot and its controller from a Structured English specification of the robot's task. The Structured English description is turned into Linear Temporal Logic (LTL) where the atomic propositions are actions of the robot. On the controller side, the authors use reactive synthesis to find a controller. On the physical side, the authors start with a common platform and add elements according to the actions: moving requires wheels or legs, grabbing an object requires a gripper, etc. They rely on a database mapping words to components, for both, actuators and sensors.

Campos et al. [13] work on design of modular manipulators to accomplish a given task. Given a task description with locations to reach and regions to avoid, their algorithm searches the space of physical design to produce a robot which accomplishes the task. At the same time they also produce the control configuration to move the manipulator. They use a controller based on local feedback similar to [12], and therefore, could also benefit from the method presented in this paper to handle a wider range of tasks' specification.

Dimensional types: On the software side, physical dimensions and units integrated into type systems have a long history [45], [46], [47]. These works use types to check that program are well-formed and use numbers in a dimensionally correct way. In this work we use the type information for synthesis.

Types for synthesis: Type information has also been used for code completion and synthesis [48], [49]. The synthesis engine searches for well-typed expressions to fill gaps within a program. Using richer typing systems [50], e.g. inductive datatypes and refinement types, it becomes possible to synthesize whole methods like sorting algorithms. Newton [34] is the only work we know that uses physical information in the type system for synthesis. We improve over Newton by having a goal directed search over the system's components with the goals initially coming from the need of a controller.

Controller synthesis with partial information: In this work, the controller is synthesized a priori without restriction. This may lead to the controller using more information than strictly necessary. The work on knowledge-based abstraction and partial information games [51], [52], [32] could be adapted to help us by finding the controller requiring the least amount of information. Or we could try, for a given set of sensors, to find a controller which optimizes the information provided by the sensors [53].

VI. Conclusion and Future Work

Inspired by the advances in controller synthesis and computational fabrication of robotic systems, we have shown how to apply dimensional analysis to a robot's physical design to meet the requirements of an automatically synthesized controller. We have applied our method to the design of a robotic arm and shown that it can find two different sensor configurations and that these sensors provide enough information to estimate the system's state.

In the future, we will work on better filtering techniques to reduce the number of spurious Π groups generated during the dimensional analysis. This will also allow better scalability, especially for larger systems. We use a correct by construction approach for the controller, but we do not consider measurement precision during the search for sensors; also, the state estimation does not provide any guarantees. We will also investigate the possibilities of obtaining end-to-end correctness guarantees about the system.

Furthermore, our method currently is sequential; once a step is finished, the choices made during that step will not be changed again. We plan to extend our method with a feed back loop, which allows for returning information about the available sensors to the controller synthesis. For instance, if we fail to find sensors for a quantity, the controller synthesis would backtrack and try to find a controller which does not depend on that quantity.

Acknowledgment

Marcus Pirron and Damien Zufferey were funded in part by the Deutsche Forschungsgemeinschaft project 389792660-TRR 248 and by the European Research Council under the Grant Agreement 610150 (ERC Synergy Grant ImPACT). Phillip Stanley-Marbell is supported by an Alan Turing Institute award TU/B/000096 under EPSRC grant EP/N510129/1 and by EPSRC grant EP/V004654/1.

References

[1] D. Mayne, J. Rawlings, C. Rao, and P. Scokaert, "Constrained model predictive control: Stability and optimality," *Automatica*, vol. 36, no. 6, pp. 789 – 814, 2000.

[2] P. Tabuada, *Verification and Control of Hybrid Systems - A Symbolic Approach*. Springer, 2009.

[3] L. Zhu, W. Xu, J. Snyder, Y. Liu, G. Wang, and B. Guo, "Motion-guided mechanical toy modeling," *ACM Trans. Graph.*, vol. 31, no. 6, pp. 127:1–127:10, 2012.

[4] B. Thomaszewski, S. Coros, D. Gauge, V. Megaro, E. Grinspun, and M. H. Gross, "Computational design of linkage-based characters," *ACM Trans. Graph.*, vol. 33, no. 4, pp. 64:1–64:9, 2014.

[5] A. M. Mehta, J. DelPreto, B. Shaya, and D. Rus, "Cogeneration of mechanical, electrical, and software designs for printable robots from structural specifications," in *2014 IEEE/RSJ International Conference on Intelligent Robots and Systems, Chicago, IL, USA, September 14-18, 2014*. IEEE, 2014, pp. 2892–2897.

[6] A. M. Mehta, J. DelPreto, and D. Rus, "Integrated codesign of printable robots," *J. Mechanisms Robotics*, vol. 7, no. 2, 2015.

[7] V. Megaro, B. Thomaszewski, M. Nitti, O. Hilliges, M. H. Gross, and S. Coros, "Interactive design of 3d-printable robotic creatures," *ACM Trans. Graph.*, vol. 34, no. 6, pp. 216:1–216:9, 2015.

[8] N. Bezzo, A. M. Mehta, C. D. Onal, and M. T. Tolley, "Robot makers: The future of digital rapid design and fabrication of robots," *IEEE Robot. Automat. Mag.*, vol. 22, no. 4, pp. 27–36, 2015.

[9] A. Schulz, C. R. Sung, A. Spielberg, W. Zhao, R. Cheng, E. Grinspun, D. Rus, and W. Matusik, "Interactive robogami: An end-to-end system for design of robots with ground locomotion," *I. J. Robotics Res.*, vol. 36, no. 10, pp. 1131–1147, 2017.

[10] A. M. Mehta, J. DelPreto, K. W. Wong, S. Hamill, H. Kress-Gazit, and D. Rus, "Robot creation from functional specifications," in *Robotics Research, Proceedings of the 17th International Symposium of Robotics Research, ISRR 2015, Sestri Levante, Italy, September 12-15, 2015, Volume 2*, ser. Springer Proceedings in Advanced Robotics, A. Bicchi and W. Burgard, Eds., vol. 3. Springer, 2015, pp. 631–648.

[11] S. Ha, S. Coros, A. Alspach, J. M. Bern, J. Kim, and K. Yamane, "Computational design of robotic devices from high-level motion specifications," *IEEE Trans. Robotics*, vol. 34, no. 5, pp. 1240–1251, 2018.

[12] M. Pirron and D. Zufferey, "MPERL: hardware and software co-design for robotic manipulators," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. IEEE, 2019, pp. 7784–7790.

[13] T. Campos, J. P. Inala, A. Solar-Lezama, and H. Kress-Gazit, "Task-based design of ad-hoc modular manipulators," in *International Conference on Robotics and Automation, ICRA 2019, Montreal, QC, Canada, May 20-24, 2019*. IEEE, 2019, pp. 6058–6064.

[14] P. Stanley-Marbell, D. Marculescu, R. Marculescu, and P. K. Khosla, "Modeling computational, sensing, and actuation surfaces," *Low-Power Processors and Systems on Chips*, pp. 16–1, 2005.

[15] J. T. Muth, D. M. Vogt, R. L. Truby, Y. Mengüç, D. B. Kolesky, R. J. Wood, and J. A. Lewis, "Embedded 3d printing of strain sensors within highly stretchable elastomers," *Advanced Materials*, vol. 26, no. 36, pp. 6307–6312, 2014.

[16] J. A. Lewis and B. Y. Ahn, "Three-dimensional printed electronics," *Nature*, vol. 518, no. 7537, pp. 42–43, Feb 2015.

[17] M. A. McEvoy and N. Correll, "Materials that couple sensing, actuation, computation, and communication," *Science*, vol. 347, no. 6228, 2015.

[18] W. Yan, Y. Yu, and A. Mehta, "Rapid design of mechanical logic based on quasi-static electromechanical modeling," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems, IROS 2019, Macau, SAR, China, November 3-8, 2019*. IEEE, 2019, pp. 5820–5825.

[19] J. Lim and P. Stanley-Marbell, "Newton: A language for describing physics," *CoRR*, vol. abs/1811.04626, 2018.

[20] O. Kaparounakis, V. Tsoutsouras, D. Soudris, and P. Stanley-Marbell, "Automated physics-derived code generation for sensor fusion and state estimation," *CoRR*, vol. abs/2004.13873, 2020. [Online]. Available: https://arxiv.org/abs/2004.13873

[21] E. Buckingham, "On physically similar systems; illustrations of the use of dimensional equations," *Phys. Rev.*, vol. 4, pp. 345–376, Oct 1914.

[22] S. Qin and T. A. Badgwell, "A survey of industrial model predictive control technology," *Control Engineering Practice*, vol. 11, no. 7, pp. 733 – 764, 2003.

[23] A. Bemporad and M. Morari, "Robust model predictive control: A survey," in *Robustness in identification and control*, A. Garulli and A. Tesi, Eds. London: Springer London, 1999, pp. 207–226.

[24] C. Belta, B. Yordanov, and E. A. Gol, *Formal Methods for Discrete-Time Dynamical Systems*. Springer, 2017.

[25] A. Girard, "Controller synthesis for safety and reachability via approximate bisimulation," *Automatica*, vol. 48, no. 5, pp. 947–953, 2012.

[26] G. Reissig, A. Weber, and M. Rungger, "Feedback refinement relations for the synthesis of symbolic controllers," *TAC*, vol. 62, no. 4, pp. 1781–1796, 2017.

[27] M. Rungger and M. Zamani, "SCOTS: A tool for the synthesis of symbolic controllers," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control, HSCC 2016, Vienna, Austria, April 12-14, 2016*, A. Abate and G. E. Fainekos, Eds. ACM, 2016, pp. 99–104.

[28] K. Hsu, R. Majumdar, K. Mallik, and A. Schmuck, "Multi-layered abstraction-based controller synthesis for continuous-time systems," in *Proceedings of the 21st International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC 2018, Porto, Portugal, April 11-13, 2018*, M. Prandini and J. V. Deshmukh, Eds. ACM, 2018, pp. 120–129.

[29] M. Khaled and M. Zamani, "pfaces: an acceleration ecosystem for symbolic control," in *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2019, Montreal, QC, Canada, April 16-18, 2019*, N. Ozay and P. Prabhakar, Eds. ACM, 2019, pp. 252–257.

[30] J. S. Shamma and K. Tu, "Set-valued observers and optimal disturbance rejection," *IEEE Trans. Automat. Contr.*, vol. 44, no. 2, pp. 253–264, 1999.

[31] O. Mickelin, N. Ozay, and R. M. Murray, "Synthesis of correct-by-construction control protocols for hybrid systems using partial state information," in *American Control Conference, ACC 2014, Portland, OR, USA, June 4-6, 2014*. IEEE, 2014, pp. 2305–2311.

[34] Y. Wang, S. Willis, V. Tsoutsouras, and P. Stanley-Marbell, "Deriving equations from sensor data using dimensional function synthesis," *ACM Trans. Embedded Comput. Syst.*, vol. 18, no. 5s, pp. 84:1–84:22, 2019.

[32] R. Majumdar, N. Ozay, and A. Schmuck, "On abstraction-based controller design with output feedback," *CoRR*, vol. abs/2002.02687, 2020.

[33] *The International System of Units (SI)*, 9th ed. Bureau International des Poids et Mesures, 2019. [Online]. Available: https://www.bipm.org/en/publications/si-brochure/

[35] G. L. Smith, L. A. M. Gee, and S. F. Schmidt, "Application of statistical filter theory to the optimal estimation of position and velocity on board a circumlunar vehicle," National Aeronautics and Space Administration, Tech. Rep. R-135, 1962.

[36] J. M. Gere and B. J. Goodno, *Mechanics of Materials*, 9th ed. Cengage Learning, 2018.

[37] M. A. Fischler and R. C. Bolles, "Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography," *Commun. ACM*, vol. 24, no. 6, pp. 381–395, 1981.

[38] R. R. Brooks and S. S. Iyengar, "Robust distributed computing and sensing algorithm," *IEEE Computer*, vol. 29, no. 6, pp. 53–60, 1996.

[39] G. Alefeld and G. Mayer, "Interval analysis: theory and applications," *Journal of Computational and Applied Mathematics*, vol. 121, no. 1, pp. 421 – 464, 2000.

[40] M. C. F. Donkers and W. P. M. H. Heemels, "Output-based event-triggered control with guaranteed $\updownarrow_\infty$-gain and improved and decentralized event-triggering," *IEEE Transactions on Automatic Control*, vol. 57, no. 6, pp. 1362–1376, 2012.

[41] R. D. Cook, *Finite Element Modeling for Stress Analysis*, 1st ed. USA: John Wiley & Sons, Inc., 1994.

[42] G. Ćwikła, C. Grabowik, K. Kalinowski, I. Paprocka, and P. Ociepka, "The influence of printing parameters on selected mechanical properties of fdm/fff 3d-printed parts," *IOP Conference Series: Materials Science and Engineering*, vol. 227, p. 012033, 08 2017.

[43] H. Makino, "Assembly robot," U.S. Patent US4 341 502A, 1979.

[44] I. E. Moyer, "Corexy cartesian motion platform," https://corexy.com/index.html, 2012, [Online; accessed 15-Apr-2020].

[45] M. Babout, H. Sidhoum, and L. Frecon, "AMPERE: a programming language for physics," *European Journal of Physics*, vol. 11, no. 3, pp. 163–171, may 1990.

[46] A. Kennedy, "Dimension types," in *Programming Languages and Systems - ESOP'94, 5th European Symposium on Programming, Edinburgh, UK, April 11-13, 1994, Proceedings*, ser. Lecture Notes in Computer Science, D. Sannella, Ed., vol. 788. Springer, 1994, pp. 348–362.

[47] Z. D. Umrigar, "Fully static dimensional analysis with C++," *SIGPLAN Notices*, vol. 29, no. 9, pp. 135–139, 1994.

[48] D. Mandelin, L. Xu, R. Bodík, and D. Kimelman, "Jungloid mining: helping to navigate the API jungle," in *Proceedings of the ACM SIGPLAN 2005 Conference on Programming Language Design and Implementation, Chicago, IL, USA, June 12-15, 2005*, V. Sarkar and M. W. Hall, Eds. ACM, 2005, pp. 48–61.

[49] T. Gvero, V. Kuncak, I. Kuraj, and R. Piskac, "Complete completion using types and weights," in *ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI '13, Seattle, WA, USA, June 16-19, 2013*, H. Boehm and C. Flanagan, Eds. ACM, 2013, pp. 27–38.

[50] N. Polikarpova, I. Kuraj, and A. Solar-Lezama, "Program synthesis from polymorphic refinement types," in *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2016, Santa Barbara, CA, USA, June 13-17, 2016*, C. Krintz and E. Berger, Eds. ACM, 2016, pp. 522–538.

[51] D. Lee and M. Yannakakis, "Online minimization of transition systems (extended abstract)," in *Proceedings of the 24th Annual ACM Symposium on Theory of Computing, May 4-6, 1992, Victoria, British Columbia, Canada*, S. R. Kosaraju, M. Fellows, A. Wigderson, and J. A. Ellis, Eds. ACM, 1992, pp. 264–274.

[52] J. Raskin, K. Chatterjee, L. Doyen, and T. A. Henzinger, "Algorithms for omega-regular games with imperfect information," *Logical Methods in Computer Science*, vol. 3, no. 3, 2007.

[53] S. Ponda, R. Kolacinski, and E. Frazzoli, *Trajectory Optimization for Target Localization Using Small Unmanned Aerial Vehicles*. American of Institute Aeronautics and Astronautics, 2006.