Robovis: Parameter Space Exploration in Robotic Design

Alistair Wick

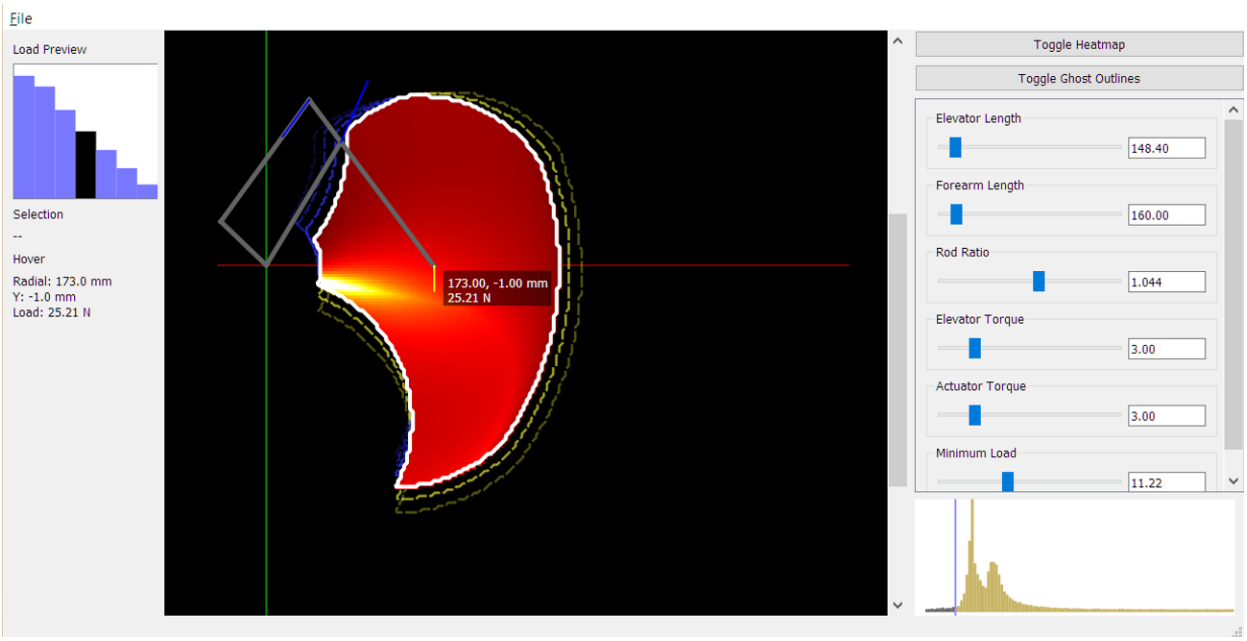

Fig. 1. The Robovis application window

Abstract—Robotic arms can be defined by a set of parameters such as their dimensions and power: Robovis is a visual parameter-space exploration tool meant to help end users find settings for these parameters which result in a robot arm that is useful for their purposes.

Index Terms—Visualization, Robotics, CPSC547

1 Introduction

Robotics applications are numerous and varied, but the design process for a robotic arm is esoteric, not accessible to the many potential users who lack the specialist skills necessary for this work. Robovis is a tool designed as the user-facing component of a larger toolchain which would partially automate this design process. It allows the user to explore the design space for a robotic arm, varying aspects of the arm's configuration such as the length of different components and the torques of the different motors, and showing them how the resulting robot arm will perform. In this prototype tool, the performance metrics of the arm are limited to the shape of the area the end effector can reach (the arm's "workspace") and the load it can carry. In an attempt to enhance the interactive iteration of the configuration, Robovis takes a novel approach to the display of the workspace: it displays the workspace of both the current configuration and several "nearby" configurations, with the aim of guiding the user towards configurations which suit their needs.

As far as I am aware, there is no other comparable tool for visually exploring the design space of a robotic arm. The work of figuring out a bespoke arm's design is typically done as-needed for each new robotics project, which may hobble uptake of robotic manipulators outside of research and high-end automation applications. The modeling of an arm and its capabilities may even be performed post-hoc, after a manipulator is bought or built: an immediately relevant example of such a case is the control application I developed for my own robotics project, from which RoboVis stems. This work was unpublished, but a screenshot of the workspace visualization is shown in Fig. 2; it is from this project that Robovis was born.

• *Alistair is a graduate student at the University of British Columbia. E-mail: alistair.wk@gmail.com.*

2 Related Work

A published example of this type of work is a project by Johari et al. [2], where they model and simulate a palletizing robot with similar mechanics to the design modelled by Robovis. They created a 3D visualization of the arm's workspace, shown in figure 7 of the paper. Their approach certainly allows exploration of the design space of an arm, but is not suited to this task: in much the same way as I was able to test different configurations using the PyIK control tool for my robot arm, their modelling application can be loaded with different configurations to see the results; however, as with my application, this is an extremely slow process, as the software must be reloaded after each change to the configuration.

In a project by Hao et al. to develop a 6-dof arm and the associated control software [1], we see an approach with 3D visualizations of the resulting arm, in custom software dubbed *SMART ARM*. They briefly discuss optimizing the design of the arm for a given task, including consideration of its reachable volume, but do not consider a targeted visualization or interaction approach as in Robovis, instead relying on the slow configuration iteration process described above.

Essentially, while robotic simulation tools abound, there are no exist-

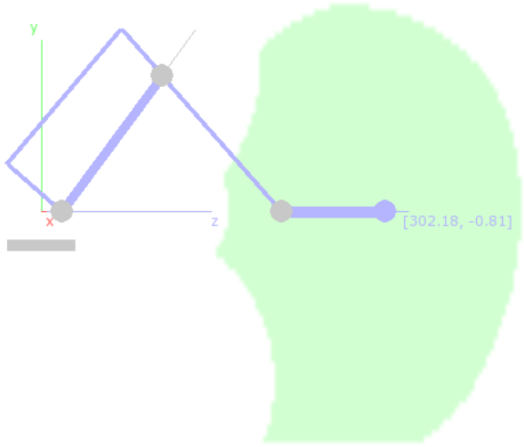

Fig. 2. Screenshot of PyIK, a bespoke robotic control application I developed for *EvoArm*, showing the workspace visualization in green

ing tools which are explicitly designed for the exploration of different possible configurations of robotic arms; rather, these tools are typically designed to explore the usage and control of *existing* designs, or are limited to testing the feasibility of a new design (rather than helping the user come up with one which is suitable).

2.1 Robotic Manipulator Analysis

Knowing precisely where a robot manipulator can reach has been an important problem in robotics for some time. In *Design Considerations for Manipulator Workspace* [7], Roth et al. explore ways of analytically determining the workspace of an arm, including high degree-of-freedom designs: those with redundant axes, which can reach not just any point but any orientation for their end effector. The basic approach taken in this paper involves taking the workspace of a point on the end of one axis, and revolving it around the previous axis, repeating for all axes in the design:

$$R_k[W_{k+1}(P)] = W_k(P)$$

For example, in the basic case of a point at the end of two links, the point is able to revolve in a circle around the second axis, and this circle is revolved around the first axis to obtain a toroidal surface reachable by the point. Adding additional degrees of freedom them forms increasingly complex 3-dimensional volumes, rather than simple surfaces. The presentation of the overall workspace used in this paper is quite similar to the main view employed in Robovis: a cross-sectional view showing the reachable area for some axes of the robot (shown in figure 6 of the paper).

One of the more interesting aspects of this analysis is the distinction drawn between the workspace in which the end effector can be placed, and the workspace in which it can be placed in any orientation. Roth et al. refer to the different workspaces as the *primary* workspace $W^p(P)$, where each point is reachable from all orientations, and the *secondary* workspace $W^s(P)$, where each point is reachable with only a limited subset of all orientations (or only one). This is an aspect of the design problem I do not deal with in Robovis: I ignore end-effector orientation, and indeed only consider the two primary axes, leaving the third, revolute axis up to the user's imagination. I am also able to ignore the case of "voids" being present in the workspace, as this is not possible with the mechanics of the design under consideration in my application.

While the framework Roth et al. present is more general than my approach, extending to arbitrary DOF and designs, it is also more idealized: it does not appear to take into account limits on the joint angles, and does not deal with other mechanical constraints such as self-interference. These are a focus for Robovis: my approach tests for reachable points using a full IK solver which checks any number of specified constraints.

2.2 Vis Design

I employ a number of fairly standard viz idioms in my tool, including scented widgets, first fully described by Willett et al. [9]. These are interactive widgets, such as sliders or input boxes, which have small visualizations attached or embedded within them. They typically outperform conventional (un-scented) user interface widgets, as they provide some indication as to the structure or composition of the data under interaction. This allows users to quickly explore data by guiding their search towards areas of interest, or away from invalid or uninteresting inputs, without the need to first "try them out" to see the full results. In my design for Robovis, a scented slider is used to show a histogram of the manipulator's load capacity over its workspace. The use of a scented slider here provides additional contextual information over a standard slider, allowing the user to quickly see how much of the overall range they will be removing by raising the minimum allowable load.

Robovis displays the shape of a robot arm configuration's reachable workspace as its primary visual cue, and it is this shape which is the user's main consideration when accepting or rejecting a configuration. In the course of using the application, the user is presented with a large number of these shape renderings. A common approach to showing a large number of closely related outputs is the small-multiple, or faceted display idiom. This type of approach can be seen in Keefe at al. [3], where ˜100 pig jaw profiles (and the outlines of their motions) are visible and selectable at once, or in Marks et al. [5], where thumbnails of CGI renders with varying parameters are displayed side-by-side. On the surface this type of display may appear a reasonable approach for Robovis, but I intentionally avoid small-multiple displays. I believe that the user must be able to understand the exact shape and scale of the reachable area: in fact, this is essential to the task. However, these workspace shapes vary widely in size and shape in ways which are irregular and unpredictable, and do not have a fixed underlying structure which would aid visual comparisons. These problems are compounded when the viewing area is constricted, and so I feel that the small-multiples view is inappropriate.

2.3 Parameter Space Exploration

While my specific application is novel, the idea of using viz to explore parameter spaces is far from new. Sedlmair et al. [8] discuss a framework for approaching this vast research space, and while their discussion focuses somewhat on avoiding the complications of computationally costly sampling (not a major problem for my tool), their insights have certainly been applicable.

My general approach to interaction in Robovis is a take on the "informed trial and error" and "local-to-global" idioms discussed in this paper. Essentially, interaction in Robovis revolves around the iterative change of a "current" configuration, with the goal of getting closer to (and eventually meeting) the goal requirements. This corresponds to the informed trial-and-error approach, but I also mix in aspects of the local-to-global approach, by sampling around the current configuration and displaying the results of these samples to the user. These results are then made selectable, so that the user may "jump" to those configurations which are more in line with their requirements. In my tool, this approach is not borne of the need to avoid real-time sampling – samples for my tool take a fraction of a second to compute. Rather, the reasoning is along the same lines as for scented widgets: the nearby results provide the "scent", guiding the user towards solutions which meet their requirements by helping to inform the process of trial-and-error refinement.

3 Data and Task Abstractions

Robovis operates on a two main sets of data: the configurations, and the IK results, with the latter generated from the former. Importantly, both data sets are *dynamic* – they change, often frequently, as the user operates the tool.

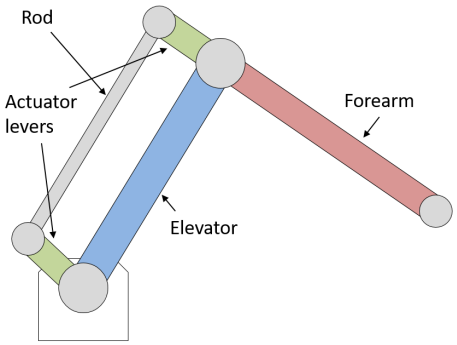

Fig. 3. Basic layout of the arm

Attribute	Description
Elevator length	First section of the arm – see Fig. 3
Forearm length	Length of the second section of the arm, to which the end effector is attached
Rod ratio	Ratio of length of the actuator rod to the elevator; affects leverage
Elevator torque	Maximum torque available to drive the elevator; limits the maximum load, and should include any gearing of the drive mechanism
Actuator torque	Maximum torque available to drive the actuator, which indirectly drives the forearm; also limits the maximum load

Table 1. Adjustable attributes in the Robovis configuration, where every attribute is a continuous, quantitative, sequential value.

3.1 Configuration Data

Assuming the basic layout shown in Fig. 3 is adhered to, the configuration of the arm is a set of parameters which together fully define the arm's physical properties, size, and performance. These include important aspects of an arm's design like the respective lengths of the different rigid sections, and the amount of torque (rotational power) available to the drive systems for the elevator and actuator portions of the arm. These properties are defined in real-world units: millimeters and Newton-meters.

A subset of five of these properties are modifiable in Robovis. This *adjustable* configuration is described in Table 1. While not a focus for direct visualization in Robovis, a configuration fully defines a solution, and so in some sense the rest of the data in the application – that which is actually displayed – is derived from the configuration data. Changes to the current configuration are at the root of the dynamism of the dataset: multiple successive changes to the current configuration can (and do) trigger total regeneration of all data in the application.

The configuration space – the space to be explored, in which all possible configurations exist – is essentially a hyperdimensional field, where any point is itself a multi-value continuous 2D field, describing the reach and performance of the arm corresponding to the configuration values which index said point. This space is discretized in a course sampling, extending outwards from the *current* configuration. The current configuration can be freely adjusted, and is initialized with a known "seed" value for further exploration.

Additional, fixed attributes which might nevertheless be considered part of the configuration include aspects of the design such as angular limits, for example between different sections of the arm, or on the drive system. These are baked into the solver, and are not made visible to the user in the presented version of Robovis. Many more explicit attributes are possible, and would not represent a significant challenge algorithmically; they would simply be new, or newly adjustable constraints in the solver.

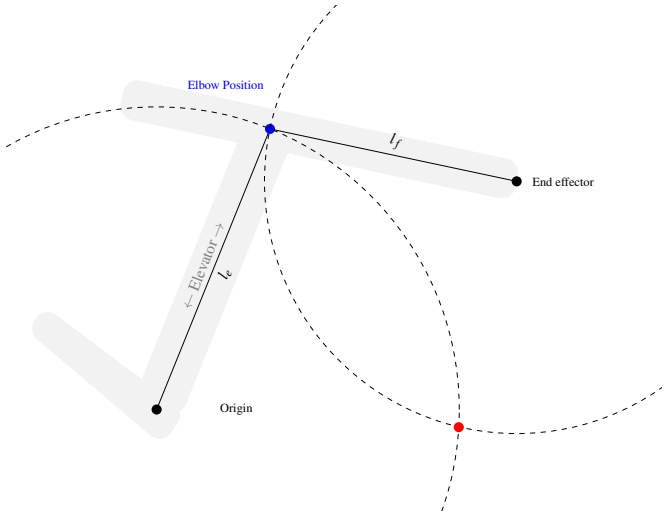

Fig. 4. Inverse Kinematics: Locating the elbow joint using the intersections of circles around the origin and goal points

3.2 Inverse Kinematics & Load Calculations

The actual process by which a configuration is transformed into workspace data is not a data set: rather, it is a data *model*, consisting of calculations which treat the configuration data as a description of a physical arm. I have chosen to include these details here as this process is arguably what the application is truly visualizing for the user; the rest of the data are merely inputs and outputs.

Inverse Kinematics (IK) is a process where a pose for a kinematic chain is found, satisfying some goal position and/or orientation of an "end effector" at the end of the kinematic chain. The IK in Robovis is adapted from the routines used in PyIK.

With the first degree of freedom of the arm ignored, the remaining two (motion of the elevator and actuator/forearm assembly) are considered within a 2D plane. The basic approach used is to analyze two circles radiating from the origin and from the goal point, with radii equal to the respective section lengths. If these circles do not intersect, no solution is possible: the goal is too far from the origin. If they do intersect, then at least one intersection may be found, corresponding to the position of the "elbow" joint between the elevator and forearm – in the case of two intersections, the higher intersection is chosen as the elbow point (Fig. 4).

With the elevator length as l_e, the forearm length as l_f, goal point G, and origin O at $(0,0)$, the solution point I for the elbow position is described in Equation 1.

$$
\begin{aligned}
a &= \frac{l_f^2 - l_e^2 + ||G||^2}{||G||^2} \\
I_1 &= \begin{pmatrix} G - ||G||^{-1}(G \cdot a - \sqrt{l_f^2 - a^2} \cdot G_y) \\ G - ||G||^{-1}(G \cdot a + \sqrt{l_f^2 - a^2} \cdot G_x) \end{pmatrix} \\
I_2 &= \begin{pmatrix} G - ||G||^{-1}(G \cdot a + \sqrt{l_f^2 - a^2} \cdot G_y) \\ G - ||G||^{-1}(G \cdot a - \sqrt{l_f^2 - a^2} \cdot G_x) \end{pmatrix} \\
I &= \begin{cases} I_1, & \text{if } I_{1_y} > I_{2_y} \\ I_2, & \text{otherwise} \end{cases}
\end{aligned}
\tag{1}
$$

The angle α between the elevator and elbow is then:

$$
\alpha = \arccos\left(\frac{V_e \cdot V_f}{||V_e||||V_f||}\right) \tag{2}
$$

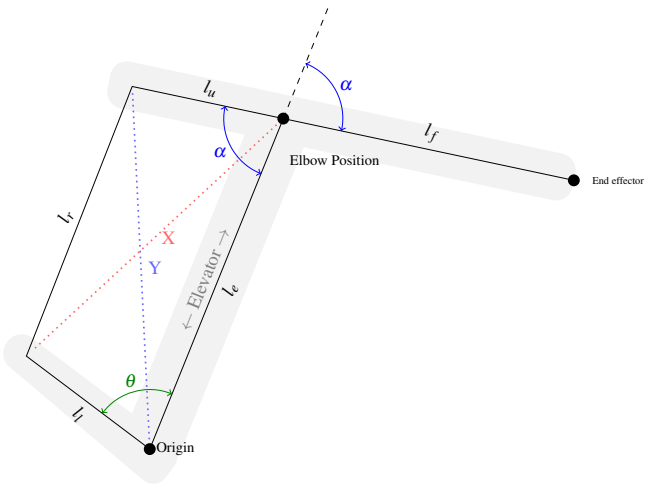

Fig. 5. Inverse Kinematics: Finding the interior angles of the arm's actuation mechanism

Once the elbow point is known, the interior angles of the actuation mechanism (the lever assembly at the rear of the arm which drives the forearm) can be found by applying the cosine rule. This is necessary to calculate the position of the lower actuator lever, which is important in the load calculations.

$$\begin{aligned} Y &= \sqrt{l_e^2 + l_u^2 - 2 l_e l_u \cos(\alpha)} \\ \theta &= \arccos\left(\frac{Y^2 + l_l^2 - l_r^2}{2 Y l_l}\right) + \arccos\left(\frac{Y^2 + l_e^2 - l_u^2}{2 Y l_e}\right) \end{aligned} \tag{3}$$

Once the pose of the arm is known for a given point: with θ_f as the angle of the forearm from vertical, θ_a as the angle of the actuator from vertical, the maximum load when considering the actuator motor may be calculated as follows:

$$\begin{aligned} \theta &= \theta_f - \frac{\pi}{2} \\ w &= \frac{l_f \cos\theta}{l_u \cos\alpha} \\ A &= \begin{pmatrix} \sin\gamma \\ \cos\gamma \end{pmatrix} \\ l_{\text{actuator}} &= \left| \frac{T_{\text{actuator}}}{l_l w \left(\cos(\theta+\alpha) A_x - \sin(\theta+\alpha) A_y\right)} \right| \end{aligned} \tag{4}$$

The maximum load when considering the elevator motor is then:

$$\begin{aligned} E &= \begin{pmatrix} \sin\theta_e \\ \cos\theta_e \end{pmatrix} \\ z &= E_y w \sin(\theta+\alpha) - E_X \left(w \cos(\theta+\alpha) + 1\right) \\ l_{\text{elevator}} &= \left| \frac{T_{\text{elevator}}}{l_e z} \right| \end{aligned} \tag{5}$$

3.2.1 Workspace Data

Strictly speaking, a robot arm's workspace is the shape in 3D space in which it is able to place its end effector: the space mapped out by its range of motion, as in Fig. 6. In Robovis, I extend the term to refer to the area in which an arm is able to place its end effector while also meeting additional constraints; in the presented tool, these constraints are on the load that the arm is able to support, meaning that an increase in the minimum allowable load may decrease the workspace. In Robovis, visual analysis of the workspace is the fundamental technique by which users evaluate a potential design, and so this data is a key component of the overall tool.

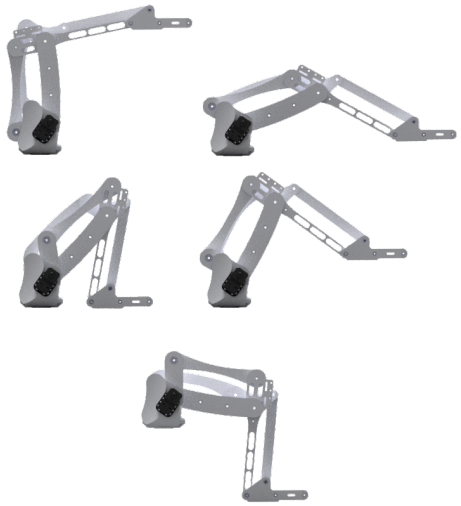

Fig. 6. Motion capabilities of the basic palletizing robot design under consideration. Images are of my robot arm design *EvoArm*, rendered using CAD software.

The raw workspace data takes the form of a grid of binary categorical samples, with square cells of uniform separation. Each cell indicates whether the configuration under consideration results in an arm which is able to place its end effector at the sample point – whether a valid IK solution could be found for the point.

This coarse sampling maps out a region of space – a shape, or several shapes – and so, with no additional information contained within this data, it was natural to derive polygonal shapes from the grid data. These are contours, wrapping around the positive regions of the grid, approximating the boundary between the arm meeting or failing the specified physical constraints.

3.3 Task

The main task is to find a configuration which matches the user's requirements: following the task abstraction guidelines in *Visualization Analysis and Design* [6], this is a parameter-space search/explore task, since neither the "location" (configuration) nor the *exact* target (arm capabilities) are known ahead of time. Notably, the hyperdimensional nature of the configuration space, in combination with the multidimensional, shape-centric output, makes providing an overview of the exploration space difficult. What is the target of the search? Since the space is continuous, any viable user requirements for an arm will typically yield an infinite number of configurations which are acceptable; however, the target of the search is simply any single configuration which fulfills the requirements, not the entire set.

The tool might also find related uses, outside of this core task: for example, one might wish to explore the space to discover the limits of practical designs.

On the spectrum of "specific" to "general" vis tools, this is fairly specific: it aims to solve one problem well.

4 Solution

The user interacts with the application (Fig. 1) by modifying a "current" configuration in an iterative fashion, using the coupled sliders and value-boxes in the parameter panel on the right hand side of the display. The user's choices are guided by various visualization elements in the user interface: the main workspace view, ghost outlines and interactive inspectors in the center of the display, the load histogram/slider in the bottom-right, and the load preview bars in the top-left.

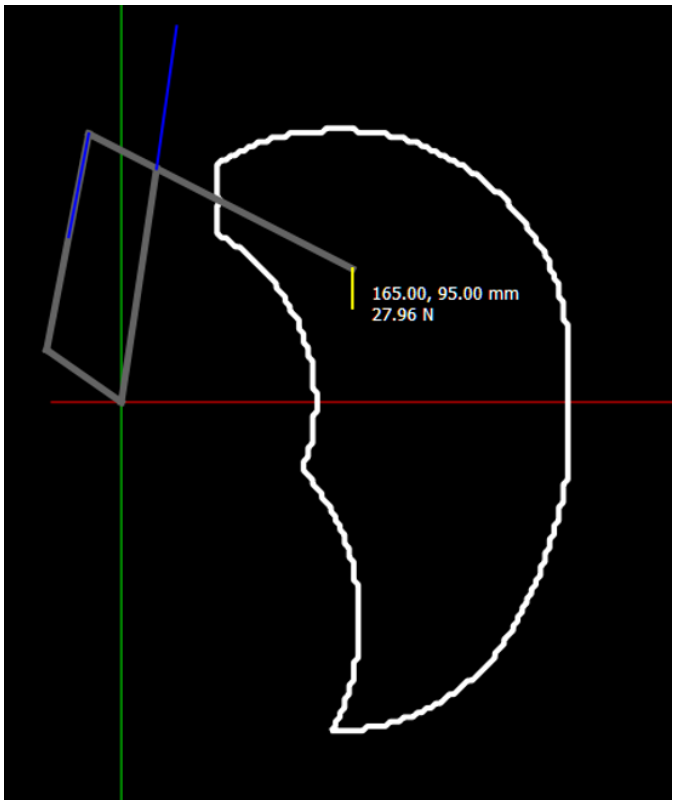

Fig. 7. Simple outline view of a workspace, with a visualization of the robot arm itself also overlaid (this appears while hovering the mouse over the workspace)

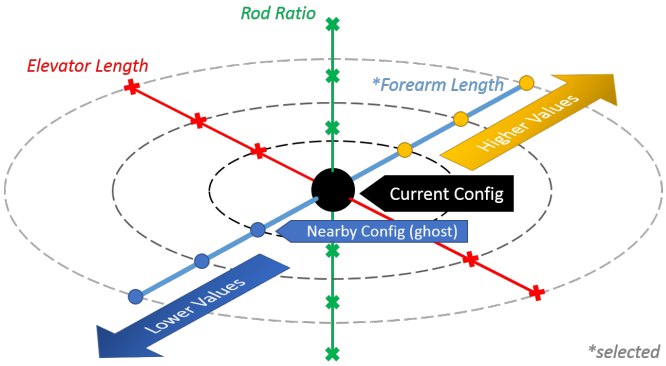

Fig. 8. A reduced 3D representation of the many possible dimensions for the configuration space for the arm; this space extends in all directions from the current configuration, and any point in the space may represent a solution for the user. Robovis samples nearby points along a single *selected* parameter, and displays the results of these samples as "Ghosts" alongside the main display.

4.1 Workspace View

The robot arm's workspace is represented by an outline view in 2D (Fig. 7) – this shows a slice of the overall volume in which the arm is able to work, in any plane which passes through the axis of rotation for the arm's base rotation motion (the 'first' of the three degrees of freedom). The motion of this first axis simply revolves the planar workspace into a roughly toroidal volume, in much the same way as described in Sect. 2.1. It is assumed that this axis aligns with gravity, and so the rotary position on this axis has no bearing on the shape of the workspace, even taking into account effects such as the minimum allowable load capacity. It is therefore of little relevance in the presentation of the workspace slice, and is ignored in Robovis' presentation.

In Robovis, a workspace essentially consists of a field of binary point samples, indicating whether or not each point is reachable by a particular configuration of the arm. The number of samples is adjustable in code, but not exposed to the user at runtime; as presented, Robovis uses a 100×200 grid, or 20,000 individual samples per workspace. One or more contours (there may be multiple 'islands' in the data) are then generated from the sample grid, wrapping around the contiguous areas within which valid IK solutions can be found.

I chose to use outlines over a filled display to enable the concurrent use of a heatmap overlay, and to allow multiple workspaces to be visualized in a collocated pattern.

4.2 Ghosts

Robovis uses a novel idiom dubbed "ghost outlines" to display nearby samples of the configuration space. In this hyperdimensional space, each parameter of the configuration is an axis, with a single parameter axis being the 'current' axis. Six sample configurations are taken at steps along this axis, fanning out in both directions from the current configuration (Fig. 8), and the workspaces which result from these configurations are displayed alongside the main outline as ghosts (Fig. 9).

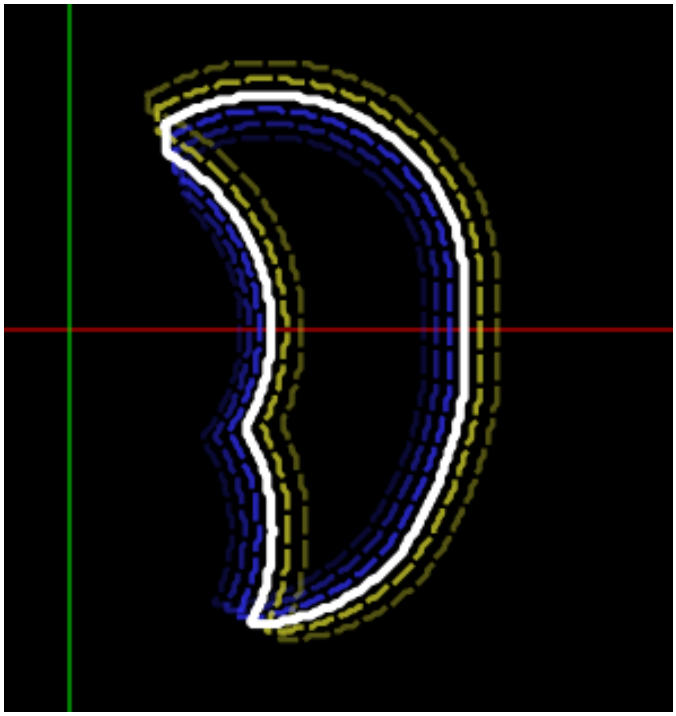

Fig. 9. Colored ghost outlines visible alongside the white main outline, here displayed for the 'forearm-length' parameter and showing a straightforward scaling effect.

As long as no other parameters are changed, these samples (and their corresponding ghosts) remain static in the configuration space. The intention is to create the impression of the current configuration *sliding through* the configuration space – the ghosts provide a visual reference for the relative change in the current workspace. Three ghosts are displayed on either side of the current configuration at all times; to maintain this, as the current configuration crosses over a ghost's sample configuration, one ghost on the positive side of the change is added, and a ghost on the negative side removed. To avoid visually jarring changes, and to provide a visual indication of the parameter-space distance, the ghosts fade in and out as the current configuration moves respectively towards or away from them. A sample motion through the 'rod-ratio' axis is shown in Fig. 10.

4.3 Inspectors – Arm Visualization

A sketch view of the arm itself is displayed, for the current configuration, in two instances: when the user hovers the mouse over the workspace area, and when the user clicks in the workspace to select a point of interest. In both cases, the arm is visualized with its goal end effector position set to the hovered or selected point. Here, I refer to these individual displays as "inspectors".

Each inspector operates a dedicated IK solver in 'point' mode, which yields results for a single goal point rather than a grid across the full range of motion (as in the workspace solver). In this mode, the solver returns sufficient information to reconstruct the arm's pose, and so the arm can be drawn by 'joining the dots' according to the computed elbow position, interior angles and configuration lengths. Optional additional annotations may also be displayed, and are displayed by default for the hover inspector. These include a readout of the goal coordinates and maximum load for the point, and a force-vector display for the load calculations, providing some indication as to the internal forces on the arm for a particular point at maximum loading.

This mode of viewing was chosen as a straightforward and complete representation of a solution at a point, and as a way of showing the user the physical structure of the arm, both for the current configuration for any point in the workspace. Selecting a point allows the user to see how the arm's structure changes as they alter configuration parameters, tying these parameters to their physical representations.

4.4 Readout and Load Preview

As the user moves the hover inspector, or selects a point in the main view, additional readouts (**??**) appear on the top-left of the application window. These readouts include explicitly laid-out information on the coordinates of the point in question, as well as a bar chart, dubbed the "load preview".

This preview chart serves a role supplementing the ghost outlines in guiding the user towards a suitable solution: rather than displaying the change in the workspace for a particular parameter, this chart shows how the *load* at a particular point will change as the parameter is altered, with each of the three bars on the left and right of the central column corresponding to the three ghosts on either side of the current configuration.

4.5 Interface Design

The Robovis application window is a largely static user interface containing controls for the current configuration, and several visualization widgets, including the main view. This view contains the key visualization elements of the tool, including the outline and heatmap views of the current configuration, the ghost outlines of nearby configurations, and any visualizations of the arm itself.

5 Implementation

Medium-level implementation description. Specifics of what you did versus what other libraries you used/built upon.

This is a major divergence from standard research paper format, providing much more detail than would normally be appropriate.

5.1 Workspace Calculations

The Inverse Kinematics and load calculations which go into generating Robovis' views were reformulated from *PyIK*'s code. In PyIK, the workspace (Fig. 2) was calculated in much the same overall way as in Robovis, with a grid of sample points; however, this is where the similarities end. The newer tool takes the original nested loops and converts them to massively parallel matrix operations: where before there were individual values inside loops, now there are values inside huge matrix data structures, with calculations handled by NumPy. For some lines this was a matter of simply switching data types – for example, $\cos(x) \cdot y$ will work in semantically identically as a statement in a loop or with x replaced with a matrix holding the old values of x.

5.2 Asynchronous Compute

As my data set was not precalculated, a substantial amount of work went into making the generation of results data execute at interactive speeds. This was achieved through a custom asynchronous compute pipeline which spreads work units ("jobs") over separate processes, and hence over the available CPU cores on the host computer. This also serves to keep the main process relatively unencumbered, ensuring the user interactions and user interface rendering which it handles are left working smoothly and responsively.

The actual "work" discussed here is the generation of workspace results for the ghost outlines. Importantly, results are generated not only for the on-screen ghosts, but for the off-screen ghosts attached to non-current parameters. Individual jobs execute within approximately 100ms on a typical PC core, but clearly the number of jobs discussed here (several dozen) precludes interactivity when computations are executed sequentially on a single thread.

The basic unit of the asynchronous pipeline is the *Process*, a representation of an operating system process provided by Python's *multiprocessing* module. Full processes are used over the more common case of within-process threads due to a phenomenon dubbed "Global Interpreter Lock"[1]. In short, in the most common Python runtimes, multiple threads within the same Python process cannot be executed concurrently. Instead, sub-processes are spawned during execution: these are separate instances of the Python program which can interact indirectly with the main process. Processes are given a standard Python function to execute, and expire when this function terminates. For convenience, *multiprocessing* provides a *Pool* system for creating and managing worker processes. Rather than terminating when their work is done, Pool workers run a function which listens for input in the form of new functions and arguments to execute on. The Pool object itself is provided work, which it automatically assigns to the next available worker. The results of this work can then be polled for completion.

The Pool seems like a full solution to the asynchronous work problem, but unfortunately falls short in the details. The main issue I found was that there was no way to modify the work queue once jobs were submitted: in an interactive application with a constantly changing state, this meant spending significant time waiting for jobs with out-of date results to complete. Tagging jobs with an incremental stamp ensured only the newest results were actually displayed, but I was still left with multi-second latency and inconsistently updated visuals. My solution was to wrap the Pool in an additional job queueing system which keeps the number of jobs in the Pool within the limits of the worker count. The system monitors the completion of previous jobs, and the Pool is provided with new jobs only once a worker has completed a previous one. In the meantime, jobs in the wrapper's queue are free to be modified, overwritten, or re-ordered. The queue as implemented is a priority queue, with the additional constraint that jobs are tagged with a 'reference', and any new jobs with the same reference as a job in the queue will replace the old job.

[1] GIL, its causes, and its full effects are far beyond the scope of this paper. A general overview of the concept is available at `https://en.wikipedia.org/wiki/Global_interpreter_lock`, and a specific discussion for CPython at `http://www.dabeaz.com/GIL/`

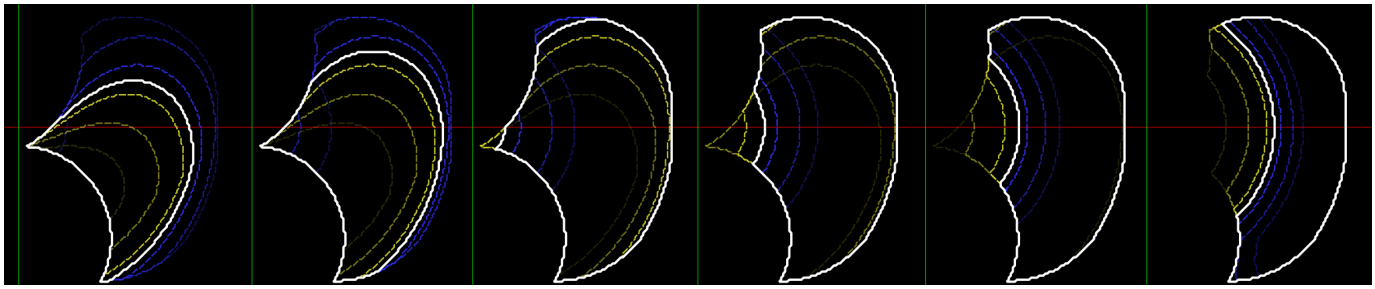

Fig. 10. Progression of main, white outline and colored ghost outlines as the 'rod-ratio' parameter is altered in the negative (blue) direction.

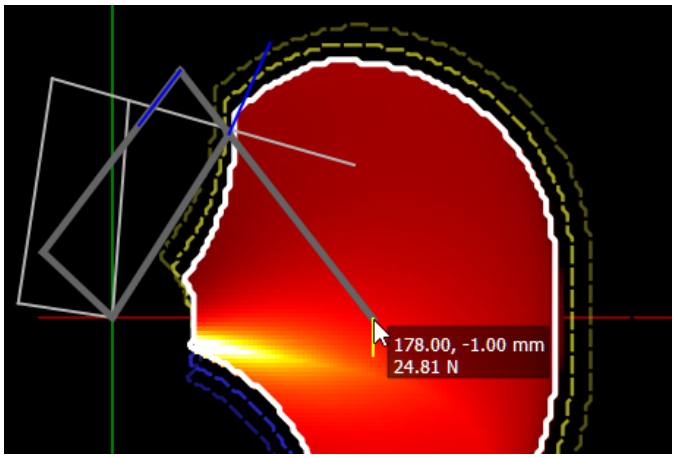

Fig. 11. Both inspectors (arm visualizations) visible at once – the upper arm for a selected point, and the lower for a hovered point

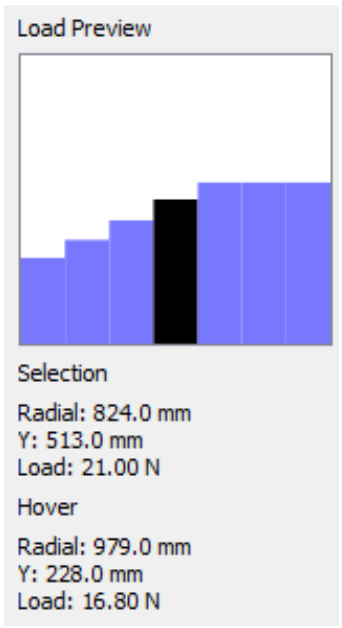

Fig. 12. Readouts for the selected and hovered points, with the load preview bar chart displaying the relative load for nearby (ghost) configurations at the selected point.

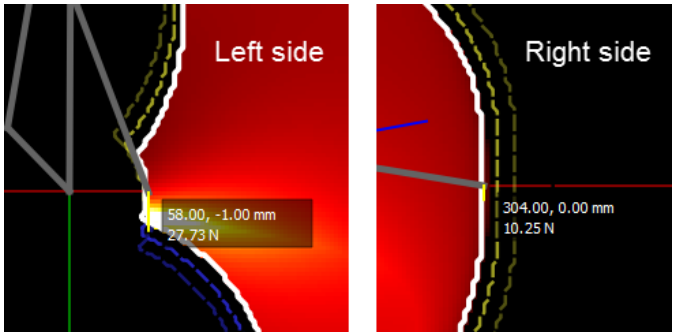

Fig. 13. Scenario 1: Hovering over the workspace to inspect the default configuration

6 Results

7 Scenarios of Use

I expect that a user will approach the application with some idea of what they want out of their own robotic arm in terms of the workspace and load requirements.

I work under the assumption that any configuration file output by the software can be automatically 'compiled' by separate software into 3D-printable files for physically constructing the arm; however, this is not a critical assumption, and Robovis does not have to be integrated with a larger toolchain to be useful. Rather, the tool could be considered as a design exploration tool for *manual* creation of a palletizing robot arm design, outputting the dimensions and motor/drive system requirements for a roboticist to design for; or, indeed, as a tool for roboticists to use themselves in the design process.

7.1 Scenario 1: Sorting Cards

A researcher has a project in which she wishes to use a robotic arm in combination with an optical tracking system to sort playing cards. She realizes that the arm needs a wide reach over a flat surface to move the cards around, but minimal carrying capacity – just enough to support a wrist joint and pneumatic holder attachment, with the weight of the cards themselves being negligible. She also notes that the arm will not have to be able to lift the cards more than a few centimeters above the work surface.

She estimates that the entire assembly at the end of the arm, including the motor, frame and pneumatic cup, will weight 120g; she decides that a load capacity of 2.0 newtons would leave a comfortable margin for her task. She measures her working area, a tabletop within the optical system's view, as 30cm by 30cm, and notes that it could be raised or lowered freely to accommodate the arm's height. She opens Robovis to begin exploring the potential designs for a robot arm.

Presented with the default view of the seed configuration, she examines its properties by hovering her mouse over the view, and moving it across the widest area of the arm's workspace (Fig. 13). She finds that the arm's reach is too small, and notes that it is annoying to have to manually calculate the distance between the sides of the workspace,

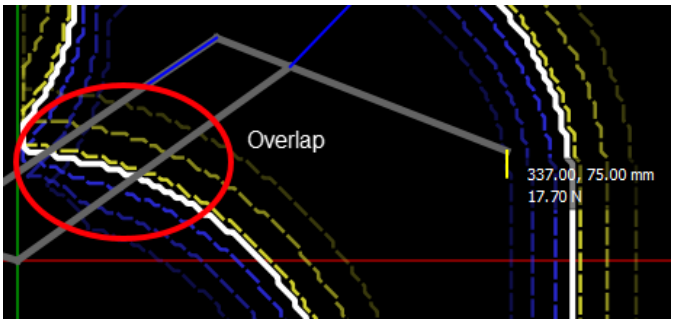

Fig. 14. Scenario 1: Arm vis shows overlap between the arm's pose and the worksurface under consideration at y=80mm

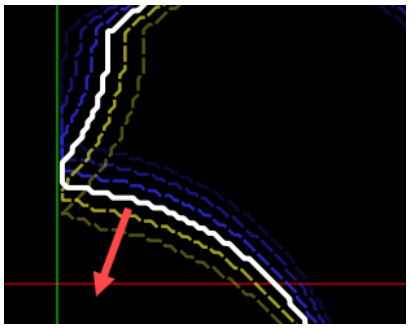

Fig. 15. Scenario 1: Ghost outlines show that increasing the forearm length will lower the wide portion of the workspace

Fig. 16. Scenario 1: Load histogram shows that the load distribution for the current configuration is comfortably above the setpoint.

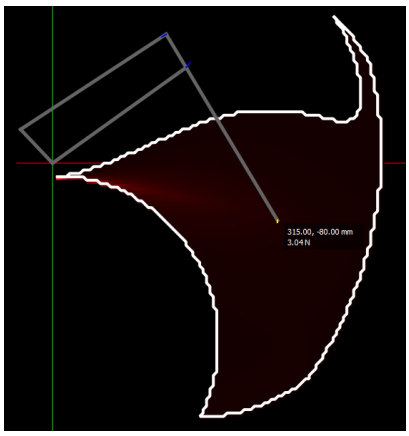

Fig. 17. Scenario 1: Shape of the final workspace for the scenario, fulfilling the researcher's requirements.

as there is no measuring tool in the application. She also notices that the arm is far more powerful than she requires, able to support some 20-30 N of load over the widest area. She decides to approach the size of the workspace as the first problem to deal with, and turns off the load heatmap using the toggle button.

Scrolling through the parameters by hovering the cursor over them, she sees that raising either the forearm or the elevator lengths will increase the workspace size, while changing the rod ratio will only shrink the area, and altering the motor torques would appear to have no effect. She decides to raise the elevator length to 230 mm, and sees that the widest portion of the workspace, at approximately +80 mm y, now meets her requirements as it is over 380 mm in width. However, as she inspects it with the hover tool, she sees that the arm's frame would intersect this area (Fig. 14), indicating that the arm would collide with the worksurface. She surmises that the working area will have to be at around y = 0 to avoid contact between the arm's mechanics and the work surface.

Scrolling through the parameters again, she sees that the ghost outlines indicate that raising the forearm's length will lower the widest portion of the workspace (Fig. 15), as she has now realized she needs to do. She raises the forearm slider to 249.2 mm, at which point this wide area sits over the line at y = 0. Inspecting the range at this line, she finds that the arm can reach radially from as close as 40 mm from its origin to as far as 475 mm away.

With no facility in the tool to view the arm's workspace from above, she resorts to pen and paper to verify that the workspace at this level will encompass her 300×300 mm worksurface. Placing the worksurface at 40 mm away from the origin horizontally, she checks that the corner of the worksurface is within the range of the arm:

$$\sqrt{340^2 + 150^2} = 371.6 < 475$$

Satisfied, she turns to the arm's load. She knows that less powerful motors tend to be cheaper, and she wants to save part of the project's budget for a selection of optically diverse playing cards – thus, she decides to try and keep the motor's power requirements as low as possible. She starts by raising the minimum load to 2 N, noting that this change comes nowhere close to the main spike in the load histogram: none of the workspace has been cut off by increasing the minimum load (Fig. 16).

She lowers both motor torque sliders, drastically at first (watching the shifting load distribution), before switching to a finer-grained refinement. She realizes that using the same motors for both axes would make sense, and she tries to establish a lower bound on the acceptable power for both. The widest portion of the working area drops slightly, but examining with the hover tool shows her that the arm will not collide with the worksurface; having the arm's origin above the surface is also acceptable. Continuing in a loop of refinement and checking, she finds that increasing the 'rod ratio' creates a wedge-shaped workspace (Fig. 17) which works well for her purposes, and allows her to drop the motor torques to just 0.5 N.m.

7.2 Scenario 2: Moving Cans

A food packaging facility operates a production line and are adding a new conveyor belt to the line where heavy, filled cans must be moved from a low conveyor belt to the new, higher one, with a 0.8 m vertical separation between the two levels. The conveyors will be placed 1 m apart, and the production manager thinks a robot arm can be placed in-between the two conveyors, where it will swing from one to the other while also raising the cans up. The cans weigh 600g when full.

The technician assigned to the task considers his options, and decides that a robot arm built in-house specifically for the task could save costs over a generic industrial arm. He opens up Robovis, and like the researcher in scenario 1, finds that the seed configuration is far from his requirements. He reasons that he will need the arm to be able to place the cans at a radius of around 600 mm – or, at least that the radii where it picks up and drops off the cans add up to around 1100mm (leaving a margin for picking and placing from across the conveyors). He sees that increasing either the elevator or forearm lengths will increase the reach of the arm; he picks the elevator to scale up.

Scaling up the elevator length, he finds that he is creating an arc-shaped workspace (Fig. 18), and considers whether this would be

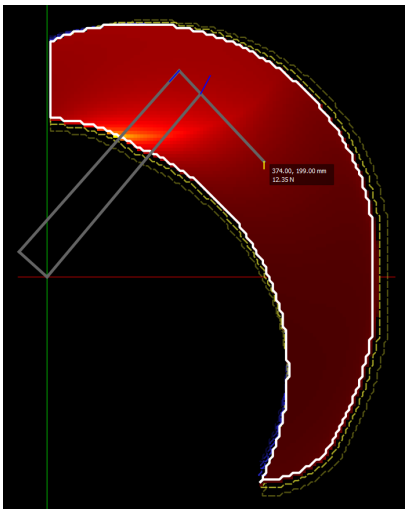

Fig. 18. Scenario 2: Arc-shaped workspace owing to a high elevator length and relatively low forearm length

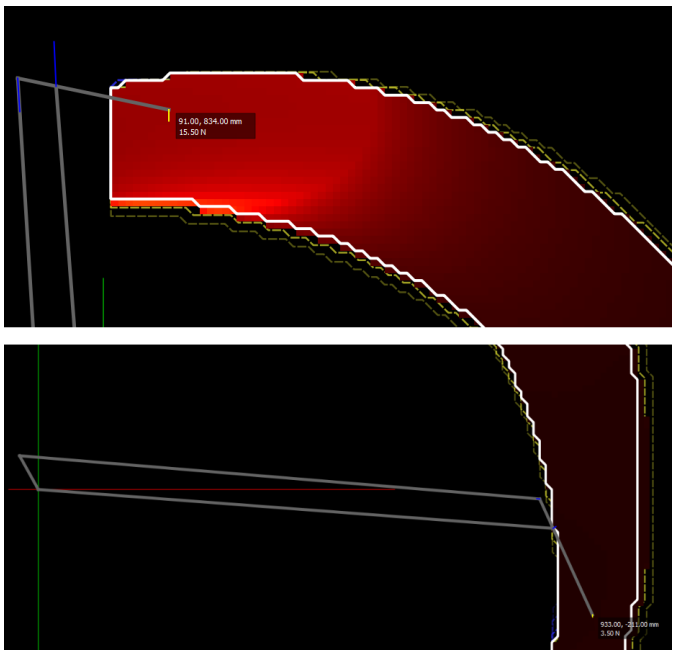

Fig. 19. Scenario 2: Examining the reach of the 'arc' robot, designed to lift objects from a distant position to place them at a nearby, but elevated position.

suitable for his purposes. He continues expanding the elevator length until the outer reaches of the arc are around 1 m away from the origin, and the upper reaches over 0.8 m higher than the zero level, where the arc bulges out the most (Fig. 19). He reasons that an arm of this type could be placed adjacent to the higher conveyor, reaching across to pick up the cans and lifting them up and around onto the higher conveyor, near the center of rotation; that is, close to radial zero.

He considers the weight of the payload. He knows that Robovis does not account for the arm's frame weight, and adds 500g to the payload weight to account for this, bringing the total up to 1.1 kg. He reasons that the payload will have to be accelerated upwards, and picks an acceleration of $3ms^{-2}$. Combining this with gravity, he arrives at a peak load of 14 N, and enters this into Robovis. Finding that this leaves only a small patch of workspace left, he scrolls through the parameters, and finds that increasing the elevator torque begins to restore the workspace size. When he raises the elevator torque past 14 N.m, the workspace is fully restored (as indicated by the load histogram). He realises that this makes sense – at the furthest reach of the workspace, the elevator motor is supporting a 14 N load at 1 m away. The final workspace for this option is shown in Fig. 20)

8 Discussion and Future Work

In my opinion, the main strength of the work I have presented here is its novel and goal-oriented approach to parameter space exploration. The 'ghost' outlines are the key idea which turn the user experience from an unguided trial-and-error exploration to a visually driven optimization task, and it is this visualization technique which I feel deserves the most recognition in this tool. I'm also pleased with the level of responsiveness I achieved, especially considering the sheer amount of numerical calculation involved in formulating the data for display; this also feeds into another mixed strength, the level of integration between the tool and its underlying data representation. As discussed at length in Sect. 5, Robovis generates its data dynamically, which expands its range of operation beyond what any static dataset could offer: both very fine refinements and extremely wide ranges are possible in the search process.

I believe the major weakness of this work is the lack of any user participation in the design process, or any formal user study – this leaves me without any real feedback (or other metrics) with which to objectively evaluate the success of my tool and approach. However, I do have some "feedback" in the form of the results of the usage scenarios walked through in Sect. 7.

There are some important shortcomings in my approach revealed by these scenarios – while not critical failures (I still found that a motivated user could reasonably complete the scenarios), there are

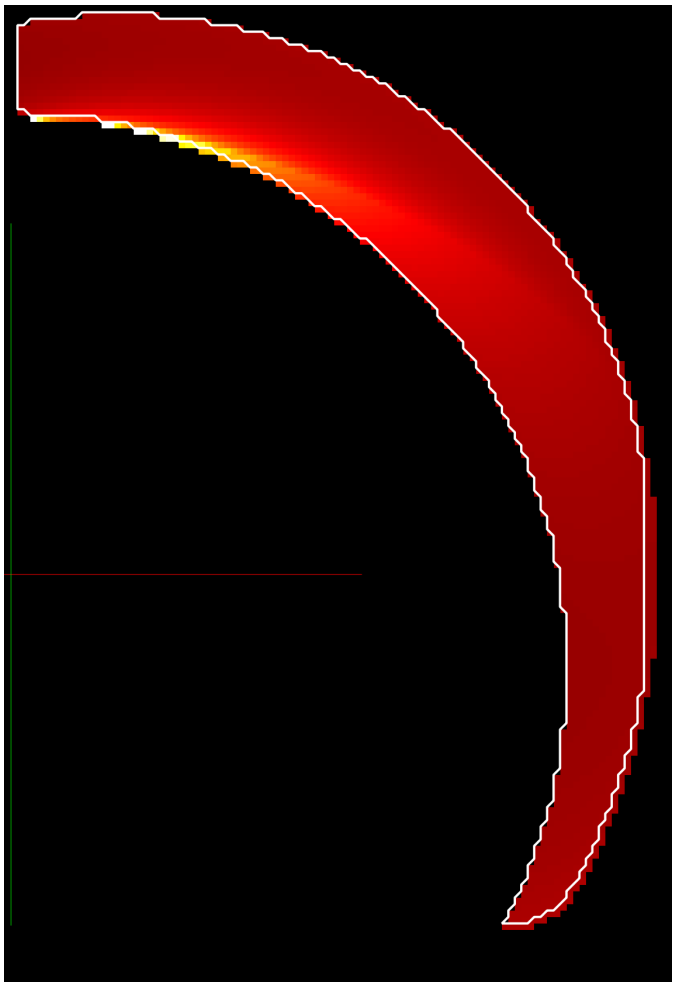

Fig. 20. Scenario 2: The final workspace for scenario 2, with a slim, arcing path and support for heavy loads at all points.

Fig. 21. A highly skewed load distribution, as shown by the scented slider

problems which hamper progress:

- The inspection tool (hovering the cursor) is essential, but provides no direct way to measure distances, leaving this up to the user. This is an easily automated task, making the absence of such a tool all the more frustrating.

- Different, but closely related parameters cannot be adjusted together. For example, in scenario 1, the researcher wants both motors to have the same torque output. Manually adjusting both to match with each change is frustrating, and again easily automated by adding a 'lock' button or multi-parameter scaling option. This type of option would also be useful for scaling the two length parameters together.

- The weights of the arm's segments being excluded from the load calculations might lead to overly optimistic specifications for the arms. This is tricky to address – there is no obvious mapping from lengths to weights without making significant assumptions about how the arm will be constructed. Having a system which converts the specifications to 3D files could help solve this, as the physical properties of parts of different lengths could then be estimated with reasonable accuracy.

8.1 Other Issues

Outside of those identified by the scenarios of use, there are several remaining problems with Robovis. In my view, most of these amount to a lack of 'polish', rather than critical failures in the approach. Problems include the lack of a key for the load heatmap, and scaling problems in general: the configuration sliders naturally have capped upper values, and the main view displays at a fixed 1 mm : 1 pixel scale, limiting the maximum dimensions of the arm under consideration according to the user's display resolution. Logarithmic scaling helps the slider problem to some degree, but there are no mitigating factors for the view scaling. In addition to not actually being displayed, the load heatmap's key is also static: the colors map to a fixed range of load values, so large and/or powerful arm designs display a washed-out heatmap which is of little use to the user. This was partly due to difficulties in establishing a dynamic range for the load values, as the distribution is typically highly skewed – this is shown on the load histogram, which itself cuts off a large portion of the distribution's upper range. When there is a large disparity between the actuator and elevator torque values, this skew can become very pronounced – see Fig. 21. A reasonable solution might be to pick a range which starts at 0, and extends to cover some percentage of the load samples (for example, 90%).

Other minor issues include the lack of numbered labels on the scented slider, and a problem with the display of the preview barchart where the middle (current) bar is out of alignment with the overall trend – see Fig. 22. For whatever reason, this error cycles on and off when the user hovers over (selects) a parameter repeatedly.

Finally, I would have liked to add the ability to alter the configuration through a selection idiom. I had originally intended to allow the user to directly select the ghost outlines and load previews, making the selected outline's configuration the new current configuration. This was not implemented due to time constraints.

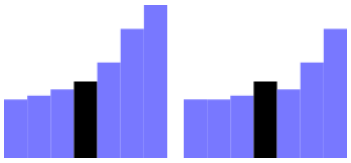

Fig. 22. Barchart artifact where the central bar does not follow the trend: the correct view is shown on the left. Inspection reveals that this is indeed an erroneous result, as the load value for the current configuration in this image does in fact sit in-between those of the surrounding configurations.

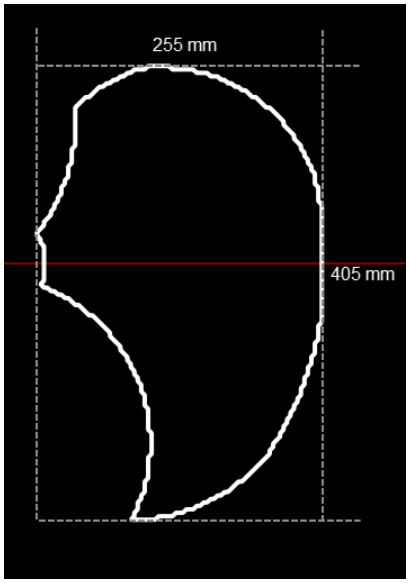

Fig. 23. Future work: Bounding box concept for displaying the dimensions of the current workspace

8.2 Future Work

Near-term future changes would mainly involve fixing the more minor problems described above, and some polishing work. In no particular order:

- Add a numbered key for the load heatmap, with a dynamic range adjusted to include the majority of the sampled values.

- Identify the cause of, and correct, the barchart display artifact.

- Add a sliding, numbered scale to both the load histogram and the preview barchart.

- Dynamically adjust the view (zoom in/out) to fit the arm's workspace.

- For the inspectors and heatmap, continue to display areas which are below the load threshold, but still physically reachable, using a grayed-out color or other visual distinction. This would provide additional context for users looking to expand the reachable range, or who are flexible with load requirements.

- Add a bounding box around the workspace to display its dimensions – Fig. 23.

More ambitious changes would involve new features and enhancements beyond simple "fixes". One obvious addition would be implementing the planned selection idiom for ghost outlines and the "preview" bar chart, allowing the current configuration to be updated by clicking the displayed items. Adding ruler guides and a spatial grid behind the main display would allow the user to maintain greater awareness of the scale of the arm they are working on.

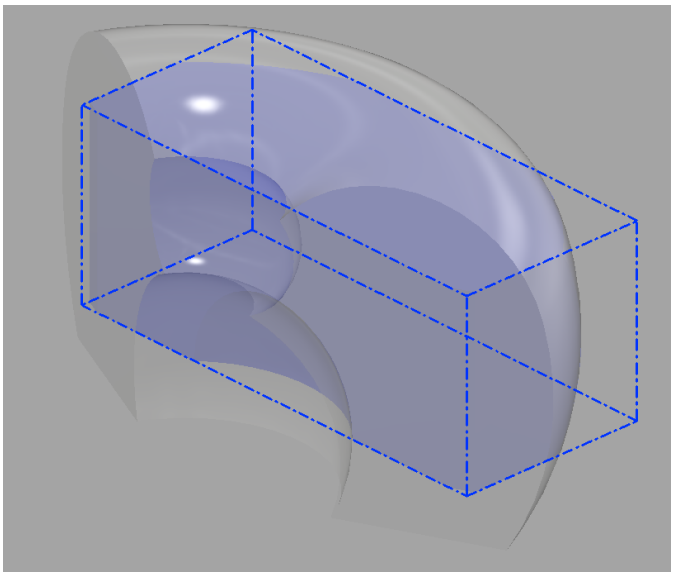

Fig. 24. Future work: 3D view mockup in CAD software, showing the intersection of a target cuboid workspace and a revolved robot arm workspace.

Even further afield, I believe the basic visualization approach taken here could be extended with additional robotic performance goals. Both maximum velocity and acceleration are relevant performance metrics which will also vary significantly over the workspace of the robot, and could be brought under consideration using much the same techniques as for the maximum load: scented widgets with adjustable minimums, and heatmap displays. Adjustable minimums would naturally incorporate these performance metrics into Robovis' ghost displays, in the same way as for the load values.

When showing Robovis to my peers, some commented on the lack of a 3D display: while this may not be particularly useful in isolation [6, Chapter 6], it would provide additional context to new users who might otherwise be unsure how to interpret the 'sliced' default view. It could also prove a powerful feature if combined with volume rendering and user-defined shapes of interest: by displaying the intersection between the workspace volume and user-defined shapes (Fig. 24), users could define their own workspace requirements in three dimensions, and intuitively evaluate how well the current configuration fits them.

I would also be interested in seeing how my approach could be extended to a greater variety of robotic arm designs, beyond the simple palletizing robot focused on here. This might include SCARA robots, which also operate primarily in a single plane [4].

On the user interaction side, undo/redo for the configuration changes would allow users to return to a known good state in the application; changes would not have to be lossy.

8.3 Lessons Learned

In general, I need to focus more on the user experience, and should perform some scenario testing as early as possible in development of a user-facing tool like this. The testing here has shown some issues with Robovis, such as the inability to measure distances directly, which are both frustrating and which would have been fairly straightforward to address.

It is possible I would have been able to add more of the 'polish' the application is missing if I had spent less time working to increase responsiveness; however, this responsiveness in the face of significant data transformation requirements is an aspect of the tool I'm particularly proud of, and without this real-time responsiveness, the tool's interface would have had to work quite differently – relying, I suspect, on iterative selection of configurations rather than fine-grained user-driven changes to the parameters. I cannot say conclusively if I made the right decision, as I cannot judge the effectiveness of hypothetical interface choices, and I am unable to say for certain how much further I would have gotten while taking a different path. Perhaps the lesson here is to prototype, in order to try different options before making a final decision: my choice was not made on evidence, but on a gut feeling that a real-time interactive application would be more engaging and yield faster iterations on arm configurations than a less responsive version.

9 Conclusions

While this tool is in some ways incomplete, lacking features – like distance measuring tools, and multiple views – which one would expect from what amounts to a specialized CAD application, the results presented here are extremely promising. As demonstrated by the usage scenarios, even lacking these additional features, Robovis allows a user to come up with novel and highly specialized designs for robotic arms in a way that is quick and intuitive. With additional work, I believe this could be turned from a promising prototype to a powerful rapid design tool.

References

[1] W. G. Hao, Y. Y. Leck, and L. C. Hun. 6-DOF PC-Based Robotic Arm with Efficient Trajectory Planning and Speed Control. In *International Conf. Mechatronics (ICOM)*, pp. 1–7, May 2011. doi: 10.1109/ICOM.2011.5937171

[2] N. A. M. Johari, H. Haron, and A. S. M. Jaya. Robotic Modeling and Simulation of Palletizer Robot Using Workspace5. In *Computer Graphics, Imaging and Visualisation (CGIV)*, pp. 217–222, Aug. 2007. doi: 10.1109/CGIV.2007.73

[3] D. Keefe, M. Ewert, W. Ribarsky, and R. Chang. Interactive Coordinated Multiple-View Visualization of Biomechanical Motion Data. *IEEE Trans. Visualization and Computer Graphics*, 15(6):1383–1390, Nov. 2009. doi: 10.1109/TVCG.2009.152

[4] H. Makino. Assembly robot, July 1982. US Patent 4,341,502.

[5] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design Galleries: A General Approach to Setting Parameters for Computer Graphics and Animation. In *Proc. 24th Annual Conf. Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pp. 389–400. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 1997. doi: 10.1145/258734.258887

[6] T. Munzner. *Visualization Analysis and Design*. CRC Press, Sept. 2014.

[7] B. Roth and K. Gupta. Design Considerations for Manipulator Workspace. *Journal of Mechanical Design*, 104:705–713, 1982.

[8] M. Sedlmair, C. Heinzl, S. Bruckner, H. Piringer, and T. Moller. Visual Parameter Space Analysis: A Conceptual Framework. *IEEE Trans. Visualization and Computer Graphics*, 20(12):2161–2170, Dec. 2014. doi: 10.1109/TVCG.2014.2346321

[9] W. Willett, J. Heer, and M. Agrawala. Scented Widgets: Improving Navigation Cues with Embedded Visualizations. *IEEE Trans. Visualization and Computer Graphics*, 13(6):1129–1136, Nov. 2007. doi: 10.1109/TVCG.2007.70589