Draft version October 26, 2019
Typeset using LaTeX default style in AASTeX63

SDSS-V Algorithms: Fast, Collision-Free Trajectory Planning for Heavily Overlapping Robotic Fiber Positioners

Conor Sayres,[1] José R. Sánchez-Gallego,[1] Ricardo Araujo,[2] David W. Hogg,[3] Juna A. Kollmeier,[4] Michael R. Blanton,[3] Sarah Tuttle,[1] and Richard W. Pogge[5]

[1]*Department of Astronomy, Box 351580, University of Washington, Seattle, WA 98195, USA*
[2]*School of Engineering, Swiss Federal Institute of Technology in Lausanne (EPFL), 1015 Lausanne, Switzerland*
[3]*Center for Cosmology and Particle Physics, Department of Physics, New York University, 726 Broadway, New York, NY 10003, USA*
[4]*Observatories of the Carnegie Institution for Science, 813 Santa Barbara Street, Pasadena, CA 91101, USA*
[5]*Department of Astronomy, Ohio State University, 140 West 18th Avenue, Columbus, OH 43210-1173*

Submitted to AJ

ABSTRACT

operationally sdss is changing significantly in sdss-v so we have new problems to solve

Keywords: surveys

1. INTRODUCTION

Robotic fiber positioners (RFPs, robots, positioners) are emerging as a promising technology in the today's landscape of wide-field multi-object spectroscopic instruments and surveys. Many projects (e.g. LAMOST, PFS, DESI, MOONS, 4MOST, MANIFEST, MSE, SDSS-V) (cite?) have adopted densely packed RFP arrays patrolling the telescope's focal plane to obtain multiplexed spectroscopic observations of hundreds to thousands of objects in a field. RFP arrays provide the ability to rapidly reconfigure between fields and exposures. Most RFP systems are designed to provide full coverage of a focal plane, which necessitates a certain amount of overlap in RFP patrol zones. A system in which RFPs may physically overlap must have a motion planning strategy ensuring that robots do not collide, wedge, or deadlock during reconfiguration. Such situations could severely impact survey performance, if not wholly cripple the system. This manuscript presents a clever and efficient algorithmic solution to the problem of collision avoidance and trajectory planning in the context of heavily-overlapping robotic fiber positioner arrays.

The work presented here has been developed for deployment with the SDSS-V Survey which is briefly summarized in Section 2. We present a generic layout describing the relevant geometries and mechanics of the robotic fiber positioners in Section 3. An overview of the collision avoidance problem and its impacts on survey performance are presented in Section 4. The algorithm is described in detail in Section 5 and its performance is discussed in Section 6. We conclude this work in Section 7, motivating the importance of algorithmic development to maximize scientific output in the current era of next-generation large scale astronomical surveys.

2. SDSS-V SURVEY OVERVIEW

The Sloan Digital Sky Survey (York et al. 2000, SDSS I/II) is currently ramping towards a fifth phase of operations. For nearly two decades, SDSS has continuously served high quality, well documented, and accessible data in a series of 15 (and counting) data releases. The most recent data release being DR16 scheduled for December 2019. The project has evolved over the years, beginning with SDSS I/II which provided imaging data from a mosiac camera that now resides at the Smithsonian[1] and multi-object optical spectroscopy through the use of plug plates[2] at Apache Point Observatory (APO). SDSS-III (Eisenstein et al. 2011) included new instrumentation at APO for conducting a near-infrared multi-object spectroscopic survey APOGEE (Majewski et al. 2017) and a radial velocity planet finding survey MARVELS (Ge et al. 2009). SDSS-IV (Blanton et al. 2017) saw the inclusion of the MaNGA integral field unit (IFU) survey of nearby galaxies (Bundy et al. 2015), and built out new infrastructure and instrumentation to extend

[1] https://www.sdss.org/instruments/#Camera
[2] https://skyserver.sdss.org/dr12/en/tools/getimg/plate.aspx

the APOGEE survey in the southern hemisphere at Las Campanas Observatory (LCO). SDSS-V (Kollmeier et al. 2017) is an all-sky multi-epoch spectroscopic mapping survey operating for 5 years in both hemispheres beginning mid-2020. At a top level, SDSS-V is broken down into three components: the Local Volume Mapper (LVM), the Black Hole Mapper (BHM), the Milky Way Mapper (MWM). Each of these are introduced in turn below.

The LVM will obtain contiguous optical spatio-spectroscopic coverage of the Milky Way and nearby galaxies using IFU design inspired by the SDSS-IV MaNGA Survey (Drory et al. 2015). The LVM will sample the Milky Way, SMC/LMC, and M31/M33 at sub-parsec, 10 parsec, and 20 parsec spatial scales respectively. This will yield a dataset well suited to investigate mechanisms in ISM over wide areas at resolutions paired to the physical scales of feedback and enrichment. It will use a completely new suite of instrumentation and telescopes dedicated to the LVM survey. Although the LVM will not use a robotic fiber positioning system, it is included here as a key component of the SDSS-V mission, and the intersections of the LVM and MWM datasets will provide spatially resolved structure of the major components of our galaxy: stars, ISM and their interactions. The LVM dataset will provide >25 million contiguous spectra covering over 3,000 deg^2 of the sky.

The BHM is an all-sky survey obtaining optical spectroscopy of >400,000 extra-galactic sources. It will obtain multi-epoch observations of quasars/AGN as well as providing spectroscopic follow up of x-ray sources from the recently launched eROSITA space telescope (Predehl et al. 2010). BHM will extend upon the reverberation mapping project of SDSS-III/IV (Shen et al. 2015), selecting ∼1500 quasars to receive hundreds of visits throughout the duration of survey to measure masses of black holes. Additionally ∼25,000 quasars will receive between 3 and 13 visits throughout the survey to monitor variability. By combining BHM data with existing BOSS measurements, BHM will provide a homogeneous time-series dataset of quasar spectra boasting decade long baselines, sensitive to variability on month-long timescales.

The MWM is an all-sky survey targeting many classes of stars in the Milky Way for optical and near-infrared spectroscopy at an impressive scale. Fundamental stellar characterization (e.g. age, chemistry, kinematics) will provide a dataset from which we can investigate the history and evolution of the galaxy's structure. Targeting over 6 million objects, the MWM will extend the observed footprint of the SDSS-IV APOGEE Survey by (x-amount?). Beyond a global spectroscopic map of Milky Way stars, the MWM survey is well-timed to leverage GAIA (Gaia Collaboration et al. 2016) and TESS (Ricker et al. 2014) space missions for interesting targets for multi-epoch observations. This massive increase in target volume with respect to previous SDSS surveys would be impossible to achieve using the existing SDSS plug plate infrastructure, predicating the need for a flexible and nimble robotic fiber positioning system to complete the survey.

To facilitate the MWM and BHM surveys, robotic fiber positioning systems will be installed in the focal planes of the 2.5 meter Sloan Foundation Telescope at APO (Gunn et al. 2006) and the 2.5 meter du Pont Telescope at LCO (Bowen & Vaughan 1973). Each focal plane system (FPS) will carry 500 RFP units arranged in a hexagonal array covering a 7 deg^2 field of view at APO and a 3 deg^2 field of view at LCO. 300 of these units will feed both the BOSS optical spectrograph (Smee et al. 2013) and the APOGEE near-infrared spectrograph (Wilson et al. 2019), while the remaining 200 units will carry fibers dedicated solely to the BOSS optical spectrograph.

SDSS-V builds on a heritage of evolution and cooperation between instrument, infrastructure, science, and data teams to obtain and deliver a wide and diverse set of data products to the community. Operationally, SDSS will see major changes with the inclusion of the FPS instruments, and teams are currently tackling the many challenges inherent in organizing, optimizing, and deploying a successful SDSS survey operating in a completely new mode. The work presented here outlines and retires one of the major risks involved with SDSS's move to an RFP array: a strategy for safe and efficient robot trajectory planning during array reconfiguration. The approach presented in further sections should be informative and applicable to other surveys operating with similar instruments and constraints.

3. FOCAL PLANE SYSTEM LAYOUT

The SDSS-V FPS instrument design consists of several elements including fiber positioners, fixed metrology fiducials, and guide camera assemblies occupying a hexagonal layout mounted on a slightly curved surface that matches the focal plane of the telescope. The majority of SDSS-V RFPs carry two science fibers to feed either the BOSS or APOGEE spectrographs such that a small offset in robot position will place an object on either fiber. Removing SDSS-V FPS design specifics to present this work in a more general context we consider a layout in which each RFP carries a single fiber, the focal surface is a Cartesian plane ($z = 0$), and robots are densely packed in a hexagonal array without any additional axillary components.

3.1. *Positioner Details*

A solid model of the SDSS-V RFP is shown in Figure 1. Fiber positioning is achieved through the rotations of two arms about two axes. The alpha arm (lower arm), rotates about the alpha axis. The alpha axis is colinear with the lower body of the robot. The beta arm (upper arm), rotates about a beta axis at the end of the alpha arm. The beta arm carries the optical fiber and (as will be discussed in further sections) risks collisions with the beta arms of neighboring robots. The rotation axes, arms, and fiber are labeled in Figure 2.

To understand how the RFP places fibers, a focal plane view is helpful. Figure 3 shows the focal plane projection of a fiber positioner centered at the robot's xy base position (x_b, y_b). Robot arms and rotation axes are indicated on the figure. The alpha arm length (l_α) is the distance from the alpha axis to beta axis (7.4 mm for SDSS-V). The beta arm length (l_β) is the distance from the beta axis to the fiber center (15 mm for SDSS-V). The robot may position a fiber anywhere in the annular patrol zone through the specification of the alpha angle θ_α and beta angle θ_β, where $0 < \theta_\alpha \leq 360$ degrees and $0 \leq \theta_\beta \leq 180$ degrees. By restricting the operating range of θ_β to be 180 degrees, there is a one to one mapping between $(\theta_\alpha, \theta_\beta)$ coordinates and xy fiber position (x_f, y_f), this is to say that our robots are always "right handed", and may never obtain a "left handed" orientation. This coordinate conversion is:

$$\begin{pmatrix} x_f \\ y_f \end{pmatrix} = \begin{pmatrix} x_b \\ y_b \end{pmatrix} + \begin{pmatrix} \cos\theta_\alpha & \cos(\theta_\alpha + \theta_\beta) \\ \sin\theta_\alpha & \sin(\theta_\alpha + \theta_\beta) \end{pmatrix} \begin{pmatrix} l_\alpha \\ l_\beta \end{pmatrix}, \quad \text{where } z_f = 0 \tag{1}$$

3.2. *Positioner Array*

To obtain full focal plane coverage the robots are spaced at a pitch of $l_\alpha + l_\beta$ (22.4 mm for SDSS-V) in a hexagonal array. This allows a robot's neighbor to patrol its central exclusion zone, leading to heavily overlapping patrol zones between a robot and its neighbors. Figure 4 shows a 19 positioner array, indicating the area covered by 1, 2, 3, and 4 fibers in the focal plane. Scaling up the hexagonal array towards hundreds of positioners, the majority of the focal surface will be covered by 3 or more fibers. Only the perimeter of the hexagonal array will have single fiber coverage with some gaps.

[this paragraph needs help] Both SDSS-V and the MOONS projects have elected to use hex-packed RFP designs in which $l_\beta > l_\alpha$ leading to heavily overlapping patrol zones. This design grants a relatively large patrol area to each robot, yet still allows sets of targets with relatively close focal plane spacing to be obtained. In contrast, programs like DESI or PFS use hex-packed RFP designs with $l_\alpha = l_\beta$ and very slight patrol zone overlap, meaning that the typical target separation on the focal plane must be about that of the pitch between positioners. The BOSS spectrograph accepts a maximum of 500 fibers (APOGEE accepts 300 fibers), and the choice of RFP arm lengths in SDSS-V allows us to cover the full focal plane of the telescope with 500 robots, while granting the largest area possible for each robot to patrol. This design choice provides greater flexibility in target assignment and field generation, as a single target may be acquired by a number of available robots. (this is sorta true, really the overlap might make it so we block other robots from achieving their targets? not sure there is something quantitative to say. why did we go with the unequal arms in the first place? was it just because thats what MOONS did?). Its true that our layout does support target clustering better...

3.3. *Collision Formalism*

Considering the alpha arm length (l_α) and pitch between robots, there can be no interaction between alpha arms of adjacent robots, and as alpha and beta arms exist in different planes (see Figure 2), they do not interact with one another. Thus all physical interference between robots in the grid happens between beta arms. We present a relatively simple geometric strategy to represent a beta arm's position in 3D space. The beta arm's orientation is described by two elements: a Cartesian line segment and a collision buffer (σ_{buff}). The beta line segment is constructed from two points: the 'elbow' point $(x_e, y_e, z_e = 0)$ where the beta axis intersects the focal plane, and the point of the fiber in the focal plane $(x_f, y_f, z_f = 0)$. The (x_f, y_f) coordinates are given by equation 1. The (x_e, y_e) coordinates are given by:

$$\begin{pmatrix} x_e \\ y_e \end{pmatrix} = \begin{pmatrix} x_b \\ y_b \end{pmatrix} + \begin{pmatrix} l_\alpha \cos\theta_\alpha \\ l_\alpha \sin\theta_\alpha \end{pmatrix}, \quad \text{where } z_e = 0 \tag{2}$$

The absolute orientation of this line segment will depend only on the robot's $(\theta_\alpha, \theta_\beta)$ and (x_b, y_b) coordinates. The collision buffer (σ_{buff}) specifies a distance from the beta line segment such that it safely encloses the physical extent

of the beta arm. For SDSS-V RFPs $\sigma_{buff} \geq 1.5$ mm accomplishes this. We refer to this buffered 3D line segment as the collision envelope, which is a cigar shaped volume. Figure 5 constructs the geometric representation of the beta arm segment, collision buffer, and collision envelope for two neighboring RFPs.

Collisions between beta arms are determined by first computing the minimum distance between two beta arm line segments in 3D space. To accomplish this we employ an algorithm adapted from Dan Sunday, which is described in detail online[3]. If the minimum distance between beta arm segments is less than twice the collision buffer, then the collision envelopes of each beta arm intersect, and the two robots are collided. An example of this intersection is indicated as the red area in Figure 5.

For this work we've selected to define a beta arm as a line segment that exists wholly in the focal plane, and in this representation no two beta arm envelopes may overlap in the focal plane projection. In reality, the beta arm's shape (see Figure 2) does allow physical overlap due to the curvature of its neck (eg Figure 6). A more exact collision envelope for a beta arm could be represented using a series of line segments closely approximating the beta neck curvature, and such a model would allow beta arm overlap. This would open up the space available for target assignment in the focal plane and only slightly increase the runtime of the collision detection and path generation routine. Ultimately however, the choice of restricting beta arms to not overlap in the focal plane projection is motivated by the fact that the science fiber hangs in the space below the head of the beta arm, so to protect the fiber we favor non-overlapping beta arms.

4. PROBLEM OVERVIEW

The SDSS-V focal plane is a very crowded space. Figure 7 emphasizes this, indicating the multiplicity of collision envelope intersections in a 19 positioner array. Operationally, positioners will have to evade up to 6 other neighbors simultaneously on their way from one target to the next: a robotic game of Twister[4] with 500 players.

The simultaneous coordination of two axes of motion for 500 robots is a path planning problem in a high dimensional space. Trajectory planning is an active field of development in the robotics literature with applications from autonomous vehicles to highly dexterous robotic arms. The typical benchmark robotics control problems are often focused on fast computations in low dimensional spaces to navigate static obstacles. Recently, the field has begun tacking multi-robot coordination problems. These so called swarm control problems are inspired by emergent behavior observed in swarming biological systems (insects, fish, bacteria, etc). Decentralized Navigation Functions (DNFs) are often employed to model coordinated swarm behavior (Ge et al. 2012; Falconi & Melchiorri 2008). The control challenge inherent in today's robotic multi-object spectrograph designs poses a similar problem.

The REACT[5] team has devoted significant effort toward solving the RFP collision-avoidance problem using a DNF algorithm (Makarem et al. 2014). Conceptually, this strategy somewhat akin to an N-body problem in which robot arms experience a repulsive force from one another, and an attractive force towards their target. A time-stepping loop is used to propagate robot arm kinematics according to their local potential fields until all robots arrive at their target positions. For array layouts with slightly overlapping fiber patrol zones (e.g. DESI or PFS), this strategy was shown to guarantee the successful convergence of robots to targets. However, when applied to heavily overlapping RFP arrays like those of SDSS-V and MOONS, this algorithm struggles: 30-40% of positioners will experience a deadlocked state in which progress towards the target halts due to the inability of combinations of positioners to push pass one another. To combat this, Tao et al. (2018) injected additional control layers in the DNF algorithm to detect and resolve deadlock situations. Using this augmented DNF strategy, the success rate improved with target convergence ranging between 80% and 95% in a series of simulated configuration transitions.

The quoted range of efficiencies raises alarms when considering implementing this strategy for the SDSS-V. The significant reduction in usable fibers to first order directly translates to either an incomplete survey or an increased survey duration. However, this problem is compounded in the context of SDSS-V target selection and field design (an interesting optimization problem in itself). As described in Section 2, SDSS-V will be designing fields shared between a diverse set of targets each with their own observational constraints. Field to field scheduling must happen on a nightly basis, and as the DNF algorithm behavior depends on transitions between subsequent fields, specific target loss would not be detectable on a time scale in which any iteration between target reassignment and path regeneration could feasibly happen. Furthermore, many targets will have strict requirements on the timing of multi-epoch observations.

[3] http://geomalgorithms.com/a07-_distance.html

[4] https://en.wikipedia.org/wiki/Twister_(game)

[5] https://www.epfl.ch/labs/react/

The random loss of targets like these could be impossible to patch with any mitigation strategy. The heart of the SDSS-V FPS is its ability to quickly and efficiently obtain new target configurations between observations, and a random loss of a significant number of targets on a nightly basis is an incredible burden to account for when designing and executing a complete astronomical survey. [this argument suffers because we could use DNF to get paths between folded and target just as we do with our own algorithm, really the issue is that we can't simply replace a target and fix things, we'll always get deadlocks with the DNF which means we cant ever transition, also the DNF works pretty well when run in reverse, like what we do (though it's way to complicated). gotta think about this one a little more, maybe something to talk about in the conclusion?]

5. ALGORITHM

We introduce a new anti-collision trajectory planning strategy for the SDSS-V RFP array: an algorithm that limits target loss to less than 0.06%. Our approach decouples the path dependence between any two target configurations by generating trajectories between a target state and a common folded state where $(\theta_\alpha = 0°, \theta_\beta = 180°)$ for every robot in the grid. For a given configuration the move from folded to target (the forward path), and the move from target to folded (the reverse path) are simple reversals of the same trajectory, thus only one trajectory per configuration needs to be computed. As a result, the full trajectory between one target state and the next will transition through a common folded state. This flow between two target states is indicated in Figure 8. The decoupling of robotic path planning from nightly field sequencing allows all trajectories to be vetted during the field design phases of survey. In the case of a deadlock event, a stuck robot may be reassigned a new target such that any deadlock is eliminated well in advance to observations.

Our algorithm uses a greedy heuristic to build a reverse path. Beginning from a target state (which is necessarily collision-free), the routine enters an outer loop in which each positioner attempts to perturb its current $(\theta_\alpha, \theta_\beta)$ positions by a small angle Δ_θ. The set of possible perturbations is the combination of $\{-\Delta_\theta, 0, +\Delta_\theta\}$ moves for each axis. The perturbation set is ranked, favoring 'folding' moves over 'unfolding' moves. The positioner selects the highest ranked move that doesn't result in a collision. If collisions inhibit all perturbations or if the robot has reached $(\theta_\alpha = 0°, \theta_\beta = 180°)$, the positioner remains static until the next loop iteration. The routine ends when either all robots have achieved a folded configuration, or a maximum outer loop iteration is reached, usually indicating a deadlock.

The key to the success of this algorithm is the realization that building a reverse path is miraculously easy, while building a forward path is quite difficult. In practice, an analogous forward path heuristic (stepping towards a target configuration rather than a folded configuration) utterly fails to achieve target convergence. [Why does directionality matter? To me the routine makes a sort of sense where in each iteration beta arms are becoming more separated, which continuously opens up more free space for motion. Maybe Hogg can come up with something beautiful involving entropy and irreversible processes]. As a simple heuristic in which all computations boil down to vector operations in a 3D space, this algorithm is quite computationally efficient. Equipped with a fast path generator, the SDSS-V FPS is a flexible and nimble system in which reconfiguration is possible on the fly, should the need arise. An obvious use case being the possibility of injecting a target of opportunity into an existing field design on short notice.

A complete pseudocode implementation of this routine is provided in Appendix A.

6. ALGORITHM PERFORMANCE

SDSS-V's Kaiju[6] package is an open source Python-wrapped C++ package implementing the collision avoidance algorithm. Kaiju routines closely follow the logic described in Section 5, though the package itself contains additional specifics related to the SDSS-V FPS hardware and layout. The results we present here were obtained using the Kaiju package [tag xxx] compiled with clang++ (version 10.0.1) using an -O3 optimization flag running on a 2.9 GHz Intel Core i9 CPU.

6.1. *Initialization*

In the analysis that follows, we consider the following parameter settings:

[6] https://github.com/sdss/kaiju

number of positioners: 547
alpha arm length (l_α): 7.4 mm
beta arm length (l_β): 15 mm
pitch (distance between adjacent robots): 22.4 mm
angular step (Δ_θ): 30 linearly spaced samples in the range [0.01, 1] degrees
collision buffer (σ_{buff}): 10 linearly spaced samples in the range [1.5, 3] mm
maxIter: 700 / Δ_θ

The arm lengths and pitch are selected to match that of the SDSS-V FPS. A hexagonal array of 547 robots was chosen as it is the smallest filled hex pattern with at least as many robots as the SDSS-V FPS. The maxIter value was set such that each robot axis may accumulate up to 700° of motion before the maximum loop iteration termination criterion is met. The σ_{buff} and Δ_θ parameters are varied.

The Δ_θ parameter is varied between 0.01 and 1 degrees. Largely, the setting of this parameter is a trade off between path smoothness and program execution time. Figure 10 will provide an intuition for the effect Δ_θ has on path smoothness. The figure plots a positioner's beta axis trajectory as it navigates around a neighbor at both the maximum and minimum settings of Δ_θ.

The σ_{buff} parameter has dramatic effect on algorithm performance. As σ_{buff} grows, the available free space in the robot array becomes choked, and chance of interference increases. We choose to investigate a range of σ_{buff} settings between 1.5 mm and 3 mm. A 1.5 mm σ_{buff} exactly encloses the physical size of an SDSS-V beta arm. For σ_{buff} greater than 3 mm, it becomes difficult to find non-colliding target orientations, indicating a robot size upper limit for feasibly carrying out target selection. The visualization in Figure 9 shows the relative area occupied by both 1.5 mm and 3 mm collision envelopes. In deployment with a real system, σ_{buff} must be chosen carefully. It must be large enough to encompass the physical extent of the beta arm, plus any uncertainty in position for a real robot along its trajectory to ensure no physical collision is possible. In practice, the σ_{buff} parameter must also be set to sufficiently defend against a 'tunneling' move in which a positioner may jump past a neighbor's beta arm.

For each Δ_θ, σ_{buff} we compile path generation results from 1000 randomly initialized grids in which every robot is assigned a target drawn uniformly from its annular patrol area. If any target assignment results in a collision with a neighbor, that robot is re-assigned a target until the collision vanishes, ensuring a completely non-colliding starting point for the routine. A random seed is set such that for each trial index, the initial target orientation is identical for all σ_{buff} values. However, the initial target orientations are not usually identical between σ_{buff} settings, due to the fact that non-colliding target orientations at a lower setting of σ_{buff} may become collided as the parameter grows.

6.2. *Runtime*

The total execution time of the collision-avoidance routine increases exponentially with decreasing Δ_θ. For the finest step size considered ($\Delta_\theta = 0.01°$) we measure a mean runtime of ∼40 seconds, while the coarsest step size ($\Delta_\theta = 1°$) has a mean runtime of ∼200 milliseconds. Trials often hit the max iteration limit for the largest settings of σ_{buff}, which skews these estimated runtimes high. Mean runtimes for the varied settings of Δ_θ and σ_{buff} are shown in Figure 11. This is a fast algorithm, which is an important result for at least two reasons: it may feasibly be used for large scale survey simulations, and trajectories may be computed on short notice, should the need arise during a nightly observing sequence.

6.3. *Target Convergence*

The most important feature of this algorithm is its extremely high efficiency at finding non-colliding paths for an RFP array. Here we define target convergence as the percentage of positioners achieving a folded configuration integrated over all trials. As shown in Figure 12, convergence depends strongly on σ_{buff} and less so on Δ_θ. In general, for σ_{buff} values less than 2 mm the routine finds a non-colliding path for over 99.7% of robots. The convergence curve turns sharply downward beyond $\sigma_{buff} = 2.25$ mm with a clear dispersion of values with Δ_θ.

6.4. *Deadlock Resolution*

This algorithm is not 100% efficient, so each path generation trial will risk some probability of deadlock. Figure 14 shows the probability that a trial will experience a deadlock as a function of the varied parameters for a grid of 547 robots. For σ_{buff} values greater than 2.25 mm deadlocks are virtually guaranteed. For σ_{buff} near the 1.5 mm lower limit, fewer than 1 in 5 trials will suffer a deadlock. Examples of deadlocked configurations over a range of σ_{buff} values

are visualized in Figure 13. Because we employ a reverse path solver, valid robot trajectories require solutions from trials in which no deadlocks exist. Bluntly, paths are useless unless all robots converge, so the utility of this routine ultimately depends on a reasonable deadlock resolution strategy.

Deadlock resolution could be injected into the main path generation routine. A proposed strategy might be to detect halted progress during the folding process and reverse the ranking of the perturbation set for a single positioner involved in a deadlock for a period of steps. This reversal in a single positioner's preferred direction tends to allow a previously deadlocked neighbor to pass. Prototype examples of this indeed show an increase in target convergence at the cost of a slightly more complicated algorithm with extra parameters to tune, and a potential for increased total reconfiguration time. The example that follows demonstrates an alternate method to resolve deadlocks requiring no modification to the algorithm presented in Section 5.

A brute force deadlock resolution method is an iterative approach that achieves as many of the original targets as possible. The procedure is: 1) run the path generator, 2) randomly select a deadlocked robot, 3) replace its target. Iterate steps 1-3 until the whole array converges. Because deadlocks typically involve 2-3 positioners, a single replacement will often eliminate 2 or more deadlocks. We test this procedure for 100 randomly selected deadlocked trials for a subrange of $\sigma_{buff} = \{1.5, 2.0, 2.5, 3.0\}$ mm. We set $\Delta_\theta = 1^\circ$ in the interest of fast computation times. The results of this test are shown in Figure 15, and may be generally summarized in two regimes: the bloated robot regime ($\sigma_{buff} > 2.25$ mm) and the skinny robot regime ($\sigma_{buff} < 2.25$ mm). Bloated robots exhibit a 100% chance of initial deadlock, and show a wide distribution of replacement attempts to resolve a deadlock. Adopting the median value of 10 replacements for an array of 547 positioners this equates to a 2% loss of targets due to collision avoidance constraints. Skinny robots exhibit a 34% chance of initial deadlock, typically requiring only a single target replacement to resolve it. For an array of 547 positioners this equates to a mere 0.06% loss of targets. SDSS-V positioners will fit safely into the skinny robot regime, and so the SDSS-V FPS should suffer negligible target loss due to anti-collision trajectory planning using a brute force deadlock resolution technique.

6.5. *Reconfiguration Time*

The final metric we present is the reorientation time between target configurations. When considering the several thousands of fields that SDSS-V will observe over the course 5 years, the speed at which the FPS transitions between target arrangements is an important factor to understand. To estimate this value we consider the robot's angular speed ($\dot{\theta} = 30$ deg/sec), the angular step size (Δ_θ) and the total steps taken to fold (n_{steps}). The time to move from target to fold is then estimated by:

$$\tau_{fold} = \frac{\Delta_\theta \mathrm{n}_{steps}}{\dot{\theta}}$$

The distributions of τ_{fold} for the deadlock resolved trials presented in Section 6.4 are shown in Figure 16. Here we see that the choice of σ_{buff} has only a slight effect on the folding time for the array. Marginalizing over all σ_{buff}, we determine the time to fold is 13 $\pm$1 seconds. So roughly we expect the target to target transition time to be 26 ± 2 seconds, regardless of σ_{buff} parameter choice. We see no dependence of τ_{fold} on Δ_θ choice. The following section will discuss additional effects that will increase reconfiguration time for the SDSS-V FPS.

A real-time example of a transition between target states is shown in Figure 17.

7. CONCLUSION

The conclusion.

APPENDIX

A. PSEUDOCODE IMPLEMENTATION

A pseudocode implementation of the anti-collision algorithm to generate a reverse path for an RFP array follows below, favoring clarity over computational efficiency. The routine assumes that the functions **dot(v1, v2)** and **norm(v1)** exist and return the dot product and Euclidean norm for input vectors **v1, v2**. Vector subtraction and scalar multiplication is defined in the normal linear algebra sense.

Algorithm 1: Reverse Path Generator

```
# ----------------------
# runtime parameters
# ----------------------

alphaArmLen ← length of alpha arm, l_α
betaArmLen ← length of beta arm, l_β
collisionBuffer ← buffer that describes the collision envelope, σ_buff
dTheta ← maximum angular step size for a positioner axis perturbation, Δ_θ
maxIter ← maximum iterations to perform


# ------------------------
# structure definitions
# ------------------------

struct Robot:
  # holds relevant coordinates and neighbors for a given positioner
  x_b ← x coordinate of positioner base position, x_b
  y_b ← y coordinate of positioner base position, y_b
  alphaAng ← alpha angle, θ_α
  betaAng ← beta angle, θ_β
  neighbors ← list of Robot instances with whom I risk collision

struct Vector3:
  # simply, a 3-vector
  x ← x coordinate
  y ← y coordinate
  z ← z coordinate

struct LineSegment:
  # a container representing a line segment between two Vector3's v0 and v1
  v0 ← an instance of Vector3
  v1 ← an instance of Vector3


# ------------------------
# function definitions
# ------------------------

function dist3D_Line_to_Line(L1, L2):
  # Almost verbatim from Dan Sunday's algorithm: http://geomalgorithms.com/a07-_distance.html
  parameters: L1, L2 each a LineSegment
  output: the minimum distance between L1 and L2.

  SMALL_NUM ← 0.00000001 # anything that avoids division overflow
  u ← L1.v1 - L1.v0
  v ← L2.v1 - L2.v0
  w ← L1.v0 - L2.v0
  a ← dot(u,u)        # always >= 0
  b ← dot(u,v)
  c ← dot(v,v)        # always >= 0
  d ← dot(u,w)
  e ← dot(v,w)
  D ← a*c - b*b       # always >= 0

  # compute the line parameters of the two closest points
  if (D < SMALL_NUM):          # the lines are almost parallel
    sc ← 0.0
```

```
    if (b > c):
      tc ← d / b
    else:
      tc ← e / c
  else:
    sc ← (b*e - c*d) / D
    tc ← (a*e - b*d) / D

  # get the difference of the two closest points
  dP ← w + (sc * u) - (tc * v)
  return norm(dP)


function betaArmSegment(jaeger):
  # This implements equations 1 and 2 in Section 3.
  parameters: jaeger, a Robot
  output: the orientation of the beta arm, a LineSegment

  elbow ← Vector3()
  elbow.x ← jaeger.x_b + alphaArmLen*cos(jaeger.alphaAng)
  elbow.y ← jaeger.y_b + alphaArmLen*sin(jaeger.alphaAng)
  elbow.z ← 0 # we work in the z=0 focal plane

  fiber ← Vector3()
  fiber.x ← elbow.x + betaArmLen*cos(jaeger.alphaAng + jaeger.betaAng)
  fiber.y ← elbow.y + betaArmLen*sin(jaeger.alphaAng + jaeger.betaAng)
  fiber.z ← 0 # we work in the z=0 focal plane

  betaSeg ← LineSegment()
  betaSeg.v0 ← elbow
  betaSeg.v1 ← fiber

  return betaSeg


function isCollided(jaeger):
  parameters: jaeger, a Robot
  output: True when input Robot is collided with a neighbor, otherwise False

  seg1 ← betaArmSegment(jaeger)

  foreach neighbor in jaeger.neighbors:
    seg2 ← betaArmSegment(neighbor)
    armSeparation ← dist3D_Line_to_Line(seg1, seg2)
    if armSeparation < 2*collisionBuffer:
        # the two robots are collided
        # exit now
        return True

  # not collided with any neighbor
  return False


function perturbRobot(jaeger):
  parameters: jaeger, a Robot
  output: modify state of input Robot

  # if the positioner is already folded, do not perturb
  if jaeger.alphaAng == 0 && jaeger.betaAng == 180:
```

```
    return

  # build the ranked list of (Δθα, Δθβ) angular perturbations
  # favor folding moves over un-folding moves
  rankedMoves ← [
    [-dTheta,  dTheta],
    [      0,  dTheta],
    [ dTheta,  dTheta],
    [-dTheta,       0],
    [ dTheta,       0],
    [-dTheta, -dTheta],
    [      0, -dTheta],
    [ dTheta, -dTheta]
  ]

  # save the current state of the robot's position
  currAlpha ← jaeger.alphaAng
  currBeta ← jaeger.betaAng

  foreach dAlpha, dBeta in rankedMoves:
    nextAlpha ← currAlpha + dAlpha
    nextBeta ← currBeta + dBeta

    # don't allow out of range moves
    if nextAlpha < 0:
      nextAlpha ← 0
    if nextAlpha > 360:
      nextAlpha ← 359.99999
    if nextBeta < 0:
      nextBeta ← 0
    if nextBeta > 180:
      nextBeta ← 180

    # set and test the new position
    jaeger.alphaAng ← nextAlpha
    jaeger.betaAng ← nextBeta
    if !isCollided(jaeger):
      # move didn't collide positioner, exit here
      return

  # no moves were possible, reset to initial positions
  jaeger.alphaAng ← currAlpha
  jaeger.betaAng ← currBeta
  return

function arrayConverged(robotList):
  parameters: robotList, a list of Robot instances
  output: True when all robots have folded, otherwise False

  foreach jaeger in robotList:
    if jaegar.betaAng != 180 or jaeger.alphaAng !=0:
      # this robot has not converged, exit here
      return False

  # all robots have converged
  return True


function recordState(robotList):
```

```
    # a function to record the current positions of all Robot
    # instances in the array, called throughout the routine
    # the record the path of each positioner
    parameters: robotList, a list of Robot instances
    output: record the current state the Robot array


# ----------------------
# begin main algorithm
# ----------------------

# procedure requires a list of each Robot initialized with:
# grid coordinates (x_b, y_b),
# non-colliding target coordinates (θ_α, θ_β),
# and a list of neighboring Robot instances that risk beta arm interference
allRobots ← list of initialized Robot instances

# record the initial state (all robots at targets)
recordState(allRobots)

# set things in motion
loopIter ← 1
while loopIter < maxIter:

    foreach jaeger in allRobots:
      perturbRobot(jaeger)

    # record new state of robots
    recordState(allRobots)

    # check for convergence
    if arrayConverged(allRobots):
      # all robots have reached a folded state, exit loop
      break

    loopIter ← loopIter + 1

# ----------------------
# end main algorithm
# ----------------------
```

REFERENCES

Blanton, M. R., Bershady, M. A., Abolfathi, B., et al. 2017, AJ, 154, 28, doi: 10.3847/1538-3881/aa7567

Bowen, I. S., & Vaughan, Jr., A. H. 1973, ApOpt, 12, 1430, doi: 10.1364/AO.12.001430

Bundy, K., Bershady, M. A., Law, D. R., et al. 2015, ApJ, 798, 7, doi: 10.1088/0004-637X/798/1/7

Drory, N., MacDonald, N., Bershady, M. A., et al. 2015, AJ, 149, 77, doi: 10.1088/0004-6256/149/2/77

Eisenstein, D. J., Weinberg, D. H., Agol, E., et al. 2011, AJ, 142, 72, doi: 10.1088/0004-6256/142/3/72

Falconi, R., & Melchiorri, C. 2008, IFAC Proceedings Volumes, 41, 44 , doi: https://doi.org/10.3182/20080408-3-IE-4914.00009

Gaia Collaboration, Prusti, T., de Bruijne, J. H. J., et al. 2016, A&A, 595, A1, doi: 10.1051/0004-6361/201629272

Ge, F., Wei, Z., Lu, Y., Tian, Y., & Li, L. 2012, Nonlinear Dynamics, 70, 571

Ge, J., Lee, B., Lee, N. D., et al. 2009, in Techniques and Instrumentation for Detection of Exoplanets IV, ed. S. B. Shaklan, Vol. 7440, International Society for Optics and Photonics (SPIE), 187 – 196, doi: 10.1117/12.826651

Gunn, J. E., Siegmund, W. A., Mannery, E. J., et al. 2006, AJ, 131, 2332, doi: 10.1086/500975

Kollmeier, J. A., Zasowski, G., Rix, H.-W., et al. 2017, arXiv e-prints, arXiv:1711.03234. https://arxiv.org/abs/1711.03234

Majewski, S. R., Schiavon, R. P., Frinchaboy, P. M., et al. 2017, AJ, 154, 94, doi: 10.3847/1538-3881/aa784d

Makarem, L., Kneib, J.-P., Gillet, D., et al. 2014, A&A, 566, A84, doi: 10.1051/0004-6361/201323202

Predehl, P., Andritschke, R., Böhringer, H., et al. 2010, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 7732, Proc. SPIE, 77320U, doi: 10.1117/12.856577

Ricker, G. R., Winn, J. N., Vanderspek, R., et al. 2014, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 9143, Proc. SPIE, 914320, doi: 10.1117/12.2063489

Shen, Y., Brandt, W. N., Dawson, K. S., et al. 2015, ApJS, 216, 4, doi: 10.1088/0067-0049/216/1/4

Smee, S. A., Gunn, J. E., Uomoto, A., et al. 2013, AJ, 146, 32, doi: 10.1088/0004-6256/146/2/32

Tao, D., Makarem, L., Bouri, M., Kneib, J.-P., & Gillet, D. 2018, in Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series, Vol. 10702, Proc. SPIE, 107028K, doi: 10.1117/12.2313962

Wilson, J. C., Hearty, F. R., Skrutskie, M. F., et al. 2019, PASP, 131, 055001, doi: 10.1088/1538-3873/ab0075

York, D. G., Adelman, J., John E. Anderson, J., et al. 2000, The Astronomical Journal, 120, 1579, doi: 10.1086/301513

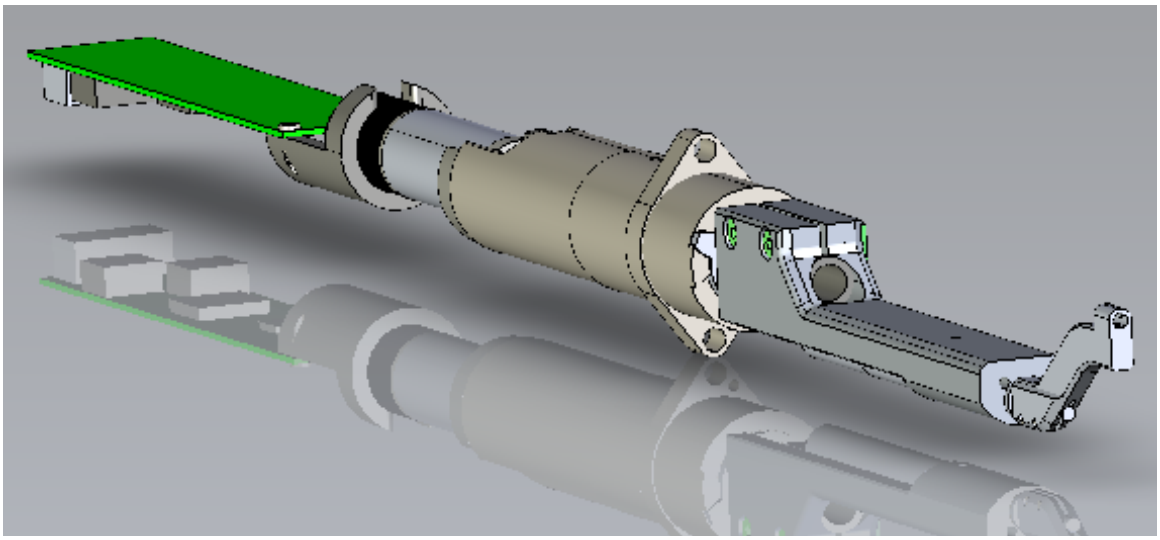

Figure 1: A solid model view of an SDSS-V robotic fiber positioner.

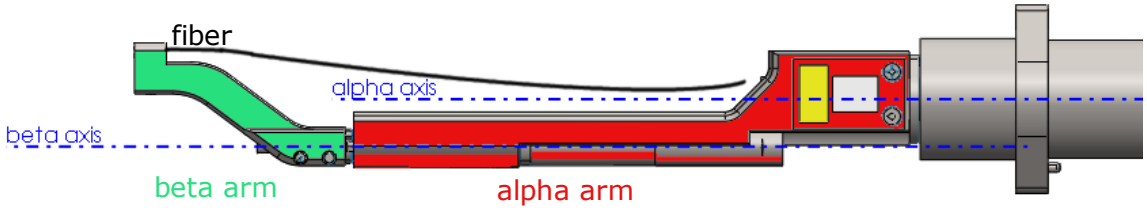

Figure 2: Side view of SDSS-V RFP indicating rotation axes, arms, and an optical fiber. The alpha arm (red) rotates about the alpha axis. The beta arm (green) rotates about the beta axis. Perhaps pick the same colors as the focal projection plots?

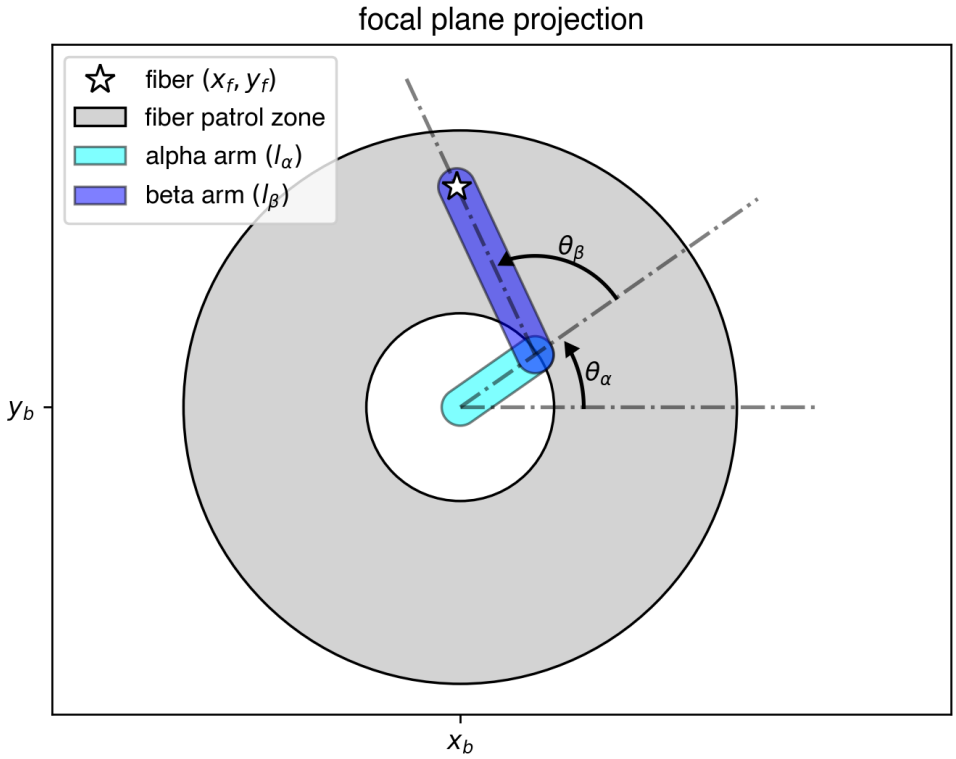

Figure 3: A focal plane projection of an SDSS-V robot, showing the annular patrol zone. Fiber position in the focal plane (x_f, y_f) is achieved through the specification of two angular rotations θ_α and θ_β of the alpha and beta arms.

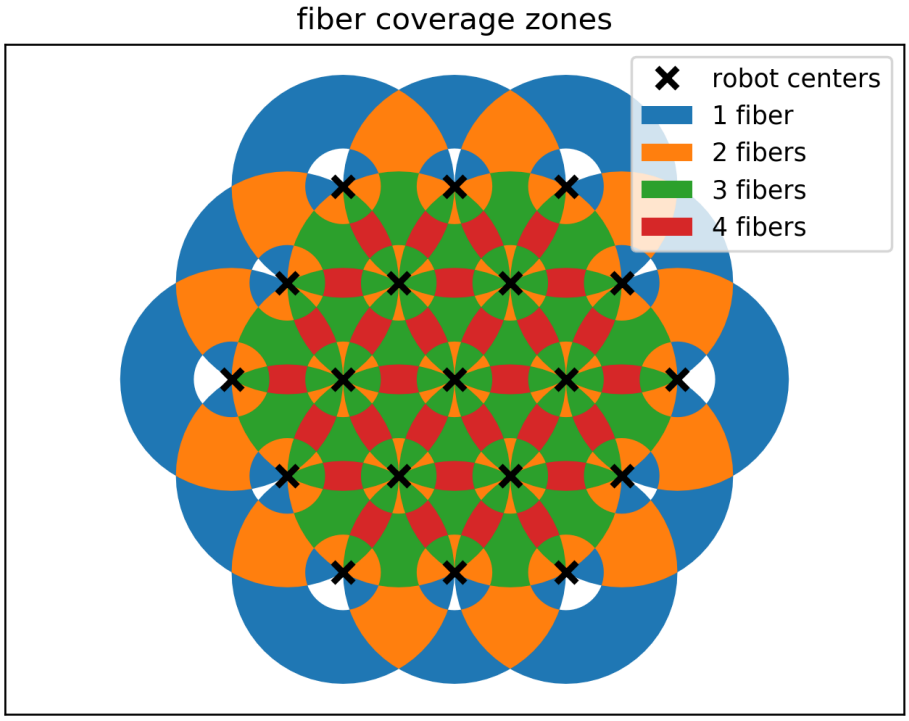

Figure 4: A 19 positioner hexagonal array, indicating the focal plane area covered by 1, 2, 3, or 4 fibers. The array perimeter will be covered by one or two fibers, and contain small gaps. Most points in the interior area will be accessible to 3 or more fibers.

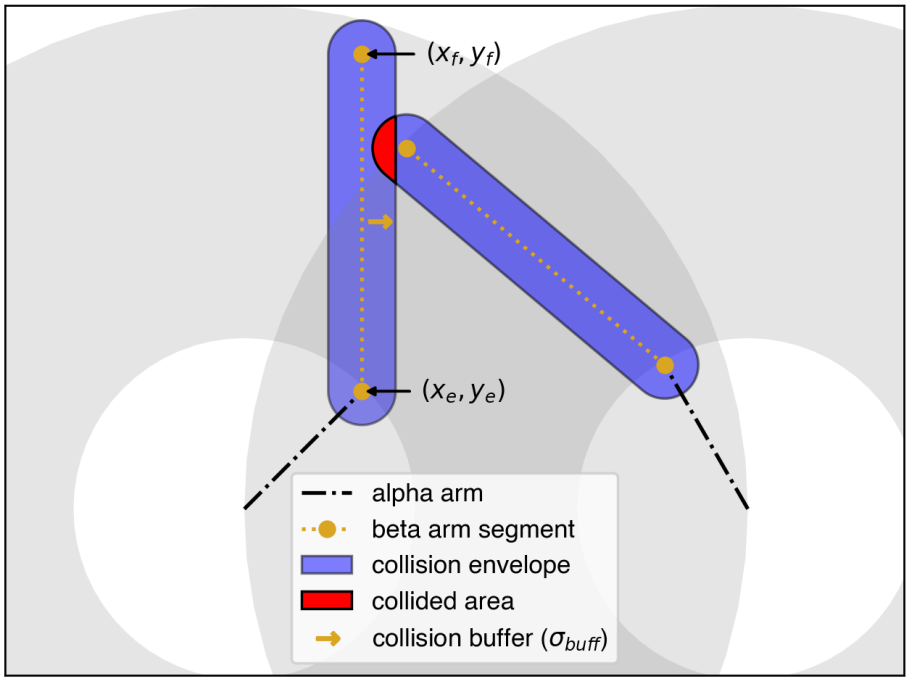

Figure 5: Geometric representations of collided beta arms. The beta arm is described by a line segment constructed of two points and a collision buffer σ_{buff} specifying a distance from the line segment. The volume generated by the line segment and σ_{buff} is the collision envelope, the collision envelope contains the physical extent of the beta arm. When collision envelops intersect (indicated as red in the figure), we say the two robots are collided.

Figure 6: Nature driven example of non-colliding, yet overlapping beta arms.

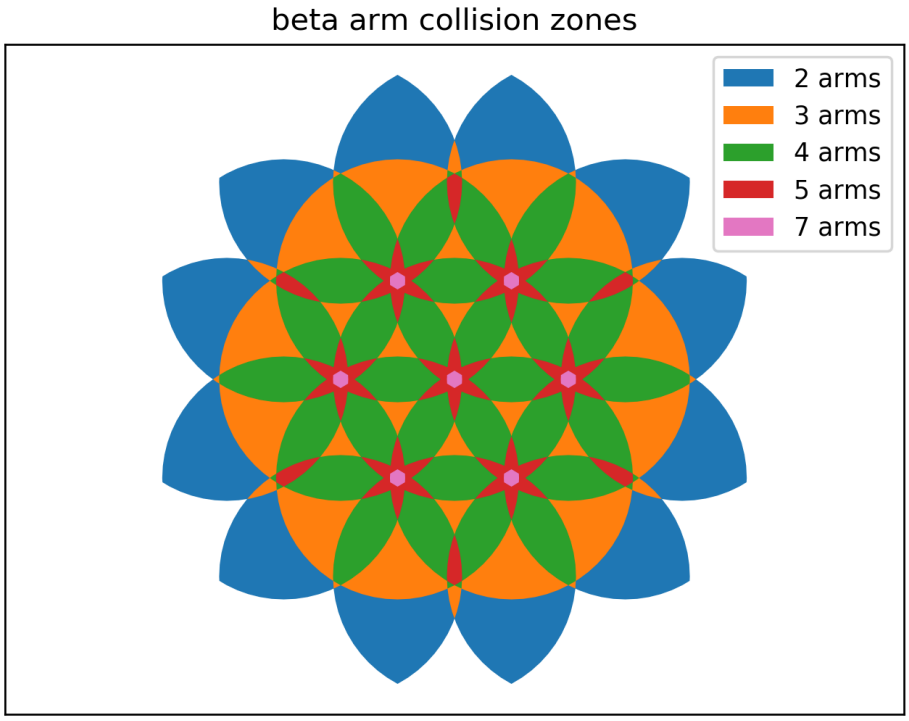

Figure 7: A 19 positioner hexagonal array mapping the intersections between 2, 3, 4, 5, and 7 beta arm collision envelopes, indicating a high collision potential within the array.

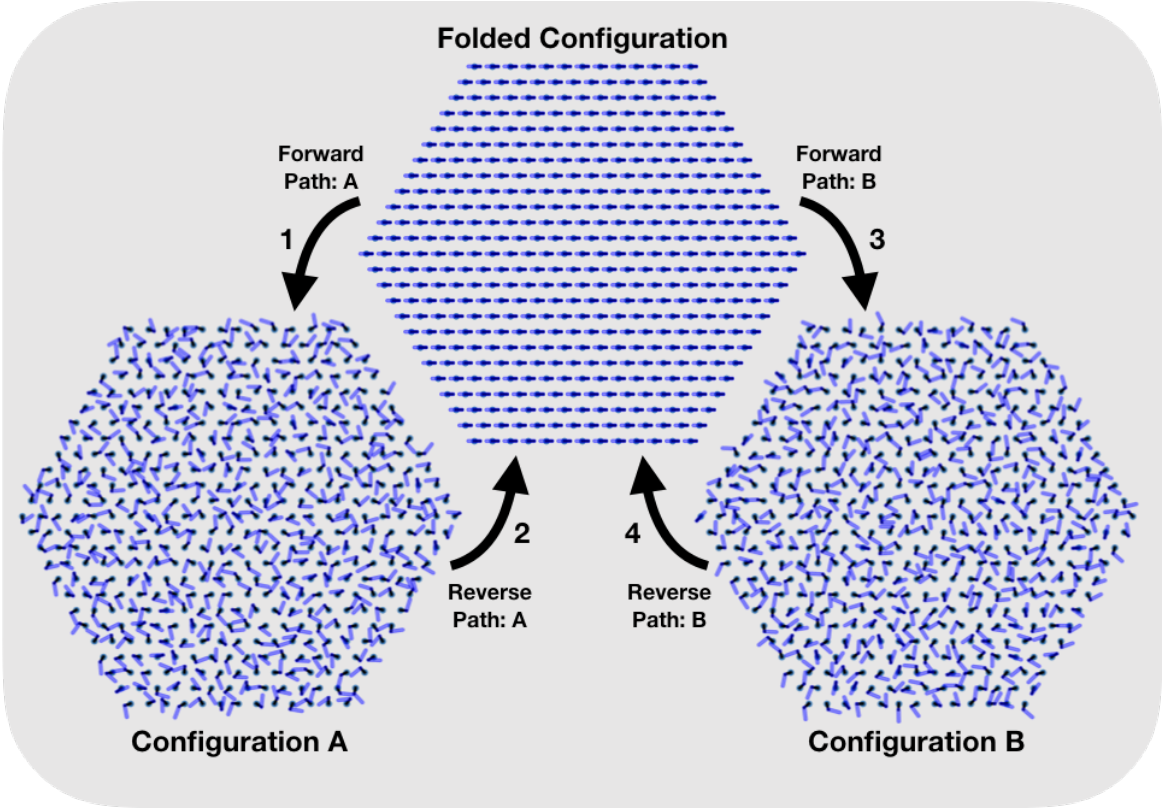

Figure 8: A flow showing the sequencing of paths required to move between one target configuration A and another target configuration B. The folded state ($\theta_\alpha = 0°, \theta_\beta = 180°$) is the common state between which all paths to all target configurations are generated. Forward and reverse paths are simple reflections of one another, so they need only be computed in one direction. 469 RFPs are included in this grid and the blue regions indicate the collision envelopes of the RFPs.

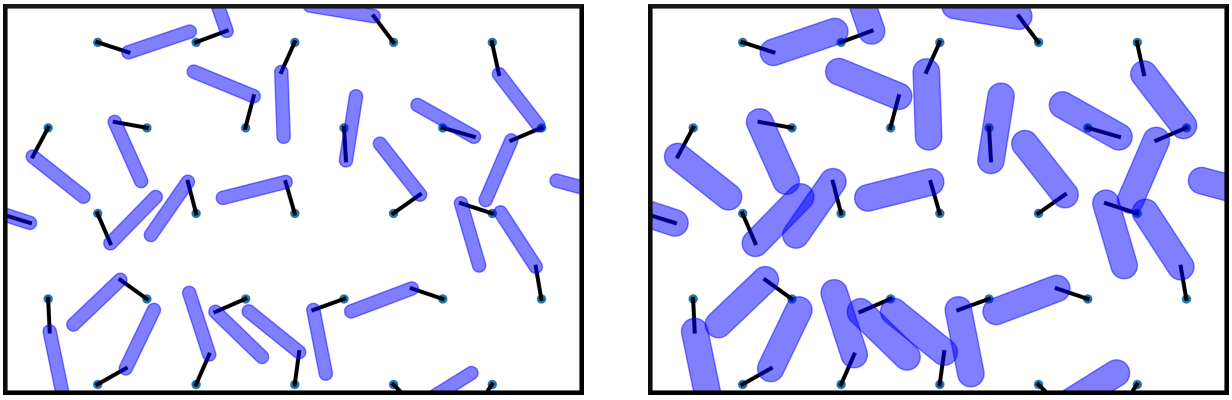

Figure 9: A visual comparison for the maximum and minimum σ_{buff} investigated. The left panel shows collision envelopes for a field of positioners with $\sigma_{buff} = 1.5$ mm. The right panel shows collision envelopes (some of them now intersecting) with a $\sigma_{buff} = 3$ mm for positioners in an identical orientation. Blue regions indicate the collision envelopes of the fiber positioners. Black lines indicate the alpha arms of the fiber positioners.

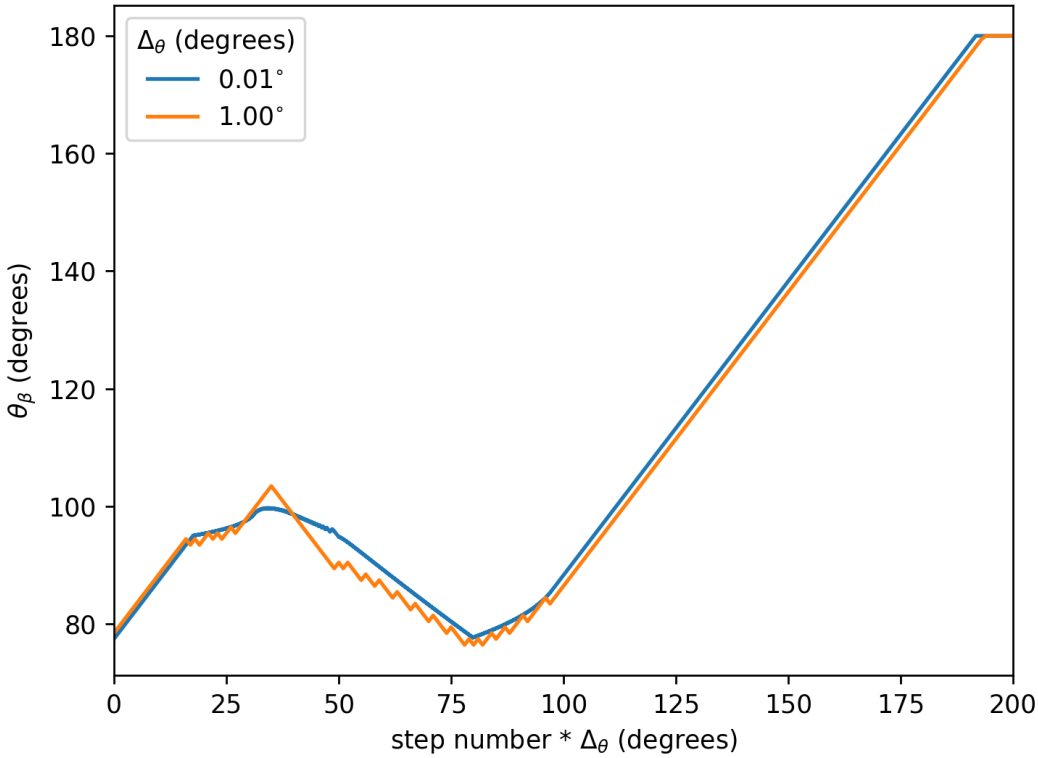

Figure 10: A positioner's beta axis trajectory solved using an angular step size (Δ_θ) of 0.01° and 1°, showing the difference in path smoothness between the two settings. The positioner begins at a target θ_β position of 78.4°, and steps towards the folded state ($\theta_\beta = 180°$). In this path example, the positioner encountered a neighbor requiring changes in beta axis velocity and direction to successfully navigate to the folded state.

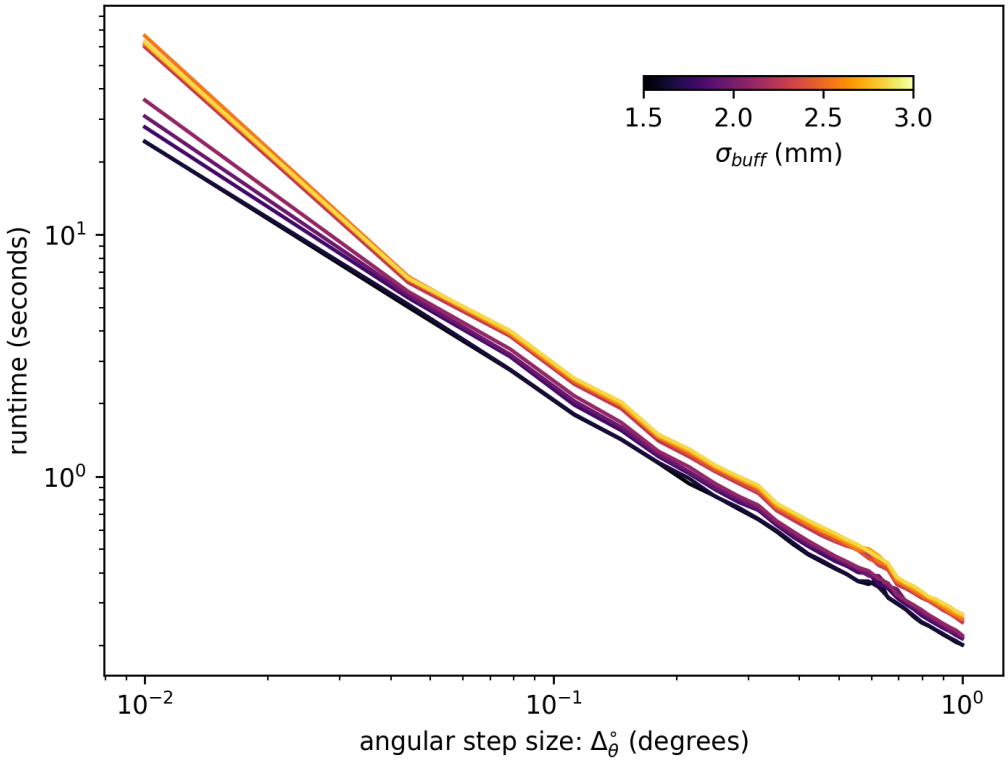

Figure 11: Mean runtimes over the varied ranges of σ_{buff} and Δ_θ, averaged over 1000 trials at each parameter combination.

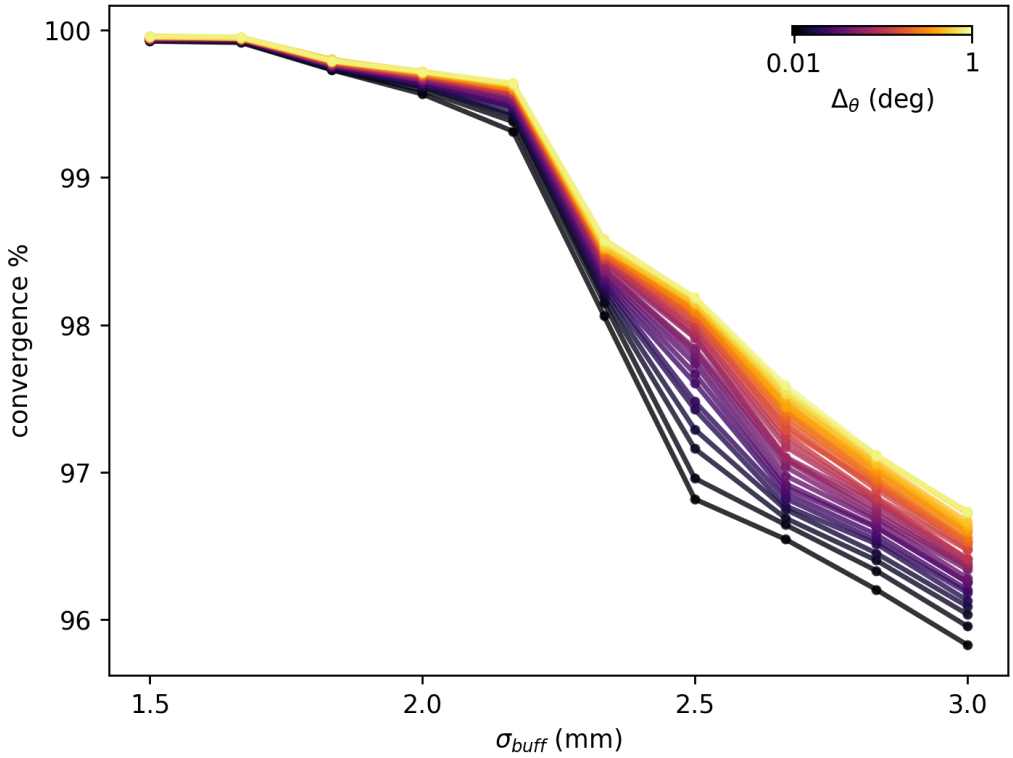

Figure 12: Percent of robots achieving a folded state computed from 1000 trials at each σ_{buff} and Δ_θ. For $\sigma_{buff} < 2$ mm we see convergence for $> 99.7\%$ of positioners over the full range of Δ_θ. Convergence decreases with σ_{buff} with a sharp downturn around $\sigma_{buff} = 2.25$ mm. Dispersion in measured convergence scales more strongly with Δ_θ for the larger values of $\sigma buff$.

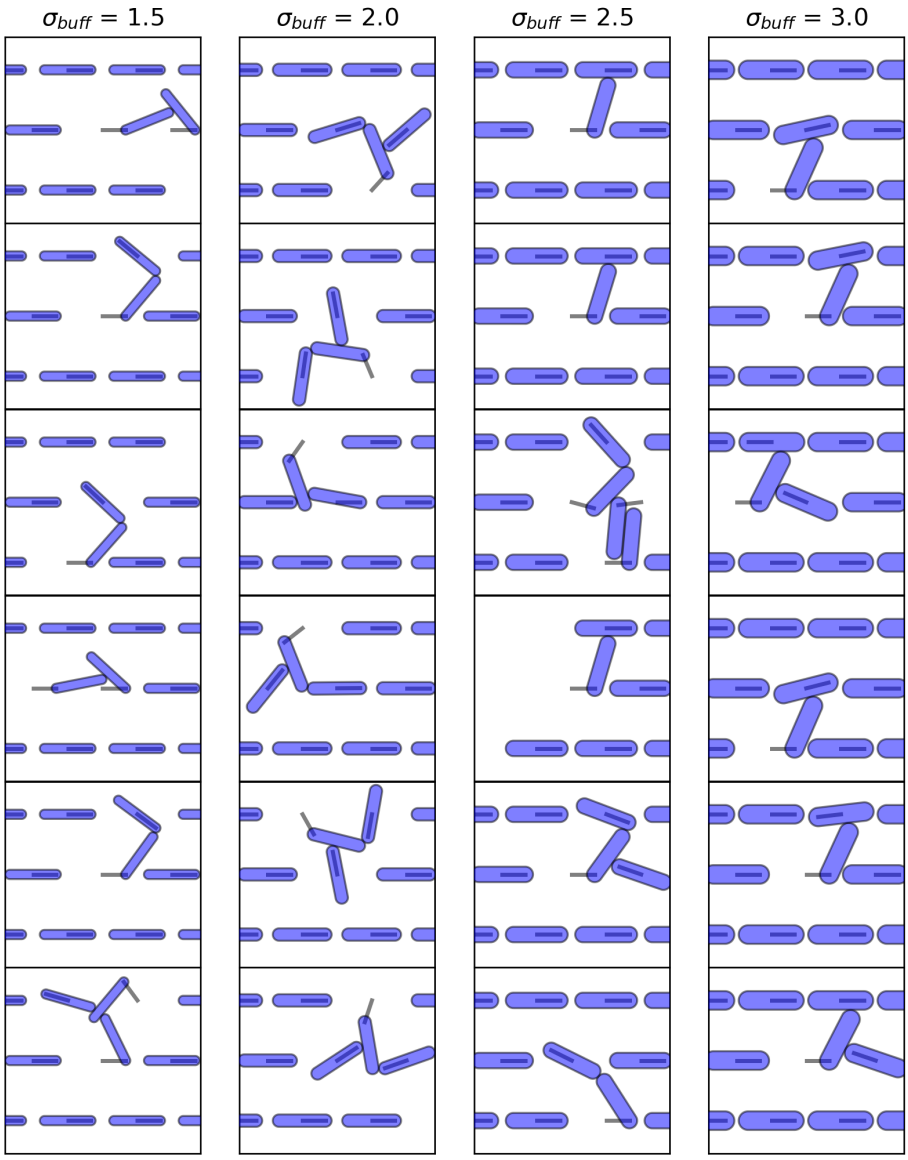

Figure 13: Examples of deadlocked positioners.

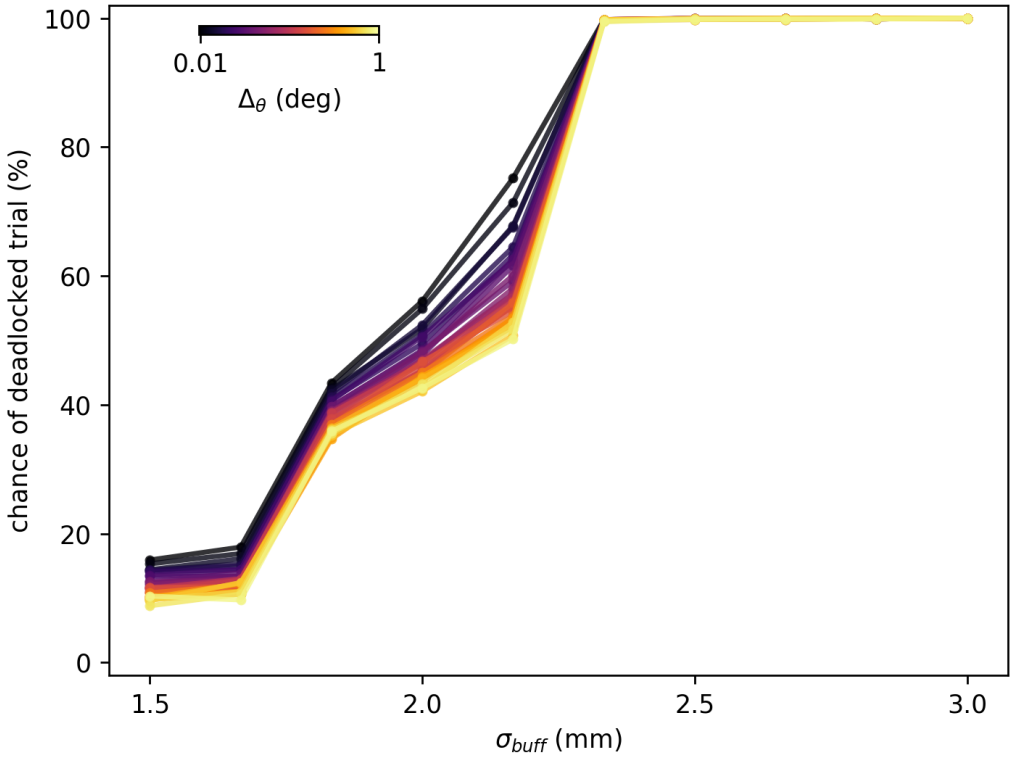

Figure 14: Chance of a trial deadlocking.

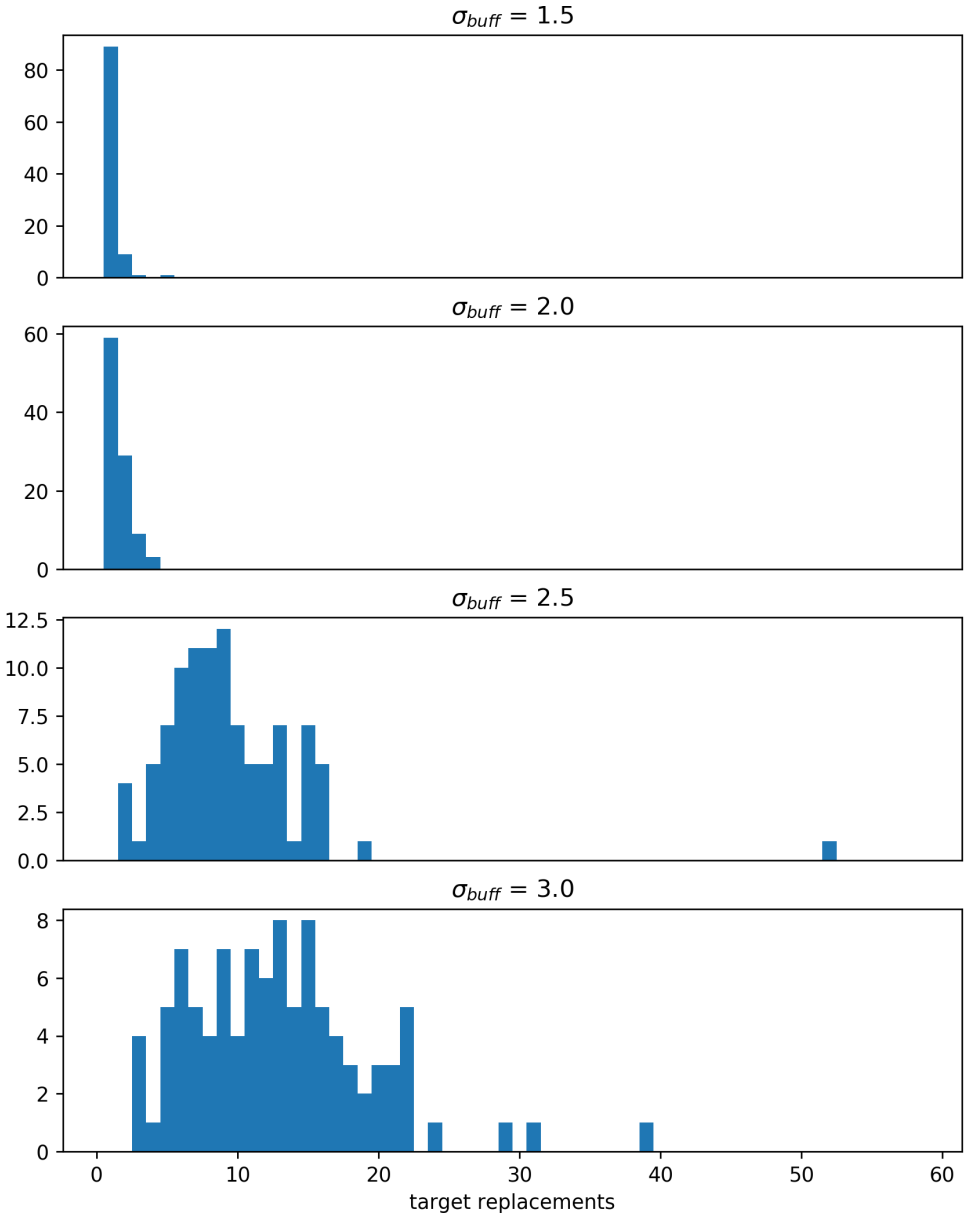

Figure 15: Required target replacements to resolve 100 deadlocked trials. Each trial uses a grid of 547 positioners over a range of σ_{buff} settings.

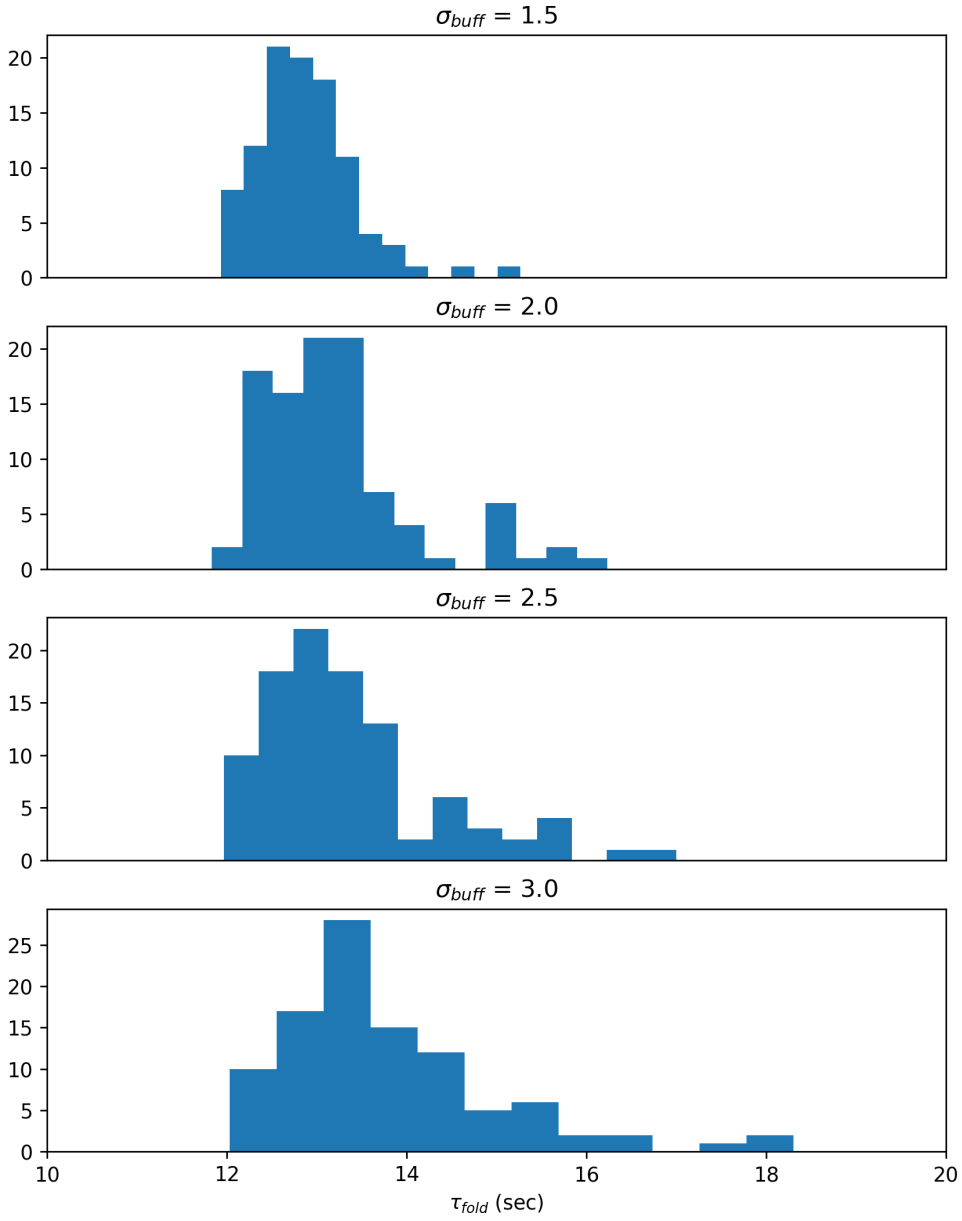

Figure 16: Histograms of folding times for 100 trials.

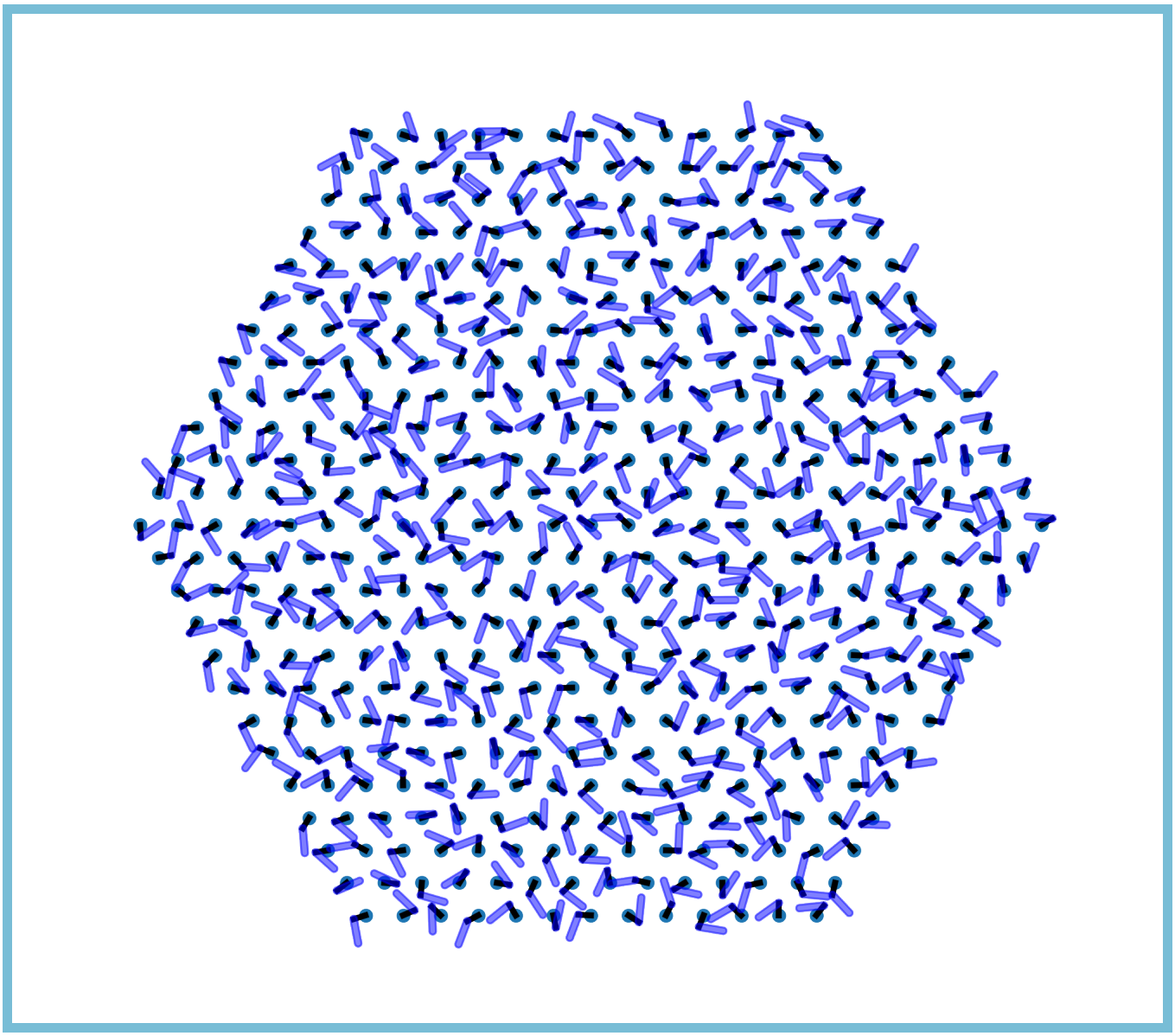

Figure 17: Real time example of transition between target states.