An Open-Source, Reproducible Tensegrity Robot That Can Navigate Among Obstacles

William R. Johnson III, *Member, IEEE*, Patrick Meng, Nelson Chen, Luca Cimatti, Augustin Vercoutere, Mridul Aanjaneya, Rebecca Kramer-Bottiglio, and Kostas E. Bekris, *Senior Member, IEEE*

***Abstract*—Tensegrity robots, composed of rigid struts and elastic tendons, provide impact resistance, low mass, and adaptability to unstructured terrain. Their compliance and complex, coupled dynamics, however, present modeling and control challenges, hindering planning and obstacle avoidance. This letter presents a complete, open-source, and reproducible system that enables navigation for a 3-bar tensegrity robot. The system comprises: (i) an inexpensive, open-source hardware design, and (ii) an integrated, open-source software stack for physics-based modeling, system identification, state estimation, path planning, and control. All hardware and software are publicly available at tensegrity.yale.edu. The proposed system tracks the robot using a static overhead camera and executes collision-free paths to a goal among known obstacle locations. System robustness is demonstrated through experiments involving unmodeled environmental challenges, including a vertical drop, an incline, and granular media, culminating in an outdoor field demonstration. To validate reproducibility, experiments were conducted using robot instances at two different laboratories. This work provides the robotics community with a complete navigation system for a compliant, impact-resistant, and shape-morphing robot. This system is intended to serve as a springboard for advancing the navigation capabilities of other unconventional robotic platforms.**

***Index Terms*—Open source hardware, open source software, path planning, soft robotics.**

I. Introduction

NAVIGATION over terrain with obstacles remains a challenge in robotics. Tensegrity robots made with rigid struts and elastic tendons are a promising mobile platform due to their durability, adaptability, low mass, and low cost [1]. Their deformations and coupled dynamics, however, create modeling and control challenges, which have hindered the demonstration of complete navigation solutions.

Received 30 October 2025; accepted 5 March 2026. Date of publication 6 April 2026; date of current version 23 April 2026. This article was recommended for publication by Associate Editor D. Shin and Editor B. Mazzolai upon evaluation of the reviewers' comments. This work was supported in part by NSF under Award IIS-1955225, Award IIS-1956027, Award CCF-2110861, Award IIS-2132972, Award IIS-2238955, and Award CCF-2312220, and in part by Red Hat, Inc. *(William R. Johnson III and Patrick Meng are co-first authors.) (Corresponding author: Kostas E. Bekris.)*

William R. Johnson, Luca Cimatti, Augustin Vercoutere, and Rebecca Kramer-Bottiglio are with Mechanical Engineering, Yale University, New Haven, CT 06520 USA (e-mail: rebecca.kramer@yale.edu).

Patrick Meng, Nelson Chen, Mridul Aanjaneya, and Kostas E. Bekris are with Computer Science, Rutgers University, Piscataway, NJ 08854 USA (e-mail: mridul.aanjaneya@rutgers.edu; kostas.bekris@cs.rutgers.edu).

This article has supplementary downloadable material available at https://doi.org/10.1109/LRA.2026.3681109, provided by the authors.

Digital Object Identifier 10.1109/LRA.2026.3681109

Fig. 1. The open-source tensegrity robot moving around obstacles outdoors starts from the top left and autonomously navigates to the bottom right.

This letter presents a complete pipeline for navigation and obstacle avoidance using a 3-bar tensegrity robot. The work makes the following **contributions**: (i) it presents and quantitatively evaluates the robustness of a newly proposed closed-loop navigation solution against an open-loop baseline; (ii) it introduces a new hardware platform designed to support reproducible research in tensegrity navigation; (iii) it presents a software framework that integrates modeling, state estimation, and planning for closed-loop navigation; (iv) it offers the hardware and software as a complete, open-source tensegrity pipeline to the research community; (v) it demonstrates the capabilities of the integrated system across four different obstacle configurations, three distinct robot platforms, four different environmental disturbances (drops, inclines, granular media, and outdoor environments) and two different laboratories.

The software component begins with system identification and the modeling of motion primitives via a differentiable physics engine. An open-loop planner uses these primitives to generate an action sequence that moves the robot from its current pose to the goal while avoiding known obstacles. State estimation given information from an overhead static camera closes the loop, enabling replanning after each executed action to correct for discrepancies between model predictions and the robot's observed state. Navigation experiments demonstrate robustness to unmodeled disturbances, including vertical drops, inclined surfaces, and granular media, and the system's effectiveness is further validated in an outdoor field environment (Fig. 1).

The hardware component centers around a new, open-source tensegrity robot design. The hardware and software are publicly available to promote reproduction by other researchers.

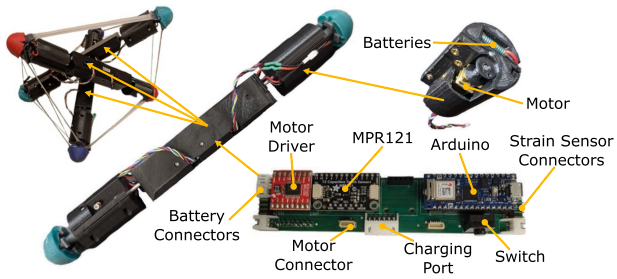

Fig. 2. Open-source design: Each strut has two motor enclosures; each housing a brushed DC motor and LiPo batteries. A custom motherboard uses commercial-off-the-shelf circuits mounted on headers, allowing for easy replacement. A power switch and charging port provide convenience. The tensegrity robot is formed by assembling three struts with strain sensors.

Reproducibility is validated through experiments at two separate laboratories that independently constructed the robot. The resulting platform supports autonomous control and teleoperation, serving as a suitable testbed for future investigations into data-driven navigation solutions.

II. Related Work

Tensegrity robotics has demonstrated diverse abilities, i.e., rolling [2], crawling [3], vibrating [4], climbing [5], and flying [6]. While many studies are constrained to laboratory settings, some have demonstrated locomotion on unstructured terrain [7] or in the field [8]. To track tensegrity robots, state estimation methods such as iterative closest point-based [9] or recent factor graph-based [10] approaches have been proposed. Tensegrity simulators have been developed either based on differential equations [11], [12], [13], or upon general-purpose physics engines [14], [15], [16], and, more recently, learning-based ones [17], [18], [19], [20], [21] to address the Sim2Real gap. Typical solutions for path planning and navigation involve geometric solutions that tile the robot's base polygons from start to goal [22], [23], often ignoring dynamics. Furthermore, hardware experiments have frequently been limited to open-loop control [24], [25].

In contrast, the approach presented here uses a physics engine to accurately model motion and utilizes the robot's current pose as feedback. This re-planning compensates for unmodeled variables, enabling effective navigation through environmental disturbances, such as vertical drops, inclines, and granular terrain, and among obstacles.

III. Robot Design

This work utilizes a 3-bar tensegrity robot, designed to be lightweight, deformable, and impact-resistant (Fig. 2). To demonstrate the generality of the proposed navigation system, experiments are conducted using three distinct versions: an initial design (described in previous work [27]), an upgraded version produced at a second laboratory, and a new open-source version developed for broad community adoption. The design files and assembly instructions for the latter two versions are available at https://tensegrity.yale.edu/.

The open-source design aims to facilitate easy reproduction and adoption by the robotics community. To this end, the new design features several key upgrades:

- A custom PCB enabling solderless (re)assembly.
- Convenient ports for peripherals and battery charging.
- WiFi communication via the Arduino Nano 33 IoT.
- A modular design where each bar contains a full electronics set, allowing the struts to be reconfigured for different tensegrity topologies.
- Open-source code and CAD files for user customization.

Mechanical Design The robot's three bars are structurally identical, differentiated only by colored end caps for pose tracking. Each bar houses two motors with winches to control tendon length. The robot features nine tendons: six are actuated by the motors, and three function as passive restoring springs. All tendons incorporate capacitive strain sensors [26]. On actuated tendons, the sensor runs in parallel with the motor's cable. On passive tendons, a more substantial sensor element is used, which dually serves as the restoring spring. These sensors enable closed-loop control of tendon lengths and assist in pose estimation. As shown in Fig. 2, the struts are assembled from 3D-printed enclosures and commercial off-the-shelf (COTS) parts. Each strut contains two motor sub-assemblies (housing motors and batteries) and a central electronics enclosure. The full robot is formed by connecting the strut assemblies with the tensegrity tendons.

Electrical Design Each bar contains an Arduino Nano 33 IoT, which communicates with an off-board base station via WiFi. The Arduino is mounted on a custom printed circuit board (PCB) that integrates a motor driver (TB6612FNG; SparkFun) and a capacitive sensing breakout board (MPR121; Adafruit). This custom PCB includes a pushbutton switch, a 2 $\times$ 2S LiPo charging port, and auxiliary ports for expansion. The compact electronics enclosure is designed to provide easy access to the switch, charging port, and Arduino's microUSB port for programming. The electrical design is open source and leverages COTS breakout boards to simplify assembly and maintenance.

Software Design The open-source software comprises a ROS package for control and the on-board Arduino code. The primary ROS control node receives strain sensor data from each Arduino to implement a low-level PID controller. This node sends a control signal to each motor based on the error between the tendon's measured length and its target length. The Arduino translates this signal into a PWM direction and speed command for the motor. A "target shape" is defined by a set of six target lengths, one for each actuated tendon. A "motion primitive" is a sequence of these target shapes that constitutes a higher-level behavior. The following sections detail the other open-source components of the navigation solution: pose estimation, modeling motion primitives, and the use of these models for path planning and navigation.

IV. Pose Estimation

The tensegrity robot's pose is tracked using a previously developed perception algorithm [9], which fuses data from an overhead camera (Intel RealSense L515) and on-board strain sensors. This algorithm operates in two steps. A local pose transformation is computed by scan-matching registered model points to observed points that fall within a maximum distance threshold and matching HSV color ranges. Then, this local transformation is jointly optimized with strain sensor measurements and known physical constraints. This process outputs the endpoints for each rod, which are used to infer the rod's pose. The rod's twist information is lost due to its radial symmetry.

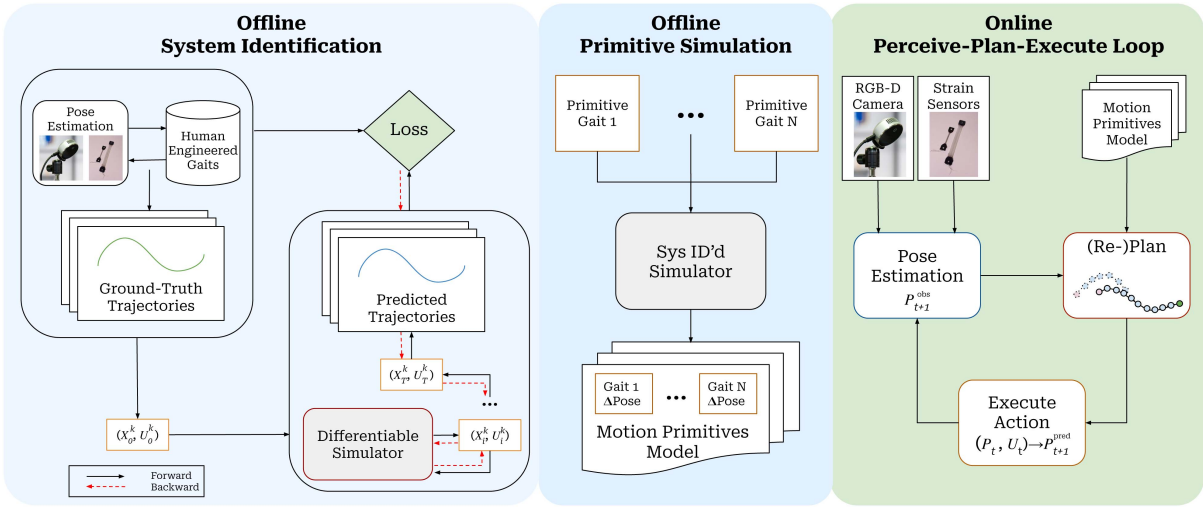

Fig. 3. (Left) Offline sysID: Trajectories are executed by chaining human-engineered gaits while tracking the robot's pose. The same start state $\mathbf{X}_0$ and controls $\mathbf{U}_{0:T-1}$ are applied to a differentiable simulator to predict these trajectories. A scalar loss between ground truth and predictions is back-propagated to update parameters. (Middle) Offline primitive modeling: The gaits are executed in the identified simulator. The change in robot pose from each primitive is stored in a motion primitive library. (Right) Online loop: As the tensegrity moves, the pose is estimated at the start and end of each primitive. Then, a plan to the goal is computed, and its first primitive action executed. This process is repeated until the tensegrity reaches the goal.

V. Modeling Motion Primitives

For path planning, the proposed navigation solution relies on predefined motion primitives and predictions of the $SE(2)$ transforms of the robot's pose that these primitives achieve. Fig. 3 highlights on the left this offline system identification (sysID) and primitive simulation process.

An accurate simulator requires identifying system parameters that best match observed data. This work leverages previous differentiable tensegrity simulators [17], [18], [19] to perform sysID via gradient-based optimization.

First-principles simulator In [17], a differentiable simulator $\mathcal{F}(\cdot)$ was developed, parameterized by Θ, with inputs of the current state and controls $(\mathbf{X}_t, \mathbf{U}_t)$, and predicts the next state $\hat{\mathbf{X}}_{t+1}$ using Newtonian physics principles and an impulse-based linear contact model:

$$\mathbf{X_{t+1}} = \mathcal{F}(\mathbf{X_t}, \mathbf{U_t}; \Theta) \tag{1}$$

Θ is optimized via gradient descent over a loss function $\mathcal{L}(\mathbf{X}_{t+1}, \hat{\mathbf{X}}_{t+1})$ between the predicted and observed output:

$$\Theta \leftarrow \Theta - \alpha \nabla_\Theta \mathcal{L} \tag{2}$$

System parameters Θ are contact parameters: (i) μ, coefficient of friction between end caps and ground; (ii) ϵ, coefficient of restitution; (iii) β, Baumgarte stabilization coefficient. Optionally, cable stiffnesses K and damping coefficients σ can also be optimized. In practice, the cables are set to have a high stiffness of 10^6 N/m.

Training process First, trajectories $(\mathcal{T}^1, \ldots, \mathcal{T}^k)$, where $\mathcal{T}^i := [(\mathbf{X}_0^i, \mathbf{U}_0^i), \ldots, (\mathbf{X}_{N-1}^i, \mathbf{U}_{N-1}^i), (\mathbf{X}_N^i)]$ are collected using human-engineered gaits. The robot's pose is tracked using the pose estimation algorithm of Section IV. Next, predicted trajectories $(\hat{\mathcal{T}}^1, \ldots, \hat{\mathcal{T}}^k)$ are generated from the ground-truth start state and recursively applying $\mathcal{F}$ N times:

$$\hat{\mathbf{X}}_N = \mathcal{F}^N(\mathbf{X}_0, \mathbf{U}_{0:N-1}) \tag{3}$$

The system parameters are then optimized over the training data using gradient descent on the mean squared error (MSE) loss between predicted and observed end cap positions.

$$\mathcal{L}_m(\mathbf{X}_m, \hat{\mathbf{X}}_m) = ||\mathbf{X}_m - \hat{\mathbf{X}}_m||_2^2 \tag{4}$$

$$\mathcal{L}(\mathcal{T}, \hat{\mathcal{T}}) = \frac{1}{M}\sum_{m=0}^{M} \mathcal{L}_m = \frac{1}{M}\sum_{m=0}^{M} ||\mathbf{X}_m - \hat{\mathbf{X}}_m||_2^2 \tag{5}$$

Addressing partial observability (Section IV), each pass rolls out the entire trajectory (M gait cycles) and measures the pose mean squared error (MSE) at the end of each cycle, after which gradients are back-propagated. The converged system parameters are then validated by comparing a new real-world trajectory against its simulated counterpart.

This training process is iterated until the desired simulation-to-reality accuracy is achieved. Once validated, motion primitives can be reliably simulated and compiled into a library for planning. In practice, convergence is typically achieved in two iterations.

Motion primitive library Eleven motion primitives were generated (see Table I) and classified into three types: (i) roll forward, (ii) turn clockwise, and (iii) turn counterclockwise. The clockwise and counterclockwise turning gaits were generated in simulation. A human-engineered roll forward primitive is modified to also generate gradual turning gaits. Specifically, the maximum tendon lengths on each side of the robot are varied,

TABLE I
Parameters for the 11 Motion Primitives

Primitive	Gait	Left Max (mm)	Right Max (mm)
0	Forward Roll	200	200
1	Forward Roll	220	220
2	Forward Roll	240	240
3	Forward Roll	200	220
4	Forward Roll	220	200
5	Forward Roll	200	240
6	Forward Roll	240	200
7	Forward Roll	220	240
8	Forward Roll	240	220
9	Counterclockwise	200	200
10	Clockwise	200	200

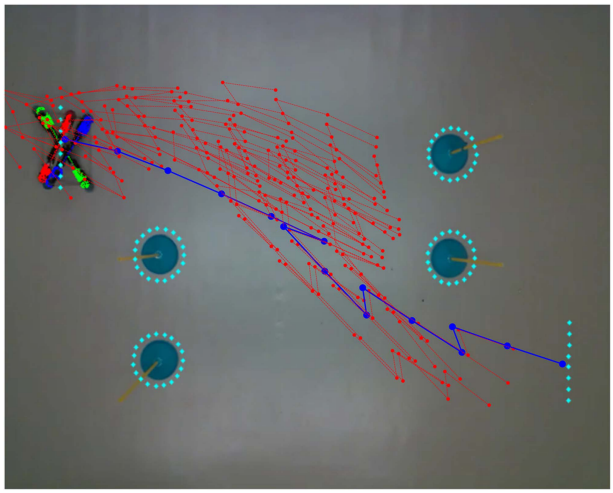

Fig. 4. Example open-loop plan (dark blue) around obstacles (light blue circles) with unused expanded configurations (red).

using values of 200 mm, 220 mm, and 240 mm based on the limits of the capacitive strain sensors to create nine distinct variations. Maximum tendon lengths less than 200 mm make it difficult for the robot to roll, while those greater than 240 mm risk damaging the sensors.

VI. Tensegrity Path Planning

Discretizing the robot's possible actions into motion primitives reduces the control problem to a discrete graph search. Each primitive is treated as an edge in a graph where nodes represent the robot's SE(2) configurations. This abstraction simplifies planning by focusing on achievable transitions between configurations rather than the detailed, low-level control inputs required to execute them.

Open-loop Planner The open-loop planner employs a modified A$*$ algorithm (Algorithm 1). In traditional A$*$, a closed list prevents revisiting identical states. In this application, however, the combination of motion primitives rarely generates the exact same pose twice. Instead, the search expands numerous similar but non-identical poses, leading to computational redundancy. To mitigate this, the planner uses a KD-Tree to efficiently query the nearest node in the closed list. This strategy allows the algorithm to prune new branches that are sufficiently close (within a predefined threshold) to an explored pose. As shown in Fig. 4, this modification significantly reduces the number of expanded poses, enhancing planner efficiency at the expense of strict optimality. Despite this concession, the resulting planner is highly effective for real-time experiments where computational efficiency is important.

Closed-loop Re-planner Physical execution of motion primitives is subject to small variances, precluding perfect prediction of the robot's movements. Consequently, open-loop plans are effective only for short, undisturbed paths. Over longer trajectories, these small discrepancies between simulated and physical execution accumulate, resulting in pose error. To mitigate this, the system employs a closed-loop planner that generates a new path after each motion primitive is executed. The online workflow (Fig. 5) involves continuous pose tracking at 7 Hz, with re-planning triggered after each primitive based on the current pose estimate. The planner may occasionally fail to find

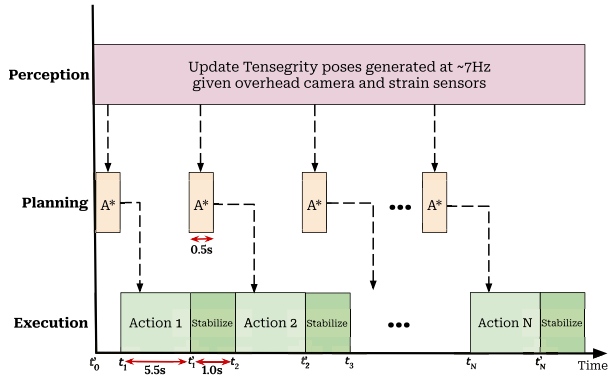

Fig. 5. Perceive-plan-execute loop: The pose is passed to an A$*$ planner at 7 Hz. The first primitive in the plan is executed. Re-planning occurs at the end of each primitive as the robot returns to a rest state.

Algorithm 1: A$*$ with Tensegrity Primitives.

Input: Start $\mathbf{S}$, Goal $\mathbf{G}$, Boundary $\mathbf{B}$, Obstacles o, Primitives $\mathbf{P}$
Output: Path composed of primitives from $\mathbf{S}$ to $\mathbf{G}$
Initialize Open List $\mathcal{O} = \{\mathbf{S}\}$ and Closed List $\mathcal{C} = \varnothing$
Initialize $G_{score}[\mathbf{S}] = 0$
Initialize heuristics h by calculating distances along grid
while $\mathcal{O}$ is not empty **do**
  $curr \leftarrow$ pop node in $\mathcal{O}$ with the lowest $F_{score}[curr]$
  if collisionDetect($curr$, $\mathbf{B}$, o) **then**
    continue
  if $curr$ within threshold of $\mathbf{G}$ **then**
    returnReconstructPath($curr$, $cameFrom$)
  if $curr$ within threshold of closest element in $\mathcal{C}$ **then**
    continue
  for each $p \in \mathbf{P}$ **do**
    Child $child = curr$.propagate(p)
    $G'_{score} = G_{score}[curr] + \text{cost}(p)$
    if $G'_{score} < G_{score}[n]$ **then**
      $cameFrom[child] = curr$
      $G_{score}[child] = G'_{score}$
      $F_{score}[child] = G_{score}[child] + h(child)$
      Add $child$ to $\mathcal{O}$
returnFailure (no path found)

a valid path, often due to an inaccurate pose estimate or close proximity to an obstacle. In such cases (a "planning failure"), the robot executes the next primitive from the last valid plan. This fallback behavior continues until re-planning can be successfully performed.

VII. Demonstrations and Experiments

Experiments demonstrate the robot's ability to autonomously navigate obstacle courses. An analysis of open-loop execution motivates closed-loop re-planning. While many experiments are conducted in a standardized environment to compare the three platforms and evaluate reproducibility, the solution is also demonstrated in increasingly diverse settings and validated against unmodeled effects, including a vertical drop, an incline, granular media, and an outdoor field. Table II provides the

TABLE II
MOTION PRIMITIVE ERROR ON VARIOUS TERRAINS

Terrain	Positional Error (m)	Angular Error (°)
SysID Baseline	0.20 ± 0.05	4.72 ± 2.66
Drop	0.26 ± 0.06	14.11 ± 0.41
Incline	0.19 ± 0.10	19.37 ± 10.98
Granular	0.23 ± 0.03	5.41 ± 2.41

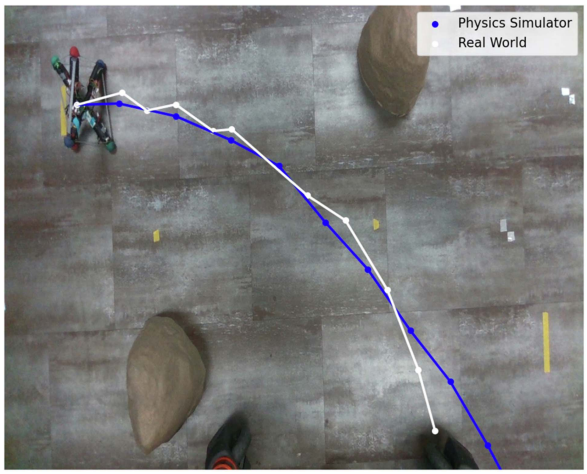

Fig. 6. Comparison of an open-loop path in simulation (blue) and real (white) after sysID. The simulation incorporates dynamic effects and accurately predicts the robot deviating from its path.

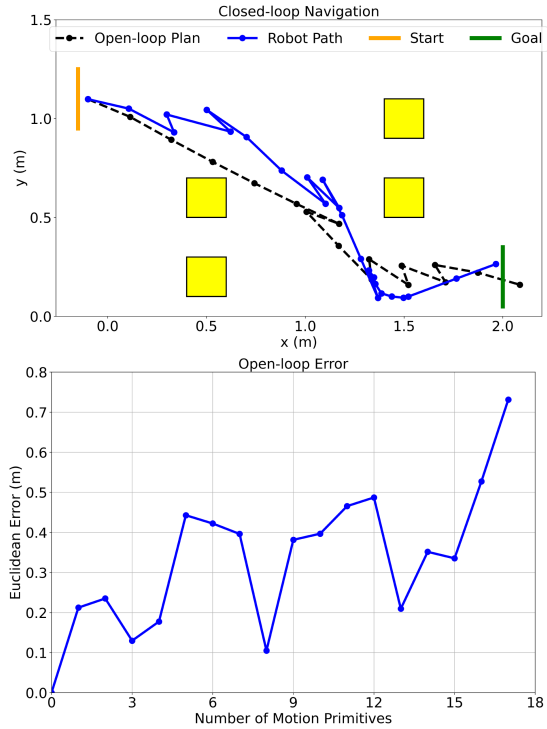

Fig. 7. The executed path deviates from the plan, motivating closed-loop control. (Top) The robot's path against the open-loop plan. (Bottom) The Euclidean distance between the open-loop plan and the robot's position as a function of motion primitives executed. As plans get longer, the reliability of initial predictions decreases.

mean and standard deviation of positional and angular errors for various terrains. The error is calculated as the absolute difference between the static $\mathbb{SE}(2)$ transform stored in the motion primitive library and the observed $\mathbb{SE}(2)$ transform in the recorded trajectories. The terrain used for SysID has a nonzero error, and the unseen terrains have moderately larger errors. Despite these model mismatches, re-planning can compensate and successfully navigate to the goal. Videos of these experiments are available in the supplementary material.

Open-loop Execution A 2D obstacle course was constructed to evaluate planner performance. Open-loop planning proved sufficient only for simple environments requiring few steps to reach the goal. As shown in Fig. 6, a robot executing an open-loop plan (post-sysID) successfully approaches the goal, but small, cumulative errors between the primitive models and the robot's physical dynamics cause it to deviate near the end of the path. This deviation highlights the necessity of a closed-loop re-planner.

Closed-loop Navigation To mitigate open-loop limitations, the system employs the re-planner described in Section VI. This re-planner adjusts the path after each primitive based on the robot's current pose, compensating for disturbances and enabling navigation through more challenging obstacle courses. Fig. 7 compares the closed-loop path to the original open-loop plan, showing how the robot's executed path progressively deviates. As shown in Fig. 8, this closed-loop approach enables the robot to reliably reach the goal. Re-planning requires approximately 0.5 s on average, which is sufficient for real-time execution.

To demonstrate repeatability, 10 closed-loop trials were compared against 10 open-loop trials on the open-source platform. Fig. 9 shows the robot's COM for the closed-loop in blue, while the open-loop is shown in red. The success rates are compared

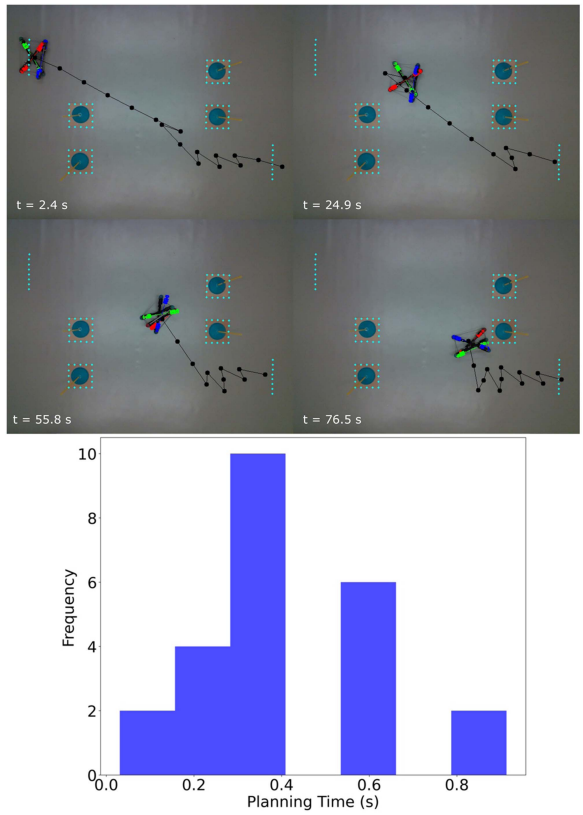

Fig. 8. Navigating among obstacles: The plan is recomputed after each motion primitive. The histogram shows the distribution of planning compute times, which depends on how close to the obstacles and the goal the robot is.

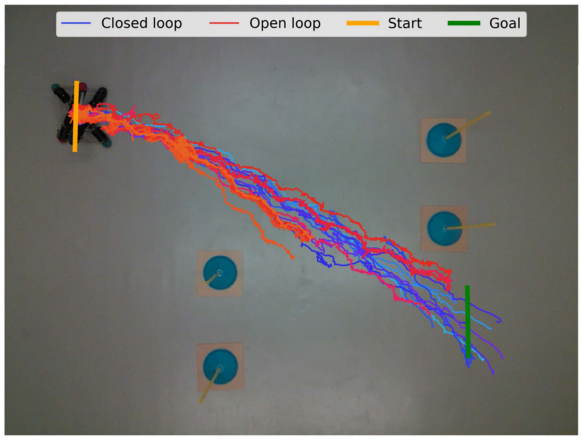

Fig. 9. The robot's COM for 10 closed-loop (blue) and 10 open-loop (red) trials. With closed-loop control, the robot repeatably arrives at the goal, violating obstacle boundaries in two out of 10 trials (but still being able to complete the task). In contrast, only one open-loop trial arrives at the goal without violating the obstacles.

TABLE III
REPEATABILITY AND SUCCESS RATE FOR 10 TRIALS

Solution	Success	Collisions	Short of Goal
Closed-loop (proposed)	80%	20%	0%
Open-loop (baseline)	10%	30%	60%

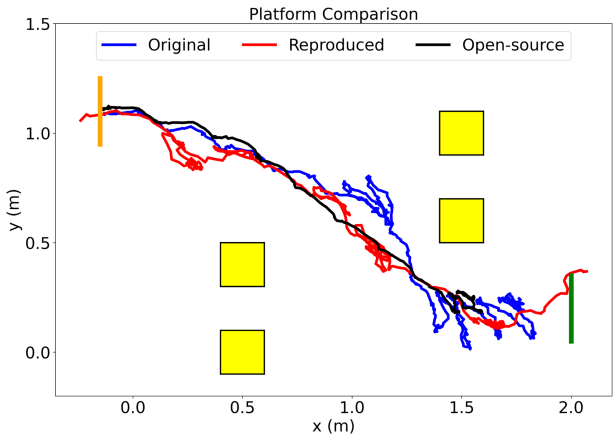

Fig. 10. Closed-loop executions by the three platforms given the same primitive model on the same obstacle course, i.e., without a re-identified model specific to each platform. Re-planning addresses the model inaccuracies.

in Table III. A trial was considered a failure if at least one of the robot's end caps physically collided with the upright obstacles. Another failure mode for the open-loop case was when primitives did not execute as predicted. Then, the robot may end up in the wrong orientation, which does not allow it to continue rolling, and finish its plan short of the goal. Successful trials were those where the robot ended within one body length of the goal without obstacle collisions. As these experiments evaluate the navigation pipeline, trials where a hardware failure occurred, such as a cable getting tangled or a WiFi disconnection, were not considered.

To demonstrate reproducibility, all three robot platforms (prior design, reproduced platform, and new open-source design) navigated an identical obstacle course (Fig. 10). Notably, all plans were derived from a motion primitive model based on

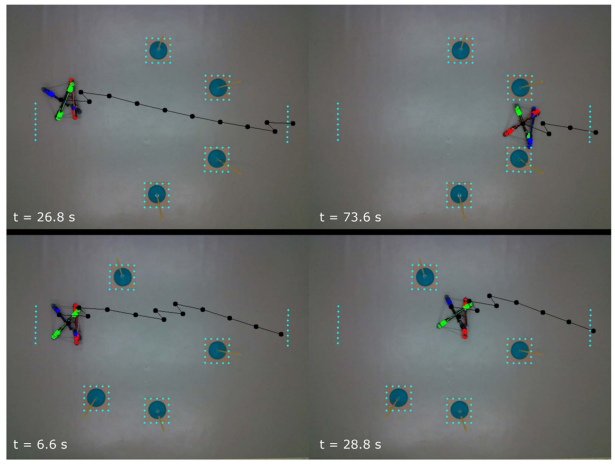

Fig. 11. Two different obstacle courses.

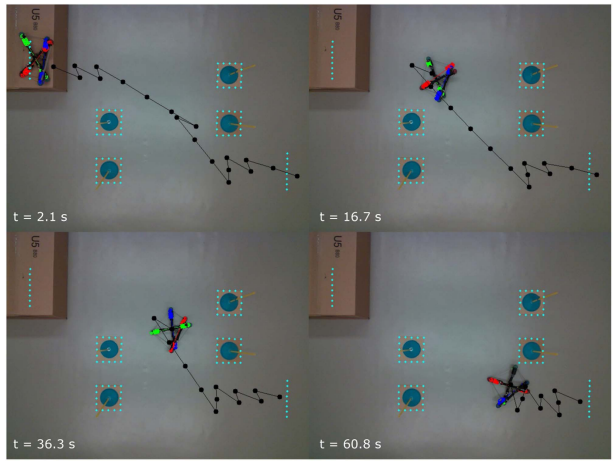

Fig. 12. Feedback makes navigation robust to disturbances in a 37 cm drop. Due to the sudden movement, pose tracking lags but recovers.

the original platform's sysID. Despite lacking platform-specific models, all robots successfully completed closed-loop paths. This demonstrates that re-planning mitigates model inaccuracies, enabling robust navigation without needing to repeat sysID for each new robot. While many experiments were conducted in a standardized environment for comparative analysis, the closed-loop system also enables robust navigation in other obstacle fields (Fig. 11).

Drop To evaluate the system's robustness to impacts, a strength of tensegrity robots [27], a trial was conducted where the robot began at a height of 37 cm above the ground (Fig. 12). Upon executing its first motion primitive, the robot rolled off the ledge, resulting in a post-drop pose significantly different from the model's prediction. The navigation pipeline proved robust to this disturbance. The pose tracker successfully estimated the robot's new position and orientation, allowing the re-planner to calculate a new sequence of primitives from the post-drop state. Primitive gaits were modified via symmetry based on the robot's perceived orientation, enabling the robot to autonomously complete the obstacle course despite the unmodeled drop.

Incline To further assess robustness, experiments were conducted on an 8° wooden ramp (Fig. 13). The incline functions as

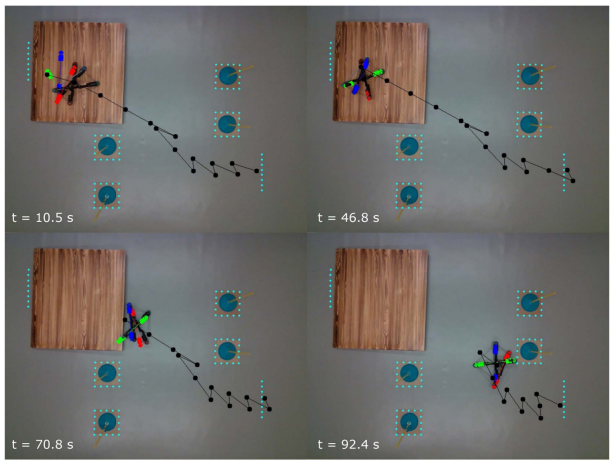

Fig. 13. The robot navigates an obstacle course with an 8° incline ramp without new sysID. The approach is robust to errors in perception: toward the start, pose tracking gives inaccurate results, but eventually recovers.

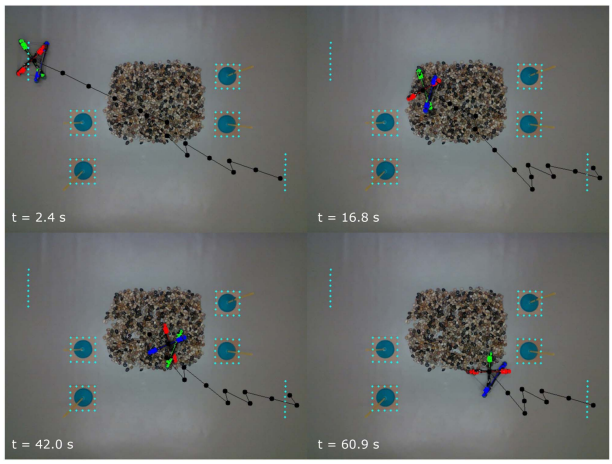

Fig. 14. Successful navigation over granular terrain, where the motion differs from the flat ground where sysID was performed.

a disturbance. The motion primitives, modeled on flat ground, do not yield the same transforms on a slope. Nevertheless, the solution successfully overcomes the incline without requiring new sysID. The system remains effective even as the pose tracking temporarily loses accuracy during the trial, an issue caused by the ramp's distracting color.

Granular Terrain Repeating sysID for the vast range of terrains encountered in the field is infeasible. Therefore, the solution's robustness to unmodeled terrain effects was evaluated in an experiment requiring the robot to traverse granular media (Fig. 14). The robot's primitive execution on granular media is significantly different from flat terrain, rendering the physics engine's predictions inaccurate. Despite this, the navigation system proved robust. The discrepancies between the model's prediction and the robot's physical execution were mitigated by the closed-loop planner, which generates a new path after each motion primitive.

Field Experiments Field tests required switching from the LiDAR-based Intel RealSense L515 to the stereo-based D435, which operates in direct sunlight. This involved mounting the camera overhead, calibrating it with an April tag [28], and re-tuning the HSV filter values to track the robot's colored end caps under this lighting. The robot successfully navigated an outdoor obstacle course (Fig. 15), since the closed-loop feedback system compensated for the unmodeled effects.

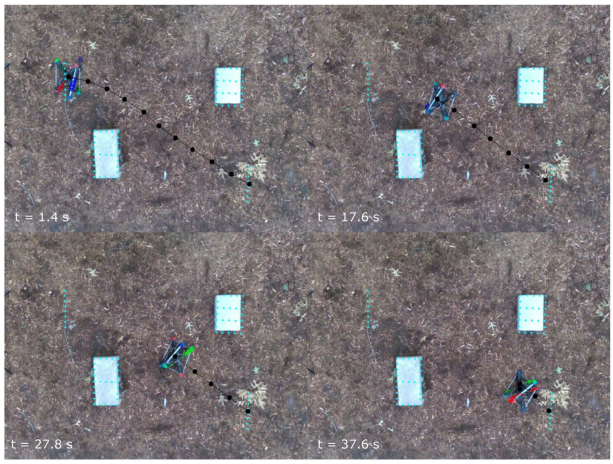

Fig. 15. The tensegrity navigates an outdoor obstacle course.

TABLE IV
PLANNING RESULTS WITH VARIOUS NUMBER OF OBSTACLES

# of Obstacles	Success Rate	# Prim.	# States Searched	Time (s)
1	0.99	12.98	166.30	0.866
2	1.00	14.39	237.99	0.854
4	1.00	15.92	381.25	0.884
6	0.90	17.04	478.46	0.848

TABLE V
PLANNING RESULTS WITH VARIOUS ENVIRONMENT DIMENSIONS

Env. Dim. (m)	Success Rate	# Prim.	# States Searched	Time (s)
4x4	1.00	25.56	907.44	2.396
6x6	0.96	37.63	243.38	5.330
8x8	1.00	56.32	338.59	9.471
10x10	1.00	80.86	1257.59	15.924

Obstacle Density Analysis To evaluate the impacts of obstacle density, two ablation studies were performed for varying numbers of obstacles (Table IV) and workspace size (Table V). For each condition, 100 simulated environments with randomized obstacle placements were generated. Results are averaged across trials. The varying obstacle number experiment used workspace dimensions that approximated the real world setup (4 m x 1.6 m). All experiments used rectangular obstacles of size 0.4 m x 0.27 m.

In the obstacle study of Table IV, the number of searched states increases with additional obstacles, while planning time increases marginally. This is due to the dominant cost of the obstacle-aware heuristic, which averaged 0.768 seconds over 100 trials. When obstacle density increases significantly (6 obstacles), success rates start declining. Planning parameters were held constant across experiments.

Workspace scale is evaluated in Table V by increasing its size. Planning times do increase significantly with size and would impact closed-loop operation in deployments. While increasing pruning aggressiveness can mitigate this issue, it may reduce success rates. For large-scale workspaces, introducing intermediate goals or checkpoints can decompose the planning problem and limit computational overhead.

Motion Primitive Analysis Table VI reports an ablation study on the number of motion primitives available to the planner. For the three-primitive tests, primitives 0, 9, and 10 from Table I were used. The seven-primitive tests employed primitives 0, 1, 4, 5, 8,

TABLE VI
SIMULATED ABLATION OF NUMBER OF PRIMITIVE TYPES

# Prim. Types	Success Rate	# Prim.	# States Searched	Time (s)
3	0.99	23.12	178.33	0.819
7	0.99	18.30	460.41	0.840
11	0.98	15.76	411.58	0.854

9, and 10, while the eleven-primitive configuration included the full primitive set. The same 100 random environments were used for each test. The environments used four randomized obstacles with essentially the same setup as the four-obstacle test in IV.

Results indicate that reducing the number of available primitives can preserve similar success rates, but this reduction leads to less efficient trajectories as measured by the number of primitives per solution (third column of Table VI). The decrease in efficiency arises from the limited maneuverability imposed by a smaller primitive set, which restricts the planner's ability to exploit more direct motions.

VIII. DISCUSSION

This work presents a complete system for tensegrity robot navigation among obstacles. The system models the robot's motion primitives using a physics engine and employs an A∗ planner to generate paths, incorporating feedback from real-time pose tracking. The pipeline demonstrates robustness to unmodeled disturbances, such as vertical drops, inclines, and granular media. Its effectiveness is validated in an outdoor field. This contribution includes a complete open-source hardware and software stack, enabling reproducibility by other labs. This platform is intended to serve as a baseline to foster further advances in tensegrity navigation and related research.

This study assumes static obstacles. Future work can focus on supporting moving obstacles and should incorporate obstacle tracking. As the planner recomputes a path after each primitive, it is primed to handle this if obstacle poses are updated and obstacle motion is similar to the robot's speed.

The solution relies on a static overhead camera, but the proposed framework can be extended to consider the case of a drone monitoring the motion of a tensegrity robot after deployment. Such solutions have been demonstrated in the literature, where a drone with an overhead camera tracks the pose of a cable-driven robot after dropping it [29]. Ultra-wideband rangefinders on the robot and in the environment have been used to track a tensegrity's pose [30]. Tracking the pose of a tensegrity given only on-board sensors remains an important, active area of research [31]. Such solutions can be integrated with the open-source framework for tensegrity navigation in the future.

REFERENCES

[1] D. S. Shah et al., "Tensegrity robotics," *Soft Robot.*, vol. 9, no. 4, pp. 639–656, 2022.

[2] M. Vespignani, J. M. Friesen, V. SunSpiral, and J. Bruce, "Design of SUPERball v2, A compliant tensegrity robot for absorbing large impacts," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 2865–2871.

[3] R. Kobayashi, H. Nabae, G. Endo, and K. Suzumori, "Soft tensegrity robot driven by thin artificial muscles for the exploration of unknown spatial configurations," *IEEE Robot. Automat. Lett.*, vol. 7, no. 2, pp. 5349–5356, Apr. 2022.

[4] J. Rieffel and J.-B. Mouret, "Adaptive and resilient soft tensegrity robots," *Soft Robot.*, vol. 5, no. 3, pp. 318–329, 2018.

[5] J. Friesen, A. Pogue, T. Bewley, M. de Oliveira, R. Skelton, and V. Sunspiral, "DuCTT: A tensegrity robot for exploring duct systems," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2014, pp. 4222–4228.

[6] S. Mintchev, D. Zappetti, J. Willemin, and D. Floreano, "A soft robot for random exploration of terrestrial environments," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2018, pp. 7492–7497.

[7] Z. Wang, K. Li, Q. He, and S. Cai, "A light-powered ultralight tensegrity robot with high deformability and load capacity," *Adv. Mater.*, vol. 31, no. 7, 2019, Art. no. 1806849.

[8] L.-H. Chen et al., "Soft spherical tensegrity robot design using rod-centered actuation and control," *J. Mechanisms Robot.*, vol. 9, no. 2, 2017, Art. no. 025001.

[9] S. Lu et al., "6N-DoF pose tracking for tensegrity robots," in *Proc. Int. Symp. Robot. Res.*, 2023, pp. 136–152.

[10] E. Granados et al., "State and trajectory estimation of tensegrity robots via factor graphs and Chebyshev polynomials," *Robosoft*, Kanazawa, Japan, Apr. 2026.

[11] J. M. Friesen, "Tensegrity matlab objects," 2015. [Online]. Available: https://github.com/Jfriesen222/Tensegrity_MATLAB_Objects

[12] V. Tadiparthi, S.-C. Hsu, and R. Bhattacharya, "STEDY: Software for tensegrity dynamics," *J. Open Source Softw.*, vol. 4, 2019, Art. no. 1042.

[13] R. Goyal, M. Chen, M. Majji, and R. E. Skelton, "MOTES: Modeling of tensegrity structures," *J. Open Source Softw.*, vol. 4, 2019, Art. no. 1613.

[14] C. Paul, F. J. V. Cuevas, and H. Lipson, "Design and control of tensegrity robots for locomotion," *IEEE Trans. Robot.*, vol. 22, no. 5, pp. 944–957, Oct. 2006.

[15] K. Caluwaerts et al., "Design and control of compliant tensegrity robots through simulation and hardware validation," *J. Roy. Soc.*, vol. 11, 2014, Art. no. 20140520.

[16] S. Wittmeier, M. Jäntsch, K. Dalamagkidis, M. Rickert, H. G. Marques, and A. Knoll, "CALIPER: A universal robot simulation framework for tendon-driven robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2011, pp. 1063–1068.

[17] K. Wang, M. Aanjaneya, and K. Bekris, "Sim2Sim evaluation of a novel data-efficient differentiable physics engine for tensegrity robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2021, pp. 1694–1701.

[18] K. Wang, M. Aanjaneya, and K. Bekris, "A recurrent differentiable engine for modeling tensegrity robots trainable with low-frequency data," in *Proc. Int. Conf. Robot. Automat.*, 2022, pp. 3230–3237.

[19] K. Wang et al., "Real2Sim2Real transfer for control of cable-driven robots via a differentiable physics engine," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2023, pp. 2534–2541.

[20] N. Chen, K. Wang, W. R. J. Johnson III, R. Kramer-Bottiglio, K. Bekris, and M. Aanjaneya, "Learning differentiable tensegrity dynamics using graph neural networks," in *Proc. 8th Conf. Robot Learn.*, P. Agrawal, O. Kroemer, and W. Burgard, Eds. vol. 270, 2025, pp. 5134–5149. [Online]. Available: https://proceedings.mlr.press/ v270/chen25j.html

[21] N. Chen, W. Johnson, R. Kramer-Bottiglio, K. Bekris, and M. Aanjaneya, "CableRobotGraphSim: A graph neural network for modeling partially observable cable-driven robot dynamics," *Learn. Dyn. Control*, Los Angeles, USA, Jun. 2026.

[22] M. Vespignani, C. Ercolani, J. M. Friesen, and J. Bruce, "Steerable locomotion controller for six-strut icosahedral tensegrity robots," in *Proc. IEEE/RSJ Int. Conf. Intell. Robots Syst.*, 2018, pp. 2886–2892.

[23] R. L. Baines, J. W. Booth, and R. Kramer-Bottiglio, "Rolling soft membrane-driven tensegrity robots," *IEEE Robot. Automat. Lett.*, vol. 5, no. 4, pp. 6567–6574, Oct. 2020.

[24] J. Chang, B. Li, W. Liu, and W. Du, "The path planning method of tensegrity robot based on A∗ Algorithm," in *Proc. IEEE 8th Annu. Int. Conf. CYBER Technol. Automat., Control, Intell. Syst.*, 2018, pp. 1502–1507.

[25] X. Feng et al., "Trajectory planning on rolling locomotion of spherical movable tensegrity robots with multi-gait patterns," *Soft Robot.*, vol. 11, pp. 725–740, 2024.

[26] W. R. Johnson, A. Agrawala, X. Huang, J. Booth, and R. Kramer-Bottiglio, "Sensor tendons for soft robot shape estimation," in *Proc. IEEE Sensors*, 2022, pp. 1–4.

[27] W. R. Johnson et al., "Impact-resistant, autonomous robots inspired by tensegrity architecture," in *Proc. Nature Mach. Intell.*, 2026.

[28] E. Olson, "AprilTag: A robust and flexible visual fiducial system," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2011, pp. 3400–3407.

[29] L. Zhao, Y. Jiang, M. Chen, K. Bekris, and D. Balkcom, "Modular shape-changing tensegrity-blocks enable self-assembling robotic structures," *Nature Commun.*, vol. 16, 2025, Art. no. 5888.

[30] K. Caluwaerts, J. Bruce, J. M. Friesen, and V. SunSpiral, "State estimation for tensegrity robots," in *Proc. IEEE Int. Conf. Robot. Automat.*, 2016, pp. 1860–1865.

[31] W. Tong, T.-Y. Lin, J. Mi, Y. Jiang, M. Ghaffari, and X. Huang, "Tensegrity robot proprioceptive state estimation with geometric constraints," *IEEE Robot. Automat. Lett.*, vol. 10, no. 4, pp. 4069–4076, Apr. 2025.