An Open-Source, Reproducible Tensegrity Robot that can Navigate Among Obstacles

William R. Johnson III[1,Y], Patrick Meng[1,R], Nelson Chen[R], Luca Cimatti[Y], Augustin Vercoutere[Y], Mridul Aanjaneya[R], Rebecca Kramer-Bottiglio[Y], Kostas E. Bekris[R]

***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 path planning and obstacle avoidance. This paper 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 https://sites.google.com/view/tensegrity-navigation/. The proposed system tracks the robot's pose and executes collision-free paths to a specified 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.**

I. Introduction

Navigation over unpredictable terrain with obstacles remains a challenge in robotics. Tensegrity robots made with rigid struts and elastic tendons are a next-generation mobile platform promising due to their durability, adaptability, low mass, and low cost [1]. Their significant deformations and coupled dynamics, however, create formidable modeling and control challenges, which have hindered the demonstration of a complete solution for tensegrity robot navigation.

This paper presents a complete pipeline for navigation and obstacle avoidance using a 3-bar tensegrity robot. It integrates methods in tensegrity design, modeling, and state estimation with path planning to achieve robust navigation around obstacles amidst environmental disturbances. The pipeline 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 to move the robot from its current pose to the goal, avoiding known obstacles. State estimation from an overhead camera closes the loop, enabling to re-plan the path after each action to correct for discrepancies between the model's prediction and the robot's state. Navigation experiments demonstrate the robot's robustness to unmodeled disturbances, including vertical drops, inclines, and granular media. The system's effectiveness is further validated through operation in an outdoor field environment (Fig. 1).

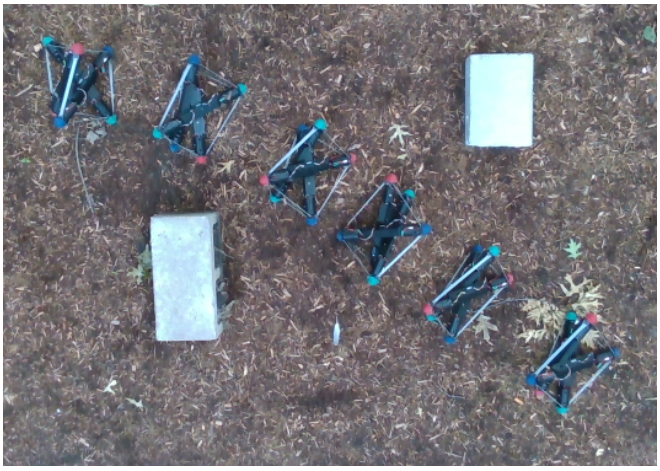

Fig. 1. The open-source tensegrity robot moving around obstacles outdoors starts from the top left and autonomously navigates to the bottom right.

This work also introduces a new, open-source tensegrity robot design. The hardware and software are publicly available to promote reproduction by other researchers. 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 research has demonstrated diverse abilities, including 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 field environments [8]. Various tensegrity simulators have been developed, those solving analytical differential equations [9]–[11], those [12]–[14] building upon general-purpose physics engines, and, more recently, learning-based simulators [15]–[18] to address the simulation-to-reality (sim2real) gap. Typical solutions for path planning and navigation involve geometric solutions that tile the robot's base polygons from start to goal [19], [20], often ignoring dynamic effects. Furthermore, real hardware experiments have frequently been limited to open-loop control [21], [22].

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.

[1]Co-first authors; [Y]Mech. Eng., Yale Univ.; [R]Comp. Sc., Rutgers Univ.

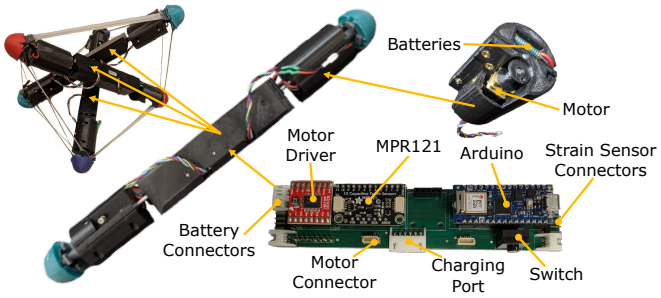

Fig. 2. Open-source design: Each strut has 2 motor enclosures; each housing a brushed DC motor and LiPo batteries. A custom motherboard uses commercial-off-the-shelf circuits mounted on headers, allowing for easy replacement. A power switch and charging port provide convenience. The tensegrity robot is formed by assembling 3 struts with strain sensors.

III. Robot Design

This work utilizes a 3-bar tensegrity robot, designed to be lightweight, deformable, and impact-resistant (Fig. 1). To demonstrate the generality of the proposed navigation system, experiments are conducted using 3 distinct versions: an initial design (described in previous work), 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://sites.google.com/view/tensegrity-navigation/.

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 3 bars are structurally identical, differentiated only by colored end caps for pose tracking. Each bar houses 2 motors with winches to control tendon length. The robot features 9 tendons: 6 are actuated by the motors, and 3 function as passive restoring springs. All tendons incorporate capacitive strain sensors [23]. 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 2 motor subassemblies (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 also integrates a motor driver (TB6612FNG; SparkFun) and a capacitive sensing (MPR121; Adafruit) breakout board. This custom PCB includes a pushbutton switch, a 2 $\times$ 2S LiPo charging port, and auxiliary ports for future expansion. The compact electronics enclosure is designed to provide easy access to the switch, charging port, and the 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 6 target lengths, 1 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 [24], which fuses data from an overhead camera (Intel RealSense L515) and on-board strain sensors. This algorithm operates in 2 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. As this process resolves, however, only the rod's center axis (via its endpoints), the rod's twist information is lost due to its radial symmetry.

V. Modeling Motion Primitives

The navigation solution requires the SE(2) transformations from the robot's motion primitives as input. As large-scale data collection on the physical robot is impractical, the system relies on simulation to obtain accurate predictions for each motion primitive. As shown in Fig. 3, this offline modeling process involves system identification (sysID) and primitive simulation.

An accurate simulator requires identifying system parameters (sysID) that best match observed data. This work leverages previous differentiable tensegrity simulators [15]–[17] to perform sysID via gradient-based optimization.

First-principles simulator In [15], a differentiable simulator $\mathcal{F}(\cdot)$ was developed, parameterized by Θ, with input 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:

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

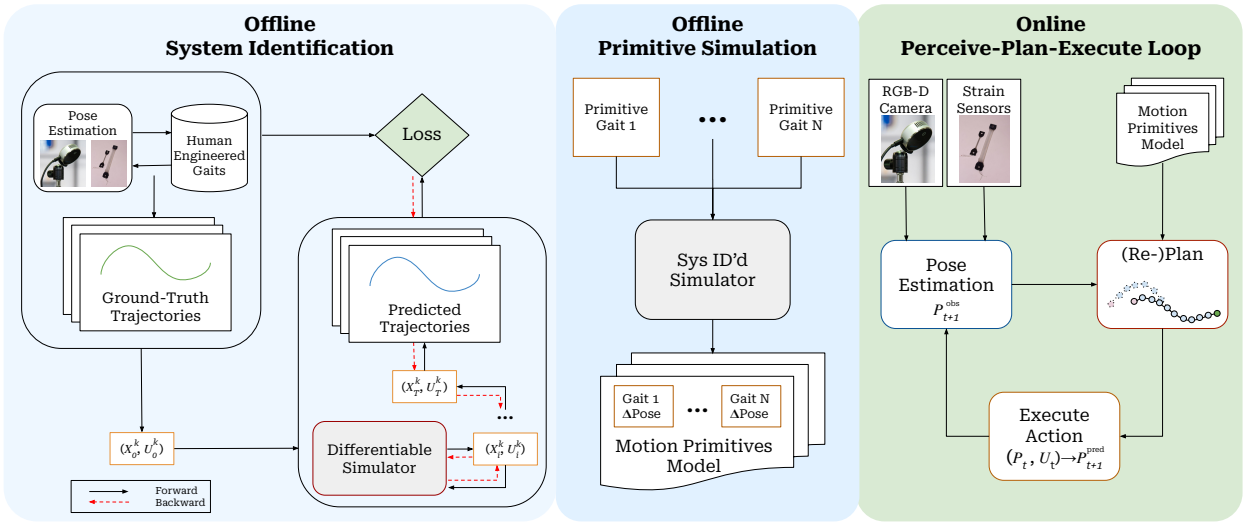

Fig. 3. (Left) Offline sysID: Real trajectories are generated by chaining human-engineered gaits and tracking the robot's pose via vision. Then, the start state $\mathbf{X}_0$ and controls $\mathbf{U}_{0:T-1}$ are autoregressively inputted into a differentiable simulator to predict corresponding trajectories. A scalar loss function between the ground truth and predicted trajectories is back-propagated to update system parameters. (Middle) Offline primitive modeling: Multiple human-engineered and in-simulation-found gaits are executed in the identified simulator. The changes in the pose of each primitive are recorded and 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 motion. Then a plan from the tensegrity's current pose to the goal is computed, and its first primitive action executed. This is repeated until the tensegrity reaches the goal.

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 the contact parameters:

- μ coefficient of friction between end caps and ground.
- ϵ coefficient of restitution.
- β Baumgarte stabilization coefficient.

Optionally, the cable stiffnesses K and damping coefficients σ can also be optimized in this process. In practice, we simply set the cables to have a high stiffness of $10^6 N/m$.

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

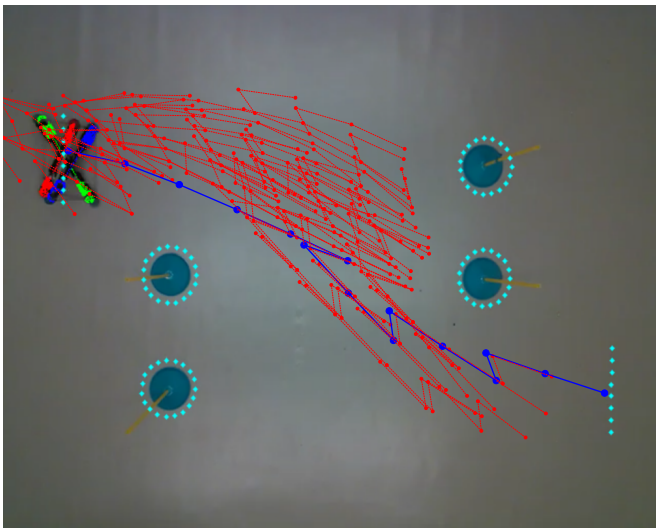

Fig. 4. Example open-loop plan (dark blue) around obstacles (light blue circles) with unused expanded configurations (red).

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 2 iterations.

To ensure path planning tractability, a motion primitive library is constructed from human-engineered and simulation-generated gaits. This library contains 11 motion primitives (Table I) classified into 3 types: (i) roll forward, (ii) turn clockwise, and (iii) turn counterclockwise. Gradual turning gaits are generated by modifying the roll-forward primitive. Specifically, the maximum tendon lengths on each side of the robot are varied, using values of 200 mm, 220 mm, and 240 mm to create nine distinct turning variations.

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

TABLE I
PARAMETERS FOR THE 11 MOTION PRIMITIVES

VI. TENSEGRITY PATH PLANNING

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

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**
        return ReconstructPath($curr$, $cameFrom$)
    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 $\mathcal{O}$
return Failure (no path found)

A. 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.

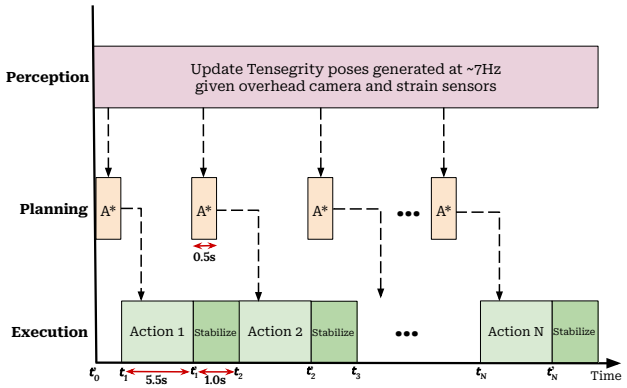

Fig. 5. Online perception-plan-execution: Pose tracking runs at 7 Hz. The pose is passed to an A* planner. The first primitive in the new plan is executed. Re-planning occurs at the last step of each primitive as the robot returns to its rest state.

B. 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 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.

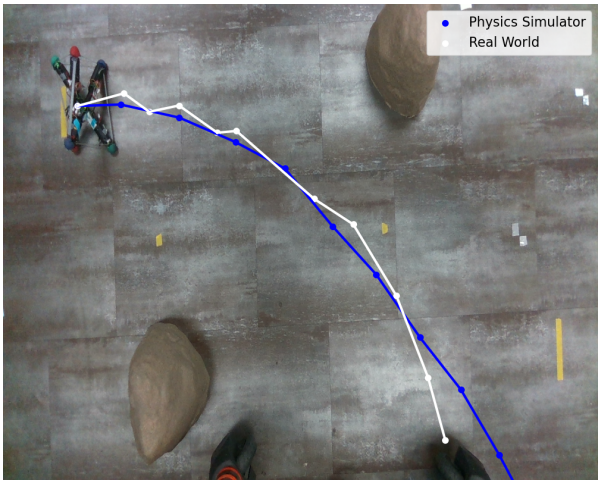

Fig. 6. Comparison of an open-loop path in simulation (blue) and real (white) after sysID. The simulation incorporates dynamic effects and accurately predicts the robot deviating from its path.

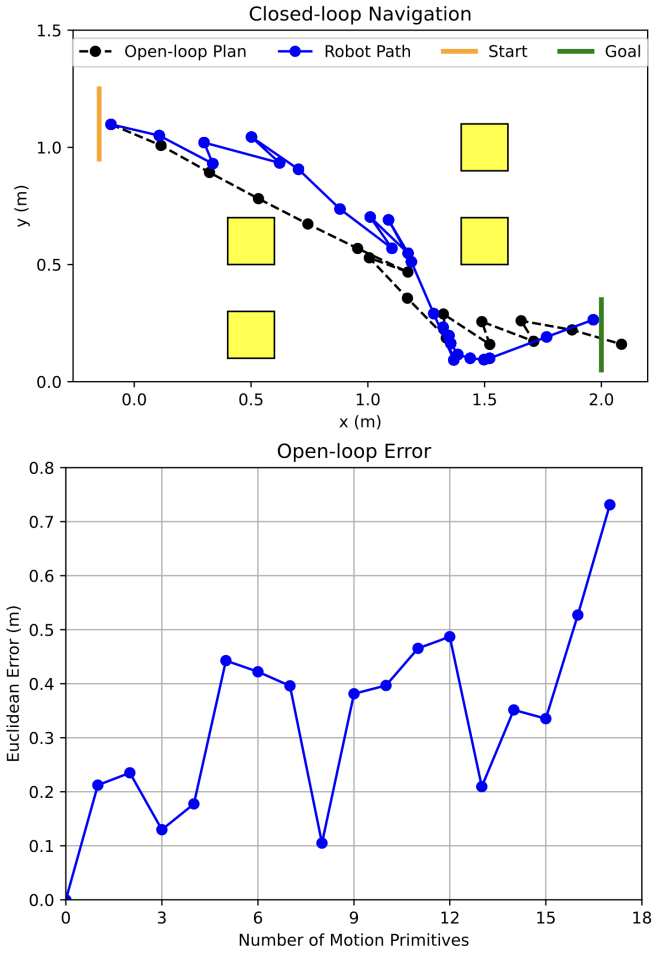

Fig. 7. The path executed deviates from the plan, necessitating closed-loop control in the presence of obstacles. (Top) The robot's path from an experiment against the open-loop path. (Bottom) The Euclidean distance between the open-loop plan and the robot's position as a function of the motion primitives executed. As plans get more complex and require more steps, reliability of open-loop predictions decreases.

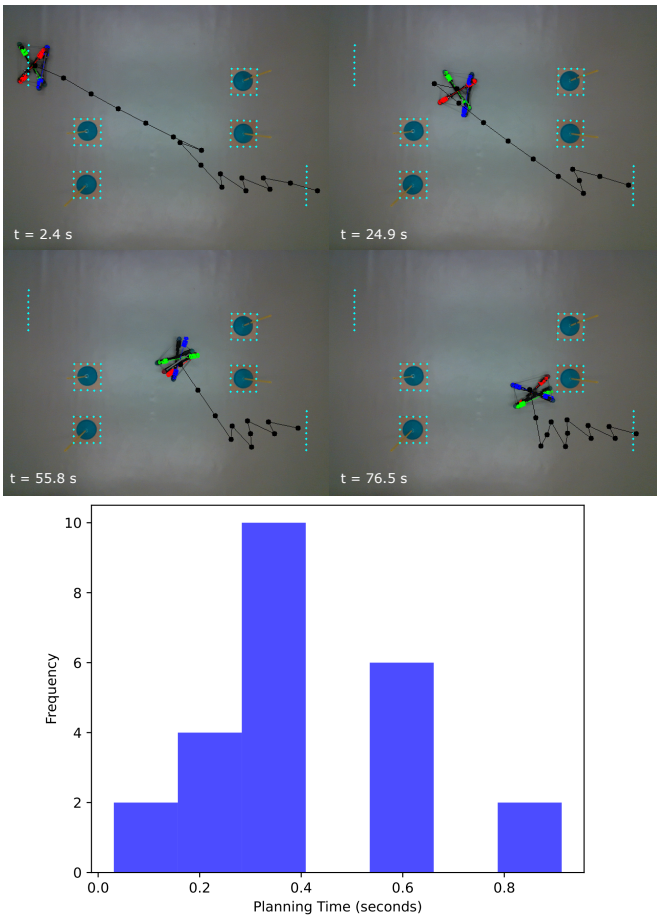

Fig. 8. Navigating an obstacle course given an overhead camera. The plan is recomputed after a motion primitive. The histogram shows the distribution of planning compute times. Planning time varies depending on where the robot is, i.e., how close to the obstacles and the goal.

VII. Demonstrations and Experiments

Experiments demonstrate the robot's ability to autonomously navigate obstacle courses. An initial analysis of open-loop plan execution motivates the necessity of closed-loop re-planning for more complex fields. While many experiments are conducted in a standardized environment to compare the 3 platforms and evaluate reproducibility, the solution's effectiveness is also demonstrated in diverse settings. Furthermore, the closed-loop system's robustness is validated against unmodeled effects, including a vertical drop, an incline, and granular media. The system is also shown operating in an outdoor field. 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 Figure 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-B. 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 reproducibility, all 3 robot platforms (prior design, reproduced platform, and new open-source design) navigated an identical obstacle course (Fig. 9). Notably, all plans were derived from a motion primitive model based on 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. 10).

Drop To evaluate the system's robustness to impacts, a

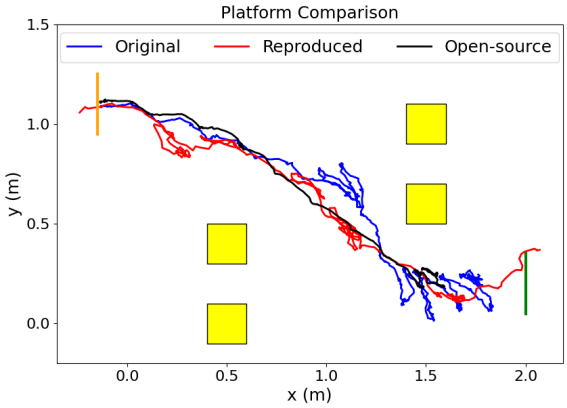

Fig. 9. Closed-loop executions by the 3 platforms given the same primitive model on the same obstacle course, i.e., without a re-identified model specific to each platform. Re-planning addresses the model inaccuracies.

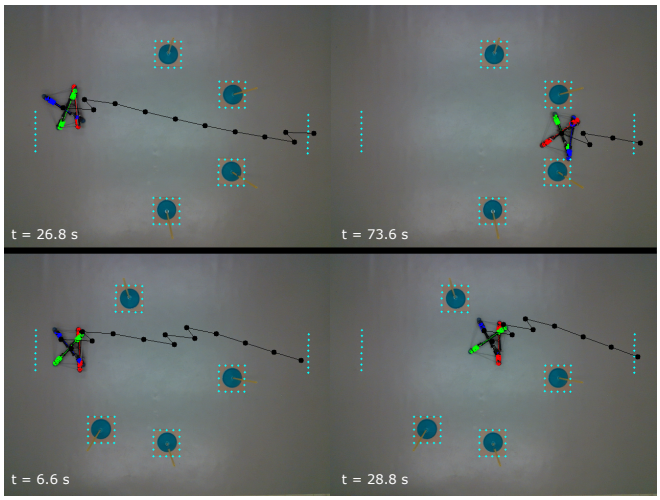

Fig. 10. Two different obstacle courses.

strength of tensegrity robots [25], a trial was conducted where the robot began at a height of 37 cm above the ground (Fig. 11). 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 immediately 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. 12). The incline functions as a disturbance, as the motion primitives, modeled on flat ground, do not yield the same SE(2) transformations on a slope. Nevertheless, the solution successfully overcomes the incline without requiring repeated sysID. The system remains effective even as the pose tracking algorithm temporarily loses accuracy at the start of 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

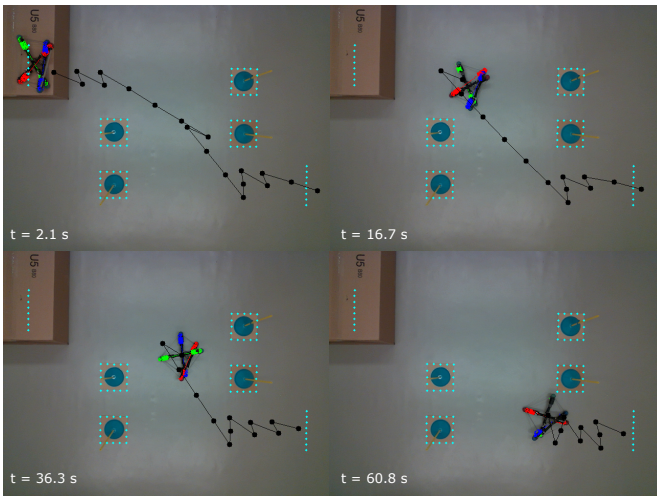

Fig. 11. Feedback makes the navigation solution robust to disturbances, such as a 37 cm drop. Due to the sudden movement from the drop, pose tracking lags but recovers.

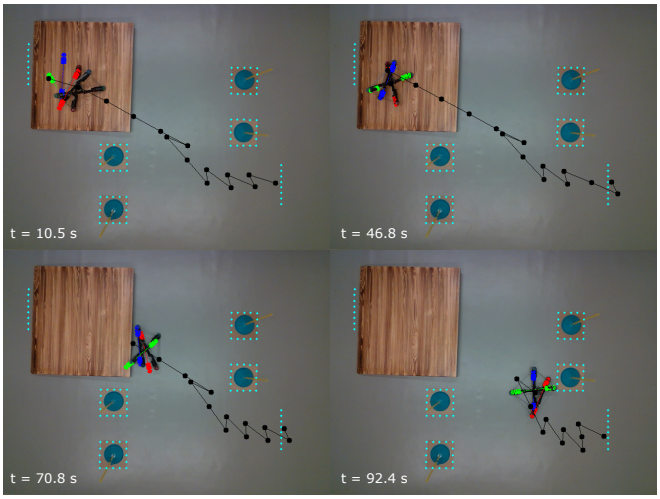

Fig. 12. The robot navigates an obstacle course with an 8° incline wooden ramp without sysID on this new surface. The approach is robust to errors in perception: toward the start, pose tracking gave inaccurate results, but eventually recovers.

solution's robustness to unmodeled terrain effects was evaluated in an experiment requiring the robot to traverse granular media (Fig. 13). 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, as the latter is operable in direct sunlight. This setup involved mounting the camera overhead, calibrating it with an April tag [26], and re-tuning the HSV filter values to track the robot's colored end caps under the new lighting conditions. Following these adjustments, the robot successfully navigated an outdoor obstacle course (Fig. 14). The closed-loop feedback system proved robust, compensating for the unmodeled effects of the outdoor environment.

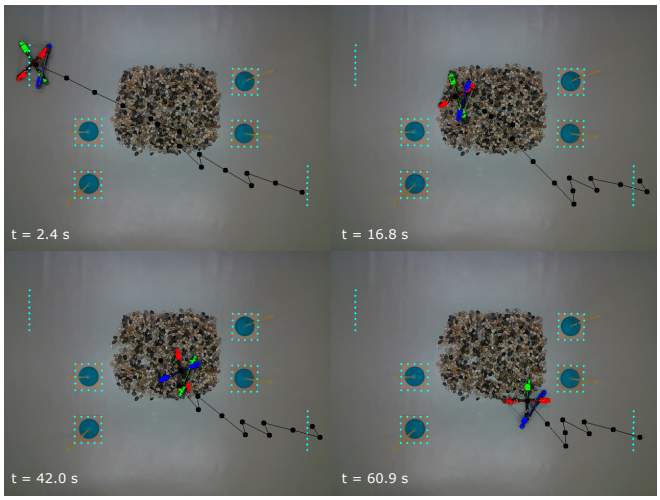

Fig. 13. The solution is robust to unmodeled effects, such as granular terrain, where the primitives substantially differ from the flat ground for which sysID was performed.

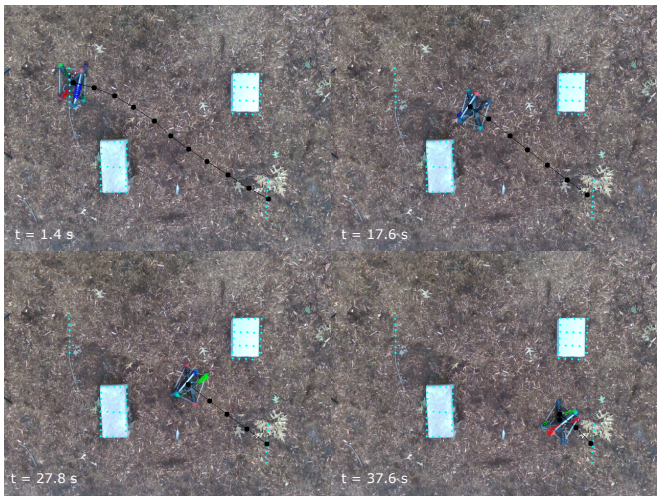

Fig. 14. The tensegrity navigates an outdoor obstacle course.

VIII. Conclusion

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

References

[1] D. S. Shah, J. W. Booth, R. L. Baines, K. Wang, M. Vespignani, K. Bekris, and R. Kramer-Bottiglio, "Tensegrity robotics," *Soft robotics*, 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 *IEEE/RSJ IROS*, 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 RA-L*, vol. 7, no. 2, pp. 5349–5356, 2022.

[4] J. Rieffel and J.-B. Mouret, "Adaptive and resilient soft tensegrity robots," *Soft robotics*, 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 *IEEE ICRA*, 2014, pp. 4222–4228.

[6] S. Mintchev, D. Zappetti, J. Willemin, and D. Floreano, "A soft robot for random exploration of terrestrial environments," in *IEEE ICRA*, 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," *Advanced Materials*, vol. 31, no. 7, 2019.

[8] L.-H. Chen, K. Kim, E. Tang, K. Li, R. House, E. L. Zhu, K. Fountain, A. M. Agogino, A. Agogino, V. Sunspiral *et al.*, "Soft spherical tensegrity robot design using rod-centered actuation and control," *Journal of Mechanisms and Robotics*, vol. 9, no. 2, 2017.

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

[10] V. Tadiparthi, S.-C. Hsu, and R. Bhattacharya, "Stedy: Software for tensegrity dynamics," *Journal of Open Source Software*, vol. 4, no. 33, p. 1042, 2019.

[11] R. Goyal, M. Chen, M. Majji, and R. E. Skelton, "Motes: Modeling of tensegrity structures," *Journal of Open Source Software*, vol. 4, no. 42, p. 1613, 2019.

[12] C. Paul, F. J. V. Cuevas, and H. Lipson, "Design and control of tensegrity robots for locomotion," *IEEE Transactions on Robotics*, vol. 22, pp. 944–957, 2006.

[13] K. Caluwaerts, J. Despraz, A. Iscen, A. Sabelhaus, J. Bruce, B. Schrauwen, and V. Sunspiral, "Design and control of compliant tensegrity robots through simulation and hardware validation," *Journal of the Royal Society*, vol. 11, 09 2014.

[14] 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 *IEEE/RSJ IROS*, 2011, pp. 1063–1068.

[15] K. Wang, M. Aanjaneya, and K. Bekris, "Sim2sim evaluation of a novel data-efficient differentiable physics engine for tensegrity robots," in *IEEE/RSJ IROS*, 09 2021, pp. 1694–1701.

[16] ——, "A recurrent differentiable engine for modeling tensegrity robots trainable with low-frequency data," in *IEEE ICRA*, 2022, pp. 3230–3237.

[17] K. Wang, W. R. Johnson, S. Lu, X. Huang, J. Booth, R. Kramer-Bottiglio, M. Aanjaneya, and K. Bekris, "Real2sim2real transfer for control of cable-driven robots via a differentiable physics engine," in *IEEE/RSJ IROS*, 2023.

[18] N. Chen, K. Wang, W. R. Johnson, R. Kramer-Bottiglio, K. E. Bekris, and M. Aanjaneya, "Learning differentiable tensegrity dynamics using graph neural networks," *ArXiv*, vol. abs/2410.12216, 2024.

[19] M. Vespignani, C. Ercolani, J. M. Friesen, and J. Bruce, "Steerable locomotion controller for six-strut icosahedral tensegrity robots," in *IEEE/RSJ IROS*, 2018, pp. 2886–2892.

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

[21] J. Chang, B. Li, W. Liu, and W. Du, "The path planning method of tensegrity robot based on a* algorithm," in *IEEE CYBER*, 2018, pp. 1502–1507.

[22] X. Feng, J. Xu, J. Zhang, M. Ohsaki, Y. Zhao, Y. Luo, Y. Chen, and X. Xu, "Trajectory planning on rolling locomotion of spherical movable tensegrity robots with multi-gait patterns," *Soft Robotics*, 2024.

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

[24] S. Lu, W. R. Johnson, K. Wang, X. Huang, J. Booth, R. Kramer-Bottiglio, and K. Bekris, "6n-dof pose tracking for tensegrity robots," in *ISRR*, 2023.

[25] W. R. Johnson, X. Huang, S. Lu, K. Wang, J. W. Booth, K. Bekris, and R. Kramer-Bottiglio, "Impact-resistant, autonomous robots inspired by tensegrity architecture," *ArXiv*, 2025.

[26] E. Olson, "Apriltag: A robust and flexible visual fiducial system," in *IEEE ICRA*, 2011, pp. 3400–3407.