Planar Cable-Driven Robot

Arzaq Khan
Hamdan Raashid

December 14, 2024

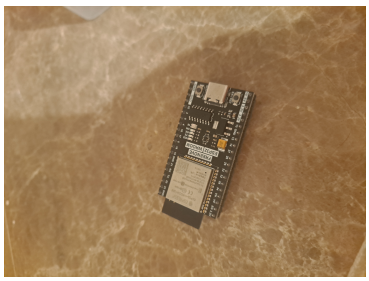
1 Introduction

1.1 Project Description

Our project is a planar cable-driven driven that continously tracks and follows objects in real time. If we had been able to implement everything according to our initial plan, we would have been able to apply the robot in a logistics setting to pick and place objects. Unfortunately, due to certain problems we encountered, which is discussed in a later section, we were unable to strictly follow our original plans.

During the planning stage, we aimed to design a planar cable-driven robot that would pick a user selected target and drop off that target to a user selected location. The user would first select the target in a video feed provided by a staionary camera and the robot would move the end-effector to where the target is located. Next, a command would be given to the end-effector to switch on the electromagnet to pick up the target. Then, the user would provide a location to drop off the target using the same video feed and the robot would drop off the target at said location. Although our actual robot is similar to what we had planned initially, there are slight differences.

The main difference between what we built and we had planned lies in the parts we used and the end-effector. We had planned to use an ESP32, a microcontroller, for its Bluetooth and WiFi features and stepper motors due to their precision. Unfortunately, we could not get the stepper motors to work. We initially figured that it was a power issue, but even after connecting the motors to an external power supply, we could not get the motors to move. As for the end-effector, we could not mount an electromagnet on it due to issues discussed in a subsequent section. To summarize, what we have is a robot that can continously track and follow objects, but cannot pick them up.



(a) An ESP32 microcontroller.

(b) A stepper motor.

(c) An electromagnet.

Figure 1: Three side-by-side figures showing an ESP32, a stepper motor, and an electromagnet.

1.2 Introduction to Cable-Driven Robots

A cable-driven robot is actuated using cables [1]. The main parts of a cable-driven robot include winches, cables, motors, and a mobile platform/end-effector. The active length of the cables are controlled by the winches, and this is what determines the position of the end-effector. Changing the active length of the cables changes the position of the end-effector.

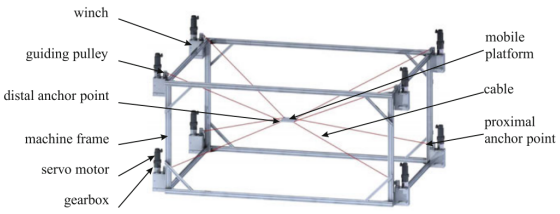


Figure 2: A cable-driven robot from [3].

Cable-driven robots offer one main advantage over serial robots: they are light weight. This is because, unlike the serial case, the motors do not have to bear the weight of other motors. Therefore, much higher speeds can be achieved.

2 Methodology

2.1 Inverse Kinematics Model

The inverse kinematics model for the robot determines the required cable lengths to achieve the desired position of the end effector. The derivation assumes a fixed base frame and utilizes geometric relations between the pulley positions and the end-effector's coordinates.

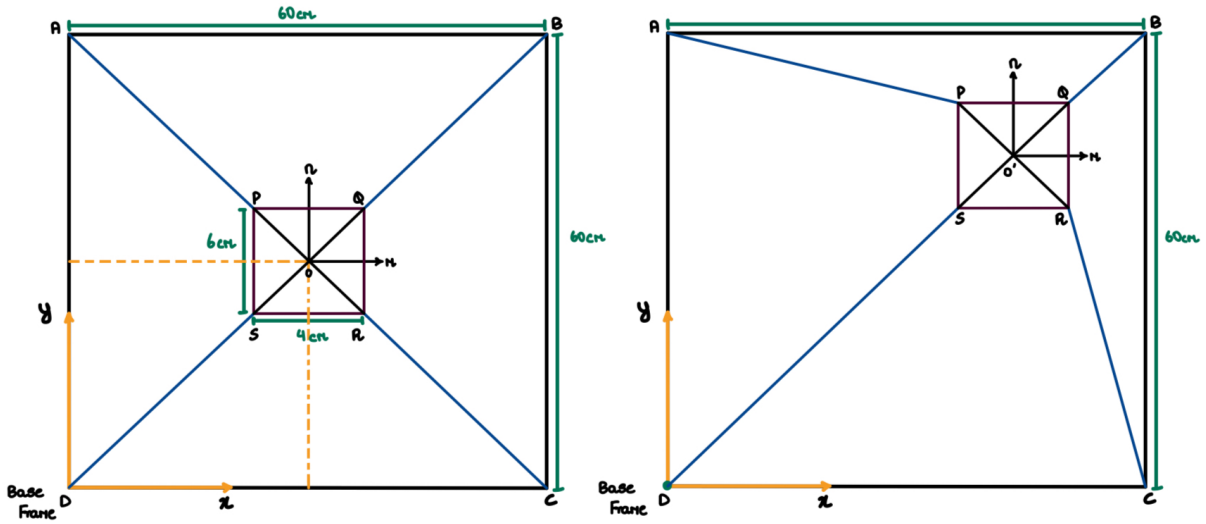


Figure 3: Inverse Kinematics Reference Diagram

Cable Length Calculation

Let the system frame be defined by four points: $A(0, 60)$, $B(60, 60)$, $C(60, 0)$, and $D(0, 0)$. The center of the end effector is initially located at $O(O_x = 30, O_y = 30)$. The coordinates of the edges of the end effector (points P, Q, R, S) are calculated with respect to this center as follows:

$$P = \begin{bmatrix} O_x - 2 \\ O_y + 3 \end{bmatrix}, \quad Q = \begin{bmatrix} O_x + 2 \\ O_y + 3 \end{bmatrix}, \quad R = \begin{bmatrix} O_x + 2 \\ O_y - 3 \end{bmatrix}, \quad S = \begin{bmatrix} O_x - 2 \\ O_y - 3 \end{bmatrix}$$

For a given end-effector center position, the cable lengths associated with each (motor - end-effector edge) pair are given by:

$$L_A = \|P - A\|, \quad L_B = \|Q - B\|, \quad L_C = \|R - C\|, \quad L_D = \|S - D\|$$

Therefore, to move the end-effector center from point 1 to point 2, we calculate the corresponding cable lengths for the two states, and then the absolute difference between the two.

$$\Delta L_A = |L_{A_{\text{new}}} - L_{A_{\text{old}}}|, \quad \Delta L_B = |L_{B_{\text{new}}} - L_{B_{\text{old}}}|$$
$$\Delta L_D = |L_{D_{\text{new}}} - L_{D_{\text{old}}}|, \quad \Delta L_C = |L_{C_{\text{new}}} - L_{C_{\text{old}}}|$$

Motor Control Logic

The motor movement in the system is defined by three parameters: the control action required, the velocity of the motor, and the angle by which it should move. Let motors A,B,C and D (the names corresponding to their positions on the system frame) be generally referred to as X.

To determine the control action for each motor:

- If $L_{\text{new}} > L_{\text{old}}$: Release cable (push)
- If $L_{\text{new}} < L_{\text{old}}$: Contract cable (pull).

The change in angle (in radians) required for each motor, with a pulley with radius r attached to it is calculated as:

$$\Delta\theta_X = \frac{\Delta L_X}{\text{r}}$$

The motor velocities are derived in the next section. By running the motors concurrently and setting variable velocity to handle each of the cable difference, the system minimizes total operation time while ensuring synchronization.

Constant Time - Variable Motor Velocity

To ensure smooth motion, a constant-time approach is used instead of constant velocities for all the motors. Fixing the cable velocity of the motor that handles the largest cable change, the velocities of the other motors are proportionally scaled down.

$$v_i = v_{\max} \cdot \frac{\Delta\theta_x}{\Delta\theta_H},$$

where $v_{\max}$ is the velocity of the motor handling the largest change in cable length, $\Delta\theta_H$ represents the largest change in angle required among the four motors to move to the desired position and $\Delta\theta_x$ represents the change in angle required in each motor X. The time required for all motors is equal:

$$t = \frac{\Delta\theta_H}{v_{\max}}.$$

This approach ensures all motors complete their movement simultaneously, reducing cable slack and ensuring smooth operation of the robot.

2.2 Planar Homography

Since the camera does not have an overhead view of the workspace, the relationship $(x, y) = s(u, v)$ does not hold. Here (x, y) is the physical coordinate, s is a scale factor, and (u, v) is the pixel coordinate. There is a linear relationship between the pixel coordinates and physical coordinates in plane described by a homography matrix [2]. The homography matrix is invertible, because unlike the general case where we a translating between a 3D point in space and a pixel in a 2D image, we are translating between a 2D plane in reality and a 2D pixel space. The homography matrix has 8 degrees of freedom, $h_1, \ldots, h_8$.

$$H = \begin{pmatrix} h_1 & h_2 & h_3 \\ h_4 & h_5 & h_6 \\ h_7 & h_8 & 1 \end{pmatrix}$$

Let X_1, X_2, X_3, X_4 be four points on a physical plane in homogeneous coordinates and x_1, x_2, x_3, x_4 the corresponding pixel coordinates in homogeneous form. Then the following equations:

$$\begin{aligned} X_1 &= Hx_1 \\ X_2 &= Hx_2 \\ X_3 &= Hx_3 \\ X_4 &= Hx_4 \end{aligned}$$

define a system of linear equations, which can be solved for $h_1, \ldots, h_8$. This is why there is a calibration phase before the robot starts moving. Now say there is a point on the video feed, p, whose physical coordinate is needed. Applying the homography matrix on the pixel coordinate, $P = Hp$, we get P, the physical coordinate corresponding to p.

2.3 Vision

We used OpenCV for object tracking and everything related to computer vision. More specifically, we used OpenCV's CSRT tracker. The figure below shows the control flow for the vision system.

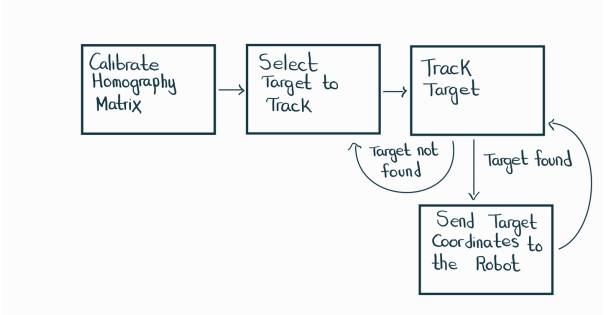


Figure 4: Control flow for the vision system.

First, marked points are selected in the video feed to calibrate the homography matrix. Next, the user selects the target to be tracked. Depending on whether OpenCV successfully tracks the object in the most recent frame, the path diverges into two. If the tracking fails, the user has to select the target again. If the tracking is successful, the server sends the physical coordinates to the EV3 and continues to track the target.

3 Results

To evaluate the performance of the robot, three distinct tests were conducted, each focusing on a specific aspect of the system's movement and precision.

3.1 Test 1: Continuous Motion Evaluation

This test involved moving the end effector through a continuous motion between four predefined points. The experiment was conducted at varying velocities across different trials to assess the robot's capability to maintain smooth and accurate movement under dynamic conditions. The following table summarizes the results obtained from the robot's movement tests at different maximum speeds. The tests involved moving the end effector (EE) to four predefined points, with the accuracy of the robot's movement being evaluated at each speed.
The actual pre-defined points are (30.5, 45.5), (45.5, 30.5), (30.5, 15.5), (15.5, 30.5).

Max Speed	**Found Points**
30	(30.7, 45.0), (45.7, 29.2), (30.7, 14.5), (16.4, 30.2)
50	(30.5, 46.0), (45.0, 30.4), (30.3, 15.0), (16.5, 30.0)
70	(30.0, 47.0), (44.0, 29.5), (29.7, 15.2), (15.5, 29.9)

Table 1: Test results for different maximum speeds and end effector movement accuracy.

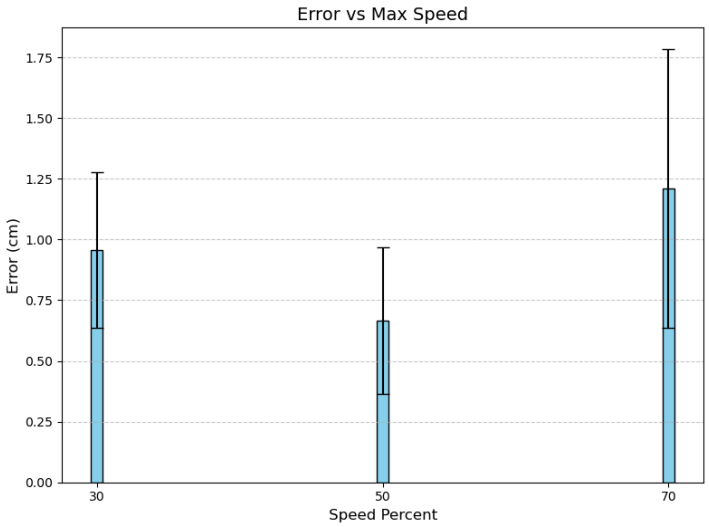


Figure 5: The graph shows the average norm differences between the actual and expected coordinates for each speed setting (30, 50, 70). These averages were calculated across four points. The black bars represent the standard deviation, indicating the variability in the norm differences at each speed.

3.2 Test 2: Repeatability Test

The aim of this test was to evaluate the robot's repeatability in reaching a specific target location. The robot was commanded to move to a randomly chosen point, $(23, 19)$, three times. For each trial, the error was calculated as the difference between the desired target position and the actual position reached by the robot. The repeatability of the system was then assessed based on the magnitude and consistency of these errors.

Trial	Displacement (x, y)	Norm Difference (Expected - Observed)
1	(+0.4, -0.7)	0.81
2	(+0.0, -0.6)	0.60
3	(+0.2, -1.3)	1.31

Table 2: Repeatability test results showing the displacement values and the norm differences.

3.3 Test 3: Homography Matrix Validation

The Homography matrix, used to map pixel coordinates to physical coordinates, was validated through this test. Four physical points were recorded, and their corresponding pixel coordinates were transformed using the Homography matrix. The test evaluated the accuracy of the transformation by comparing the computed physical coordinates with the actual recorded physical points. The average error produced by the homography transformation is 1.72 (measured using the norm of the difference).

Actual Coordinates	Homography Coordinates
(30.5, 45.5)	(31.00, 44.68)
(45.5, 30.5)	(45.03, 28.58)
(30.5, 15.5)	(29.49, 13.45)
(15.5, 30.5)	(15.13, 28.90)

Table 3: Comparison of Actual Coordinates and Homography Coordinates.

4 Discussion

4.1 Errors

We expected the errors to increase as the max speed was increased, just like it's the case for mobile rovers and serial robot arms. The data does not support this. Increasing the max speed did not have a significant effect on the accuracy of the cable-driven robot. We did, however, notice the system is more unstable and the end-effector is more prone to vibration with increase in speed.

Increasing the speed causes inaccuracies with the motor angles. Suppose the error in the motor angle is α degrees. In radians that would be 0.017α. Since the radius of the winches is 2cm, the error in the cable length is going to be $2 \cdot 0.017\alpha$. As an example, taking the error in the motor angles to be 10 degrees, the error in the cable is only going to be 0.34 cm.

4.2 Pulley Approximation

Initially, the cylindrical pulleys were custom-designed and 3D-printed. However, the printed pulleys proved unreliable, as achieving the required precision for a secure fit onto the motor shafts was challenging. The pulleys frequently slipped on the motor shafts, leading us to abandon this approach.

Instead, we approximated the cylindrical pulley using square shaped blocks using the connector pins from the EV3 kit (as seen in the System Picture), which were fixed to the motors. While this workaround addressed the slipping issue, it introduced a new source of error. Unlike cylindrical pulleys, which allow smooth and predictable cable movement based on the relationship

$$s = l \cdot \theta$$

where s is the length of cable released, l is the radius, and θ is the angular displacement, the rectangular blocks resulted in non-linear changes in cable length. This inconsistency likely caused irregularities in cable retraction and release, further impacting system performance.

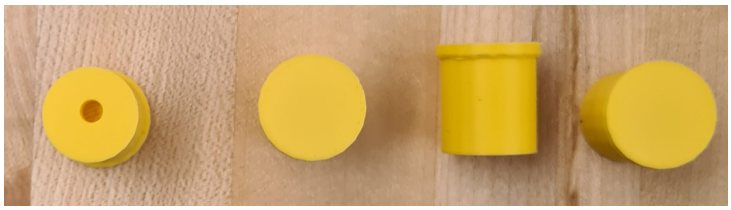


Figure 6: Initial Pulley Attempt

4.3 Other Implications

4.3.1 End-Effector Weight Distribution

As mentioned in the introduction, we were unable to execute the pick-and-drop plan of the project due to cable slack during motion. Since this is a planar robot, the four cables provide tension along the plane, whereas any load (such as an electromagnet, controller, or object) on the end-effector acts perpendicular to this plane. A simple model would be unable to counteract such a significant force, resulting in additional cable slack and undesired outcomes.

We devised a mechanical solution to address this issue, enabling the outer system to bear the load while allowing the end-effector to move freely within the outer shell above it. However, implementing this solution was beyond our initial project scope, and due to time constraints, we decided not to proceed with it.

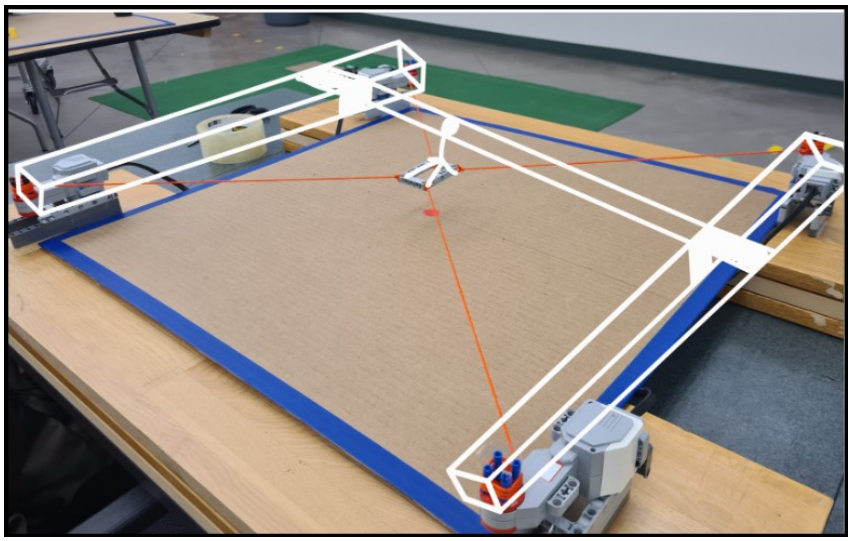


Figure 7: Proposed Mechanical Solution

4.3.2 Difficulty in Applying Broyden's Method

Applying Broyden's method to this robot presented significant challenges. A key issue was the sensitivity of the method to the accuracy of the initial Jacobian estimate. If the initial Jacobian was not sufficiently precise, the resulting movement of the end-effector would be highly unstable. Given that the end-effector is under tension from all sides, any instability could lead to excessive stress on the cables and the structure, potentially causing system failure during trials.

Furthermore, the iterative nature of Broyden's method requires updates to the Jacobian approximation based on system responses. In our case, the tension-based constraints and the inherent instability made it infeasible to conduct the necessary iterations without risking damage to the robot. As such, we opted not to proceed with this approach.

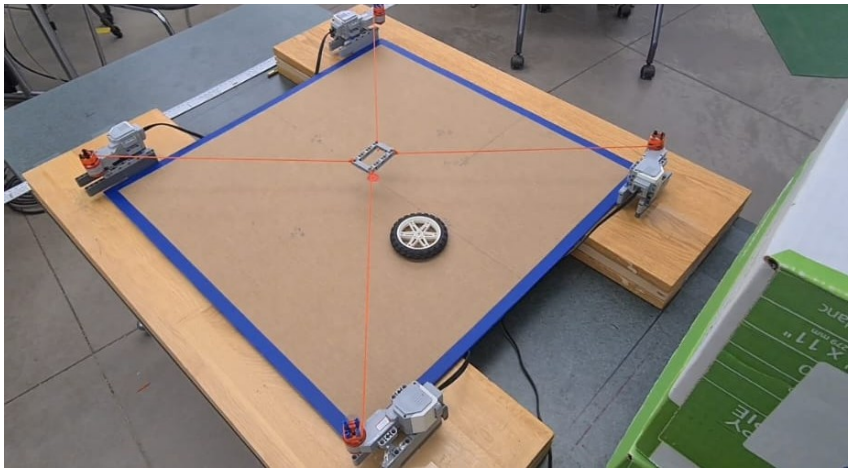


Figure 8: Picture of the cable-driven robot.

5 Conclusion

In our project we designed a cable-driven robot that could continously track and follow objects in real time. Although we were unable to implement our initial plan, the robot still successfully performs real-time object tracking. We were not far off from implementing our initial plans. The results highlight the advantages of cable-driven robots over serial robot arms, in terms of the speeds that can be achieved and the accuracy along with it. At high speeds, a serial robot arm may be inaccurate, but this is not the case for a cable-driven robot. Errors in motor angles have a minimal impact on end-effector positioning. However, increasing motor speed was found to introduce instability due to cables slipping off the pulleys. Some future improvements include incorporating cable tension into our model to reduce cable slack incorporating a mechanical counterbalance to offset gravitational forces, enabling the system to handle weight more effectively.

References

[1] M. Arslan and L. Birglen. Design, control, and experiments of a low-cost open-source planar cable-driven parallel robot. In *Proceedings of the CCToMM Symposium on Mechanisms, Machines, and Mechatronics*. CCToMM, 2023.

[2] Peter I. Corke. *Robotics, Vision and Control: Fundamental Algorithms in MATLAB*. Springer, third edition, 2022.

[3] Andreas Pott. *Cable-Driven Parallel Robots: Theory and Application*. Springer, 2018.

A Appendix: Additional Details

A.1 Setting Up Scrcpy and OBS Studio

Install Scrcpy, and OBS Studio in your laptop, follow these steps:

1. Enable USB Debugging on your phone under `Settings > Developer Options`.
2. Connect your phone to the computer via USB.
3. Run `scrcpy` in the terminal to mirror the phone's screen to the computer.
4. In OBS Studio, Start Virtual Camera after adding scrcpy source.

This setup allows you to use your phone as an external camera in the code.

A.2 Working Directory Setup

```
src/
|-- ev3-client/
|   |-- sample_client.py              Client Code for performing Visual Tracking
|   |-- System_constT.py              Attributes and Functions for using System
|   |-- System_InverseKinematics.py   Client Code for running direct Inv. Kin. Motion
|-- vision-server/
    |-- myenv/
    |-- vision_server                 Code for doing Homography fn. computation
```