Applied Robotics

FRTN85

Project report:

Self-balancing robot
(Stewart Platform)

Group 6

Authors:

Tomás Esteves	to5130es-s@student.lu.se
Gabriel Carreira Figueiredo	ga5333fi-s@student.lu.se
Juan José Quintero Sarmiento	ju7658qu-s@student.lu.se
Simon Ziegenbalg	si6258zi-s@student.lu.se
Mark Peter Lundager	ma4713lu-s@student.lu.se

Autumn semester 2025

Contents

1 Introduction

A Stewart platform, also referred to as Stewart-Gough platform, is a parallel robot that consists of a base, a platform and six actuated links between the two. The six actuators provide six degrees of freedom to the platform.

1.1 Background

Stewart originally proposed this mechanism as an elegant design for a flight simulator [5]. Other interesting applications include motion compensation in offshore applications [7] and spacecraft docking [4]. Linear actuators, either hydraulic or electric, are what is most commonly used in industrial applications and studied in literature. The Stewart platform used in this project instead uses rotary servo motors to acuate the platform.

1.2 Project aim

The project aims to realize a Stewart platform that keeps the platform horizontal when the base is tilted.

The problem can be broken down into different aspects, namely

- the hardware,
- the actuation and sensing,
- the inverse kinematics,
- the forward kinematics,
- the control.

2 Hardware Overview

The experimental Stewart platform is a six-degrees-of-freedom (6-DOF) parallel manipulator composed of a fixed base plate and a movable top plate connected by six identical leg links. Motion of the top plate (translations along and rotations about the three principal axes) is achieved by coordinated actuation of the six legs. In our project, each leg consists of a servo-driven base horn, a rigid connecting rod, and a ball-joint link to the top plate. The physical layout and geometry allow for the avoidance of mechanical interference while providing adequate workspace for typical small-scale attitude adjustments.

2.1 Geometry and mechanical layout

The top plate's radius is 7.8 cm, while the base plate's is 9.3 cm. The base plate's six servo mounting points feature a separation angle of 25° between nearby servos and 95° between servos that are further apart. The top plate attachment pattern is complimentary, with 15° separating the nearest linkages and 105° separating the further-flung links. Each servo drives a rigid rod that is 17 cm long to the top plate using a lever arm that is 1.64 cm long.

2.2 Actuation – servomotors.

Six MG995 servomotors, one on each leg, are installed on the base plate to produce actuator action. The top platform is moved by each servo rotating its horn, which alters the effective leg length and position. Their main function in the system is to translate electrical control commands into accurate angular displacements at each leg's base; the intended platform position is achieved through coordinated management of these angular displacements. An external power source is linked to the servo motors.

2.3 Sensing – IMU.

A single inertial measurement unit (IMU) of the type MPU-6050 is responsible for orientation sensing of the top plate. The upper platform's current roll and pitch are determined by fusing accelerometer and gyroscope signals provided by the IMU. The feedback signal that the controller uses to adjust and stabilize the platform's position is created from these attitude readings.

The IMU is rigidly attached to the moving platform using a 3D-printed mount. Figure 1 illustrates the structure of the IMU mount. The IMU's outer edge is supported by a surrounding frame, and it is clamped between the two parts of the mount at the contact areas labeled (a) and (b) in Figure 1. The flat surface (c) ensures that the IMU plane remains parallel to the platform plane. Alignment pin (d) fits into a corresponding hole in the platform to set the horizontal position. A cable tie, threaded through two holes in the platform and a groove (e) in the mount, secures the mount in place and simultaneously defines its orientation. This ensures that the IMU's pitch and roll axes are aligned with those of the platform. Figure 2 shows the final 3D-printed mount.

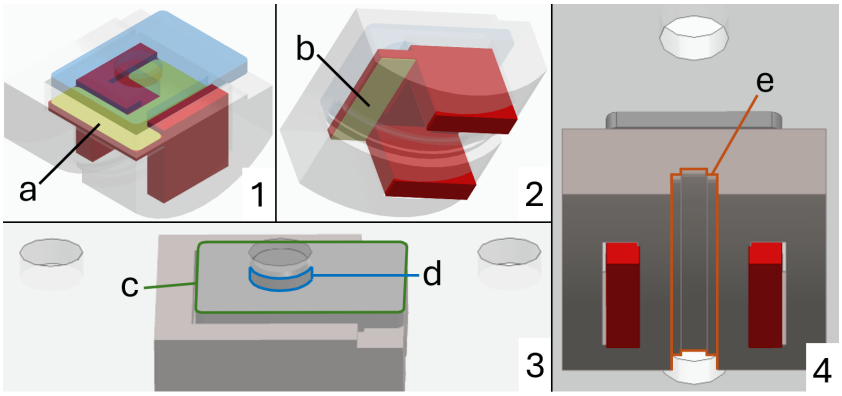


Figure 1: Different depictions of the IMU mount illustrating both the connection between IMU and mount and the connection between mount and platform

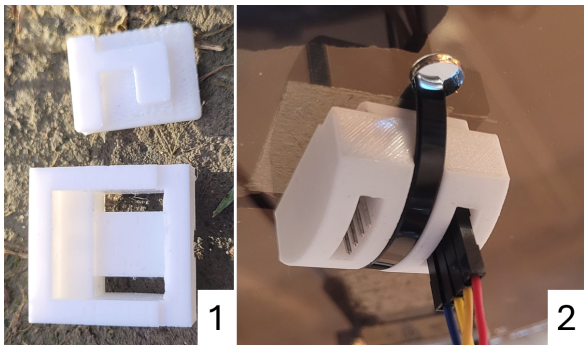


Figure 2: 3D printed mount detached and attached to the underside of the platform

2.4 Computation and control.

An Arduino Uno interfaces with the IMU and the six servos. It continuously reads the IMU, computes the required leg adjustments, and issues commands to each servo motor. In this closed-loop arrangement, the micro controller implements the control law that maps desired platform orientation and position to the servo motors, using real-time IMU feedback to close the loop.

3 Actuation and sensing

3.1 Driving of servo motors

The servo motors are driven by commands from the Arduino. Each servo has a signal wire connected to one of the Arduino's digital pins. In the software, the communication is handled by the Servo library for Arduino [1], which moves each servo to a specified angle.

3.2 Measurement of platform angles

The IMU includes a gyroscope and an accelerometer. The gyroscope measures the angular velocity around all three axes, while the accelerometer measures linear acceleration, which for a stationary object is the gravitational acceleration.

Firstly, every time the Arduino is powered on, the system calibration starts by setting all motors to the neutral position so the platform is horizontal, which is assumed as zero roll and pitch. Then, 200 readings are taken from the IMU's gyroscope and accelerometer. The gyroscope is calibrated while the IMU is stationary by averaging the bias on each axis, and the accelerometer is calibrated in this leveled state to compute reference offsets that align the readings with the assumed zeros.

Secondly, the pitch and roll angles of the platform are estimated separately from the gyroscope measurements and the accelerometer measurements. For the gyroscope, this is achieved by

integrating the angular velocities. For the accelerometer, the following equations are used [3].

$$\theta = \tan^{-1}\left(\frac{-a_x}{\sqrt{a_y^2 + a_z^2}}\right)$$
$$\psi = \tan^{-1}\left(\frac{a_y}{\sqrt{a_x^2 + a_z^2}}\right)$$

These equations calculate two independent angles describing the inclination of the x- and y-axis relative to the horizontal.
The IMU's measurements are visually verified through a virtual depiction of the measured orientation (Figure 3).

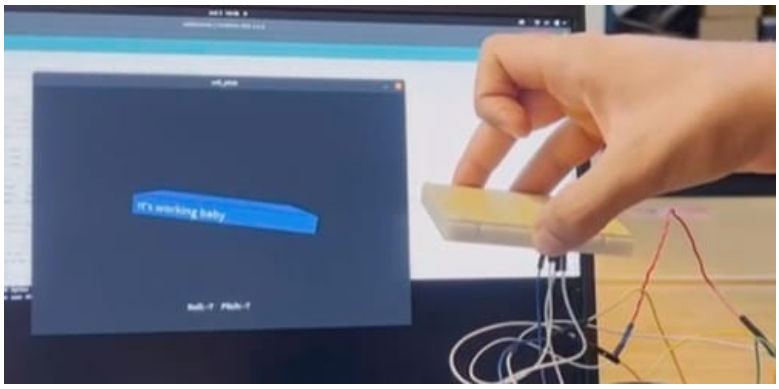


Figure 3: Visual validation of the IMU measurements

Finally, the angular position obtained from integrating the gyroscope data drifts over time but remains stable during platform movements, including vibrations. The accelerometer, on the other hand, provides absolute measurements but is affected by accelerations other than gravity. Therefore, the two angle estimates are combined using a complementary filter, where they are weighted to take advantage of the strengths of both sensors. Figure 4 shows the progression of a measurement obtained in this way, which exhibits low noise and returns to zero when the platform is returned to a horizontal position.

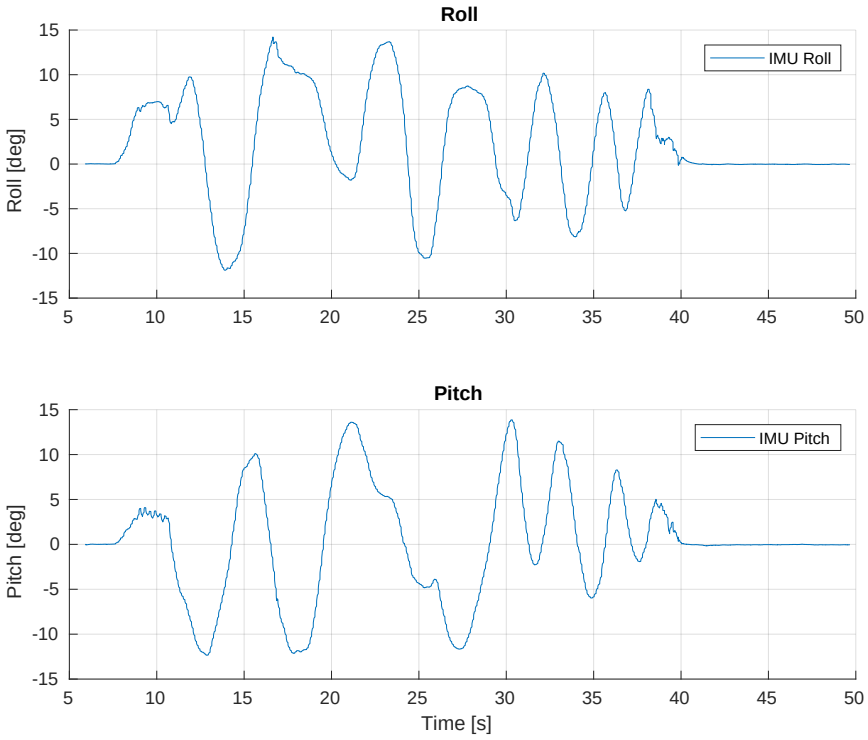


Figure 4: Exemplary IMU measurement

4 Inverse kinematics

4.1 Algorithm used

In order to be able to control the robot correctly, we needed to find a function that maps the joint values (corresponding to the respective six motor angles from each servo) to the position and orientation of the upper platform. The algorithm described is taken from [2].

In order to find this mapping, a geometric relation between the joint angles and the translation/orientation of the upper platform has to be found. We start by defining the base frame ${}^{b}\mathcal{O}$ and the platform frame ${}^{p}\mathcal{O}$, which are related through a translation vector T and a rotation matrix R_0 [2].

We can obtain the desired length for each leg of the platform ($\mathbf{l_k}$), given that we know what value for translation and rotation of the upper platform frame we want with respect to the base frame. In addition, we need to measure the distances from the origin of the frames to the anchor points of the motors (p_k) and for the platform (b_k). From Figure 5, we can get a direct expression for the desired length ($\mathbf{l_k}$) as a vector:

$$l_k = T + R_0 p_k - b_k \tag{1}$$

For the purposes of this project, we use a constant translation vector T that corresponds to the height of the platform frame from the base frame in the z direction of this last frame. This

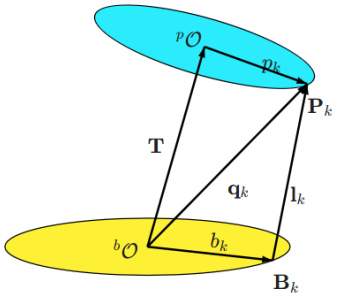
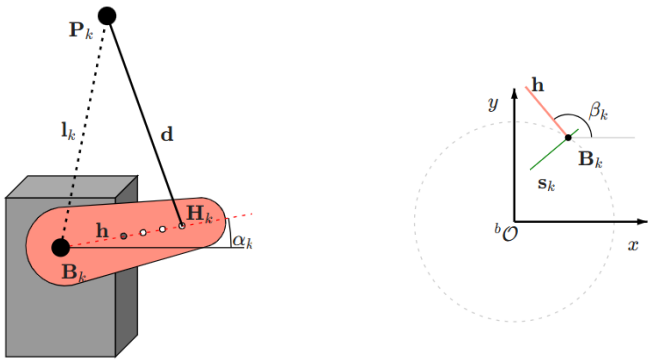


Figure 5: Kinematic parameters of the Stewart platform. Taken from [2].

means that both frames will be aligned on a vertical axis without deviating, so the platform will only tilt but not move around. In addition, we only take into account the roll and pitch angles of the platform when defining the rotation matrix, since these angles are the only kinematic measurements needed to maintain the platform horizontal at all times.

For a Stewart platform that uses linear actuators on its legs, this procedure will be enough, but we need to calculate the respective angle of the motor (α_k) for each respective leg length of the Stewart platform [2]. In Figure 6a and Figure 6b, the other parameters that will be used to obtain the angles of the motors (α_k) are defined. $\mathbf{h}$ is the vector that goes from the center of the motor B_k to the anchor of the rod with the lever arm H_k, P_k is the anchor of the rod to the platform, $\mathbf{d}$ is the length of the rod attached to the anchor of the motor and the platform and β_k is the angle between the horizontal x axis of the base frame and the vector $\mathbf{h}$ of the servomotor [2]. The measurements from subsection 2.1 are used.



(a) Servomotor and rod connections.

(b) View from above of the base platform

Figure 6: Kinematic parameters of the servomotor attached to the rod. Taken from [2].

With the measurements of the platform, the inverse kinematics using trigonometric functions for finding the angles α_k are given as [2]:

$$\alpha_k = \sin^{-1}\left(\frac{g_k}{\sqrt{e_k^2 + f_k^2}}\right) - \text{atan2}(f_k, e_k) \tag{2}$$

Where:

$$e_k = 2|\mathbf{h}|\, l_k^{(z)} \tag{3}$$

$$f_k = 2|\mathbf{h}|\left(\cos(\beta_k)\, l_k^{(x)} + \sin(\beta_k)\, l_k^{(y)}\right) \tag{4}$$

$$g_k = |l_k|^2 - \left(|\mathbf{d}|^2 - |\mathbf{h}|^2\right) \tag{5}$$

Where we use the x,y and z components of the vector $\mathbf{l_k}$ and the respective magnitudes of $\mathbf{d}$ and $\mathbf{h}$. All the information regarding the geometric calculations for finding the respective inverse kinematics equations can be found on reference [2].

4.2 Validation and test

After defining the inverse kinematics equations, we used them to build, from scratch, a Python simulator to visually verify their correctness, as shown in Figure 7. In this simulator, we could interactively vary the platform's desired roll and pitch and observe the corresponding servo angles produced by the equations, plotting the whole system for visual support.

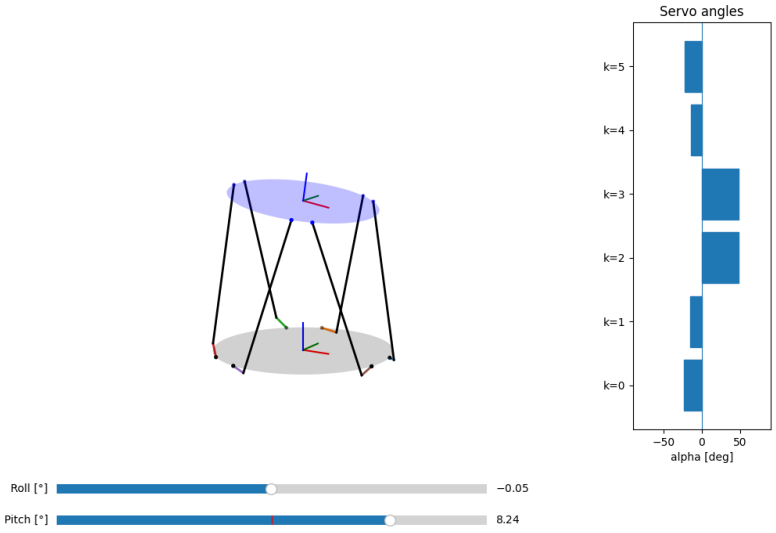


Figure 7: Inverse kinematics python simulator

After confirming that the attitude changes resulted in output angles that made physical sense, we implemented the same equations in C++ to run on the real robot.

To validate the equations on the real platform, we verified whether the manually introduced roll and pitch values matched the IMU readings. We also ensured that the IMU reference frame was correctly aligned with the inverse kinematics platform frame, so that both measurements represented the same physical orientation. The results showed that the roll and pitch angles introduced in the inverse kinematics function matched the IMU readings within a maximum

error of 10%, confirming that the inverse kinematics behavior was consistent with the real platform.

Below, we can see the serial monitor output when a command of pitch = 0° and roll = 3° was given. The IMU measurements demonstrate good accuracy, staying close to the commanded values.

```
IMU r=2.73 p=-0.16 | CMD r=3.00 p=0.00
IMU r=2.73 p=-0.15 | CMD r=3.00 p=0.00
IMU r=2.73 p=-0.16 | CMD r=3.00 p=0.00
IMU r=2.72 p=-0.15 | CMD r=3.00 p=0.00
IMU r=2.72 p=-0.15 | CMD r=3.00 p=0.00
IMU r=2.73 p=-0.15 | CMD r=3.00 p=0.00
IMU r=2.73 p=-0.14 | CMD r=3.00 p=0.00
IMU r=2.77 p=-0.11 | CMD r=3.00 p=0.00
IMU r=2.77 p=-0.12 | CMD r=3.00 p=0.00
IMU r=2.75 p=-0.12 | CMD r=3.00 p=0.00
IMU r=2.74 p=-0.12 | CMD r=3.00 p=0.00
```

5 Forward Kinematics

The forward kinematics (FK) problem for our version of the Stewart platform consists of determining the pitch and roll based on the angles of each servo motor. Yaw is ignored since it is irrelevant for maintaining the platform perpendicular to gravity (balanced). We were inspired by the article *A Geometric Approach for Real-Time Forward Kinematics of the General Stewart Platform* [6], where the FK is solved numerically.

5.1 Algorithm

5.1.1 The Residual

The FK algorithm takes the points H_k and p_k, an initial guess $\mathbf{T}$ and $\mathbf{R}$, a maximum number of iterations, an error tolerance, and an update tolerance. The servomotor angles α_k do not appear explicitly, as they are incorporated into

$$H_k(\alpha_k, \beta_k) = B_k + h(\alpha_k, \beta_k),$$

where β_k is fixed by the geometry.

First, we compute the current leg length error and the unit direction vector for each leg:

$$\begin{aligned} r_i &= \|\mathbf{P}_i - \mathbf{H}_i\| - d, \\ \hat{\mathbf{u}}_i &= \frac{\mathbf{P}_i - \mathbf{H}_i}{\|\mathbf{P}_i - \mathbf{H}_i\|}. \end{aligned} \tag{6}$$

Since $\mathbf{p}_k$ is expressed in the local frame of the top platform, performing the subtraction $\mathbf{p}_k - \mathbf{H}_k$ directly would not make sense, as $\mathbf{H}_k$ is defined in the base frame. We therefore transform $\mathbf{p}_k$ into the base frame:

$$\mathbf{P}_k = \mathbf{R}_i\, \mathbf{p}_k + \mathbf{T}_i. \tag{7}$$

We use a simple initial guess for the rotation and translation: the nominal platform height z_0 as the translation, and no rotation for $\mathbf{R}$. The subscript i denotes the current iteration.

$$\mathbf{T}_0 = \begin{bmatrix} 0 \\ 0 \\ z_0 \end{bmatrix}, \qquad \mathbf{R}_0 = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

If $\|\mathbf{r}\|$ is smaller than the error tolerance, the algorithm terminates here.

5.1.2 The Jacobian

Our goal is to generate the Jacobian and solve Equation 8, which describes how to modify $\mathbf{T}$ and $\mathbf{R}$ so that the constraint in (6) is satisfied. We construct the Jacobian as shown in (9). The Jacobian, in our case a 6×6 matrix (six legs and six platform parameters), maps small rotations and translations of the top platform to changes in the leg lengths.

$$\begin{aligned} \mathbf{r} + \mathbf{J}\,\Delta\mathbf{x} &= 0, \\ \Delta\mathbf{x} &= \begin{bmatrix} \omega_x & \omega_y & \omega_z & t_x & t_y & t_z \end{bmatrix}^\top. \end{aligned} \tag{8}$$

Each row of the Jacobian $\mathbf{J}$ is given by

$$\mathbf{J}_i = \begin{bmatrix} -\,\hat{\mathbf{u}}_i^\top\,\mathbf{R}_i[\mathbf{p}_i]_\times & \hat{\mathbf{u}}_i^\top \end{bmatrix}, \tag{9}$$

where $[\mathbf{p}_i]_\times$ denotes the skew-symmetric matrix of $\mathbf{p}_i$.

To handle potential ill-conditioning, we compute the update $\Delta\mathbf{x}$ by solving the damped least-squares (Levenberg–Marquardt) system:

$$(\mathbf{J}^\top\mathbf{J} + \lambda^2\mathbf{I})\,\Delta\mathbf{x} = -\,\mathbf{J}^\top\mathbf{r}. \tag{10}$$

Finally, we update $\mathbf{R}_i$ and $\mathbf{T}_i$ using $\Delta\mathbf{x}$:

$$\begin{aligned} \mathbf{T}_{\text{new}} &= \mathbf{T}_{\text{old}} + \Delta\mathbf{x}[3:6], \\ \mathbf{R}_{\text{new}} &= \mathbf{R}_{\text{old}}\ \exp([\Delta\mathbf{x}[0:3]]_\times), \end{aligned} \tag{11}$$

where $[\cdot]_\times$ denotes the skew-symmetric operator that maps a 3-vector to its corresponding matrix in $\mathfrak{so}(3)$.

If $||\Delta x|| < Tolerance_{update}$ we terminate the algorithm. Otherwise we compute our P_k points and residues again to continue into the next iteration until we either hit a tolerance or the maximum number of iterations.

5.2 Validation and test

Since the inverse kinematics (IK) were implemented prior to FK, we could use that to validate the FK by inserting a pitch and roll of our choosing into IK, take the output and put it in FK. Then we compare the output of FK with the input of IK. As we initially implemented both FK and IK in python, we also performed this test in python and the FK produced the correct values.

5.3 Practical limitations

While the forward kinematics implementation was successfully completed and verified numerically against the inverse kinematics, we realized that the servomotors did not provide angle feedback. Hence, we could not utilize FK to balance the platform. Nonetheless, it would be interesting to compare the performance of an FK–IK-based control approach with our current approach. Since drift will accumulate due to commands not being executed completely, drift will accumulate in the total roll and pitch. With FK, the orientation would instead rely on the accuracy of the feedback from the servo motors.

Similar to the simulator of the IK, we implemented a simulator of FK in an attempt to see differences between the two approaches. However, accurately modeling effects such as servo lag, drift, and IMU noise to compare is difficult and a better comparison would be to acquire servomotors with built-in feedback and experiment on the physical platform.

6 Attitude Controller

6.1 The control challenge

At this stage, the inverse kinematics (IK), sensors, and actuators are fully calibrated and operational, allowing us to begin controlling the platform to maintain its balance when the base is tilted. However, the main challenge is that the IMU provides the absolute attitude of the platform with respect to the gravity reference, while the inverse kinematics operate in the reference frame of the **base**. Therefore, we cannot simply apply the IMU error (difference from zero roll/pitch) directly to the IK, since that would ignore the relative orientation between the base and the platform.

To address this, we define three main angles:

- **IMU reading** (θ_{IMU}): the current absolute attitude of the platform with respect to the gravity reference.
- **Current IK** ($\theta_{\text{IK,prev}}$): the attitude command applied in the previous control step, representing the last known relative orientation between the platform and the base.
- **Base angle** (θ_{base}): the absolute orientation of the base with respect to the gravity reference. Since only one IMU is mounted on the platform, this value is not directly measured but estimated as the difference between the IMU reading and the last IK command.

The relationship between these quantities is:

$$\theta_{\text{base}} = \theta_{\text{IMU}} - \theta_{\text{IK,prev}} \tag{12}$$

To maintain the platform level relative to the base, the new target attitude for the IK must compensate for this base tilt:

$$\theta_{\text{IK,target}} = -\theta_{\text{base}} \tag{13}$$

In other words, the controller must keep track of the previous IK command to correctly compute the new reference from the IMU measurement. This ensures that small attitude errors are accumulated properly, maintaining the correct relative alignment between the platform and the base.

Figure 8 illustrates the relationship between the IMU reading, current IK, base angle, and target IK.

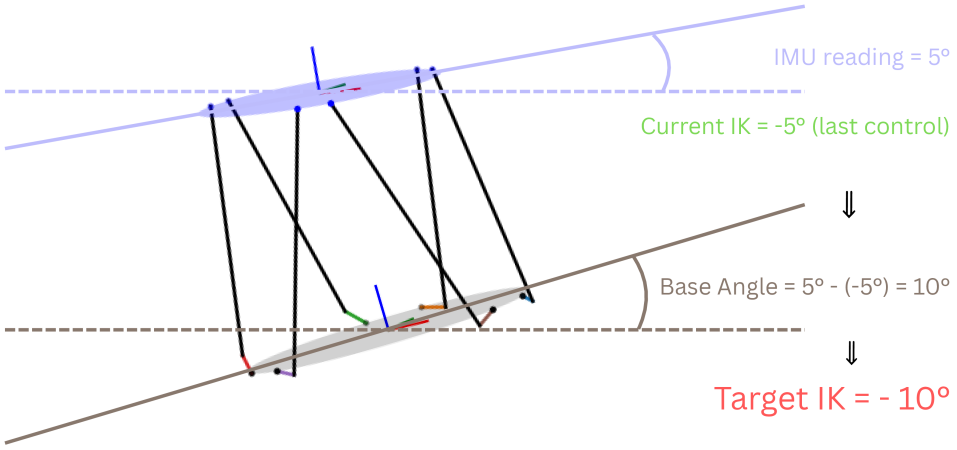


Figure 8: Example illustration

We used this approach because we were working only with one IMU mounted on the platform. If the IMU was installed on the base, it would directly measure θ_{base}, removing the need to keep track of the current IK value. In that case, the base IMU measurement could be used directly as the input (target) for the inverse kinematics, making the control conceptually simpler. However, this would also require the IK model to be perfectly accurate, since any modeling error would directly affect the balance. A more robust configuration could include two IMUs, one on the base and one on the platform, to both track the absolute base orientation and verify that the IK outputs truly keep the platform level.

6.2 Controller Design

The actuator modules already include internal control, implementing their own factory-tuned *inner control loop*. This allows each actuator to accurately follow a reference command given directly in degrees. Therefore, our task was to design an *outer control loop* that determines the necessary attitude references to keep the platform horizontal. These attitude references are then passed through the inverse kinematics, which convert them into the corresponding motor angle commands.

We started with a proportional controller, multiplying the IMU attitude error by a gain to update the attitude reference angles as described before. After tuning, we added an integral term to eliminate steady-state errors and ensure the platform remained level once stabilized. As shown in Figure 9, this kept the platform near level (within a 0.5 degrees dead-zone), since the IMU roll and pitch returned toward zero when the base was disturbed. The traces *REF_roll*

and *REF_pitch* are the controller attitude outputs fed to the inverse kinematics. However, this design produced large oscillations.

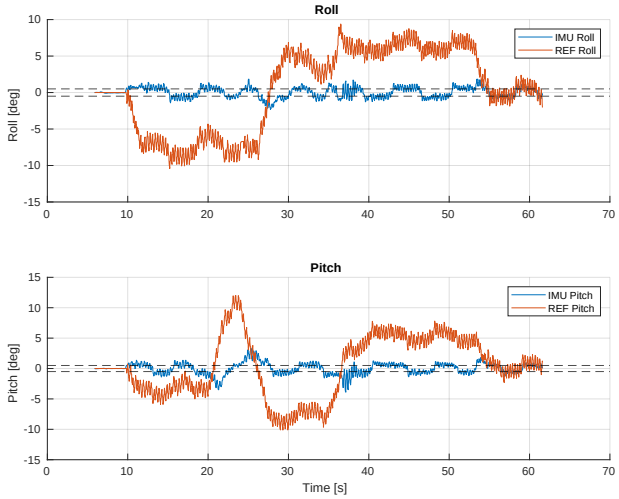


Figure 9: IMU measurements and controller references

This occurred because new attitude references were being applied while the actuators were still responding to the previous ones, resulting in accumulated dynamic lag due to actuator inertia.

To mitigate this issue, we added a derivative term to the controller to provide damping and counteract the inertia of the system. This reduced oscillations but did not eliminate them completely. Further debugging showed that the control loop was running at a much higher frequency than the mechanical response time of the actuators, causing new commands to be issued faster than the motors could physically react.

By significantly reducing the controller frequency to 100 Hz, the oscillations were largely suppressed without noticeable degradation in performance. We also observed that the derivative term provided no noticeable improvement in performance, so it was ultimately removed. The gains chosen in final tuning are $K_P = 2$ and $K_I = 8$.

Figure 10 shows the final performance of the implemented controller. The oscillations were significantly reduced, and the IMU measurements remain well contained within the $\pm 0.5^\circ$ dead zone when the platform is tilted. This confirms the correct and properly calibrated operation of the designed controller.

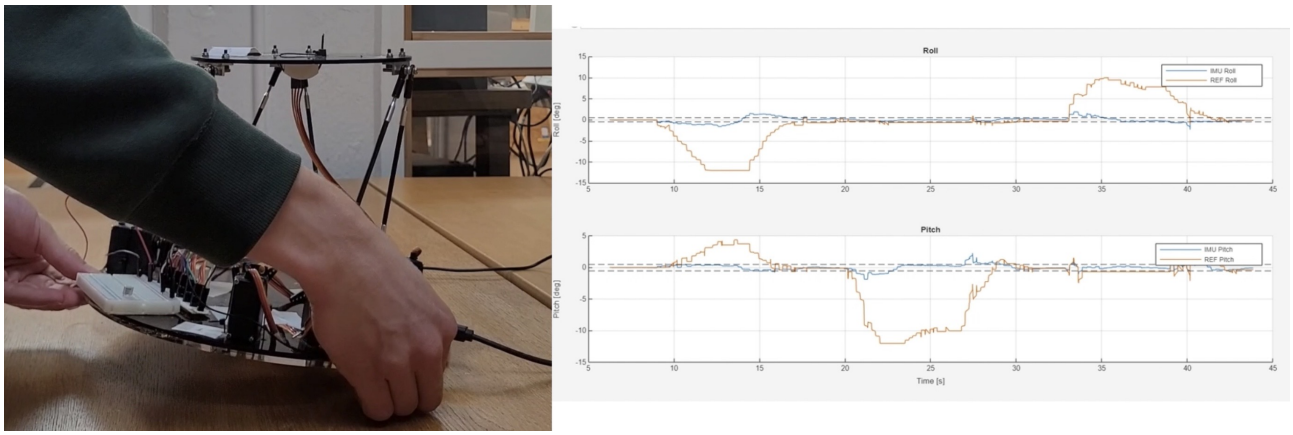


Figure 10: IMU measurements and controller outputs after oscillation suppression.

7 Conclusion

To conclude , it was possible to create a self-balancing Stewart platform that keeps the top plate level even when the base tilts. Precise inverse kinematics, efficient sensor integration, and an Arduino micro controller-based responsive attitude control system were used to accomplish this. The platform aggressively counteracts base disturbances to stay almost horizontal through continual servo adjustments and real-time IMU feedback. The top plate remained within around 0.5° of level during operation, according to final tests, demonstrating the efficiency of the control approach and appropriate system calibration.

To supplement the inverse kinematics, a forward kinematics method was also created to infer the platform's orientation from the servo angles. Although this technique was effectively put into practice, its application in the active balancing process was limited because it was not completely incorporated into the live control loop. There is still room for development in this area in the future. The platform's performance and dependability could be further increased by including the forward kinematics approach into later versions, which could also improve system calibration and offer redundant feedback.

8 Contributions

Gabriel: Simulator, Forward Kinematics, Attitude Controller Design and Final Tuning
Mark: Simulator, Forward Kinematics, Attitude Controller Design, Hardware Setup
Juan: IMU Filtering and Calibration, Inverse Kinematics Implementation, Attitude Controller Design and Final Tuning
Tomás: IMU Filtering and Calibration, Inverse Kinematics Implementation, Attitude Controller Design and Implementation
Simon: Research of Inverse Kinematics and Arduino libraries, IMU mount, Attitude Controller Design and Final Tuning

9 Source Code Repository

The complete source code developed throughout this project is publicly available on GitHub. It contains all scripts and functions, used for implementation, testing, and analysis. The repository can be accessed at https://github.com/MarkLundager/FRTN85-Pproject-Group6

References

[1] Arduino-Libraries. *Servo Library for Arduino*. https://github.com/arduino-libraries/Servo. Accessed: 2025-10-22. 2024.

[2] Robert Eisele. *Inverse Kinematics of a Stewart Platform*. https://raw.org/research/inverse-kinematics-of-a-stewart-platform/. Accessed: October 23, 2025. Feb. 2019.

[3] Christopher J. Fisher. *AN-1057: Using an Accelerometer for Inclination Sensing*. Tech. rep. Accessed: 2025-10-22. Analog Devices, 2007.

[4] Michael Hardt et al. "Simulation and control of an active Stewart platform for docking and berthing of space vehicles". In: 6 (Jan. 2006), pp. 4196–4203.

[5] D. Stewart. "A Platform with Six Degrees of Freedom". In: *Proceedings of the Institution of Mechanical Engineers* 180.1 (1965), pp. 371–386. DOI: 10.1243/PIME_PROC_1965_180_029_02.

[6] Fangfang Yang et al. "A Geometric Approach for Real-Time Forward Kinematics of the General Stewart Platform". In: *Sensors* 22.13 (2022), p. 4829. DOI: 10.3390/s22134829.

[7] Cai Yunfei et al. "Model Analysis and Modified Control Method of Ship-Mounted Stewart Platforms for Wave Compensation". In: *IEEE Access* PP (Dec. 2020), pp. 1–1. DOI: 10.1109/ACCESS.2020.3047063.