6 Degree-of-Freedom Ball Stabilizer

MMA4004 Mechatronics and System Integration

Modestas Sukarevicius[a], Jon Urcelay[a]

[a]*Department of ICT and Natural Sciences, Norges teknisk-naturvitenskapelige universitet, Ålesund, Norway.*

November, 2025

Abstract

The goal and outcome of this project is the creation of a 6-degree-of-freedom platform, including simulation, that is able to control the position of a given ball, compensating for movements of the base. The platform is able to stabilize the ball in the center or make it follow a circular trajectory. For this purpose, two separate controllers have been implemented, Proportional Integral Derivative (PID) and Linear Quadratic Regulator (LQR), which can be selected and tuned from a user interface.

1. Introduction

A 6-degree-of-freedom (6-DoF) platform, often referred to as a Stewart platform or hexapod, is a parallel manipulator capable of precise motion in all six spatial degrees of freedom: three translational (surge, sway, and heave) and three rotational (roll, pitch, and yaw).

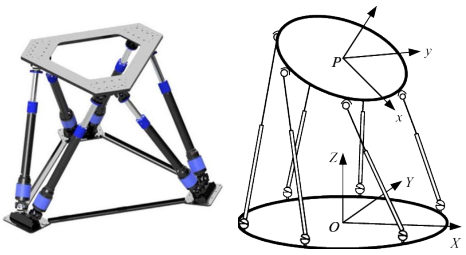


Figure 1: 6-degree-of-freedom platform.

1.1. Project objective

The primary objective of the project is to design and build a custom 6-DoF platform that automation students at NTNU can use. The idea is that students will replicate the platform, learning about prototyping, formulating the Inverse Kinematics (IK) and implementing control theory. For completing this task, the next list of objectives are defined:

- Solve the IK, for obtaining the six motor angles from the 6 platform coordinates.
- Obtain a state space model of the system, and use it for creating a simulation of the platform.
- Implement PID and LQR control in both simulation and real world, for stabilizing the ball in the center of the platform and for following a circular trajectory.
- Integrate an IMU in the base of the platform, to compensate for external rotations(roll and pitch).
- Make an all-in-one user interface, from where the platform can be controlled.

1.2. Report structure

The report is structured as follows: Section 2 reviews the theoretical background; Section 3 details the methods and materials used for the project; Section 4 presents and discusses the results for the different prototypes and control systems; and Section 5 concludes with insights, limitations, and recommendations for future enhancements.

2. Theory

2.1. PID Control

Proportional-Integral-Derivative (PID) controllers minimize tracking error $e(t) = r(t) - y(t)$ using a control input in the following form:

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau)\,d\tau + K_d \frac{de(t)}{dt}, \quad (1)$$

where K_p, K_i, and K_d are proportional, integral, and derivative gains.

2.2. *LQR Control*

Linear Quadratic Regulator controllers minimize tracking error $e(t) = r(t) - y(t)$ using a control input in the following form:

$$u(t) = -\mathbf{K}x(t), \tag{2}$$

where $\mathbf{K}$ is the optimal state-feedback gain for the linear system $\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}\mathbf{u}$, which minimizes the following cost equation:

$$J = \int_0^\infty \left(\mathbf{x}^T\mathbf{Q}\mathbf{x} + \mathbf{u}^T\mathbf{R}\mathbf{u}\right) dt, \tag{3}$$

where $\mathbf{Q} \geq 0$ penalizes states and $\mathbf{R} > 0$ penalizes control inputs. The solution for the optimal gain $\mathbf{K}$ is given by:

$$\mathbf{K} = \mathbf{R}^{-1}\mathbf{B}^T\mathbf{P}, \tag{4}$$

where $\mathbf{P} > 0$ is the unique symmetric positive-definite solution to the *continuous-time Algebraic Riccati Equation (ARE)*:

$$\mathbf{A}^T\mathbf{P} + \mathbf{P}\mathbf{A} - \mathbf{P}\mathbf{B}\mathbf{R}^{-1}\mathbf{B}^T\mathbf{P} + \mathbf{Q} = \mathbf{0}. \tag{5}$$

Numerical solvers compute $\mathbf{P}$ iteratively due to the nonlinearity of the ARE.

2.3. *Forward and Inverse Kinematics*

Forward kinematics (FK) computes the end-effector pose $\mathbf{p} \in \mathrm{SE}(3)$ from joint variables $\mathbf{q} \in \mathbb{R}^m$:

$$\mathbf{p} = f(\mathbf{q}) \tag{6}$$

Inverse kinematics (IK) determines the joint variables $\mathbf{q}$ required to achieve a desired end-effector pose $\mathbf{p}$:

$$\mathbf{q} = f^{-1}(\mathbf{p}) \tag{7}$$

For a general 6-DoF platform, the FK is complex and nonlinear, requiring iterative numerical solutions, while the IK is closed-form and efficient. The pose is defined as

$$\mathbf{p} = \begin{bmatrix} x & y & z & \phi & \theta & \psi \end{bmatrix}^T \in \mathbb{R}^6 \tag{8}$$

where $[x, y, z]$ are translations and $[\phi, \theta, \psi]$ are rotations (typically ZYX Euler angles), and the joint vector is

$$\mathbf{q} = \begin{bmatrix} l_1 & l_2 & l_3 & l_4 & l_5 & l_6 \end{bmatrix}^T \in \mathbb{R}^6 \tag{9}$$

representing the six actuator lengths.

2.4. *State-Space System Definition*

A state-space model represents a dynamic system using first-order differential equations, enabling modern control design like LQR and Kalman filter. In continuous time, the state-space model comes defined as follows:

$$\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t), \tag{10}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t), \tag{11}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the state vector, $\mathbf{u} \in \mathbb{R}^m$ is the control input, $\mathbf{y} \in \mathbb{R}^p$ represents the measurements, and $\mathbf{A}$, $\mathbf{B}$, $\mathbf{C}$, $\mathbf{D}$ are constant system matrices of appropriate dimensions.

3. Methods and Equipment

3.1. *6-DoF platform adapted to motors*

The configuration that has been used for creating the 6-DoF platform, is an adapted version that uses 6 motors instead of 6 linear actuators, as shown in Figure 3.

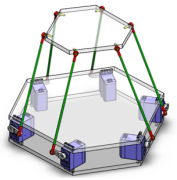
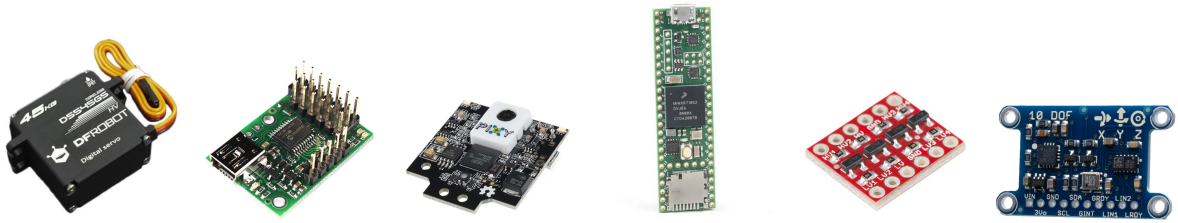


Figure 3: Motor version of 6-DoF platform.

3.2. *Components*

The complete list of components is referenced in Appendix A. The following list names the main electronic components, which create the core of the 6-DoF platform:

- Six DFRobot SER0061 servo motors (Figure 2(a)). Working Voltage range: 6.0 to 8.4V. Maximum torques: 35 kg·cm at 6V, and 45 kg·cm at 8.4V. No-load speeds: 0.12sec/60° at 6V, and 0.11sec/60° at 8.4V.
- Pololu Micro Maestro 6-Channel USB Servo Controller (Figure 2(b)). Drives all six servos. It supports USB, TTL serial, and scripting. Powered via 5–16 V or USB. Configured using Maestro Control Center, with speed and acceleration limits.
- CMUcam5 Pixy 2 Camera (Figure 2(c)). Resolution of 320×200 pixels at 60 fps. Processes images internally. Red pixel detection for ball pose estimation.



(a) Servo Motor (b) Servo Controller (c) Camera (d) Teensy 4.1 (e) LLC (f) IMU

Figure 2: Electronic components

- Teensy 4.1 Microcontroller (Figure 2(d)). Powered by the NXP i.MX RT1062 (600 MHz ARM Cortex-M7) with FPU. It features 1 MB RAM, 8 MB flash (via QSPI), 55 GPIO pins, 14 PWM outputs, and native USB. Compatible with Arduino by using Teensyduino.
- BOB-12009 Bidirectional Logic Level Converter (Figure 2(e)). Interfaces 3.3V Teensy GPIO with 5V servo TTL.
- CNHL Racing Series 2S 6200mAh LiPo battery. 7.4V nominal voltage, 100C continuous discharge. Main power of the system. Dimensions 139×47×25mm. Equipped with Deans connector.
- Adafruit 10-DOF Inertial Measurement Unit (Figure 2(f)). Integrates three sensors on a single I2C breakout: 3-axis gyroscope; 3-axis accelerometer and magnetometer; barometric pressure and temperature sensor. It has an onboard 3.3 V regulator and level shifters support 3–5 V logic.

3.3. ***Software tools***

The software tools used for the 6-DoF platform include:

- PyCharm IDE. Used for: inverse kinematics, simulation, real-time control calculations and user interface generation.
- Arduino IDE with Teensyduino. Used for compiling and uploading C++ code to Teensy.
- Maestro Control Center. Used to configure servo IDs, baud rate, and motion limits.
- PixyMon v2. Used for tuning color tracking and streaming marker centroids.

4. Results and discussion

Two versions of the 6-DoF platform have been obtained in this project. This section presents the design for each of the versions, showing their performance and discussing the reasons to build a second platform.

This section is organized as follows. First, section 4.1 presents the control system used in both versions. Second, section 4.2 shows the self-designed user interface that controls the platforms. Third, section 4.3 presents platform version 1, measuring it's performance and discussing the problems with this version. Finally, section 4.4 presents platform version 2, measuring it's performance and comparing it with version 1.

4.1. ***Control-loop Design***

The control loop is the same for both versions, being the IMU the only difference, as it is implemented in the second version of the platform but not in the first.

A general overview of the control loop design is shown in Figure 4. As it can be observed, the control loop starts with taking the measurements from the camera, which are filtered through the Kalman Filter to obtain a better estimation of the position and velocity of the ball. Then, the position estimations are compared to the reference position, to obtain the position errors. The errors and velocities are then passed to the controller, which can be either PID or LQR. The controller will compute the next roll and pitch for the platform to reduce the errors, and therefore stabilize the ball. Finally, the inverse kinematics takes the roll and pitch from the controller, along with the rest of hardcoded coordinates ($x = 0$, $y = 0$, $z = 0$, $\psi = 0$), and by computing the Z Correction algorithm obtains the desired motor angles, which are passed to the servos or to the simulation.

In addition, for version 2, the IMU estimates the orientations of the base of the platform. These

estimations are subtracted to the roll and pitch that come from the controller, and the result is passed to the inverse kinematics.

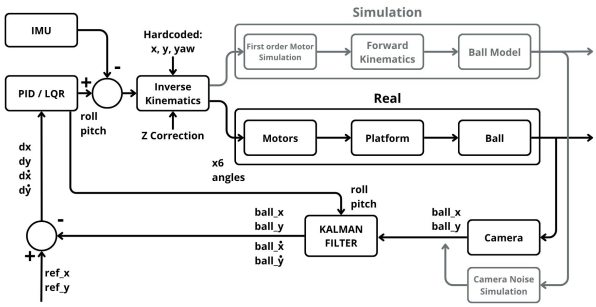


Figure 4: Platform Control Schematic.

Details about Inverse Kinematics, Forward Kinematics, Simulation, Kalman Filter, PID, LQR and Z Correction can be found, respectively, in Appendix B, Appendix C, Appendix D, Appendix E, Appendix F, Appendix G and Appendix H. In addition, the code is presented in Appendix K.

Figure 5 shows the complete electrical and data flow of the real 6-DoF platform.

The battery powers the system via a switch. It supplies the voltage regulator (for the camera, 5V) and the Pololu Micro Maestro servo controller. The Maestro's internal regulator provides 5V to both the Teensy and the Logic Level Converter (LLC).

The Polulu drives the six servo motors with 7.4V power (from battery), and 5V PWM signals, encoding target position, speed, and acceleration.Each servo uses an internal closed-loop controller to reach the commanded pose. The resulting platform motion displaces the ball.

The camera detects the ball's 2D position in real time and transmits it to the Teensy via SPI. Simultaneously, the IMU (powered by the Teensy at 5V) sends accelerometer, gyroscope, and magnetometer data over I2C.

The Teensy forwards all sensor data to the PC via UART (serial). The PC runs the control loop (including Kalman filtering, PID/LQR, inverse kinematics, and Z-correction) as detailed in Figure 4. It computes the next set of motor angles and sends them back to the Teensy over the same serial link.

Finally, the Teensy transmits the target angles to the Pololu Maestro using TTL serial. Since the Teensy uses 3.3V logic and the Maestro expects 5V, a Logic Level Converter (LLC safely shifts both TX and RX lines while preserving signal integrity.

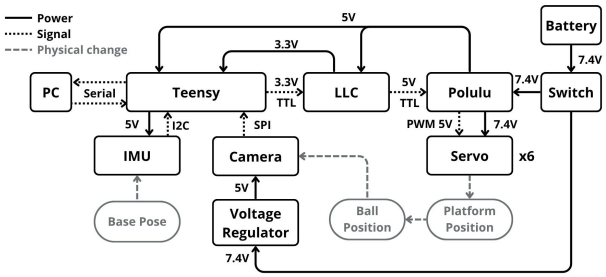


Figure 5: Platform Electronic Schematic.

4.2. Control User Interface

Two graphical user interfaces (GUI) have been developed for the platform. The first, the full feature GUI, has been used for development, and is described in Appendix L. The second, the unified GUI, has been created a simplified version for final use, and is described in this section.

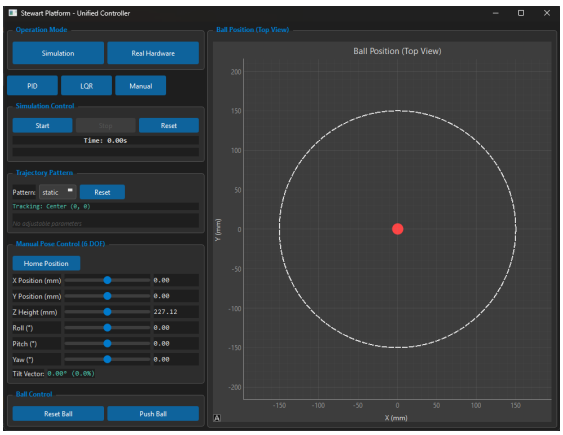


Figure 6: Unified GUI for platform Version 2.

Figure 6 shows the unified GUI for version 2 platform (backward compatible with version 1). The right panel displays a real-time top-view plot of the ball position (red circle) within the platform workspace (white dashed circle), with axes in millimeters. The left panel provides controls for:

- Operation mode: simulation or real.
- Control method: manual, PID or LQR, with tunable parameters for the last two.
- Trajectory configuration: center, circle, figure-eight, or star. Each of them has adjustable parameters, for example the radius and period (time for a full 360º loop) for the circle.
- Sliders visually showing LQR/PID control outputs, when in Manual mode sliders are used to control the platform pose directly.

4.3. ***Platform Version 1***

The first version of the platform can be observed in Figure 7. As a first prototype, this version has been constructed in a smaller size, being the main goal to be able to control it as desired with manual control, PID and LQR.

In addition, this version of the platform has been used to develop a project for robotics students. All the components and steps have been put together in a Github repository, which is explained in Appendix J.

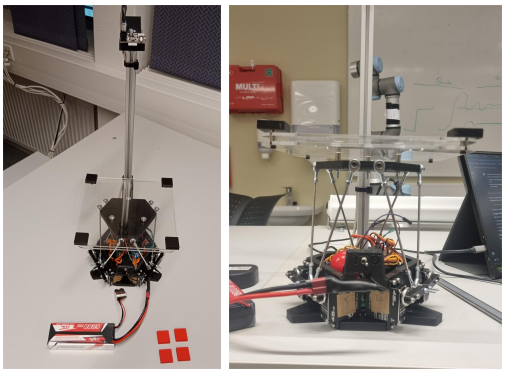


Figure 7: Platform Version 1.

The components that have been used are specified in Appendix A. In addition, a video of version 1 is linked in Appendix I.2.

Figure 8 compares the settling behavior of both controllers from similar initial conditions (approximately 105 mm from center in the X-axis). The PID controller reaches the center significantly faster, reaching steady state around 3.0 seconds with brief overshoot. The LQR controller reaches the center slower, more gradual but faster convergence, taking approximately 1.2 seconds to settle. After settling, both maintain comparable steady-state errors around 10-15 mm, with PID showing slightly more oscillation.

Figure 9 demonstrates trajectory tracking performance on Platform V1. In both simulation and real hardware, PID exhibits noticeable overshoot. While LQR maintains tighter tracking throughout the circular path in both environments. The hardware results show increased noise and irregularity compared to simulation, with PID's deviation more pronounced, consistent with the oscillatory behavior observed in Figure 8.

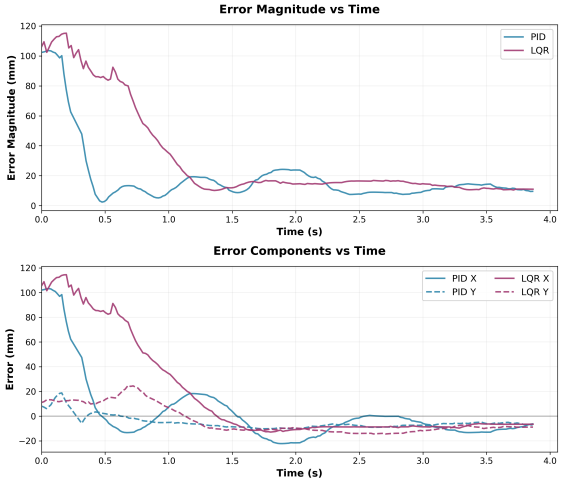


Figure 8: Platform Version 1: Error magnitude comparison of PID and LQR controllers centering the ball from an initial position approximately 105 mm in the X-axis from center. Top: total error magnitude. Bottom: X and Y error components.

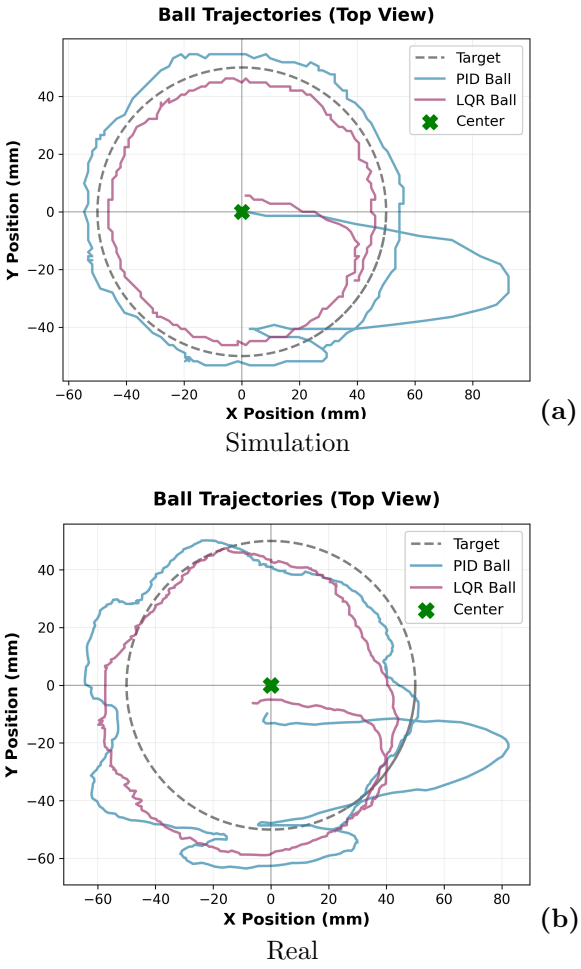


Figure 9: Platform Version 1: Trajectory tracking comparison for a circular reference (radius = 50 mm, period = 10 s) starting from center over 12 seconds. (a) Simulation results. (b) Real world results.

The main problems with platform version 1 are summarized in four main points.

First, the surface in contact with the ball does not let the ball bounce, as the material that makes contact with the ball (acrylic) absorbs all the energy from the bounce. In addition, due to the transparency from the acrylic, false detections may be performed from the red wires in the base of the platform.

Second, the rapid movements from the motors induce vibrations in the base of the platform and in the camera holding stick, which leads to unstable control.

Third, the ball joint limits constrain the workspace boundaries for the platform. In addition, the rods will bend if the platform is moved to the limit configurations, as it can be observed in Figure 10.

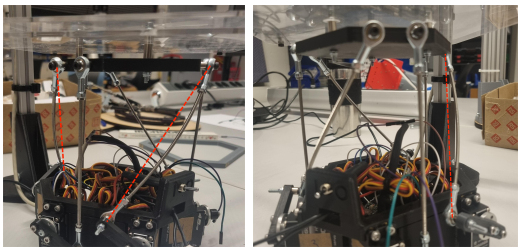


Figure 10: Platform Version 1 bending the rods when moving to a limit position.

Forth and final, there is limited space for the electronic components in the base, which complicates IMU integration.

These problems are the main reason to build a second version of the platform, which is introduced in the following section.

*4.4. **Platform Version 2***

The second version of the platform can be observed in Figure 11. As the second prototype, this version has been constructed with the main goal of fixing the problems related to version 1.

The components that have been used for this second version are specified in Appendix A. In addition, several videos of version 2 are linked in Appendix I.

The total size and length comparisons are shown in Table 1 and 2, respectively. The physical meaning of each of the lengths is defined in Appendix B.

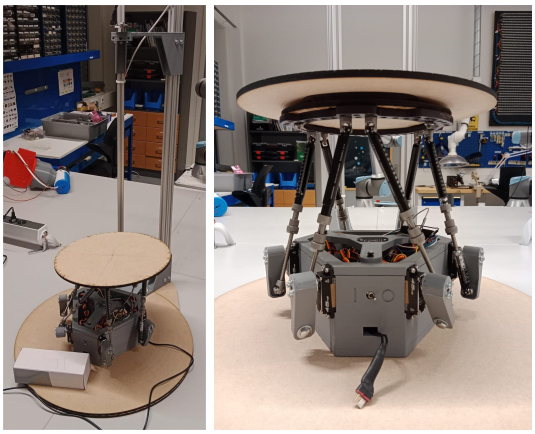


Figure 11: Platform Version 2.

	Width	Lenght	Height
V1	205	330	530
V2	500	600	1020

Table 1: Total size comparison Version 1 vs 2 [mm].

	l_0	d_1	l_f	m	p_1	p_2
V1	73.0	36.9	67.8	12.7	31.8	145.0
V2	116.4	64.8	84.1	12.5	45.4	205.0

Table 2: Lenghts comparison Version 1 vs 2 [mm].

Table 3 presents a comprehensive comparison of workspace limits between the two versions of the platform under different motor angle (α) constraints. Each configuration is represented by two rows, showing positive and negative direction limits for translations (x, y, z in mm) and rotations (ϕ, θ, ψ in degrees).

The second platform (V2) demonstrates superior workspace capabilities compared to the first (V1) at the same 40° motor limit, with approximately 27% larger translational range and 11% greater rotational range.

When operating V2 at the extended 70° servo limit, which is not possible with V1, the workspace increases substantially, reaching ±79 mm in x and y translation and up to ±54° in rotation.

The final configuration, V2(z), employs dynamic z-axis correction as explained in Appendix H, resulting in dramatic workspace expansion: x and y translational limits extend to ±103 mm (a 49% increase), and yaw rotation achieves the full ±90° range (a 67% increase).

	α	x	y	z	ϕ	θ	ψ
V1	40	43	52	23	18	18	38
		-43	-41	-18	-15	-18	-38
V2	40	55	59	32	20	22	38
		-55	-59	-27	-21	-22	-38
V2	70	79	79	48	31	33	54
		-79	-89	-38	-47	-33	-54
V2(z)	70	103	111	48	44	36	90
		-103	-118	-38	-48	-36	-90

Table 3: Limit comparison Version 1 vs 2.

Focusing on the second version of the platform, Figure 12 compares the settling behavior of both controllers from similar initial conditions. The PID controller reaches the center faster, but exhibits significant overshoot and sustained oscillations. The LQR controller demonstrates smoother convergence with minimal overshoot, settling within approximately 1.6 seconds. Both controllers maintain steady state errors below 20 mm, though PID continues oscillating while LQR exhibits more stable behavior.

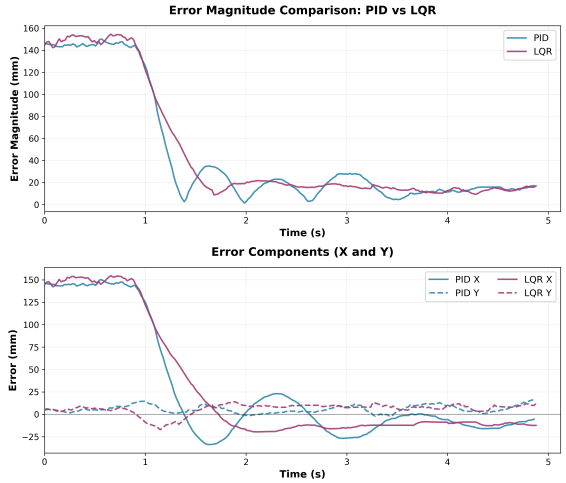


Figure 12: Platform Version 2: Error magnitude comparison of PID and LQR controllers centering the ball from an initial position approximately 150 mm from center. Top: total error magnitude. Bottom: X and Y error components.

Figure 13 demonstrates trajectory tracking performance. The PID controller overshoots in both simulation and experiment, consistent with the oscillatory behavior observed in previous version of the platform and Figure 12. The LQR controller maintains tighter tracking with minimal overshoot, confirming its superior damping characteristics and robustness to model uncertainties.

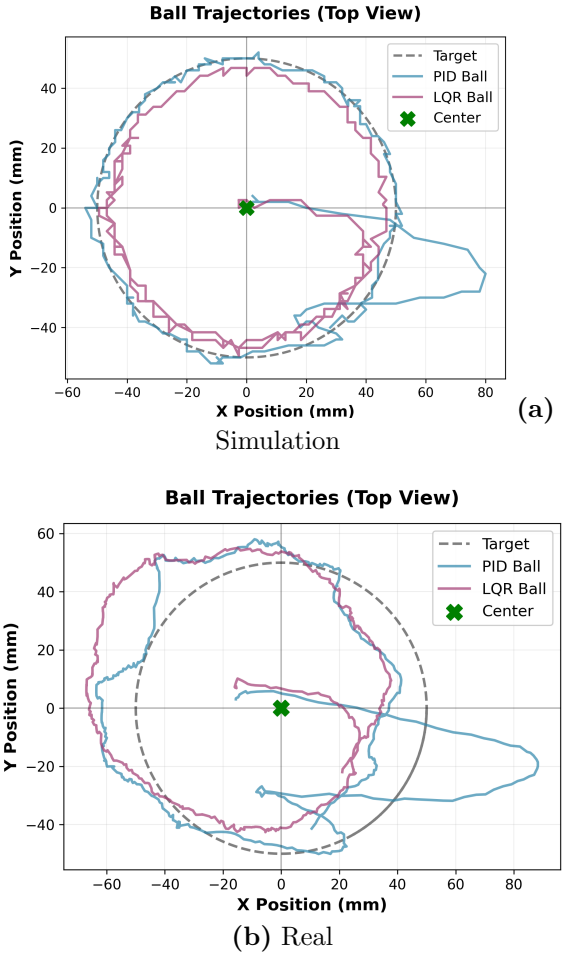


Figure 13: Platform Version 2: Trajectory tracking comparison for a circular reference (radius = 50 mm, period = 10 s) starting from center over 12 seconds. (a) Simulation results. (b) Real world results.

Figure 14 compares roll angle estimates during a 30° step response, showing the commanded angle from inverse kinematics (dashed purple), the estimated platform angle computed from first order servo estimation positions via forward kinematics (solid blue), and the Kalman-filtered IMU estimate (dash-dot red). The IMU-based estimate follows the overall trajectory but exhibits noticeable noise during rapid transitions, particularly visible at $t \approx 8$s where it peaks near 40° in wrong direction before settling. The Kalman filter parameters shown for the IMU estimate are: accelerometer noise of 1.0 m/s^2 and gyroscope noise of 0.0224 rad/s.

Table 4 presents the maximum base tilt angle at which each controller successfully maintains the ball on the platform. Adding IMU tilt correction proves highly advantageous, increasing the maximum tolerable tilt from 10.0° to 24.6° for LQR (146% improvement) and from 14.1° to 27.7° for PID (96% improvement).

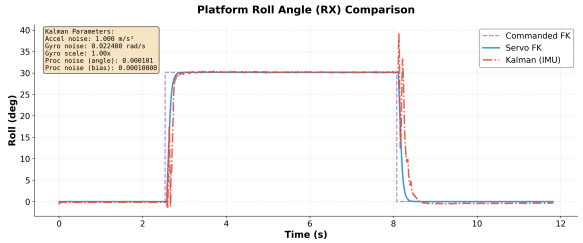


Figure 14: IMU-based platform angle estimation comparison

Controller	Without IMU	With IMU
LQR	10.0°	24.6°
PID	14.1°	27.7°

Table 4: Controller performance comparison.

5. Conclusion

This project successfully developed and implemented a complete real-time control system for a 6-DoF Stewart platform, achieving stable closed-loop performance.

The inverse kinematics solution enabled full 6-DoF motion control while respecting actuator constraints. Combined with dynamic Z-axis optimization, this approach proved effective for real-world operation by expanding the usable workspace by maintaining all servo angles within safe operating limits.

The custom Extended Kalman Filter fusing gyroscope, accelerometer, and magnetometer data provided drift-free orientation estimates.

A high-fidelity simulation environment was developed in Python to support controller design and validation. The model included first-order actuator dynamics, full nonlinear forward kinematics solved via Newton-Raphson, nonlinear state-space representation, platform rigid-body dynamics integrated with Runge-Kutta 4, and a realistic camera measurement noise and grid simulation.

All controllers were first validated in simulation under aggressive trajectories and sensor degradation, achieving close performance to hardware, confirming the simulator's predictive accuracy.

A linearized state-space model of the full nonlinear platform dynamics was derived around the home posture, which was used for LQR controller design, as well as for discrete-time Kalman filter for vision-based pose estimation from the camera.

Both control strategies, PID and LQR, were successfully deployed, tracking complex trajectories (circles, figure-8, or star).

Two platform versions have been developed during the project. In one hand, Version 1 has successfully been used as a project for master students in mechatronics and automation at NTNU Ålesund, for the course of AIS4001 Cybernetics and Robotics. In the other hand, Version 2 has been created by solving mechanical problems from Version 1, with the intention of continuing working on it, and using it as a research tool for real-time nonlinear control and for testing different sensors, as mentioned in Section 6. However, when creating Version 2, the joints between the motors and the platform introduced structural noise, resulting in audible noise and some performance issues.

6. Future Work

For further performance gains and new capabilities, several promising directions are proposed for future work, which are listed as follows:

Eliminate vibrations

As concluded in Section 5, the second hardware revision introduced audible high-frequency vibrations, which were specially notable under aggressive commands. Future work should identify root causes (gear backlash, bearing play, or structural resonance) and eliminate them.

Non-linear Model Predictive Control (NMPC)

The use real-time NMPC with the full nonlinear kinematics and dynamics. The ACADOS Toolkit auto-generated solver can be used here.

Reinforcement Learning (RL) Control

Investigate dual-layer RL policies, one for high-level policy commands desired platform pose, and another for low-level policy directly outputs motor angles or voltages, which bypasses inverse kinematics entirely and may discover non-intuitive, energy-optimal gaits.

High Resolution Stereo Camera

Replace low resolution single-camera 2D detection with a high resolution stereo camera, to be able to estimate 3D ball position. This would enable dynamic catching tasks, as well as the possibility to implement ball bouncing with controlled height and spin.

Event Cameras, Spiking Neural Networks and Neuromorphic Computing

Finally, the platform would be a solid experiment environment for testing with event cameras and spiking neural networks (SNN), training the SNN directly on events and using neuromorphic hardware. This could potentially enable ultra-fast reactive behaviors under low-power consumption.

Appendix A. Component List

The electronic components are the same for both versions of the platform, except the IMU, which is officially incorporated in version 2. See the list of electronic components on Table A.5

Component	Price [NOK]	Sold Units	Needed Units	Cost [NOK]
Servo Motor	548.03	1	6	3288.18
Teensy 4.1	852.10	1	1	852.10
Polulu Maestro	251.28	1	1	251.28
Pixy 2	1515.75	1	1	1515.75
3.3V to 5V LLC	56.25	1	1	56.25
10-DoF IMU	303.68	1	1	303.68
Custom PCB*	325.00	5	1	65.00
LiPo Battery 7.4V	439.00	1	1	439.00
Battery Alarm	115.00	1	1	115.00
Deans T-Plug Male	19.00	1	1	19.00
5V Regulator	150.24	5	1	30.05
Screw Termination	72.71	10	1	7.26
5 Core Cable	1172.74	20	2	117.27
18 AWG Wire	661.00	100	0.5	3.31
Fem. to fem. jumpers	66.14	10	10	66.14
USB A to USB B Micro Cable	52.59	1	2	105.18
USB A to USB B Mini Cable	37.11	1	1	37.11
Male Pins 1 Row	205.9	10	5	102.95
Female Pins 1 Row	60.69	1	4	242.76
Male Pins 2 Rows	32.26	1	2	64.52
Female Pins 2 Rows	49.94	1	2	99.88
Toggle Switch	57.30	1	1	57.30
Total [NOK]				**7838.97**

Table A.5: Electronic component list.

The customized PCB has been designed using Autodesk Fusion. The schematic and the board are shown in Figure A.15.

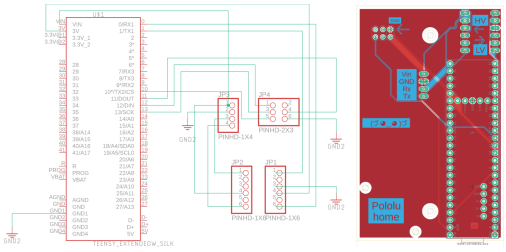


Figure A.15: PCB schematic (left) and board (right).

The mechanical components used in version 1 and 2 of the platform are different, and they are listed, respectively, in Table A.6 and A.7. In addition, the CAD files are attached in `CAD_files.zip`, separated for Version 1 and 2 of the platform. These folders include the CAD files in STL format for 3D printing, as well as the DXF files for laser cutting. See the 3D modeled files in Figure A.16.

Component	Price [NOK]	Sold Units	Needed Units	Cost [NOK]
PLA Filament	637.02	1000	369	235.06
Acrylic 3mm	910.99	1	1	910.99
M3 Threaded Rod	372.20	5	1	74.44
M3 x 27mm Rod End	663.10	1	12	7957.20
M3 x 6mm Inserts	466.76	100	40	186.70
M3 x 5mm Standoff	202.25	50	22	88.99
M3 Locknuts	186.18	100	24	44.68
M3 x 5mm Screws	264.82	100	24	63.56
M3 x 14mm Screws	105.54	50	30	63.32
M3 x 25mm Screws	353.46	50	12	84.83
20 x 20mm Aluminium Bar	374.85	2	0.6	112.46
M5 T-Slot	223.88	10	8	179.10
M5 x 12mm Screw	332.88	50	8	53.26
Total [NOK]				**10054.60**

Table A.6: Mechanical component list for platform version 1.

Component	Price [NOK]	Sold Units	Needed Units	Cost [NOK]
PLA Filament	637.02	1000	916	583.51
Plywood Board 6mm	239.00	1	2	478.00
Smith & Nephew Arms and Circle	10648.68	1	1	5324.34
M3 x 6mm Inserts	466.76	100	32	149.36
M3 x 10mm Standoff	108.88	20	3	16.33
M3 x 5mm Screws	264.82	100	6	15.89
M3 x 14mm Screws	105.54	50	29	61.21
M3 x 25mm Screws	353.46	50	27	190.87
M3 Washers	92.06	100	24	22.09
M2 x 12mm Screws	83.22	100	6	4.99
M2 Nuts	180.88	250	6	4.34
M6 x 30mm Screws	259.38	50	15	77.81
M6 Washers	120.92	100	48	58.04
M6 Locknuts	258.64	100	15	38.80
20 x 20mm Aluminium Bar	374.85	2	2	374.85
M5 T-Slot	223.88	10	24	537.31
M5 x 12mm Screw	332.88	50	24	159.78
Total [NOK]				**8097.54**

Table A.7: Mechanical component list for platform version 2.

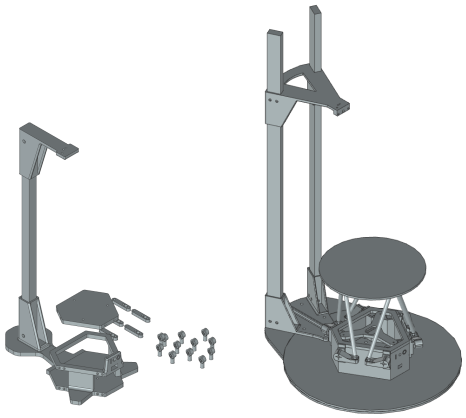


Figure A.16: CAD files for Version 1 (left) and Version 2 (right).

Appendix B. Inverse Kinematics

The Inverse Kinematics (IK) for the 6-DoF platform take as input the desired position and orientation of the platform, and return the six motor angles that achieve the desired pose.

$$\boldsymbol{\alpha} = IK(h_x, h_y, h_z, \phi, \theta, \psi), \tag{B.1}$$

where $\boldsymbol{\alpha} = (\alpha_0 ... \alpha_5)$ are defined as the six motor angles, (h_x, h_y, h_z) are the three position coordinates of the center point of the platform (h), and (ϕ, θ, ψ) are the XYZ Roll-Pitch-Yaw angles of the platform (or equivalently, the Euler ZYX angles).

Let's define the base coordinate system with the origin at the center of the base, as shown in Figure B.17, with the z-axis pointing upwards, y-axis pointing to the front of the platform, and the x-axis pointing at the right.

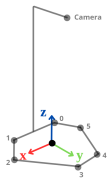


Figure B.17: Origin definition.

Figure B.18 illustrates the base and top planes. l_0 and d_1 are used to define the base plane, and l_f and m to define the top plane. t and w are defined as intermediate variables in the top plane, as defined in Equations B.2 and B.3, respectively.

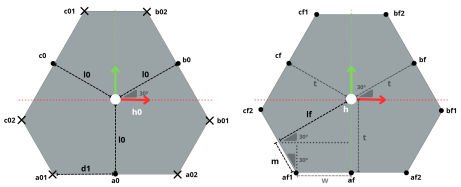


Figure B.18: Base plane (left) and top plane (right).

$$t = l_f sin(30°) + m\ cos(30°) = \frac{l_f}{2} + \frac{m\sqrt{3}}{2} \tag{B.2}$$

$$w = l_f cos(30°) - m\ sin(30°) = \frac{l_f\sqrt{3}}{2} - \frac{m}{2} \tag{B.3}$$

$(a_{01}, a_{02}, b_{01}, b_{02}, c_{01}, c_{02})$ are defined as the points where the motor shafts $(0, 1, 2, 3, 4, 5)$ intersect their respective short motor arms, respectively.

$(a_{f1}, a_{f2}, b_{f1}, b_{f2}, c_{f1}, c_{f2})$ are the points where each of the long motor arms intersect the top plane. P_1 and P_2 are defined as the lengths of the short and the long arm, respectively.

The IK are explained in 3 stages, explained in sections Appendix B.1, Appendix B.2, and Appendix B.3. This explanation is also available in slide format, attached as `6 DOF Ball Balancing Robot - Inverse Kinematics.pdf`

Appendix B.1. ***IK Stage I***

Let $\mathbf{h}$ be the position vector of point h:

$$\mathbf{h} = \begin{bmatrix} h_x & h_y & h_z \end{bmatrix}^T \tag{B.4}$$

Stage I computes $\mathbf{a_f}$, $\mathbf{b_f}$ and $\mathbf{c_f}$, which are defined as:

$$\mathbf{a_f} = \mathbf{h} + \mathbf{ha} \tag{B.5}$$
$$\mathbf{b_f} = \mathbf{h} + \mathbf{hb} \tag{B.6}$$
$$\mathbf{c_f} = \mathbf{h} + \mathbf{hc} \tag{B.7}$$

where,

$$\mathbf{ha} = Rot(\phi, \theta, \psi)\mathbf{ha_i} \tag{B.8}$$
$$\mathbf{hb} = Rot(\phi, \theta, \psi)\mathbf{hb_i} \tag{B.9}$$
$$\mathbf{hc} = Rot(\phi, \theta, \psi)\mathbf{hc_i} \tag{B.10}$$

where,

$$Rot(\phi, \theta, \psi) = Rot(\hat{z}, \phi)Rot(\hat{y}, \theta)Rot(\hat{x}, \psi) \tag{B.11}$$

The rotation matrix for yaw (ψ) about the z-axis is:

$$\mathbf{Rot}(\hat{z}, \psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{B.12}$$

The rotation matrix for pitch (θ) about the y-axis is:

$$\mathbf{Rot}(\hat{y}, \theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{B.13}$$

The rotation matrix for roll (ϕ) about the x-axis is:

$$\mathbf{Rot}(\hat{x}, \phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{B.14}$$

The initial vector $\mathbf{ha}_i$, $\mathbf{hb}_i$ and $\mathbf{hc}_i$ are:

$$\mathbf{ha}_i = \begin{bmatrix} 0 & -t & 0 \end{bmatrix}^T \tag{B.15}$$

$$\mathbf{hb}_i = \begin{bmatrix} \frac{\sqrt{3}t}{2} & \frac{t}{2} & 0 \end{bmatrix}^T \tag{B.16}$$

$$\mathbf{hc}_i = \begin{bmatrix} -\frac{\sqrt{3}t}{2} & \frac{t}{2} & 0 \end{bmatrix}^T \tag{B.17}$$

Appendix B.2. ***IK Stage II***

Stage II uses a_f, b_f and c_f from Stage 1 to compute vectors a_1, a_2, b_1, b_2, c_1, and c_2. These come defined as:

$$\mathbf{a_1} = \mathbf{a_{f1}} - \mathbf{a_{01}} \tag{B.18}$$
$$\mathbf{a_2} = \mathbf{a_{f2}} - \mathbf{a_{02}} \tag{B.19}$$
$$\mathbf{b_1} = \mathbf{b_{f1}} - \mathbf{b_{01}} \tag{B.20}$$
$$\mathbf{b_2} = \mathbf{b_{f2}} - \mathbf{b_{02}} \tag{B.21}$$
$$\mathbf{c_1} = \mathbf{c_{f1}} - \mathbf{c_{01}} \tag{B.22}$$
$$\mathbf{c_2} = \mathbf{c_{f2}} - \mathbf{c_{02}} \tag{B.23}$$

where a_{01}, a_{02}, b_{01}, b_{02}, c_{01}, and c_{02}, as in Figure B.18, are defined as

$$\mathbf{a01} = \begin{bmatrix} -d_1 & -l_0 & 0 \end{bmatrix}^T \tag{B.24}$$
$$\mathbf{a02} = \begin{bmatrix} d_1 & -l_0 & 0 \end{bmatrix}^T \tag{B.25}$$
$$\mathbf{b01} = \begin{bmatrix} \frac{l_0\sqrt{3}}{2} + \frac{d_1}{2} & \frac{l_0}{2} - \frac{d_1\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.26}$$
$$\mathbf{b02} = \begin{bmatrix} \frac{l_0\sqrt{3}}{2} - \frac{d_1}{2} & \frac{l_0}{2} + \frac{d_1\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.27}$$
$$\mathbf{c01} = \begin{bmatrix} -\frac{l_0\sqrt{3}}{2} + \frac{d_1}{2} & \frac{l_0}{2} + \frac{d_1\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.28}$$
$$\mathbf{c02} = \begin{bmatrix} -\frac{l_0\sqrt{3}}{2} - \frac{d_1}{2} & \frac{l_0}{2} - \frac{d_1\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.29}$$

a_{f1}, a_{f2}, b_{f1}, b_{f2}, c_{f1}, and c_{f2} are defined as:

$$\mathbf{a_{f1}} = \mathbf{a_f} + \mathbf{a_f}_{\rightarrow}\mathbf{a_{f1}} \tag{B.30}$$
$$\mathbf{a_{f2}} = \mathbf{a_f} + \mathbf{a_f}_{\rightarrow}\mathbf{a_{f2}} \tag{B.31}$$
$$\mathbf{b_{f1}} = \mathbf{b_f} + \mathbf{b_f}_{\rightarrow}\mathbf{b_{f1}} \tag{B.32}$$
$$\mathbf{b_{f2}} = \mathbf{b_f} + \mathbf{b_f}_{\rightarrow}\mathbf{b_{f2}} \tag{B.33}$$
$$\mathbf{c_{f1}} = \mathbf{c_f} + \mathbf{c_f}_{\rightarrow}\mathbf{c_{f1}} \tag{B.34}$$
$$\mathbf{c_{f2}} = \mathbf{c_f} + \mathbf{c_f}_{\rightarrow}\mathbf{c_{f2}} \tag{B.35}$$

where,

$$\mathbf{a_f}_{\rightarrow}\mathbf{a_{f1}} = Rot(\phi,\theta,\psi)\ \mathbf{a_f}_{\rightarrow}\mathbf{a_{f1}}_i \tag{B.36}$$
$$\mathbf{a_f}_{\rightarrow}\mathbf{a_{f2}} = Rot(\phi,\theta,\psi)\ \mathbf{a_f}_{\rightarrow}\mathbf{a_{f2}}_i \tag{B.37}$$
$$\mathbf{b_f}_{\rightarrow}\mathbf{b_{f1}} = Rot(\phi,\theta,\psi)\ \mathbf{b_f}_{\rightarrow}\mathbf{b_{f1}}_i \tag{B.38}$$
$$\mathbf{b_f}_{\rightarrow}\mathbf{b_{f2}} = Rot(\phi,\theta,\psi)\ \mathbf{b_f}_{\rightarrow}\mathbf{b_{f2}}_i \tag{B.39}$$
$$\mathbf{c_f}_{\rightarrow}\mathbf{c_{f1}} = Rot(\phi,\theta,\psi)\ \mathbf{c_f}_{\rightarrow}\mathbf{c_{f1}}_i \tag{B.40}$$
$$\mathbf{c_f}_{\rightarrow}\mathbf{c_{f2}} = Rot(\phi,\theta,\psi)\ \mathbf{c_f}_{\rightarrow}\mathbf{c_{f2}}_i \tag{B.41}$$

using $Rot(\phi,\theta,\psi)$ as defined in equation B.11. The initial vectors are defined as:

$$\mathbf{a_f}_{\rightarrow}\mathbf{a_{f1}}_i = \begin{bmatrix} -w & 0 & 0 \end{bmatrix}^T \tag{B.42}$$
$$\mathbf{a_f}_{\rightarrow}\mathbf{a_{f2}}_i = \begin{bmatrix} w & 0 & 0 \end{bmatrix}^T \tag{B.43}$$
$$\mathbf{b_f}_{\rightarrow}\mathbf{b_{f1}}_i = \begin{bmatrix} \frac{w}{2} & -\frac{w\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.44}$$
$$\mathbf{b_f}_{\rightarrow}\mathbf{b_{f2}}_i = \begin{bmatrix} -\frac{w}{2} & \frac{w\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.45}$$
$$\mathbf{c_f}_{\rightarrow}\mathbf{c_{f1}}_i = \begin{bmatrix} \frac{w}{2} & \frac{w\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.46}$$
$$\mathbf{c_f}_{\rightarrow}\mathbf{c_{f2}}_i = \begin{bmatrix} -\frac{w}{2} & -\frac{w\sqrt{3}}{2} & 0 \end{bmatrix}^T \tag{B.47}$$

Appendix B.3. ***IK Stage III***

Stage III uses vectors a_1, a_2, b_1, b_2, c_1, and c_2 from Stage II to compute the motor angles $\boldsymbol{\alpha} = (\alpha_0, \alpha_1, \alpha_2, \alpha_3, \alpha_4, \alpha_5)$.

For simplicity, only the equations for α_2 will be derived, using $\mathbf{b_1}$ from Stage II. See Figure B.19 for a visual representation.

$$\alpha_2 = (A + B)\frac{180°}{\pi} \tag{B.48}$$

where angles A defined as:

$$A = acos\left(\frac{\mathbf{b_{1proj}}.\mathbf{b_{01\rightarrow 02}}}{2d_1|\mathbf{b_{1proj}}|}\right) \tag{B.49}$$

Angle B is defined using the law of cosines as:

$$P_{2proj}^2 = |\mathbf{b_{1proj}}|^2 + P_1^2 - 2|\mathbf{b_{1proj}}|P_1 cos(B) \tag{B.50}$$

$$B = acos\left(\frac{|\mathbf{b_{1proj}}|^2 + P_1^2 - P_{2proj}^2}{2|\mathbf{b_{1proj}}|P_1}\right) \tag{B.51}$$

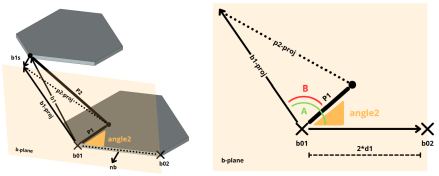


Figure B.19: Stage III visual representation.

An the remaining variables are defined as:

$$|\mathbf{b_{1s}}| = |\mathbf{b_1}.\mathbf{n_b}| \tag{B.52}$$
$$\mathbf{b_{1s}} = \mathbf{n_b}|\mathbf{b_{1s}}| \tag{B.53}$$
$$\mathbf{b_{1proj}} = \mathbf{b_1} + \mathbf{b_{1s}} \tag{B.54}$$
$$|P_{2proj}| = \sqrt{P_2^2 - |\mathbf{b_{1s}}|^2} \tag{B.55}$$

The remaining angles are computed using the same formulas, but substituting $\mathbf{n_b}$, $\mathbf{b_1}$ and $\mathbf{b_{01\rightarrow 02}}$ with the corresponding for each angle.

Appendix C. Forward Kinematics

The Forward Kinematics (FK) for the 6-DoF platform take as input the six motor angles and return the resulting position and orientation of the moving platform.

$$(h_x, h_y, h_z, \phi, \theta, \psi) = FK(\boldsymbol{\alpha}), \tag{C.1}$$

where $\boldsymbol{\alpha} = (\alpha_0 \dots \alpha_5)$ are the six motor angles measured by the encoders, (h_x, h_y, h_z) is the position of the platform center point h, and (ϕ, θ, ψ) are the XYZ Roll-Pitch-Yaw angles (Euler ZYX convention).

The FK are solved iteratively using the Newton-Raphson method with damped least squares to ensure robustness near singular configurations.

The numerical procedure proceeds as follows:

- Use the previous FK solution as the initial guess $\mathbf{q}_0 = \hat{\mathbf{q}}(k-1)$ or $\mathbf{q}_0 = (0, 0, h_{\text{home}}, 0, 0, 0)$ if no prior exists, where h_{home} is the platform's home height.
- Use the initial guess pose q_0, or the pose of the current iteration q_n, to compute the motor angles, using the already defined IK: $\boldsymbol{\alpha}_n = IK(\mathbf{q}_n)$.
- The jacobian $\mathbf{J}(\mathbf{q}_n) \in \mathbb{R}^{6\times 6}$ is computed numerically, once per iteration, using finite differences (forward difference, with 6 IK total calls). For this, a perturbation $\delta = 10^{-2}$ is chosen. For each of the 6 pose variables $j = 0 \dots 5$, $\mathbf{e}_j$ is the unit vector in direction j. Then, each row of the Jacobian $\mathbf{J}_{:,j}$ is defined as:

$$\mathbf{J}(\mathbf{q}_n) = \left.\frac{\partial \boldsymbol{\alpha}}{\partial \mathbf{q}}\right|_{\mathbf{q}_n} \tag{C.2}$$

$$\mathbf{J}_{:,j} = \frac{\boldsymbol{\alpha}(\mathbf{q}_n + \delta \mathbf{e}_j) - \boldsymbol{\alpha}(\mathbf{q}_n)}{\delta} \tag{C.3}$$

  Each entry $J_{i,j}$ tells how much motor angle α_i changes when pose coordinate q_j is varied by one unit.
- Compute the pose update $\Delta\mathbf{q}$ using a damped least squares approach (Levenberg-Marquardt):

$$\Delta\mathbf{q} = \mathbf{J}(\mathbf{q}_n)^{-1}(\boldsymbol{\alpha}_{\text{desired}} - \boldsymbol{\alpha}_n) \tag{C.4}$$

$$\mathbf{J}(\mathbf{q}_n)^{-1} = (\mathbf{J}^\top \mathbf{J} + \lambda \mathbf{I})^{-1} \mathbf{J}^\top \tag{C.5}$$

  where $\lambda = 0.01$ is the damping factor. The system is solved using LU decomposition. If the solve fails due to singularity, λ is increased by a factor of 10 and the iteration is retried.
- Update the pose for the next iteration:

$$\mathbf{q}_{n+1} = \mathbf{q}_n + \Delta\mathbf{q} \tag{C.6}$$

- Apply physical constraints to ensure the solution remains within valid bounds. The horizontal position must satisfy $|(h_x, h_y)| \leq 150$ mm, the vertical position must be within $0 \leq h_z \leq 300$ mm, the tilt angles (roll and pitch) are constrained to $|\phi|, |\theta| \leq 60$, and the yaw angle is limited to $|\psi| \leq 120$. If any constraint is violated, the iteration fails and no solution is returned.
- Iterate until the angle error $|\boldsymbol{\alpha}_{\text{desired}} - \boldsymbol{\alpha}_n| < 10^{-6}$. Final $\mathbf{q}^*$ is the true platform pose corresponding to the target motor angles $\boldsymbol{\alpha}_{\text{desired}}$.

Appendix D. Simulation

To achieve a successful simulation of the 6-DoF platform, the next for points are implemented.

First, each of the servo motors is simulated as a separate first-order system. This is done to simulate the servos in a more realistic way, avoiding to go to the desired angle immediately, which would be the ideal scenario that doesn't match the reality. This is explained in Appendix D.1

Second, a state space model of the position of the ball needs to be used for computing the position of the ball in the next iteration of the simulation. This model is developed for physics and presented in Appendix D.2.

Third, a numerical method needs to be used to compute the position of the ball in the next iteration, as described by the model. For this, Runge-Kutta 4 is used and explained in Appendix D.3.

Finally, once the position of the ball can be computed for the next iteration, a more realistic simulation of the camera is used by introducing measurement noise. This is explained in Appendix D.4.

Appendix D.1. ***Servos as First-Order System***

A first-order linear time-invariant (LTI) system is defined in the frequency domain by the transfer function

$$G(s) = \frac{K}{1 + \tau s}, \tag{D.1}$$

where K is the DC gain and $\tau > 0$ is the time constant. In the time domain, the system satisfies the differential equation:

$$\tau \dot{y}(t) + y(t) = K u(t) \tag{D.2}$$

The unit step response ($u(t) = 1$ for $t \geq 0$) is

$$y(t) = K\left(1 - e^{-t/\tau}\right), \quad t \geq 0, \tag{D.3}$$

reaching $(1 - e^{-1}) \approx 63\%$ of K at $t = \tau$ and $(1 - e^{-3}) \approx 95\%$ at $t = 3\tau$, as shown in Figure D.20.

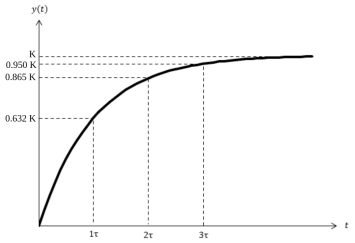


Figure D.20: First-Order System for Step Input.

In top of this, a initial delay τ_i and a maximum angular velocity $\dot{\alpha}$ are defined.

$$\alpha(t) = K\left(1 - e^{-t/\tau}\right), \quad t \geq \tau_i, \tag{D.4}$$

$$\dot{\alpha} \leq \dot{\alpha}_{max} \tag{D.5}$$

where

$$K = 1.0 \tag{D.6}$$

$$\tau = 0.05s \tag{D.7}$$

$$\tau_i = 0.04s \tag{D.8}$$

$$\dot{\alpha}_{max} = 545°/s \tag{D.9}$$

To obtain these values, system identification has been performed by placing an IMU in the top platform. In this way, given a desired platform pose, the IMU measured the platform's position in each timestep in the transition. From these measured poses in the transition, Inverse Kinematics was performed, obtaining servo angles for each timestep of the transition. Finally, these servo angle values were compared to the ones obtained from the simulation, fine tuning the parameters manually until the simulated servo angles matched them.

Appendix D.2. ***State Space Model***

The total kinetic energy of a rotating ball can be computed as:

$$E_k = \frac{1}{2}Mv^2 + \frac{1}{2}I\omega^2 \tag{D.10}$$

where M is the mass of the ball in kg, v is the velocity of the ball in m/s, I is the moment of inertia of the ball in $kg\ m^2$, and ω is the angular velocity of the ball in rad/s. The moment of inertia for a thin spherical shell and the angular velocity can be defined as:

$$I = \frac{2}{3}MR^2 \qquad \omega = \frac{v}{R} \tag{D.11}$$

where R is the radius of the ball in m. The kinetic energy can be rewritten as:

$$E_k = \frac{1}{2}(M + \frac{I}{R^2})v^2 \tag{D.12}$$

In this way, the efficient mass of the ball is defined as:

$$m_{eff} = M + \frac{I}{R^2} = \frac{5}{3}M \tag{D.13}$$

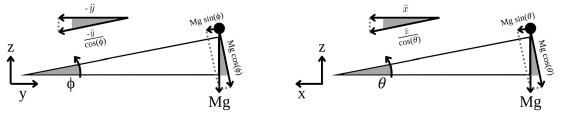


Figure D.21: Ball Model Visual Representation.

By using Newton's first law ($\Sigma F = ma$) with the angles as shown in Figure D.21:

$$Mgsin(\theta) = m_{eff}\frac{\ddot{x}}{cos(\theta)} \tag{D.14}$$

$$Mgsin(\phi) = m_{eff}\frac{-\ddot{y}}{cos(\phi)} \tag{D.15}$$

In this way, the equations that describe the acceleration of the ball in the x and y axis can be derived as follows:

$$\ddot{x} = K_g sin(\theta)cos(\theta) \tag{D.16}$$

$$\ddot{y} = -K_g sin(\phi)cos(\phi) \tag{D.17}$$

where K_g is a constant defined as a function of $g = 9800\frac{mm}{s^2}$:

$$K_g = \frac{3}{5}g = 5880\frac{mm}{s^2} \tag{D.18}$$

The second order equations can be converted into first order differential equations in the form of $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{x_1} \\ \dot{y_1} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ K_g sin(\theta)cos(\theta) \\ -K_g sin(\phi)cos(\phi) \end{bmatrix} \tag{D.19}$$

Appendix D.2.1. Linearization

The objective of having a linearized system is to be able to use it with LQR (Appendix G) and Kalman-Filter for the camera (Appendix E.1).

Equation D.19 is a non-linear system that cannot be represented as $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$. $\dot{\mathbf{x}}(t)$, $\mathbf{A}$, $\mathbf{x}(t)$ and $\mathbf{u}(t)$ can be defined, but not $\mathbf{B}$, due to

the non-linear terms.

$$\dot{\mathbf{x}} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{x_1} \\ \dot{y_1} \end{bmatrix} \quad A = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix} \quad \mathbf{x} = \begin{bmatrix} x \\ y \\ x_1 \\ y_1 \end{bmatrix} \quad \mathbf{u} = \begin{bmatrix} \phi \\ \theta \end{bmatrix} \tag{D.20}$$

This is solved by using the Taylor expansion, linearizing around $\phi = 0$ and $\theta = 0$. The Taylow expansion is given by:

$$f(x) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \cdots \tag{D.21}$$

where $a = 0$ and $f(x) = sin(x)cos(x)$. By considering the expansion up to the second order terms, this leads to:

$$f(x) = f(0) + f'(0)(x) + \frac{f''(0)}{2}(x)^2 \tag{D.22}$$

The derivatives are defined as:

$$f'(x) = cos^2(x) - sin^2(x) \tag{D.23}$$
$$f''(x) = -4cos(x)sin(x) \tag{D.24}$$

leading to:

$$f(0) = 0 \; f'(0) = 1 \; f''(0) = 0 \tag{D.25}$$

In this way $f(x)$ is linearized as $f(x) = x$, redefining $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mathbf{u}(t))$ as:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{x_1} \\ \dot{y_1} \end{bmatrix} = \begin{bmatrix} x_1 \\ y_1 \\ K_g\theta \\ -K_g\phi \end{bmatrix} \tag{D.26}$$

This makes possible to define $\mathbf{B}$ from $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t) + \mathbf{B}\mathbf{u}(t)$. $\dot{\mathbf{x}}(t)$, $\mathbf{A}$, $\mathbf{x}(t)$ as:

$$\mathbf{B} = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & K_g \\ -K_g & 0 \end{bmatrix} \tag{D.27}$$

Appendix D.3. ***Runge-Kutta 4***

The fourth-order Runge-Kutta (RK4) method numerically integrates ordinary differential equations (ODEs) of the form $\dot{\mathbf{x}} = f(t, \mathbf{x})$ over a time step h, providing higher accuracy than Euler's method. The RK4 update from $\mathbf{x}(t)$ to $\mathbf{x}(t+h)$ is defined by:

$$\mathbf{k}_1 = hf(t, \mathbf{x}(t)), \tag{D.28}$$
$$\mathbf{k}_2 = hf(t + \frac{h}{2}\mathbf{x}(t) + \frac{\mathbf{k}_1}{2}) \tag{D.29}$$
$$\mathbf{k}_3 = hf(t + \frac{h}{2}, \mathbf{x}(t) + \frac{\mathbf{k}_2}{2}) \tag{D.30}$$
$$\mathbf{k}_4 = hf(t + h, \mathbf{x}(t) + \mathbf{k}_3) \tag{D.31}$$
$$\mathbf{x}(t+h) = \mathbf{x}(t) + \frac{1}{6}(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4) \tag{D.32}$$

where $\dot{\mathbf{x}} = f(t, \mathbf{x})$ is defined in Equation D.19.

Appendix D.4. ***Camera Noise***

In the simulation, if the camera is not taken into account, the output state from one iteration is used straight-forward as an input for the next iteration. These conditions are ideal, but they don't represent the reality.

For improving the quality of the simulation and making it closer to the real system, the camera is simulated, including two main features.

First, Gaussian noise is included in the measurements from the camera.

$$x_{cam}(k) = x_{sim}(k) + w(k) \tag{D.33}$$
$$y_{cam}(k) = y_{sim}(k) + v(k) \tag{D.34}$$
$$w(k),\, v(k) \sim \mathcal{N}(0, \sigma^2), \tag{D.35}$$

where σ^2 is the noise variance.

In addition, the camera is limited to a pixel grid of 320x200 pixels. The average pixel size of the ball in the platform at the home position is shown in Table D.8. These values define the grid sizes for both platforms.

Version	Pixel Size (mm)	Noise σ (mm)	Resolution
1	1.4	0.4	316×208 px
2	2.0	1.0	316×208 px

Table D.8: Camera simulation parameters.

In this way, when a new ball position is computed from the simulation, the gaussian noise is added, and finally the camera takes it's closest values on the grid.

Appendix E. Kalman Filter

The Kalman Filter provides optimal state estimation $\hat{\mathbf{x}}(k|k)$ for discrete-time linear systems with

Gaussian noise:

$$\mathbf{x}(k+1) = \mathbf{A}\mathbf{x}(k) + \mathbf{B}\mathbf{u}(k) + \mathbf{w}(k) \tag{E.1}$$

$$\mathbf{y}(k) = \mathbf{C}\mathbf{x}(k) + \mathbf{v}(k) \tag{E.2}$$

where $\mathbf{w}(k) \sim \mathcal{N}(0, \mathbf{Q})$ is process noise and $\mathbf{v}(k) \sim \mathcal{N}(0, \mathbf{R})$ is measurement noise, both zero-mean and white. The filter operates in two steps: prediction (equations E.3 and E.4) and correction (equations E.5-E.7).

$$\hat{\mathbf{x}}(k|k-1) = \mathbf{A}\hat{\mathbf{x}}(k-1|k-1) + \mathbf{B}\mathbf{u}(k-1) \tag{E.3}$$

$$\mathbf{P}(k|k-1) = \mathbf{A}\mathbf{P}(k-1|k-1)\mathbf{A}^T + \mathbf{Q} \tag{E.4}$$

$$\mathbf{K}(k) = \mathbf{P}(k|k-1)\mathbf{C}^T \left[\mathbf{C}\mathbf{P}(k|k-1)\mathbf{C}^T + \mathbf{R}\right]^{-1} \tag{E.5}$$

$$\hat{\mathbf{x}}(k|k) = \hat{\mathbf{x}}(k|k-1) + \mathbf{K}(k) \cdot \left[\mathbf{y}(k) - \mathbf{C}\hat{\mathbf{x}}(k|k-1)\right] \tag{E.6}$$

$$\mathbf{P}(k|k) = \left(\mathbf{I} - \mathbf{K}(k)\mathbf{C}\right)\mathbf{P}(k|k-1) \tag{E.7}$$

$\mathbf{K}(k)$ is the Kalman gain, minimizing the trace of the posterior error covariance $\mathbf{P}(k|k)$.

Appendix E.1. ***Kalman Filter for Camera***

The Kalman Filter is applied with camera measurements in the real platform, using $\mathbf{A}$, $\mathbf{x}$, $\mathbf{B}$, and $\mathbf{u}$ as defined in Appendix D.2. The camera can measure x and y coordinates of the ball, so $\mathbf{C}$ and $\mathbf{y}$ are defined as:

$$\mathbf{y} = \begin{bmatrix} x \\ y \end{bmatrix}, \ \mathbf{C} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{E.8}$$

Appendix E.2. ***Kalman Filter for IMU***

The Kalman Filter fuses data from the accelerometer, gyroscope, and magnetometer from the IMU, by estimating the platform's base orientation $\boldsymbol{\theta} = [\phi, \theta, \psi]^T$ (roll, pitch, yaw) and gyroscope bias $\mathbf{b}_\omega \in \mathbb{R}^3$:

$$\mathbf{x}(k) = \begin{bmatrix} \phi(k) & \theta(k) & \psi(k) & \mathbf{b}_\omega(k) \end{bmatrix}^T \tag{E.9}$$

Prediction Step (Gyroscope)

The prediction is made using the measured angular rates (rad/s) from the gyroscope $\boldsymbol{\omega}_m = [p_m, q_m, r_m]^T$:

$$\hat{\mathbf{x}}(k|k-1) = f(\hat{\mathbf{x}}(k-1|k-1)) \tag{E.10}$$

$$\hat{\mathbf{x}}(k|k-1) = \hat{\mathbf{x}}(k-1|k-1) + \begin{bmatrix} \dot{\hat{\boldsymbol{\theta}}}(k-1) \\ \mathbf{0} \end{bmatrix} \Delta t \tag{E.11}$$

where $\dot{\hat{\boldsymbol{\theta}}}(k-1)$ is defined as:

$$\dot{\hat{\boldsymbol{\theta}}}(k-1) = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \mathbf{M}(\phi, \theta)(\boldsymbol{\omega}_m - \mathbf{b}_\omega) \tag{E.12}$$

where $\mathbf{M}(\phi, \theta)$ is the kinematic transformation matrix from body angle rates to Euler angle rates:

$$\mathbf{M} = \begin{bmatrix} 1 & \sin\phi\tan\theta & \cos\phi\tan\theta \\ 0 & \cos\phi & -\sin\phi \\ 0 & \sin\phi/\cos\theta & \cos\phi/\cos\theta \end{bmatrix}_{\hat{\boldsymbol{\theta}}(k-1|k-1)} \tag{E.13}$$

Correction Step (Accelerometer + Magnetometer)

The correction is made by using the measurements from the accelerometer and the magnetometer. The measurement residual $\mathbf{z}(k) \in \mathbb{R}^4$ compares predicted measurements with actual sensor readings, and is defined as:

$$\mathbf{z}(k) = h(\hat{x}(k|k-1)) \tag{E.14}$$

$$\mathbf{z}(k) = \begin{bmatrix} \mathbf{a}_m(k) - \mathbf{h}_a(\hat{\boldsymbol{\theta}}(k|k-1)) \\ \psi_{\text{mag}}(k) - \hat{\psi}(k|k-1) \end{bmatrix} \tag{E.15}$$

The first three values of the residual vector $\mathbf{z}(k)$ are calculated by computing the difference between the acceleration measurements $\mathbf{a_m}$ and the acceleration vector calculated from the gyroscope's predicted angles $\mathbf{h}(\hat{\boldsymbol{\theta}}(k|k-1)$.

The accelerometer measures accelerations (m/s^2) as $\mathbf{a}_m \in \mathbb{R}^3$, ideally from gravity when the platform is stationary. The predicted gravity vector in the body frame $\mathbf{h}(\hat{\boldsymbol{\theta}}(k|k-1)$, is computed using the predicted angles from E.11:

$$\mathbf{h_a}(\hat{\boldsymbol{\theta}}(k|k-1)) = \begin{bmatrix} -g\sin\hat{\theta} \\ g\cos\hat{\theta}\sin\hat{\phi} \\ g\cos\hat{\theta}\cos\hat{\phi} \end{bmatrix}_{\hat{\boldsymbol{\theta}}(k|k-1)} \tag{E.16}$$

The fourth value of the residual vector $\mathbf{z}(k)$ is calculated by computing the difference between the magnetometer's yaw estimate $\psi_{\text{mag}}(k)$ and the gyroscope's predicted yaw angle $\hat{\psi}(k|k-1)$.

The magnetometer measures Earth's magnetic field $\mathbf{m}_m$, which is used to estimate yaw $\psi_{\text{mag}}(k)$ via tilt-compensated heading.

$$\psi_{\text{mag}} = \text{atan2}\Big(m_y\cos\phi + m_z\sin\phi,\ m_x\cos\theta + m_y\sin\theta\sin\phi - m_z\sin\theta\cos\phi\Big)_{\hat{\boldsymbol{\theta}}(k|k-1)} \tag{E.17}$$

Kalman Gain and Update

The final step consist on calculating the Kalman gain $\mathbf{K}(k) \in \mathbb{R}^{6\times4}$ and using it to update $\hat{\mathbf{x}}(k|k)$.

The Kalman Gain $\mathbf{K}(k)$, along the predicted covariance $\mathbf{P}(k|k-1)$, are defined as:

$$\mathbf{P}(k|k-1) = \mathbf{F}(k)\mathbf{P}(k-1|k-1)\mathbf{F}(k)^T + \mathbf{Q} \tag{E.18}$$

$$\mathbf{K}(k) = \mathbf{P}(k|k-1)\mathbf{H}(k)^T \cdot \left[\mathbf{H}(k)\mathbf{P}(k|k-1)\mathbf{H}(k)^T + \mathbf{R}\right]^{-1} \tag{E.19}$$

In the one hand, the State Transition Jacobian $\mathbf{F}(k) \in \mathbb{R}^{6\times6}$ linearizes the prediction model around $\hat{\mathbf{x}}(k-1|k-1)$:

$$\mathbf{F}(k) = \left.\frac{\partial \mathbf{f}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}(k-1|k-1)} = \begin{bmatrix} \mathbf{I}_3 & -\mathbf{M}(\hat{\phi},\hat{\theta})\Delta t \\ \mathbf{0}_{3\times3} & \mathbf{I}_3 \end{bmatrix} \tag{E.20}$$

This is derived as follows:

- Top-left (3×3): $\frac{\partial \boldsymbol{\theta}(k)}{\partial \boldsymbol{\theta}(k-1)} = \mathbf{I}_3 \rightarrow$ angles evolve from previous angles.
- Top-right (3×3): $\frac{\partial \boldsymbol{\theta}(k)}{\partial \mathbf{b}_\omega} = -\mathbf{M}\Delta t \rightarrow$ bias error causes orientation drift via integration.
- Bottom-left: $\frac{\partial \mathbf{b}_\omega(k)}{\partial \boldsymbol{\theta}} = \mathbf{0} \rightarrow$ bias is not affected by angles.
- Bottom-right: $\frac{\partial \mathbf{b}_\omega(k)}{\partial \mathbf{b}_\omega} = \mathbf{I}_3 \rightarrow$ bias modeled as random walk (constant over one step).

In the other hand, the Measurement Jacobian $\mathbf{H}(k) \in \mathbb{R}^{4\times6}$ linearizes the measurement function $\mathbf{h}(\mathbf{x})$ (as defined in E.14) around $\hat{\mathbf{x}}(k|k-1)$:

$$\mathbf{H}(k) = \left.\frac{\partial \mathbf{h}}{\partial \mathbf{x}}\right|_{\hat{\mathbf{x}}(k|k-1)} = \begin{bmatrix} \frac{\partial \mathbf{h}_a}{\partial \phi} & \frac{\partial \mathbf{h}_a}{\partial \theta} & \mathbf{0} & \mathbf{0}_{3\times3} \\ \frac{\partial \psi_{\text{mag}}}{\partial \phi} & \frac{\partial \psi_{\text{mag}}}{\partial \theta} & 0 & \mathbf{0}_{1\times3} \end{bmatrix}. \tag{E.21}$$

where:

$$\frac{\partial \mathbf{h}_a}{\partial \phi} = \begin{bmatrix} 0 \\ g\cos\hat{\theta}\cos\hat{\phi} \\ -g\cos\hat{\theta}\sin\hat{\phi} \end{bmatrix} \tag{E.22}$$

$$\frac{\partial \mathbf{h}_a}{\partial \theta} = \begin{bmatrix} -g\cos\hat{\theta} \\ -g\sin\hat{\theta}\sin\hat{\phi} \\ -g\sin\hat{\theta}\cos\hat{\phi} \end{bmatrix} \tag{E.23}$$

$$\frac{\partial \psi_{\text{mag}}}{\partial \phi} = \frac{m_z\cos\hat{\phi} - m_y\sin\hat{\phi}}{d} \tag{E.24}$$

$$\frac{\partial \psi_{\text{mag}}}{\partial \theta} = \frac{-m_x\sin\hat{\theta} + n_y\sin\hat{\phi}\cos\hat{\theta}}{d\cos^2\hat{\theta}} \tag{E.25}$$

$$n_y = m_y\cos\hat{\phi} + m_z\sin\hat{\phi} \tag{E.26}$$

$$n_x = m_x\cos\hat{\theta} + m_y\sin\hat{\theta}\sin\hat{\phi} - m_z\sin\hat{\theta}\cos\hat{\phi} \tag{E.27}$$

$$d = \sqrt{n_y^2 + n_x^2} \tag{E.28}$$

Derivatives of $\psi_{\text{mag}} = \text{atan2}(n_y,\ n_x)$ come from applying the chain rule:

$$\frac{\partial \psi_{\text{mag}}}{\partial \phi} = \frac{\partial \psi_{\text{mag}}}{\partial n_y}\frac{\partial n_y}{\partial \phi} + \frac{\partial \psi_{\text{mag}}}{\partial n_x}\frac{\partial n_x}{\partial \phi} \tag{E.29}$$

$$\frac{\partial \psi_{\text{mag}}}{\partial \theta} = \frac{\partial \psi_{\text{mag}}}{\partial n_y}\frac{\partial n_y}{\partial \theta} + \frac{\partial \psi_{\text{mag}}}{\partial n_x}\frac{\partial n_x}{\partial \theta} \tag{E.30}$$

where:

$$\frac{\partial \psi_{mag}}{\partial n_y} = \frac{\partial}{\partial n_y}\text{atan2}(n_y, n_x) = \frac{n_x}{n_x^2 + n_y^2} \tag{E.31}$$

$$\frac{\partial \psi_{mag}}{\partial n_x} = \frac{\partial}{\partial n_x}\text{atan2}(n_y, n_x) = -\frac{n_y}{n_x^2 + n_y^2} \tag{E.32}$$

Note that the bias $\mathbf{b}_\omega$ does not appear in $\mathbf{h}(\mathbf{x})$, so the last three columns of $\mathbf{H}(k)$ are zero. In addition, yaw ψ does not affect gravity vector, so the third column in $\frac{\partial \mathbf{h}_a}{\partial \phi}$ and $\frac{\partial \mathbf{h}_a}{\partial \theta}$ are zero.

Finally, $\hat{\mathbf{x}}(k|k)$ is updated, as well as the covariance $\mathbf{P}(k|k)$:

$$\hat{\mathbf{x}}(k|k) = \hat{\mathbf{x}}(k|k-1) + \mathbf{K}(k)\mathbf{z}(k) \tag{E.33}$$

$$\mathbf{P}(k|k) = (\mathbf{I} - \mathbf{K}(k)\mathbf{H}(k))\mathbf{P}(k|k-1) \tag{E.34}$$

Appendix F. PID

The designed PID controller for the 6-DoF platform takes as an input the error values in the x and y axes of the camera (e_x, e_y), and outputs the values of ϕ (roll) and θ (pitch) to control the platform.

A variation in ϕ corresponds to a change in the y position of the ball, while a variation in θ corresponds to a change in the x position. In this way, the control values from PID are computed as follows:

$$\phi = K_P e_y + K_D\frac{de_y}{dt} + K_I\int e_y(\tau)d\tau \tag{F.1}$$

$$\theta = K_P e_x + K_D\frac{de_x}{dt} + K_I\int e_x(\tau)d\tau \tag{F.2}$$

$$\tag{F.3}$$

where e_x, e_y, $\frac{de_x}{dt}$, and $\frac{de_y}{dt}$ are the outputs from the Kalman filter from the camera, as explained in Appendix E.1. The integral terms are computed in discrete time using rectangular integration:

$$\int e_x(\tau)\,d\tau \approx I_x[k] = I_x[k-1] + e_x[k]\Delta t, \tag{F.4}$$

$$\int e_y(\tau)\,d\tau \approx I_y[k] = I_y[k-1] + e_y[k]\Delta t, \tag{F.5}$$

with anti-windup limits $|I_x|, |I_y| \leq 100$ deg·s to prevent saturation.

K_P, K_D and K_I are tunable parameters, which were found to work well with $K_P = 0.1$, $K_D = 0.04$ and $K_I = 0$ (or $K_I = K_P$).

Appendix G. LQR

The designed LQR controller for the 6-DoF platform takes as an input the state $\mathbf{x} = [x, y, \dot{x}, \dot{y}]^T$ (which is the output from the Kalman Filter from the camera, as explained in Appendix E.1), and outputs $\mathbf{u} = [\phi, \theta]^T$ (roll and pitch) to control the platform.

$$\mathbf{u} = -\mathbf{K}\mathbf{x} \tag{G.1}$$

The $\mathbf{A}$ and $\mathbf{B}$ matrices used in LQR to compute $\mathbf{K}$ are the ones defined in Appendix D.2, while $\mathbf{Q}$ and $\mathbf{R}$ are defined as:

$$\mathbf{Q} = \begin{bmatrix} q_x & 0 & 0 & 0 \\ 0 & q_y & 0 & 0 \\ 0 & 0 & q_{\dot{x}} & 0 \\ 0 & 0 & 0 & q_{\dot{y}} \end{bmatrix} \quad \mathbf{R} = \begin{bmatrix} r_\phi & 0 \\ 0 & r_\theta \end{bmatrix} \tag{G.2}$$

q_x, q_y, $q_{\dot{x}}$, $q_{\dot{y}}$, r_ϕ and r_θ are tunable parameters, which were found to work well with $q_x = q_y = 100$, $q_{\dot{x}} = q_{\dot{y}} = 0.01$, and $r_\phi = r_\theta = 0.01$.

Appendix H. Z correction

The purpose of Z-correction is to maximize the usable workspace in x, y, ϕ, θ, and ψ by dynamically adjusting the platform height z so that the six motor angles remain as centered as possible within their stroke limits.

After the controller (PID or LQR) outputs the desired ϕ and θ, the inverse kinematics are first evaluated at the home height:

$$\boldsymbol{\alpha}(0) = IK(0,\ 0,\ h_{\text{home}},\ \phi,\ \theta,\ 0). \tag{H.1}$$

Without Z-correction, these angles are sent directly to the servos. With Z-correction enabled, an optimization procedure re-calculates the motor angles to minimize servo angle imbalance.

Optimization Objective:

The servo angle imbalance at a given Z height is defined as:

$$f(z) = \alpha_{\max}(z) + \alpha_{\min}(z) \tag{H.2}$$

where $\alpha_{\max}$ and $\alpha_{\min}$ are the maximum and minimum servo angles from $\boldsymbol{\alpha} = IK(0, 0, z, \phi, \theta, 0)$. The goal is to find z^* such that $f(z^*) \approx 0$, meaning the servo angles are perfectly balanced around neutral.

Multi-Stage Optimization Algorithm:

The optimization uses a robust multi-stage approach that combines dense sampling with Brent's method for fast convergence:

Stage 1: Valid Range Identification

Perform dense sampling across the initial search range $[z_0 - 30\text{mm}, z_0 + 30\text{mm}]$ with 41 uniformly spaced points. For each sample z_i:

$$f_i = f(z_i) = \alpha_{\max}(z_i) + \alpha_{\min}(z_i) \tag{H.3}$$

Collect all samples where IK succeeds ($|f_i| < 10^5$) as the valid Z range.

Stage 2: Root Finding or Minimization

If a sign change exists within valid samples ($f_{\min} \cdot f_{\max} < 0$):

- Apply Brent's method to find the root: $f(z^*) = 0$
- Convergence tolerance: $|f(z^*)| < 0.1$

If no sign change exists:

- Apply bounded minimization to find: $z^* = \arg\min_z |f(z)|$
- Uses scipy's `minimize_scalar` with bounds method

Stage 3: Extended Range Search (if needed)

If no valid Z is found in the initial range, expand the search to the full mechanical range $[100\text{mm}, 300\text{mm}]$ with coarse sampling (31 points). Return the first valid solution found.

Stage 4: Fallback

If optimization fails, return the best solution encountered during any sampling stage (solution with minimum $|f(z)|$).

Implementation Details

The optimized Z height ensures that for the commanded ϕ and ψ, the motor angles $\boldsymbol{\alpha}$ operate farthest from their physical limits ± 70, which simultaneously enlarges the usable working space for x, y, ϕ, θ, and ψ.

Brent's method is preferred over fixed-point iteration because:

- Guaranteed convergence when a root exists (inverse quadratic interpolation + bisection)
- Typical convergence in 3-5 iterations vs. hundreds for fixed-point
- Robust handling of edge cases (no valid Z, narrow ranges, extreme angles)

Appendix I. Videos

Appendix I.1. ***Unedited videos of Version 2***

LQR bouncing - This video shows platform version 2 using LQR control, with unexpected bouncing of the ball.

IMU tilt correction - This video shows how the IMU in the base of the platform is used to keep the top platform in horizontal orientation.

Ball balancing PID, with IMU tilt correction - This video shows the second version of the platform centering the ball under external rotations of the base, showcasing the capabilities of the IMU.

Appendix I.2. ***Edited videos***

6-DOF Stewart Platform Version 1 Robot - This video shows the results achieved with the first version of the platform, comparing (1) manual control with and without Z Correction, (2) PID and (3) LQR control for centering the ball and following a circular trajectory, and (4) some final clips catching the ball being thrown to the platform. During the video, it can be noticed how the rods are bent during some movements of the platform.

6-DOF Stewart Platform Version 2 Robot - This video shows the results achieved with the second version of the platform, comparing (1) manual control with and without Z Correction, (2) PID vs LQR control for centering the ball, (3) PID vs LQR control for following a circular trajectory, (4) IMU OFF vs IMU ON with PID control for centering the ball, and (5) some final clips catching the ball being thrown to the platform.

Appendix J. Development Pack for Student

As an exercise project for robotic students, the following Github repository has been created: 6-dof-ball-balancing-robot.

This repository is a complete exercise for students, providing all the necessary files for building and controlling version 1 of the platform. The exercise focuses on PID control for centering the ball in the center of the platform, without any further complications of LQR control neither following circular trajectory. As this pack is an exercise for student, the complete code for Inverse Kinematics and for PID control is not solved.

The Github repository contains the next parts:

- Arduino files that need to be downloaded and installed to be able to run the Polulu Maestro Motor Driver and the Pixy 2 Camera.
- Arduino code for: calibrating the camera; sending the servos home for calibrating them; doing a demonstration to check the the IK are correct; and a sample code for implementing PID.
- CAD files in STL format to be able to 3D print them.
- Lasercut files in DXF format to be able to lasercut them.
- Step by step tutorial to build the platform and do the initial configurations.
- List of materials with all the necessary components to buy.

Appendix K. Code

The code that has been developed during the project is shown in two different sections. First, Appendix K.1, shows a simple code that runs on the Teensy, without any external PC computation, which was initially used in the project. Second, Appendix K.2, shows the complete final code, which runs on a PC connected to the Teensy, and includes the full final version of the code with all the features.

Appendix K.1. ***Arduino Code Running on Teensy***

This code is attached inside `1_Arduino_Code_V1.zip`. It can be opened in Arduino IDE, compiling it and uploading it to the Teensy. Once it has been uploaded, it runs directly on the Teensy, without relying on a PC. This approach was initially used in the project, but was later replaced with the approach presented in Appendix K.2, to be able to include more computationally exhaustive features like simulation, GUI and more complex control features.

The following code files are found inside the ZIP folder:

- `Pixy2_Calibrate.ino`

  This file is used for calibrating the center of the camera and the radius of the platform, by using the red markers.

- `Home_Servos.ino`

  This file is used for calibrating the servo motors in the horizontal home position, by fine tunng the offset for each of them.

- `Demo.ino`

  This file is used to check that the Inverse Kinematics work correctly, by sending hard-coded translations and rotations in the three

spatial axes, as well as a predefined circular trajectory if desired.

- `Ball_Balance_PID.ino`

  This file performs ball balance in the center of the platform using PID controller.

- `Ball_Balance_LQR.ino`

  This file partly performs ball balance in the center of the platform using LQR controller. The reason for not being able to achieve balance is that it doesn't include the Kalman Filter for the camera measurements. Using this code it was found that LQR would fail without using a proper filter for the camera measurements (like Kalman Filter) due to the high values of the derivative terms without the filter. After this, the Kalman Filter was directly implemented in the final code (Appendix K.2), and this code was saved as a draft.

Appendix K.2. ***Python Code Running on PC***

This code is attached inside `2_Python_Code.zip`. It can be opened usin any Python IDE (PyCharm, VS Code, etc.), connecting the PC to the Teensy via USB. This code uses the Teensy as an input / output device, running all computation on the PC. The complete code for this project, including all functionalities and features has been developed using this architecture.

The ZIP folder is distributed as follows:

Main Entry Points:

- `full_c.py`: Main controller with complete GUI interface. Supports mode switching (simulation/hardware), controller selection (PID/LQR/Manual), real-time parameter tuning, and data recording. Default settings: Ball Kalman filter enabled, Z-optimization disabled, IMU tilt correction disabled, 4-DOF orientation tracking.
- `min_c.py`: Minimal controller interface for rapid development. Streamlined GUI with essential controls only. Default settings: All advanced features enabled (Ball Kalman filter, Z-optimization, IMU tilt correction, PID Kalman derivative).

Core Modules (`core/`): Foundation algorithms shared by simulation and hardware modes:

- `__init__.py`: Package initialization
- `core.py`: Physical models (servo dynamics, Stewart platform kinematics, ball physics with RK4 integration, camera noise model, trajectory patterns)
- `control_core.py`: Controllers (PID, LQR) and state estimation (Kalman filters for ball position/velocity and IMU orientation with 4-DOF/6-DOF modes)
- `utils.py`: Configuration constants (platform geometry, controller gains, physics parameters, noise parameters, timing)

GUI System (`gui/`): Modular PyQt6 interface system:

- `__init__.py`: Package initialization
- `gui_builder.py`: Layout construction from configuration dictionaries
- `gui_modules.py`: 20+ reusable GUI components (control modules, display modules, configuration modules, hardware modules)

Setup Classes (`setup/`): Base classes for simulation and hardware modes:

- `__init__.py`: Package initialization
- `base_simulator.py`: Foundation class with physics simulation loop, GUI management, controller abstraction
- `base_hardware.py`: Hardware integration with serial communication, thread-based control loop, IMU data streaming

Arduino Code (`arduino/`): Firmware for Teensy microcontroller:

- `control_v1/control_v1.ino`: Initial firmware version with basic servo control and sensor reading
- `control_v2/control_v2.ino`: Updated firmware with improved IMU integration and communication protocol

Documentation and Configuration:

- `README.md`: Complete project documentation including architecture, installation instructions, usage guide, and parameter references
- `requirements.txt`: Python package dependencies. Install using:

  ```
  pip install -r requirements.txt
  ```

- `__init__.py`: Root package initialization

Key Features:

- Dual operation: Identical control algorithms run in simulation or on hardware
- Control modes: PID and LQR controllers with real-time tuning
- State estimation: Kalman filtering for ball dynamics and IMU orientation (4-DOF roll/pitch or 6-DOF with magnetometer yaw tracking)
- Advanced features: Dynamic Z-optimization, IMU tilt correction with yaw control (±45° limit), decoupled magnetometer update
- Real-time performance: 50 Hz control loop with IK calculation caching

Installation:

1. Create Python virtual environment:
   `python -m venv venv`
2. Activate virtual environment:
   `venv\Scripts\activate` (Windows)
3. Install dependencies:
   `pip install -r requirements.txt`

Running the Code:

- Simulation mode: Select "Simulation" in mode selector, choose controller (PID/LQR), configure parameters, click "Start"
- Hardware mode: Connect Teensy via USB (upload appropriate Arduino firmware first), select COM port, initialize IMU, select "Hardware" mode, start controller

Appendix L. Full Feature GUI

The `full_c.py` and `min_c.py`controller implements a dynamic, modular GUI system that adapts to operation modes (simulation/hardware) and controller types (PID/LQR/Manual). In this appendix `full_c.py` configuration will be presented.

Appendix L.1. ***Architecture***

Tokhe GUI uses three core components: **GUIBuilder** (layout factory), **Module Registry** (type-to-class mapping), and **Layout Configuration** (declarative module definitions). The system dynamically instantiates modules based on current mode and controller type.

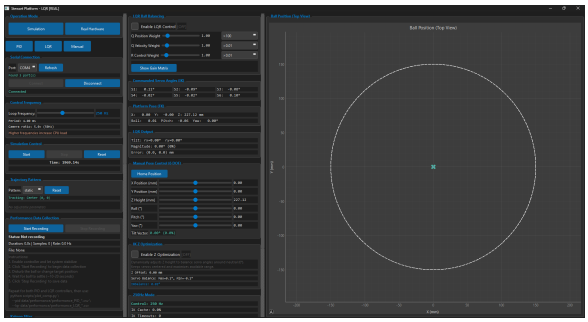


Figure L.22: Complete GUI interface showing LQR control mode with real hardware configuration. Left column contains control/input modules, middle column displays monitoring outputs and controller parameters, right column visualizes ball position tracking.

Appendix L.2. ***Module Registry***

Available GUI modules:

Common Modules	
`mode_selector`	Mode switch
`controller_selector`	Controller selection
`simulation_control`	Start/stop, time
`controller`	Parameter sliders
`trajectory_pattern`	Pattern selection
`kalman_filter`	State estimator
`manual_pose`	DOF control
`servo_angles`	Servo display
`platform_pose`	Platform display
`controller_output`	Tilt, error
`ik_z_optimization`	Dynamic Z
`performance_data`	CSV recording
`debug_log`	Event log

Table L.9: Common GUI modules

Mode-Specific	
Hardware Only:	
`serial_connection`	Port, connect
`control_frequency`	Loop frequency
`performance_stats`	Timing stats
`imu_kalman_parameters`	IMU filter
`imu_motion_detection`	Motion rejection
Simulation Only:	
`ball_control`	Ball physics
`pixy2_camera`	Camera noise

Table L.10: Mode-specific modules

Appendix L.3. ***Layout Configuration***

The `get_layout_config()` method builds a two-column layout. Left column contains control/input modules (mode selectors, serial connection, trajectory settings, filters). Right column contains output/monitoring modules (servo angles, platform pose, controller output, debug log).

Appendix L.4. ***Controller-Specific Widgets***

PID: Adds "Use Kalman Velocity for Derivative" checkbox to use filtered velocity instead of numerical differentiation.

LQR: Adds "Show Gain Matrix" button displaying the 2×4 gain matrix K with state vector $[x, y, v_x, v_y]$.

Appendix L.5. ***Callback System***

Callbacks connect GUI events to methods:

Category	Callbacks
Mode/Controller	`mode_change`, `controller_type_change`
Hardware	`connect`, `disconnect`, `frequency_change`
Kalman	`kalman_enable_change`, `kalman_reset`
Recording	`start_recording`, `stop_recording`
IMU	`imu_tilt_correction_toggle`, `imu_kalman_param_change`

Table L.11: Callback categories

Appendix L.6. ***Mode Switching***

Mode/controller changes trigger complete GUI rebuild: stop simulation, disable controller, clean resources, update mode, recreate configuration, rebuild GUI, reinitialize controller.

Appendix L.7. ***State Updates***

The `update_gui_modules()` method distributes system state to modules:

Category	Keys
Timing	`simulation_time`, `frequency`
Ball	`ball_pos`, `ball_vel`
Platform	`dof_values`, `fk_rotation`
Controller	`controller_output`, `controller_error`
Kalman	`kalman_position`, `kalman_velocity`
IK	`z_optimization_enabled`, `z_offset`
IMU	`imu_orientation`, `imu_bias`

Table L.12: State dictionary categories

Appendix L.8. ***Adding New Modules***

To add a module: (1) Create class in `gui/gui_modules.py`, (2) Add to `module_registry`, (3) Add to `get_layout_config()`, (4) Add callbacks if needed, (5) Add state keys to `update_gui_modules()`.